SunOS リファレンスマニュアル 1 : ユーザーコマンド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817–1782–10
2003 年 4 月

原典: *man pages section 1 : User Commands*

Part No: 817-0659-10

Revision A

Adobe PostScript

030305@5533

目次

はじめに

概要

SunOS リファレンスマニュアルは、初めて SunOS を使用するユーザーやすでにある程度の知識を持っているユーザーのどちらでも対応できるように解説されています。このマニュアルを構成するマニュアルページは一般に参照マニュアルとして作られており、チュートリアルな要素は含んでいません。それぞれのコマンドを実行すると、どのような結果が得られるかについて、詳しく説明されています。なお、各マニュアルページの内容はオンラインでも参照することができます。

このマニュアルは、マニュアルページの内容によっていくつかのセクションに分かれています。各セクションについて以下に簡単に説明します。

- セクション 1 は、オペレーティングシステムで使えるコマンドを説明します。
- セクション 1M は、システム保守や管理用として主に使われるコマンドを説明します。
- セクション 2 は、すべてのシステムコールについて説明します。ほとんどのシステムコールに 1 つまたは複数のエラーがあります。エラーの場合、通常ありえない戻り値が返されます。
- セクション 3 は、さまざまなライブラリ中の関数について説明します。ただし、UNIX システムプリミティブを直接呼び出す関数については、セクション 2 で説明しています。
- セクション 5 は、文字セットテーブルなど他のセクションには該当しないものについて説明します。

以下に、このマニュアルの項目を表記されている順に説明します。ほとんどのマニュアルページが下記の項目からなる共通の書式で書かれていますが、必要でない項目については省略されています。たとえば、記述すべきバグがコマンドにない場合などは、「使用上の留意点」という項目はありません。各マニュアルページの詳細は各セクションの `intro` を、マニュアルページの一般的な情報については `man`(1) を参照してください。

名前　コマンドや関数の名称と概略が示されています。

形式　コマンドや関数の構文が示されています。標準パスにコマンドやファイルが存在しない場合は、フルパス名が示されます。字体は、コマンド、オプションなどの定数にはボールド体 (bold) を、引数、パラメータ、置換文字などの変数にはイタリック体 (Italic) または <日本語訳> を使用しています。オプションと引数の順番は、アルファベット順です。特別な指定が必要な場合を除いて、1 文字の引数、引数のついたオプションの順に書かれています。

以下の文字がそれぞれの項目で使われています。

- [ ]　このかっこに囲まれたオプションや引数は省略できます。このかっこが付いていない場合には、引数を必ず指定する必要があります。
- . . .　省略符号。前の引数に変数を付けたり、引数を複数指定したりできることを意味します (例: ‘ `filename` . . .’)。
- |　区切り文字 (セパレータ)。この文字で分割されている引数のうち 1 つだけを指定できます。
- { }　この大かっこに囲まれた複数のオプションや引数は省略できます。かっこ内を 1 組として扱います。

プロトコル　この項が使われているのは、プロトコルが記述されているファイルを示すサブセクション 3R だけです。パス名は常にボールド体 (bold) で示されています。

機能説明　コマンドの機能とその動作について説明します。実行時の詳細を説明していますが、オプションの説明や使用例はここでは示されていません。対話形式のコマンド、サブコマンド、リクエスト、マクロ、関数などに関しては「使用法」で説明します。

IOCTL　セクション 7 だけに使用される項です。`ioctl`(2) システムコールへのパラメータは `ioctl` と呼ばれ、適切なパラメータを持つデバイスクラスのマニュアルページだけに記載されています。特定のデバイスに関する `ioctl` は、(そのデバイスのマニュアルページに) アルファベット順に記述されています。デバイスの特定のクラスに関する `ioctl` は、`mtio`(7I) のように `io` で終わる名前が付いているデバイスクラスのマニュアルページに記載されています。

オプション　各オプションがどのように実行されるかを説明しています。「形式」で示されている順に記述されています。オプションの引数はこの項目で説明され、必要な場合はデフォルト値を示します。

オペランド　コマンドのオペランドを一覧表示し、各オペランドがコマンドの動作にどのように影響を及ぼすかを説明しています。

出力　コマンドによって生成される出力 (標準出力、標準エラー、または出力ファイル) を説明しています。

戻り値　値を返す関数の場合、その値を示し、値が返される時の条件を説明しています。関数が 0 や -1 のような一定の値だけを返す場合は、値と説明の形で示され、その他の場合は各関数の戻り値について簡単に説明しています。void として宣言された関数はこの項では扱いません。

エラー　失敗の場合、ほとんどの関数はその理由を示すエラーコードを errno 変数の中に設定します。この項ではエラーコードをアルファベット順に記述し、各エラーの原因となる条件について説明します。同じエラーの原因となる条件が複数ある場合は、エラーコードの下にそれぞれの条件を別々のパラグラフで説明しています。

使用法　この項では、使用する際の手がかりとなる説明が示されています。特定の決まりや機能、詳しい説明の必要なコマンドなどが示されています。組み込み機能については、以下の小項目で説明しています。

コマンド
修飾子
変数
式
入力文法

使用例　コマンドや関数の使用例または使用方法を説明しています。できるだけ実際に入力するコマンド行とスクリーンに表示される内容を例にしています。例の中には必ず example% のプロンプトが出てきます。スーパーユーザーの場合は example# のプロンプトになります。例では、その説明、変数置換の方法、戻り値が示され、それらのほとんどが「形式」、「機能説明」、「オプション」、「使用法」の項からの実例となっています。

環境　コマンドや関数が影響を与える環境変数を記述し、その影響について簡単に説明しています。

終了ステータス　コマンドが呼び出しプログラムまたはシェルに返す値と、その状態を説明しています。通常、正常終了には 0 が返され、0 以外の値はそれぞれのエラー状態を示します。

ファイル　マニュアルページが参照するファイル、関連ファイル、およびコマンドが作成または必要とするファイルを示し、各ファイルについて簡単に説明しています。

属性　属性タイプとその対応する値を定義することにより、コマンド、ユーティリティ、およびデバイスドライバの特性を一覧しています。詳細は attributes(5) を参照してください。

関連項目　関連するマニュアルページ、当社のマニュアル、および一般の出版物が示されています。

診断	エラーの発生状況と診断メッセージが示されています。メッセージはボールド体 (bold) で、変数はイタリック体 (Italic) または <日本語訳> で示されており、C ロケール時の表示形式です。
警告	作業に支障を与えるような現象について説明しています。診断メッセージではありません。
注意事項	それぞれの項に該当しない追加情報が示されています。マニュアルページの内容とは直接関係のない事柄も参照用に扱っています。ここでは重要な情報については説明していません。
使用上の留意点	すでに発見されているバグについて説明しています。可能な場合は対処法も示しています。

SunOS リファレンスマニュアル 1 : ユーザーコマンド

名前

Intro, intro – コマンドおよびアプリケーションプログラムの序章

機能説明

本セクションでは、当該オペレーティングシステムで使用できるコマンドについて アルファベット順に説明します。

特別な範疇に収まるコマンドについては、以下のように区別しています。

1B	SunOS/BSD 互換性パッケージにだけ存在するコマンド。 詳細は、『*SunOS/BSD Compatibility Package*』を参照してください。
1C	他のシステムと通信するためのコマンド
1F	フォームとメニュー言語インタプリタ (FMLI) に関連するコマンド
1S	SunOS システムにだけ存在するコマンド

その他のセクションについて

詳細は、SunOS リファレンスマニュアルの各セクションを参照してください。

- システム管理のコマンドについては、本マニュアルのセクション 1M を参照してください。
- ファイルの形式については、本マニュアルのセクション 4 を参照してください。
- 公式に使用できるファイルやそのほかのさまざまな情報については、 本マニュアルのセクション 5 を参照してください。
- コンピュータのデモンストレーションについては、 本マニュアルのセクション 6 を参照してください。

上記のコマンドや使用方法については、 以下のマニュアルを参照してください。

- 『*OpenWindows* ユーザーズガイド *(*上級編*)*』

マニュアルページのコマンドの構文

特に説明しないかぎり、マニュアルページの「形式」の項で記述されるコマンドは、以下の構文に従って、 オプションやその他の引数を受け付けます。 そして、以下のように解釈されなければなりません。

name [–*option*...] [*cmdarg*...]

[ ]	必須でない *option* (オプション) や *cmdarg* (引数) を囲みます。
...	*option* (オプション) や *cmdarg* (引数) が複数回発生することを意味します。
name	実行可能ファイルの名前です。
{ }	中括弧で囲まれた、オプションまたは引数 (および両方) は独立しており、 括弧内のすべてを 1 つの単位として扱わなければなりません。
option	(常に “–” が先行します。) *noargletter*... または、 *argletter optarg*[, ...]
noargletter	引数が必要ないオプション 1 文字を表します。 複数の *noargletter* を指定する場合、 1 つの “–” の後にまとめて指定できます (後述のルール 5)。

argletter 引数が必要なオプション 1 文字を表します。

optarg *argletter* に必要なオプション引数 (文字列) です。 複数の *optargs* を *argletter* に指定する場合、コンマで区切る、または、タブか空白文字で区切って 引用符で囲まなければなりません (後述のルール 8)。

cmdarg “-” で始まらないパス名 (または他のコマンドの引数)。 “-” だけを指定すると標準入力を表します。

コマンド構文規格：ルール

ここで説明するコマンド構文のルールは、 既存のコマンドすべてに適用されているわけではありません。 しかし、新規のコマンドはすべてこのルールに従う予定です。 すべてのシェルプロシージャは `getopts`(1) を使って、定位置パラメタを構文解析し、 オプションが合法かどうかチェックしなければなりません。 `getopts`(1) は、以下に説明するルール 3 から 10 までをサポートします。 その他の規則については、コマンド自身がチェックしなければなりません。

1. コマンド名 (上記の *name*) は、 2 文字から 9 文字までの長さでなければなりません。
2. コマンド名は、小文字と数字だけで構成されなければなりません。
3. オプション名 (上記の *option*) は、 1 文字でなければなりません。
4. オプションには “-” が先行しなければなりません。
5. 引数なしのオプションは、 1 つの “-” の後に複数個まとめて指定できます。
6. オプションとオプションの最初の引数 (上記の *optarg*) の間は、 タブか空白文字で区切らなければなりません。
7. オプションの引数は、必ず指定しなければなりません。
8. オプションに複数のオプションの引数が続く場合、 それぞれをコンマで区切る、または、タブか空白文字で区切って 引用符で囲まなければなりません (たとえば、`-o xxx,z,yy` や `-o "xxx z yy"` など)。
9. コマンド行上では、オプションはオペランド (上記の *cmdarg*) より前に指定しなければなりません。
10. “--” を使って、オプションの終わりを示すことができます。
11. オプションの相対的な順番は問題になりません。
12. オペランド (上記の *cmdarg*) の相対的な順番は、 その位置によって、コマンドが決めた意味に影響します。
13. “-” の前後に空白文字を指定した場合 (“-”だけを指定した場合) 、標準入力を表します。

属性

このセクション中にリストされている属性については `attributes`(5) のマニュアルページを参照してください。

関連項目

`getopts`(1), `wait`(1), `exit`(2), `getopt`(3C), `wait`(3UCB), `attributes`(5)

診断

終了時、すべてのコマンドは状態を表す 2 バイトを返します。 1 つは、システムから提供され、終了の原因を示します。 もう 1 つ (正常な終了において) は、プログラムから提供されます (`wait`(3UCB) および `exit`(2) を参照)。 前のバイトが 0 の場合、正常な終了を表します。 後のバイトが 0 の場合、正常な実行を表します。 後のバイトがゼロでない場合、 間違った引数を指定した、 または不良で受け入れることができないデータを指定した などの障害を示します。 このバイトは、「終了コード」、「終了状態」、「リターンコード」など さまざまな呼ばれ方をします。そして、 特別な使い方がある場合に限って説明されます。

警告

ヌル文字を含むファイルを処理しているときに、 予測していなかった結果を出すコマンドがあります。 通常このようなコマンドはテキスト入力行を文字列として扱っているので、 行中のヌル文字 (つまり文字列の終端) に出合うと混乱してしまうからです。

コマンド一覧

名前	説明
`Intro`(1)	コマンドおよびアプリケーションプログラムの序章
`acctcom`(1)	プロセスアカウンティングファイルの検索と出力
`adb`(1)	汎用デバッガ
`addbib`(1)	参考文献データベース (bibliographic database) の作成または拡張
`alias`(1)	コマンドまたはコマンド群の別名または省略形の生成と削除
`answerbook2`(1)	オンライン文書システム
`apptrace`(1)	Solaris 共有ライブラリに対するアプリケーション関数呼び出しの追跡
`apropos`(1)	特定のキーワードを持つマニュアルページの検索
`ar`(1)	移植可能なアーカイブまたはライブラリの保守
`arch`(1)	現在のホストのアーキテクチャの表示
`as`(1)	アセンブラ
`asa`(1)	FORTRAN キャリッジ制御出力の印刷可能形式への変換
`at`(1)	指定した時刻にコマンドを実行
`atq`(1)	指定時刻に実行するために待ち行列に入っているジョブの表示
`atrm`(1)	at または batch によってスプールされたジョブの削除
`audioconvert`(1)	オーディオファイル形式の変換
`audioplay`(1)	オーディオファイルの再生
`audiorecord`(1)	オーディオファイルの記録
`awk`(1)	パターン走査およびパターン処理の言語

banner(1)	ポスターの作成
basename(1)	パス名の部分的な抽出
batch(1)	at(1) を参照
bc(1)	任意精度の演算言語
bdiff(1)	大規模ファイルの diff
bfs(1)	大型ファイル用のスキャナ
bg(1)	jobs(1) を参照
break(1)	while、for、foreach、until の各ループ制御から抜け出す、または続行するためのシェル組み込み関数
cal(1)	カレンダの表示
calendar(1)	リマインダーサービス
cancel(1)	印刷要求の取消し
case(1)	shell_builtins(1) を参照
cat(1)	ファイルの連結と表示
cd(1)	現在の作業用ディレクトリの変更
cdrw(1)	CD の読み取りと書き込み
chdir(1)	cd(1) を参照
checkeq(1)	eqn(1) を参照
checknr(1)	nroff と troff 用の入力ファイルをチェックして誤りを報告
chgrp(1)	ファイルのグループ所有権の変更
chkey(1)	ユーザーの Secure RPC 鍵ペアの変更
chmod(1)	ファイルのアクセス権モードの変更
chown(1)	ファイルの所有者の変更
ckdate(1)	日付の入力要求とその検証
ckgid(1)	グループ ID の入力要求とその検証
ckint(1)	整数値の入力要求とその検証
ckitem(1)	メニューの作成、およびメニュー項目の入力要求とその検証
ckkeywd(1)	キーワードの入力要求とその検証
ckpath(1)	パス名の入力要求とその検証
ckrange(1)	整数の入力要求とその検証

ckstr(1)	文字列の入力要求とその検証
cksum(1)	ファイルのチェックサムとサイズの出力
cktime(1)	時刻の入力要求とその検証
ckuid(1)	ユーザー ID の入力要求とその検証
ckyorn(1)	yes/no の入力要求とその検証
clear(1)	端末画面の消去
cmp(1)	2 つのファイルの比較
col(1)	逆方向改行フィルタ
comm(1)	2 つのファイルに共通な行または共通でない行の表示
command(1)	単純コマンドの実行
compress(1)	ファイルの圧縮、圧縮解除、または圧縮解除結果の表示
continue(1)	break(1) を参照
cp(1)	ファイルのコピー
cpio(1)	アーカイブからのファイルの抽出または復元
cpp(1)	C 言語プリプロセッサ
crontab(1)	ユーザーの crontab ファイル
crypt(1)	ファイルの暗号化または復号化
csh(1)	C に似た構文を持つシェル・コマンドインタプリタ
csplit(1)	指定した文脈に従ってファイルを分割
ctags(1)	ex および vi で使用するタグファイルの作成
cut(1)	ファイルの各行から選択されたフィールドをカット
date(1)	日付と時刻の出力と設定
dc(1)	電卓機能
deroff(1)	nroff/troff、tbl、および eqn 構造体の削除
dhcpinfo(1)	DHCP を介して受信されたパラメータ値の表示
diff(1)	2つのファイルの比較
diff3(1)	3 つのファイルの内容比較
diffmk(1)	troff 入力ファイル間の差分の表示
digestp(1)	mailp(1) を参照
dircmp(1)	ディレクトリの比較

dirname(1)	basename(1) を参照
dirs(1)	cd(1) を参照
dis(1)	オブジェクトコードの逆アセンブラ
disable(1)	enable(1) を参照
dispgid(1)	すべての有効なグループ名リストの表示
dispuid(1)	すべての有効なユーザー名リストの表示
dos2unix(1)	DOS 形式から ISO 形式へのテキストファイルの変換
download(1)	ホスト上の PostScript フォントのダウンローダ
dpost(1)	PostScript プリンタ用の troff ポストプロセッサ
du(1B)	ディレクトリまたはファイルごとに使用されるディスクブロックの数の表示
dump(1)	オブジェクトファイルの選択部分のダンプ
dumpcs(1)	現在のロケールのコードセット一覧表を表示
echo(1)	引数の出力
ed(1)	テキストエディタ
edit(1)	テキストエディタ (ex の臨時ユーザー用の変形版)
egrep(1)	完全な正規表現を使用したファイル内のパターン検索
eject(1)	ドライブからの媒体 (CD-ROM やフロッピーなど) の取り出し
elfdump(1)	オブジェクトファイルの選択部分のダンプ
enable(1)	LP プリンタを使用可能または不可能に変更
env(1)	コマンド実行のための環境の設定
eqn(1)	数学的記述のタイプセット
errange(1)	ckrange(1) を参照
errdate(1)	ckdate(1) を参照
errgid(1)	ckgid(1) を参照
errint(1)	ckint(1) を参照
erritem(1)	ckitem(1) を参照
error(1)	ソース行の右側へのコンパイラエラーメッセージの挿入
errpath(1)	ckpath(1) を参照
errstr(1)	ckstr(1) を参照

errtime(1)	cktime(1) を参照
erruid(1)	ckuid(1) を参照
erryorn(1)	ckyorn(1) を参照
eval(1)	exec(1) を参照
ex(1)	テキストエディタ
exec(1)	他のコマンドを実行するためのシェル組み込み関数
exit(1)	シェルの連続した処理を分岐して実行するための シェル組み込み関数
expand(1)	タブ文字を空白文字に展開する、またはその反対
export(1)	set(1) を参照
expr(1)	引数を式として評価する
exstr(1)	ソースファイルからの文字列の抽出
face(1)	FACE (Framed Access Command Environment Interface) 用の実行可能ファイル
factor(1)	素因数の分解
fc(1)	history(1) を参照
fdformat(1)	フロッピーディスクまたは PCMCIA メモリーカードの フォーマット
fg(1)	jobs(1) を参照
fgrep(1)	ファイル内の固定文字列のみを検索
file(1)	ファイルタイプの判別
file(1B)	ファイルの内容を検査してファイルタイプを識別
filep(1)	mailp(1) を参照
filesync(1)	通常ファイル、ディレクトリファイル、特殊ファイル の同期
filofaxp(1)	mailp(1) を参照
find(1)	ファイルの検索
finger(1)	ローカルユーザーとリモートユーザーに関する情報の 表示
fmli(1)	FMLI の呼び出し
fmt(1)	簡単なテキストフォーマッタ
fmtmsg(1)	stderr またはシステムコンソールへのメッセージ表示

fnattr(1)	FNS 名前付きオブジェクトに関連する属性の更新と検査
fnbind(1)	FNS 名への参照のバインド
fnlist(1)	FNS コンテキストにバインドされている名前と参照の表示
fnlookup(1)	NFS 名にバインドされている参照の表示
fnrename(1)	FNS 名のバインド名の変更
fnsearch(1)	指定した属性による FNS オブジェクトの検索
fnunbind(1)	FNS 名からの参照のバインド解除
fold(1)	行の折り返し用フィルタ
for(1)	shell_builtins(1) を参照
foreach(1)	shell_builtins(1) を参照
franklinp(1)	mailp(1) を参照
ftp(1)	ファイル転送プログラム
function(1)	shell_builtins(1) を参照
gencat(1)	書式付きメッセージカタログの生成
getopt(1)	コマンドオプションの解析
getoptcvt(1)	コマンドオプションを解析するために getopts に変換
getopts(1)	ユーティリティオプションの解析
gettxt(1)	テキスト文字列のメッセージデータベースからの検索
glob(1)	単語リストを展開するためのシェル組み込み関数
goto(1)	exit(1) を参照
grep(1)	ファイルにおけるパターンの検索
groups(1)	ユーザーのグループメンバーシップの出力
hash(1)	ディレクトリの内容の内部ハッシュテーブルの評価
hashstat(1)	hash(1) を参照
head(1)	ファイルの最初の数行の表示
helpdate(1)	ckdate(1) を参照
helpgid(1)	ckgid(1) を参照
helpint(1)	ckint(1) を参照
helpitem(1)	ckitem(1) を参照
helppath(1)	ckpath(1) を参照

helprange(1)	ckrange(1) を参照
helpstr(1)	ckstr(1) を参照
helptime(1)	cktime(1) を参照
helpuid(1)	ckuid(1) を参照
helpyorn(1)	ckyorn(1) を参照
history(1)	コマンドの履歴リストの処理
iconv(1)	コードセット変換ユーティリティ
if(1)	shell_builtins(1) を参照
jobs(1)	プロセスの実行の制御
join(1)	リレーショナルデータベース演算子
jsh(1)	sh(1) を参照
kill(1)	プロセスの終了またはシグナル送出
ksh(1)	Korn シェル。標準/制限付きコマンドとプログラミング言語
ld(1)	オブジェクトファイル用リンカー
ldd(1)	実行可能ファイルまたは共有オブジェクトの動的依存関係の表示
let(1)	1 つ以上の算術式を評価するためのシェル組み込みコマンド
lex(1)	字句解析プログラムの生成
limit(1)	現在のシェルとそのシェルから起動されたプロセスで利用できるシステム資源の制限値を設定または取得
ln(1)	ファイルへのハードリンクまたはシンボリックリンクの作成
locale(1)	ロケール固有の情報の取得
login(1)	システムへのログイン
logname(1)	ユーザーのログイン名の応答
logout(1)	ログインのセッションから抜け出すためのシェル組み込み関数
lp(1)	印刷要求の送信
lpr(1B)	印刷要求の提出
lpstat(1)	LP 印刷サービスの状態に関する情報の表示
ls(1)	ディレクトリの内容を一覧表示

mail(1)	メールの読み取りまたはユーザーへのメールの送信
mailp(1)	テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド
man(1)	マニュアルページの表示
mesg(1)	メッセージの許可または禁止
mkdir(1)	ディレクトリの作成
mkmsgs(1)	gettxt でアクセスできるメッセージファイルの作成
more(1)	テキストファイルの表示またはページング
mp(1)	テキストからプリンタ記述言語 (PDL) へのプリティプリントフィルタ
mt(1)	磁気テープの制御
mv(1)	ファイルの移動
nawk(1)	パターン走査およびパターン処理の言語
neqn(1)	eqn(1) を参照
netscape(1)	Solaris 版 Netscape Communicator の起動
newform(1)	テキストファイル形式の変更
newgrp(1)	新たなグループへのログイン
news(1)	新規項目の表示
newsp(1)	mailp(1) を参照
nice(1)	コマンドを変更されたスケジューリング優先順位で実行
nm(1)	オブジェクトファイルのネームリストを表示
nohup(1)	ハングアップおよび停止の影響を受けないコマンドの実行
notify(1)	jobs(1) を参照
nroff(1)	ディスプレイまたはラインプリンタ用に文書をフォーマット
od(1)	8 進ダンプ
onintr(1)	trap(1) を参照
pack(1)	ファイルの圧縮および復元
page(1)	more(1) を参照
passwd(1)	ログインパスワードおよびパスワード属性の変更
paste(1)	複数のファイルの対応する行および以降の行のマージ

pcat(1)	pack(1) を参照
perl(1)	抽出および出力を行う言語 (Practical Extraction and Report Language)
pg(1)	CRT 用のファイル閲覧フィルタ
pgrep(1)	名前または他の属性によるプロセスの検出またはシグナル送信
pkginfo(1)	ソフトウェアパッケージ情報の表示
pkill(1)	pgrep(1) を参照
popd(1)	cd(1) を参照
pr(1)	ファイルの出力
print(1)	画面またはウィンドウに文字を出力するシェル組み込み関数
priocntl(1)	指定したプロセスのスケジューリングパラメータの表示または設定
ps(1)	プロセスの状態報告
pushd(1)	cd(1) を参照
pwd(1)	作業中のディレクトリの出力
rcp(1)	リモートファイルコピー
rdist(1)	リモートファイル配布プログラム
read(1)	標準入力から 1 行を入力
readonly(1)	再表示の場合、もとの値の書き換えを防ぐシェル組み込み関数
red(1)	ed(1) を参照
regcmp(1)	正規表現のコンパイル
rehash(1)	hash(1) を参照
remote_shell(1)	rsh(1) を参照
remsh(1)	rsh(1) を参照
repeat(1)	shell_builtins(1) を参照
return(1)	exit(1) を参照
rksh(1)	ksh(1) を参照
rlogin(1)	リモートログイン
rm(1)	ディレクトリのエントリの削除
rmail(1)	mail(1) を参照

rmdir(1)	rm(1) を参照
rmformat(1)	再書き込み可能な取り外し可能媒体のフォーマット
rsh(1)	リモートシェル
rusers(1)	リモートマシン上にログインしているユーザーの表示
script(1)	端末セッションの記録の生成
sdiff(1)	2つのファイル間の違いを並べて出力
sed(1)	ストリームエディタ
select(1)	shell_builtins(1) を参照
set(1)	現在のシェルおよびそこから起動されたプロセスでの環境変数の特性を決定するシェル組み込み関数
setenv(1)	set(1) を参照
settime(1)	touch(1) を参照
sh(1)	標準シェルとジョブ制御シェルおよびコマンドインタプリタ
shell_builtins(1)	シェル組み込み関数
shift(1)	シェルの引数のリスト、またはフィールドで区切られた単語をずらすためのシェル組み込み関数
sleep(1)	実行の一定期間保留
sort(1)	テキストファイルのソート、マージ、順序の確認
source(1)	exec(1) を参照
split(1)	ファイルを複数に分割
srchtxt(1)	メッセージデータベースの内容表示、またはそこからのテキスト文字列の検索
stop(1)	jobs(1) を参照
strchg(1)	ストリーム構成の変更または照会
strconf(1)	strchg(1) を参照
strings(1)	オブジェクトファイルおよびバイナリファイルからの印字可能な文字列の検索
stty(1)	端末用オプションの設定
sum(1)	ファイルのチェックサムおよびブロックカウントの出力
suspend(1)	現在のシェルを停止させるためのシェル組み込み関数
switch(1)	shell_builtins(1) を参照

tabs(1)	端末へのタブ設定
tail(1)	ファイルの最終部分の出力
tar(1)	テープアーカイブの作成およびファイルの追加または抽出
tbl(1)	nroff または troff 用のテーブルの書式化
tee(1)	標準出力の複写
test(1)	条件の評価
time(1)	単純コマンドの時間測定
timemanp(1)	mailp(1) を参照
times(1)	現在のシェルの時間使用を報告するためのシェル組み込み関数
timesysp(1)	mailp(1) を参照
touch(1)	ファイルのアクセス日時および更新日時の変更
tput(1)	端末の初期化または terminfo データベースの照会
tr(1)	文字の変換
trap(1)	(ハードウェア) シグナルに応答するためのシェル組み込み関数
troff(1)	ドキュメントのタイプセットと清書
truss(1)	システムコールとシグナルの追跡
tty(1)	ユーザーの端末名の応答
type(1)	コマンドのタイプの検査
typeset(1)	シェル変数と関数の属性と値を設定または取得するためのシェル組み込み関数
ulimit(1)	limit(1) を参照
umask(1)	ファイルモード生成マスクの獲得と設定
unalias(1)	alias(1) を参照
uname(1)	現在のシステム名の表示
uncompress(1)	compress(1) を参照
unexpand(1)	expand(1) を参照
unhash(1)	hash(1) を参照
uniq(1)	ファイルの中の重複行の報告またはフィルタへの出力
unlimit(1)	limit(1) を参照

unpack(1)	pack(1) を参照
unset(1)	set(1) を参照
unsetenv(1)	set(1) を参照
until(1)	shell_builtins(1) を参照
vacation(1)	メールへの自動返信
valdate(1)	ckdate(1) を参照
valgid(1)	ckgid(1) を参照
valint(1)	ckint(1) を参照
valpath(1)	ckpath(1) を参照
valrange(1)	ckrange(1) を参照
valstr(1)	ckstr(1) を参照
valtime(1)	cktime(1) を参照
valuid(1)	ckuid(1) を参照
valyorn(1)	ckyorn(1) を参照
vedit(1)	vi(1) を参照
vi(1)	ex に基づいたスクリーン指向の (ビジュアル) ディスプレイエディタ
view(1)	vi(1) を参照
volcheck(1)	ドライブ内の媒体の検査と、デフォルトでのすべてのフロッピー媒体の検査
wait(1)	他のプロセスの終了を待つ
wc(1)	ファイルの中の行数、単語数および文字数の表示
whatis(1)	キーワードに関する要約の表示
whence(1)	typeset(1) を参照
while(1)	shell_builtins(1) を参照
who(1)	システムにログインしているユーザーの表示
whois(1)	インターネットのユーザー名ディレクトリサービス
write(1)	他のユーザーへのメッセージの書き込み
xargs(1)	引数リストを構築してコマンドを実行
yacc(1)	構文解析プログラムの生成
ypcat(1)	NIS データベース内の値の出力
ypmatch(1)	NIS マップ内の 1 つまたは複数のキーの値の出力

`yppasswd`(1)	NIS データベース内のネットワークパスワードの変更
`ypwhich`(1)	NIS サーバーまたはマップマスターの名前の表示
`zcat`(1)	`compress`(1) を参照

名前 | acctcom – プロセスアカウンティングファイルの検索と出力

形式 | **acctcom** [-abfhikmqrtv] [-C *sec*] [-e *time*] [-E *time*] [-g *group*] [-H *factor*] [-I *chars*] [-l *line*] [-n *pattern*] [-o *output-file*] [-O *sec*] [-s *time*] [-S *time*] [-u *user*] [*filename...*]

機能説明 | acctcom は、 *filename* 引数で指定されたファイル、標準入力、もしくは /var/adm/pacct の内容を acct(3HEAD) で規定した形式で読み込み、 条件に従って選択したレコードを標準出力に書き出します。 個々のレコードは、1 つのプロセスの実行を表します。 必ず出力されるのは、 COMMAND NAME、USER、TTYNAME、START TIME、END TIME、REAL (SEC)、1CPU (SEC)、およびMEAN SIZE (K) です。また指定された場合にだけ出力されるのは、 F(fork()/exec() フラグ: 1 は exec() なしの fork())、 STAT(システム終了状態)、HOG FACTOR、KCORE MIN、CPU FACTOR、CHARS TRNSFD、およびBLOCKS READ(読み書きされたブロックの合計数) です。

スーパーユーザー特権で実行されたコマンドについては、 コマンド名の前に ' # ' が付加されます。 また認識されている端末の いずれにも関連していないプロセスについては、 TTYNAME フィールドに ' ? ' が出力されます。

filename 引数を指定しないとき、 標準入力が端末もしくは/dev/nullに対応していれば (シェル中で ' & 'を使う場合と同じ状態)、/var/adm/pacct が読み込まれます。対応していなければ標準入力が読み込まれます。

filename 引数を指定すると、 記述した順序でそれらのファイルが読み込まれます。 各ファイルは正方向に、 すなわちプロセスの終了時刻の昇順に読まれます。通常 /var/adm/pacct は、現在の検査対象ファイルを表します。 処理量の多いシステムでは、 このようなファイルを複数個用意して、 現在のファイル以外のファイルを /var/adm/pacct*incr* として存在させることが必要となる場合があります。

オプション

-a 選択されたプロセスに関する平均値の統計情報を表示します。 この統計情報は、全出力レコードの後に出力されます。

-b レコードを逆方向に、 すなわち最新のコマンドから順番に読み込みます。入力先が標準入力の場合には、 このオプションは無視されます。

-f 出力上に fork()/exec() フラグとシステム終了状態情報を書き出します。 このオプションによる出力では、 数値はいずれも 8 進数表示となります。

-h 平均メモリーサイズの代わりに、 プロセスが実行中に使用した合計 CPU 時間の割合を表示します。この値は「ホッグ係数」 と呼ばれ、(合計 CPU 時間) / (経過時間) で計算されます。

-i 出力上に、入出力回数を示す欄を表示します。

-k メモリサイズの代わりに、実行時に使われたキロバイト/分ごとのコアサイズの合計 を表示します。

-m 平均コアサイズを表示します (デフォルト)。

`-q`	出力レコードはいっさい表示せずに、 `-a` オプションで指定するような統計情報だけを出力します。
`-r`	CPU 係数を出力します。これは (ユーザー時間) / ((システム時間) + (ユーザー時間)) で計算されます。
`-t`	システム CPU 時間とユーザー CPU 時間を区別して出力します。
`-v`	各欄のヘッダを出力しません。
`-C` *sec*	(システム時間) + (ユーザー時間) で計算される合計 CPU 時間が *sec* で示す秒数を超えているプロセスだけを選択します。
`-e` *time*	*time* で示す時刻もしくはそれ以前に存在していた プロセスだけを選択します。
`-E` *time*	*time* で示す時刻もしくはそれ以前に終了したプロセスだけを選択します。なお `-S` と `-E` の両オプションを指定し、 *time* として同じ時刻を指定した場合には、 その時刻に存在していた プロセスだけが選択されます。
`-g` *group*	*group* で示すグループに属しているプロセスだけを選択します。 *group* 引数には、グループ ID とグループ名のどちらでも指定できます。
`-H` *factor*	「ホッグ係数」値が *factor* で示す値を超えているプロセスだけを選択します。「ホッグ係数」に関しては、前述の `-h` オプションの項を参照してください。
`-I` *chars*	*chars* で示す値を超える数の文字を転送する プロセスだけを選択します。
`-l` *line*	`/dev/term/line` 端末に属するプロセスだけを選択します。
`-n` *pattern*	*pattern* で示すパターンに一致するコマンドだけを選択します。 *pattern* 引数として、 `regcmp`(3C) で述べるような正規表現も指定できます。 ただし + 記号は「1 回以上の出現」を表します。
`-o` *output-file*	選択されたプロセスレコードを、 *output-file* で示すファイルに入力データ形式でコピーします。 標準出力への書き出しは行いません。
`-O` *sec*	システム CPU 時間が *sec* で示す秒数を超えているプロセスだけを選択します。
`-s` *time*	*time* で示す時刻もしくはそれ以降に存在していた プロセスだけを選択します。 *time* の指定形式は *hr* [:*min* [:*sec*]] です。
`-S` *time*	*time* で示す時刻もしくはそれ以降に開始された プロセスだけを選択します。
`-u` *user*	*user* で示すユーザーに属しているプロセスだけを選択します。 *user* 引数として指定できるのは、 ユーザー ID 、 ログイン名 (これはユーザー ID に変換される)、 ‘ # ’ (スーパーユーザー特権で実行されたプロセスを表す)、 および ‘ ? ’ (認識されていないユーザー ID に対応しているプロセスを表す) です。

ファイル

`/etc/group`	システムグループファイル
`/etc/passwd`	システムパスワードファイル

/var/adm/pacct*incr*　　稼動中のプロセスアカウンティングファイル

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWaccu
CSI	対応済み

関連項目　ps(1), acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), su(1M), acct(2), regcmp(3C), acct(3HEAD), utmp(4), attributes(5)

『*Solaris* のシステム管理 (基本編)』

注意事項　acctcom は、すでに終了したプロセスに関する情報を出力するものです。 現在稼動中のプロセスについての情報は、 ps(1) により得ることができます。

名前 | adb – 汎用デバッガ

形式 | **adb** [-kw] [-I *dir*] [-P *prompt*] [-V *mode*] [*object* [*core*]]

機能説明 | adb ユーティリティは、対話型の汎用デバッガです。このユーティリティは、ファイルの検査に使用でき、プログラムを実行するための制御環境を提供します。

Solaris 9 リリースでは、adb ユーティリティは、mdb(1) ユーティリティへのリンクとして実装されています。mdb(1) は、実行中のオペレーティングシステムやオペレーティングシステムのクラッシュダンプ、ユーザープロセスを検査するために使用できる、下位レベルのデバッガです。新しい mdb ユーティリティは、adb のマクロファイル処理を含む、adb の既存の構文や機能との完全な下位互換性を備えています。adb 互換モードなどの mdb の機能については、『*Solaris* モジューラデバッガ』 と mdb(1) のマニュアルページを参照してください。adb 互換モードは、adb リンクが実行されている場合にデフォルトで有効になります。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmdb (32 ビット) SUNWmdbx (64 ビット)

関連項目 | mdb(1), attributes(5)

Solaris モジューラデバッガ

名前 | addbib – 参考文献データベース (bibliographic database) の作成または拡張

形式 | **addbib** [-a] [-p *promptfile*] *database*

機能説明 | addbib が起動したときに、最初の Instructions? プロンプトに対して y と応答すると指示が与えられます。n または RETURN を入力すると、それらの指示をスキップします。addbib は、さまざまな参考文献に関するフィールドを示し、端末からの応答を読み取って、出力レコードを*database* へ送信します。NULL 応答 (RETURN のみ) は、そのフィールドを省略することを意味します。– (マイナス符号) は、前のフィールドに戻ることを意味します。行末にバックスラッシュを付けると、フィールドを次の行に続けることができます。Continue? プロンプトが繰り返されているときは、y または RETURN) を入力して再開するか、n または q と入力して現在のセッションを終了するか、あるいは任意のシステムエディタ (vi(1)、ex(1)、ed(1) を参照) を使用して *database* を編集できます。

オプション | 次のオプションを指定できます。

-a　抄録に対するプロンプトを抑制します。デフォルトでは、抄録が求められます。抄録は、CTRL–D で終了します。

-p *promptfile*　*promptfile* に定義された新しいプロンプトの枠組みを使用します。このファイルには、プロンプト文字列、TAB、*database* に書き込まれるキー文字が含まれます。

参考文献キー文字 | 以下に、最も一般的なキー文字とその意味を示します。addbib は英語でプロンプトを表示するため、ユーザーはこれらのキー文字を知っている必要はありませんが、参考文献ファイルを後で編集する場合にはこれらの情報が必要になります。

%A　著者名

%B　参考資料を収録する文献

%C　都市 (発行場所)

%D　発行日

%E　参考資料を収録した文献の編集者

%F　脚注番号またはラベル (refer で与えられる)

%G　政府発注番号

%H　参考資料の前に印刷された見出し解説

%I　発行者 (発行元)

%J　参考資料を収録した定期刊行物

%K　参考資料の検索に使用するキーワード

%L　refer の -k オプションで使用されるラベルフィールド

%M　ベル研究所によるメモ (未定義)

%N　ボリューム内の番号

%O	参考資料の最後に印刷された他の解説
%P	ページ番号
%Q	企業または外国の著者 (著作権なし)
%R	レポート、ペーパー、または論文 (未公表)
%S	シリーズのタイトル
%T	資料または文献のタイトル
%V	ボリューム番号
%X	抜粋 — refer ではなく、roffbib で使用される
%Y,Z	refer では無視される

使用例

例 **1** 参考文献ファイルの編集

A を除いて、各フィールドを 1 回だけ指定する必要があります。関連フィールドだけを提供する必要があります。

```
%A    Mark Twain
%T    Life on the Mississippi
%I    Penguin Books
%C    New York
%D    1978
```

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目

ed(1), ex(1), indxbib(1), lookbib(1), refer(1), roffbib(1), sortbib(1), vi(1), attributes(5)

名前 | alias, unalias – コマンドまたはコマンド群の別名または省略形の生成と削除

形式 | **/usr/bin/alias** [*alias-name* [= *string*...]]

/usr/bin/unalias *alias-name*...

/usr/bin/unalias -a

csh | **alias** [*name* [*def*]]

unalias *pattern*

ksh | **alias** [-tx] [*name* [= *value*]...]

unalias *name*...

機能説明 | alias および unalias ユーティリティは、コマンドまたはコマンド群の別名あるいは省略形を作成または削除します。これらのユーティリティの動作は、C シェル環境と Korn シェル環境では異なります。

/usr/bin/alias | alias ユーティリティは、別名定義を作成または再定義するか、 あるいは既存の別名定義を標準出力に書き出します。 別名定義は、コマンド名を置き換える文字列を指定するものです。

別名定義は、現在のシェルの実行環境、およびそのシェルの全サブシェル の実行環境に影響を及ぼします。 このマニュアルに記述されているように用いれば、 別名定義は現在のシェルの親プロセスにも、 シェルが呼び出すユーティリティの環境にも、影響は与えません。

/usr/bin/unalias | unalias ユーティリティは、指定された別名の定義を削除します。 それにより、現在のシェルの実行環境から別名が削除されます。

csh | alias は別名 *name* に *def* で指定した別名の定義を割り当てます。 *def* はワードの並びで、エスケープされた ヒストリ置換のメタシンタックスを含んでいてもかまいません。 *name* に alias または unalias を使用することはできません。 *def* を省略すると、別名 *name* が現在の定義と共に表示されます。 *name* と *def* の両方を省略すると、現在あるすべての別名が表示されます。

実装状態の制限によって、別名の定義は、それが使われる前のコマンド行 で入力されていなければなりません。

unalias は *pattern* が示すファイル名置換パターン に一致する別名を破棄します。 'unalias *' と指定すると、すべての別名が破棄されます。

ksh | 引数なしの場合、このコマンドは標準出力上に *name=value* という形式の別名のリストを表示します。*value* が指定された名前に対しては別名を定義します。*value* の後方に空白があると、次のワードが別名置換指定かどうかをチェックします。 -t フラグは、検索済みの別名を設定または一覧表示します。検索済み別名の値は、指定した *name* に対応する完全パス名になります。PATH の値を再設定すると この値は未定義になりますが、 別名は検索済みのままです。-t フラグを省略すると、 *value* が指定されていない引数リスト内の各 *name* について、 別名の名前と値を表示します。 -x フラ

グは、 エクスポートされた別名を設定または表示します。 エクスポートされた別名は、 名前で起動されるスクリプト用に定義されます。 *name* が指定されているが、 *value* は指定されておらず、 *name* に対しての別名も定義されていない場合は、 終了状態は 0 以外になります。

`unalias` を指定して *name* が示す別名を別名リストから削除します。

オプション

`unalias` では、以下のオプションが指定できます。

`-a` 現在のシェルの実行環境から、すべての別名定義を削除します。

オペランド

以下のオペランドが指定できます。

alias *alias-name* 別名定義を標準出力に書き出します。

unalias *alias-name* 削除する別名を指定します。

alias-name=string *alias-name* で示す別名に、 *string* で示す文字列を割り当てます。

オペランドを 1 つも指定しないと、すべての別名定義が 標準出力に書き出されます。

出力

オペランドがすべて省略された場合、または *name* オペランドだけが指定された場合の、別名の表示形式は次のとおりです。

`"%s=%s\n"` *name, value*

文字列 *value* は、シェルへ再入力できるよう、適切な引用符を付加して出力されます。

使用例

例 1 コマンドの出力を変更する

次の例は、`ls` ユーティリティの出力を複数カラム形式の注釈付きに変更します。

```
example% alias ls="ls -CF"
```

例 2 コマンド履歴ファイルにある直前のエントリを繰り返す

次の例は、コマンドのヒストリファイルにある直前の入力を繰り返す単純な “redo” コマンドを生成します。

```
example% alias r='fc -s'
```

例 3 コマンドの出力オプションを指定する

次の例は、`du` ユーティリティがディスク出力を 1024 バイト単位にまとめるようにします。

```
example% alias du='du -k'
```

例 **4** 引数自身も別名であるような引数を処理する

次の例は、引数自身も別名であるような引数を処理できるよう、 nohup ユーティリティを設定します。

```
example% alias nohup="nohup "
```

環境

alias と unalias の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　正常終了

alias

>0　　*alias-name* オペランドで指定した名前の 1 つが別名定義を持っていなかった、もしくはエラーが発生した。

unalias

>0　　*alias-name* オペランドで指定した名前の 1 つが正しい別名定義を表していなかった、もしくはエラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), shell_builtins(1), attributes(5), environ(5)

名前 | answerbook2 – オンライン文書システム

形式 | **/usr/dt/bin/answerbook2** [-h]

機能説明 | AnswerBook2 サーバー製品は、Solaris および Solaris DOCUMENTATION CD からは削除されました。Solaris のマニュアルは現在、HTML および PDF の形式で Solaris 9 DOCUMENTATION CD に含まれています。これらのマニュアルは、AnswerBook2 サーバーなしで参照できます。

`answerbook2` ユーティリティは、デフォルトの Web ブラウザを起動して、ローカルマシンにインストールされているマニュアルへのリンクを含む HTML ページを表示します。AnswerBook2 サーバーが定義されている場合は、この HTML ページに AnswerBook2 コレクションへのリンクも表示されます。

デフォルトの AnswerBook2 サーバーを定義するには、環境変数 `AB2_DEFAULTSERVER` を使用します。

この機能は、CDE フロントパネルの「ヘルプ」メニューで AnswerBook2 オプションを使用しても利用できます。

AnswerBook2 サーバーが必要な場合は、http://www.sun.com から AnswerBook2 サーバーソフトウェアをダウンロードできます。

オプション | 次のオプションを指定できます。

`-h` 使用法の説明を表示します。

環境 | `AB2_DEFAULTSERVER` 使用するデフォルトの AnswerBook2 サーバーを識別するための完全指定の URL。たとえば、`http://imaserver.eng.sun.com:8888/` です。

属性 | 次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWab2m

関連項目 | `attributes`(5)

注意事項 | AnswerBook2 製品の詳細については、オンラインヘルプシステムを使用してください。Web ブラウザが起動すると、AnswerBook2 ライブラリを表示できる状態になります。

名前 apptrace – Solaris 共有ライブラリに対するアプリケーション関数呼び出しの追跡

形式 **apptrace** [-f] [-F [!] *tracefromlist*] [-T [!] *tracetolist*] [-o *outputfile*]
[[-tv] [!] *call* ,...] *command* [*command arguments*]

機能説明 apptrace ユーティリティは、command で指定された実行可能プログラムを実行して、プログラムの command が Solaris 共有ライブラリに対して行うすべての呼び出しを追跡します。apptrace における追跡では、プログラムによる呼び出しごとに、呼び出されたライブラリインタフェースの名前、渡された引数の値、および戻り値が報告されます。

デフォルトでは、apptrace は、実行可能オブジェクトからそれが依存するすべての共有オブジェクトに対する直接呼出しを追跡します。間接呼び出し (つまり、実行可能プログラムが依存する共有オブジェクト間で行われた呼び出し) は、デフォルトでは報告されません。

追加共有オブジェクトとの間で実行される呼び出しは、-F または -T オプションを指定することにより追跡できます (下記参照)。

デフォルトのレポート形式は、呼び出しごとに 1 行であり、参照によって渡された引数またはデータ構造体のフォーマット出力はありません。

追加引数の詳細を示すフォーマット出力は、-v オプションを使用すると得られます (下記参照)。

デフォルトでは、共有オブジェクトによって提供されるインタフェースは、呼び出された場合、すべて追跡されます。ただし、-t または -v オプションのいずれか、あるいは両方を使用することにより、追跡するインタフェース群を制限できます。

通常、実行時にリンクされる動的オブジェクト (実行可能オブジェクトと依存対象のすべての共有オブジェクト) 間の呼び出しを追跡することができるため、追跡された各呼び出しのレポートには、呼び出し元のオブジェクトの名前が示されます。

apptrace は、プロシージャリンケージテーブルを介して動的オブジェクト間で発生するすべてのプロシージャ呼び出しを追跡します。そのため、そのテーブルを介して結合されているプロシージャ呼び出しだけが追跡されます。『リンカーとライブラリ』を参照してください。

オプション 次のオプションを指定できます。

-f
fork(2) によって作成されたすべての子プロセスを追跡します。また、各行の先頭にプロセス ID を表示します。

-F [!] *tracefromlist*
コンマで区切られた共有オブジェクトのリストからの呼び出しを追跡します。これらの共有オブジェクトからの呼び出しだけが追跡されます。デフォルトでは、メインの実行可能オブジェクトからの呼び出しだけが追跡されます。共有オブジェクトのベース名だけが必要になります。たとえば、*libc* は */usr/lib/libc.so.1* に一致します。また、fnmatch(5) で説明されているように、

	シェルのワイルドカード文字がサポートされています。! が前に付いたリストは、そこからの呼び出しが追跡から除外されるオブジェクト群を定義します。*command* からの呼び出しの追跡が必要な場合、*command* は *tracefromlist* のメンバーでなければなりません。
`-o` *outputfile*	`apptrace` の出力を *outputfile* に送ります。デフォルトでは、`apptrace` の出力は、追跡されるプロセスの stderr ストリームに送られます。
`-t` [`!`]*call, . . .*	関数呼び出しを追跡、または除外します。コンマで区切られたリスト *call* に指定された呼び出しが追跡されます。このリストが ! で始まる場合、指定された関数呼び出しが追跡出力から除外されます。デフォルト値は `-t *` です。シェルスタイルのワイルドカードを使用できます。
`-T` [`!`] *tracetolist*	コンマで区切られた共有オブジェクトのリストへの呼び出しを追跡します。デフォルトでは、すべての共有オブジェクトへの呼び出しが追跡されます。上記と同様、ベース名だけが必要であり、ワイルドカードを使用できます。! が前に付いたリストは、そこへの呼び出しが追跡から除外されるオブジェクト群を示します。
`-v` [`!`]*call, . . .*	指定された関数呼び出しの引数と戻り値を示す冗長な、フォーマット出力を提供します (前述の `-t` オプションと同様)。`truss`(1) と違って、`-v` オプションで指定した呼び出しは、`-t` オプションで指定する必要はありません。たとえば、`apptrace -v open` は、`truss -t open -v open` と同じです。

使用例

例 **1** date コマンドの追跡

```
% apptrace date
date    → libc.so.1:atexit(func = 0xff3ba1c8) = 0x0
date    → libc.so.1:atexit(func = 0x117e4) = 0x0
date    → libc.so.1:setlocale(category = 0x6, locale = "") = "C"
date    → libc.so.1:textdomain(domainname =
                  "SUNW_OST_OSCMD") = "SUNW_OST_OSCMD"
date    → libc.so.1:getopt(argc = 0x1, argv = 0xffbeed5c,
                  optstring = "a:u") = 0xffffffff errno = No error
date    → libc.so.1:time(tloc = 0x21ecc) = 0x371397c3
date    → libc.so.1:nl_langinfo(item = 0x3a) = "%a %b %e %T %Z %Y"
date    → libc.so.1:localtime(clock = 0x21ecc) = 0xff03c928
date    → libc_psr.so.1:memcpy(0xffbeeccc, 0xff03c928, 0x24)
date    → libc.so.1:strftime(s = "Tue Apr 13 15:15:15 ",
                  maxsize = 0x400, format = "%a %b %e %T %Z %Y",
                  timeptr = 0xffbeeccc) = 0x1c
date    → libc.so.1:puts(Tue Apr 13 15:15:15 EDT 1999
                  s = "Tue Apr 13 15:15:15 ") = 0x1d
```

例 **1** date コマンドの追跡　*(続き)*

```
date    → libc.so.1:exit(status = 0)
```

例 **2** 冗長性セットによる特定のインタフェース群の追跡

```
% apptrace -v '*gid*' id -a
id    → libc.so.1:getgid() = 0xa
     return = (gid_t) 10   (0xa)
id    → libc.so.1:getegid() = 0xa
     return = (gid_t) 10   (0xa)
id    → libc.so.1:getgrgid(gid = 0xa) = 0x2238c
   gid = (gid_t) 10        (0xa)
   return = (struct group *) 0x2238c (struct group) {
     gr_name:   (char *) 0x223a0 "staff"
     gr_passwd: (char *) 0x223a6 ""
     gr_gid:    (gid_t) 10 (0xa)
     gr_mem:    (char **) 0x2239c
}

id    → libc.so.1:getgrgid(gid = 0xa) = 0x2238c
   gid = (gid_t) 10        (0xa)
   return = (struct group *) 0x2238c (struct group) {
     gr_name:   (char *) 0x223a0 "staff"
     gr_passwd: (char *) 0x223a6 ""
     gr_gid:    (gid_t) 10 (0xa)
     gr_mem:    (char **) 0x2239c
}

id    → libc.so.1:getgrgid(gid = 0x3) = 0x2238c
   gid = (gid_t) 3         (0x3)
   return = (struct group *) 0x2238c (struct group) {
     gr_name:   (char *) 0x223b4 "sys"
     gr_passwd: (char *) 0x223b8 ""
     gr_gid:    (gid_t) 3  (0x3)
     gr_mem:    (char **) 0x2239c
}

id    → libc.so.1:getgrgid(gid = 0x29) = 0x2238c
   gid = (gid_t) 41       (0x29)
   return = (struct group *) 0x2238c (struct group) {
     gr_name:   (char *) 0x223a4 "opcom"
     gr_passwd: (char *) 0x223aa ""
     gr_gid:    (gid_t) 41  (0x29)
     gr_mem:    (char **)  0x2239c
}

id    → libc.so.1:getgrgid(gid = 0xe) = 0x2238c
   gid = (gid_t) 14        (0xe)
   return = (struct group *)  0x2238c (struct group) {
     gr_name:   (char *) 0x223a0 "sysadmin"
     gr_passwd: (char *) 0x223a9 ""
     gr_gid:    (gid_t) 14  (0xe)
     gr_mem:    (char **) 0x2239c
}
```

例 2 冗長性セットによる特定のインタフェース群の追跡　(続き)

```
id    → libc.so.1:getgrgid(gid = 0xd3) = 0x2238c
   gid = (gid_t) 211        (0xd3)
   return = (struct group *) 0x2238c (struct group) {
     gr_name:   (char *) 0x223a8 "test"
     gr_passwd: (char *) 0x223ad ""
     gr_gid:    (gid_t) 211 (0xd3)
     gr_mem:    (char **) 0x2239c
}

uid=44013(georgn) gid=10(staff) groups=10(staff),3(sys),
          41(opcom),14(sysadmin),211(test)
```

ファイル

apptrace の基本実行時サポートは Solaris 実行時リンカー (ld.so.1(1)) のリンク監査機能によって提供され、apptrace コマンドは、/usr/lib/abi に格納されている監査オブジェクト (apptrace.so.1) に依存してこの機能を使用します。

呼び出しを追跡するときに (-v オプションによって選択された) 引数のフォーマット出力を実行するには、呼び出されたインタフェースに指定された引数の数とデータ型がapptrace でわかっていなければなりません。apptrace がフォーマット出力を実行するために利用する、特殊な実行時サポート共有オブジェクトが用意されています。実行時サポートオブジェクトが Solaris の共有ライブラリごとに提供されており、各共有ライブラリにはその共有ライブラリ内の各インタフェースに対する「インターセプタ (interceptor)」関数が組み込まれています。これらのサポート共有オブジェクトは、/usr/lib/abi に格納されています。apptrace は、ライブラリインタフェースの名前を、ライブラリのサポート冗長追跡共有オブジェクトのインターセプタ関数の名前にマップする単純なアルゴリズムを備えています。ライブラリのサポート追跡共有オブジェクトにインターセプタ関数がない場合、apptrace は、そのインタフェースの引数の数、またはデータ型のどちらかを判別できません。この場合、apptrace は、呼び出し追跡レポートのデフォルト出力 (最初の 3 つの引数を示す 16 進形式の出力) を使用します。

制限事項

一般に、apptrace は、変数引数リストを受け付ける関数への呼び出しを追跡することはできません。一部の特別な場合、特に printf および scanf ファミリでは、この制限事項に対処するために、適切なコーディング方法がいくつか用いられています.

スタックを調べようとする関数、または呼び出し元に関する情報を抽出しようとする関数は追跡できません。たとえば、 [gs]etcontext()、 [sig]longjmp()、 [sig]setjmp()、 vfork() がこれに相当します。

exit(2) などの値を返さない関数では、奇妙な出力が生成される場合があります。また、自分が復帰する前に他の追跡されている関数を呼び出す関数でも、若干おかしい出力が生成されます。

セキュリティ上の理由から、setuid/setgid プログラムに対して apptrace を実行できるのは root だけです。

varargs.h の取り込みを必要とする関数 (vwprintw(3XCURSES) や vwscanw(3XCURSES) など) を追跡するときは、引数の出力は書式化されません。

属性

次の属性については、attributes(5)のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcstl (32 ビット) SUNWcstlx (64 ビット)

関連項目

ld.so.1(1), truss(1), vwprintw(3XCURSES), vwscanw(3XCURSES), attributes(5), fnmatch(5)

リンカーとライブラリ

名前 | apropos – 特定のキーワードを持つマニュアルページの検索

形式 | **apropos** *keyword...*

機能説明 | apropos コマンドは、 *keyword* を「名前」の項目に含んでいるマニュアルページを参照し、そのヘッダ行を表示します。 これらの情報は、 catman(1M) により生成される /usr/share/man/windex データベース中に含まれています。なお catman(1M) が実行されていない場合、および -n オプション付きで実行されている場合、 apropos コマンドは失敗します。複数の語をキーワードに指定すると、 それらすべてを含んだマニュアルページが見つけ出されます。 大文字小文字の区別はありません。 指定した語が他の語の一部となっている場合、ともに検索対象となります。 たとえばキーワードとして 'compile' を指定すると、 'compiler' という語を「名前」の項目に含んだコマンドも見つけ出されます。

apropos は機能的には -k オプションを指定した man(1) コマンドと同等です。

次の 2 つのコマンドを実行してみてください。

```
example% apropos password
example% apropos editor
```

「名前」の項目が '*filename*(*section*) . . .' で始まっているファイルに関しては、 'man -s *section filename*' を実行することによりその *filename* が示すファイルのマニュアルページを出力することができます。 次の 2 つのコマンドを実行してみてください。

```
example% apropos format
example% man -s 3s format printf
```

これにより printf() サブルーチンのマニュアルページが得られます。

ファイル | /usr/share/man/windex　目次とキーワードのデータベース

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | man(1), whatis(1), catman(1M), attributes(5)

診断 | /usr/share/man/windex: No such file or directory　データベースが存在しません。 catman(1M) を実行してデータベースを生成する必要があります。

名前 | ar – 移植可能なアーカイブまたはライブラリの保守

形式 | **/usr/ccs/bin/ar** -d [-Vv] *archive file...*

/usr/ccs/bin/ar -m [-abiVv] [*posname*] *archive file...*

/usr/ccs/bin/ar -p [-sVv] *archive* [*file...*]

/usr/ccs/bin/ar -q [-cVv] *archive file...*

/usr/ccs/bin/ar -r [-abciuVv] [*posname*] *archive file...*

/usr/ccs/bin/ar -t [-sVv] *archive* [*file...*]

/usr/ccs/bin/ar -x [-CsTVv] *archive* [*file...*]

/usr/xpg4/bin/ar -d [-Vv] *archive file...*

/usr/xpg4/bin/ar -m [-abiVv] [*posname*] *archive file...*

/usr/xpg4/bin/ar -p [-sVv] *archive* [*file...*]

/usr/xpg4/bin/ar -q [-cVv] *archive file...*

/usr/xpg4/bin/ar -r [-abciuVv] [*posname*] *archive file...*

/usr/xpg4/bin/ar -t [-sVv] *archive* [*file...*]

/usr/xpg4/bin/ar -x [-CsTVv] *archive* [*file...*]

機能説明 | ar ユーティリティは、1 つのアーカイブファイルとして結合された ファイル群の保守を実行します。 このコマンドは、おもにライブラリファイルの 作成および更新のために使用されますが、他の似たような目的で使うこともできます。ar が使用するマジック文字列とファイルヘッダーは、 印刷可能な ASCII 文字で構成されます。 アーカイブが印刷可能なファイルの集まりであれば、アーカイブ全体も印刷可能です。

ar はアーカイブを作成する際に、すべてのマシン間で移植できるような形式でヘッダーを生成します。移植可能なアーカイブの形式や構造については、ar(3HEAD) で詳しく説明されています。 アーカイブシンボルテーブル (ar(3HEAD) で説明) は、リンクエディタ ld(1) がオブジェクトファイルのライブラリ群に またがった複数の受け渡しを効率よく有効にするために 使用します。アーカイブシンボルテーブルは、アーカイブ中にオブジェクトファイルが 1 つでも 存在している場合にだけ、ar により作成または保守されます。アーカイブシンボルテーブルは、特殊な名前が付けられたファイル内にあり、そのファイルは常にアーカイブの先頭にあります。このファイルに関してはユーザーには解説しておらず、ユーザーがアクセスすることはできません。ar を使ってアーカイブを生成または更新するたびに、シンボルテーブルは再構築されます。また後述する -s オプションを使えば、 シンボルテーブルを強制的に再構築することができます。

オプション | 以下のオプションを指定できます。

-a　*file* が示す新たなファイルを、 *archive* が示すアーカイブ中の *posname* が示すファイルの後に配置します。

`-b` *file* が示す新たなファイルを、 *archive* が示すアーカイブ中の *posname* が示すファイルの前に配置します。

`-c` デフォルトでは、 *archive* で示すアーカイブが生成されるときに標準エラー出力に 診断メッセージが書き出されますが、このオプションはそのメッセージ出力を抑止します。

`-C` 抽出されたファイルが、ファイルシステム中の同一名のファイルを 置き換えないようにします。特に `-T` オプションも指定した場合にはファイル名の後半が切り捨てられてしまうので、 この `-C` オプションにより前半部分が一致するファイルを置き換えてしまう ことを防げます。

`-d` *file* が示すファイルを *archive* が示すアーカイブから削除します。

`-i` *file* が示す新たなファイルを、 *archive* が示すアーカイブ中の *posname* が示すファイルの前に配置します。 (`-b` オプションと同機能)

`-m` *file* が示すファイルを移動します。移動先は、 `-a`、 `-b`、または `-i` オプションが *posname* とともに指定されていればそれに従い、 指定されていなければアーカイブの最後尾となります。

`-p` *archive* が示すアーカイブ中の、 *file* が示すファイルの内容を標準出力に印刷します。 *file* が 1 つも指定されていなければ、そのアーカイブ中の すべてのファイルの内容が、アーカイブ内でのファイルの順序に従って 出力されます。

`-q` *file* が示すファイルを、 *archive* が示すアーカイブの最後尾に追加します。位置を指定するオプション `-a` 、 `-b` 、および `-i` はいずれも無効です。 指定されたファイルと同じ名前のファイルが すでにそのアーカイブ中に存在しているかどうかは、 チェックされません。 このオプションを使えば二次的な動作を抑止できるので、 大きなアーカイブを少しずつ作成していく場合に便利です。

`-r` *archive* が示すアーカイブ中で、 *file* が示すファイルの置換または追加を行います。 指定したアーカイブが存在していなければ、 新たにアーカイブを作成し (`-c` オプション指定時を除く) 診断メッセージを標準エラー出力に書き出します。 アーカイブは存在していて、ファイル名が 1 つも指定されなかった場合の 実行結果は定義されていません。 ファイルの置換が発生しても、アーカイブ中でのファイルの 順序は変わりません。 この `-r` オプションとともに `-u` オプションも指定されていると、最終更新日時が アーカイブ中の対応するファイルより 新しいファイルだけが置き換えられます。 `-a` 、 `-b` 、または `-i` オプションが指定された場合、 *posname* 引数の指定も必須となります。このとき、新規ファイルは *posname* で示すファイルの後 (`-a` の場合) または前 (`-b` および `-i` の場合) に置かれます。これらの位置決めオプションが 指定されていなければ、新規ファイルは アーカイブの最後尾に置かれます。

`-s` `ar` を呼び出したコマンドに、アーカイブの内容を 更新するようなオプションが指定されていない場合でも、 アーカイブのシンボルテーブルを強

制的に再構築します。 このオプションは、アーカイブ上で `strip`(1) コマンドを実行した後でアーカイブシンボルテーブルの内容を 復元するのに便利です。

`-t` アーカイブの内容を示す目次を生成します。 *file* で指定したファイルが目次に含まれます。 *file* を 1 つも指定しないと、 アーカイブ中の全ファイル名が、アーカイブ中での順番に 含まれます。

`-T` 抽出されたファイルのアーカイブ名の長さが、 ファイルシステムでサポートする最大長を超えている場合、 超過した部分を切り捨てます。 デフォルトでは、最大長を超える長さの名前を持つ ファイルを抽出しようとするとエラーとなり、ファイルの抽出は 行われず診断メッセージが出力されます。

`-u` 古いファイルを更新します。 `-r` オプションと一緒に指定すると、 最終更新日時が *archive* 中の対応するファイルよりも 新しいファイルについてだけ、置換が行われます。

`-V` 自身のバージョン番号を標準エラー出力に印刷します。

/usr/ccs/bin/ar

`-v` 詳細な情報を出力します。他のオプション `-d`、`-r`、または `-x` も一緒に指定すると、アーカイブやファイルの生成と保守作業に ついてファイルごとの詳細な情報を出力します。

`-p` オプションも一緒に指定すると、ファイルの内容の 前にファイルの名前を標準出力に書き出します。

`-t` オプションも一緒に指定すると、アーカイブ中のファイルに関する 大量の情報が出力されます。

`-x` オプションも一緒に指定すると、抽出作業の前に ファイル名が出力されます。 アーカイブに書き込むと、 メッセージを標準エラーに書き出します。

/usr/xpg4/bin/ar

`-v` アーカイブに書き込んだ場合 (メッセージを標準エラーに書き出しません) を除いて、`/usr/bin/ar` 版と同じです。

`-x` *file* で指定したファイル群を *archive* が示すアーカイブから抽出します。 アーカイブの内容は変わりません。 *file* を 1 つも指定しないと、アーカイブ中の 全ファイルが抽出されます。 抽出されたファイルの名前が、出力先のディレクトリが サポートしている最大長を超えている場合、結果は定義されていません。 抽出されたファイルの最終更新時刻は、 アーカイブから抽出された時刻に設定されます。

オペランド

以下のオペランドを指定できます。

archive アーカイブのパス名。

file パス名。アーカイブ中のファイル名と比較する際には、 ファイル名の最終コンポーネントだけが使用されます。 最終パス名コンポーネント (`basename`(1) を参照) に同一の名前が複数指定された場合、 その結果は定義されていません。 アーカイブに追加また

は置換されたファイルの名前は、 それが正当なものであれば、 実装先システムのアーカイブの形式により切り捨てられることはありません。

posname アーカイブ中での配置位置を相対的に表すための ファイルの名前。詳しくは `-m` と `-r` オプションの説明を参照してください。

環境

`ar` の実行に影響を与える環境変数 `LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、 `environ`(5) を参照してください。

戻り値

以下の終了ステータスが返されます。

`0` 正常終了

`>0` エラーが発生した。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/ar

属性タイプ	属性値
使用条件	SUNWbtool

/usr/xpg4/bin/ar

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目

`basename`(1), `cc`(1B), `cpio`(1), `ld`(1), `lorder`(1), `strip`(1), `tar`(1), `ar`(3HEAD), `a.out`(4), `attributes`(5), `environ`(5), `XPG4`(5)

注意事項

引数リスト中に同じファイル名を 2 度記述すると、 アーカイブ中にそのファイルが 2 個置かれます。

慣習として、アーカイブのファイル名には `.a` という接尾辞を付けるのが一般的です。

名前 | arch – 現在のホストのアーキテクチャの表示

形式 | **arch** [-k | *archname*]

機能説明 | arch は、現在のホストシステムのアプリケーションアーキテクチャを表示します。このコマンドは、従来オプションなしで使用されてきたため、SunOS 5.*x* SPARC ベースのシステムはすべて、アプリケーションアーキテクチャとして sun4 を返します。このコマンドは使用しないようにしてください。「注意事項」の項を参照してください。

システムは、どの実行可能プログラムがどのマシンで実行されるかを定義するアーキテクチャによって大まかに分類できます。カーネルアーキテクチャとアプリケーションアーキテクチャ (通常は単に「アーキテクチャ」) は、区別することができます。基本となるハードウェアが異なるために稼動するカーネルが異なるマシンでも、同じアプリケーションプログラムを実行できる場合があります。

オプション | 次のオプションを指定できます。

-k　sun4m,、sun4c などのカーネルアーキテクチャを表示します。このオプションは、マシンで稼動する特定の SunOS カーネルを定義し、カーネルに明示的に依存するプログラム (たとえば ps(1)) に対してのみ意味を持ちます。

オペランド | 次のオペランドを指定できます。

archname　*archname* を使用すると、このアプリケーションアーキテクチャ用のアプリケーションバイナリを現在のホストシステムで実行できるかどうかを判断できます。*archname* は、sun4、i86pc などの有効なアプリケーションアーキテクチャでなければなりません。

archname 用のアプリケーションバイナリを現在のホストシステムで実行できる場合は、TRUE (0) が返されます。実行できない場合は、FALSE (1) が返されます。

終了ステータス | 次の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性 | 次の属性については、attributes(5) のマニュアル ページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | mach(1), ps(1), uname(1), attributes(5)

注意事項 | このコマンドは以前のリリースとの互換性のために提供されており、使用しないようにしてください。代わりに、uname コマンドを使用してください。使用法については、uname(1) のマニュアルページを参照してください。

名前 | as – アセンブラ

SPARC | **as** [-b] [-K PIC] [-L] [-m] [-n] [-o *outfile*] [-P] [-D*name*] [-D*name=def*] [-I*path*] [-U*name...*] [-q] [-Q*y* | *n*] [-s] [-S [*a* | *b* | *c* | *l* | *A* | *B* | *C* | *L*]] [-T] [-V] [-xarch=v7 | -xarch=v8 | -xarch=v8a | -xarch=v8plus | -xarch=v8plusa | -xarch=v9 | -xarch=v9a] [-xF] *filename...*

IA | **as** [-b] [-K PIC] [-L] [-m] [-n] [-o *outfile*] [-P] [-D*name*] [-D*name=def*] [-I*path*] [-U*name...*] [-Q*y* | *n*] [-s] [-S [*a* | *b* | *c* | *l* | *A* | *B* | *C* | *L*]] [-T] [-V] *filename...*

機能説明 | as コマンドは、アセンブリ言語ソースファイルからオブジェクトファイルを生成します。

共通オプション | 次のフラグは、SPARC と IA の両方に共通しています。これらのフラグは、任意の順序で指定できます。

-b
: Sun ソースブラウザ用の追加のシンボルテーブル情報を生成します。

-K PIC
: 位置独立型のコードを生成します。

-L
: 領域を節約するために通常は破棄される一時ラベルを含む、すべてのシンボルを ELF シンボルテーブルに保存します。

-m
: アセンブラへの入力時に m4(1) マクロプロセッサを実行します。

-n
: アセンブル実行時にすべての警告を抑制します。

-o *outfile*
: アセンブリの出力を *outfile* に格納します。デフォルトでは、出力ファイル名は、入力ファイル名から .s 接尾辞 (もしあれば) を削除して、.o 接尾辞を付加することによって作成されます。

-P
: アセンブル中のファイルに対して、C プリプロセッサ cpp(1) を実行します。このプリプロセッサは、入力ファイルを結合したものではなく、各入力ファイルに対して実行されます。

-D*name*
-D*name=def*
: -P オプションが有効な場合、これらのオプションは、as コマンドでは解釈されずに cpp(1) プリプロセッサに渡されます。-P オプションが無効な場合は無視されます。

-I*path*
: -P オプションが有効な場合、このオプションは、as コマンドでは解釈されずに cpp(1) プリプロセッサに渡されます。-P オプションが無効な場合は無視されます。

	-U*name*	-P オプションが有効な場合、このオプションは、as コマンドによって解釈されずに cpp(1) プリプロセッサに渡されます。-P オプションが無効な場合は無視されます。
	-Q*y*	*n*	*n* を指定した場合、このオプションは、出力オブジェクトファイルの注釈セクションにアセンブラバージョン情報を作成します。*y* を指定すると、この情報は抑制されます。
	-s	すべてのスタブを .stabs セクションに置きます。デフォルトでは、スタブは、静的リンカー ld(1) によって最終実行中に削除される stabs.excl セクションに置かれます。-s オプションを使用すると、.stab セクションが静的リンカーによって削除されないため、スタブは最終実行可能プログラム中に残ります。
	-S[*a*|*b*|*c*|*l*|*A*|*B*|*C*|*L*]	出力されたコードの逆アセンブリを標準出力に送ります。次の各文字を -S オプションに付けると、次の処理が実行されます。 *a* アドレスによる逆アセンブル *b* 「.bof」による逆アセンブル *c* 注釈による逆アセンブル *l* 行番号による逆アセンブル 大文字を指定すると、対応するオプションのスイッチが無効になります。
	-T	これは、5.*x* システムでアセンブルされる 4.*x* アセンブリファイル用の移行オプションです。このオプションを使用すると、4.*x* アセンブリファイルのシンボル名は、5.*x* シンボル名として解釈されます。
	-V	実行されるアセンブラーのバージョン番号を標準エラー出力に書き込みます。
	-xF	Sun WorkShop アナライザを使用して、実行可能プログラムのパフォーマンス解析を行うための追加情報を生成します。入力ファイルにスタブ (デバッグ指令) が含まれていない場合、アセンブラは Sun WorkShop アナライザで必要となるいくつかのデフォルトスタブを生成します。Sun WorkShop で利用可能な dbx のマニュアルページも参照してください。
SPARC だけのオプション	-q	高速アセンブリを実行します。-q オプションを使用すると、多くのエラー検査が実行されません。注 : このオプションは、多くのエラー検査を無効にするので、

手書きのアセンブリ言語をアセンブルする場合には使用しないようにしてください。

`-xarch=v7`
: このオプションはアセンブラに対して、SPARC バージョン 7 (V7) アーキテクチャで定義されている命令を受け付けるように指示します。この結果生成されるオブジェクトコードは、ELF 形式になります。

`-xarch=v8`
: このオプションはアセンブラに対して、4 倍精度浮動小数点指示を除く、SPARC-V8 アーキテクチャで定義されている命令を受け付けるように指示します。この結果生成されるオブジェクトコードは、ELF 形式になります。

`-xarch=v8a`
: このオプションはアセンブラに対して、4 倍精度浮動小数点命令および *fsmuld* 命令を除く、SPARC-V8 アーキテクチャで定義された命令を受け付けるように指示します。この結果生成されるオブジェクトコードは、ELF 形式になります。これは、`-xarch=`*options* のデフォルトの選択です。

`-xarch=v8plus`
: このオプションはアセンブラに対して、4 倍精度浮動小数点命令を除く、SPARC-V9 アーキテクチャで定義されている命令を受け付けるように指示します。この結果生成されるオブジェクトコードは、ELF 形式になります。オブジェクトコードは、Solaris V8 システム (V8 プロセッサを持つマシン) では実行できません。Solaris V8+ システムでは実行できます。これは、SPARC 64-ビット プロセッサと 32 ビット OS の組み合わせです。

`-xarch=v8plusa`
: このオプションはアセンブラに対して、4 倍精度浮動小数点命令を除く、SPARC-V9 アーキテクチャで定義されている命令と、Visual Instruction Set (VIS) の命令を受け付けるように指示します。この結果生成されるオブジェクトコードは、V8+ ELF 形式になります。オブジェクトコードは、Solaris V8 システム (V8 プロセッサを持つマシン) では実行できません。Solaris V8+ システムでは実行できます。

`-xarch=v9`
: このオプションは、命令セットを SPARC-V9 アーキテクチャに制限します。この結果生成される .o オブジェクトファイルは 64 ビット ELF 形式となり、同じ形式の他のオブジェクトファイルとのみリンクできます。生成された実行可能プログラムは、64 ビットカーネルを持つ 64 ビット Solaris を実行する 64 ビット SPARC プロセッサ上でのみ実行できます。

`-xarch=v9a`
: このオプションは、命令セットを SPARC-V9 アーキテクチャに制限し、UltraSPARC プロセッサに固有の

Visual Instruction Set (VIS) と拡張機能を追加します。この結果生成される .o オブジェクトファイルは 64 ビット ELF 形式となり、同じ形式の他のオブジェクトファイルとのみリンクできます。生成された実行可能プログラムは、64 ビットカーネルを持つ 64 ビット Solaris を実行する 64 ビット SPARC プロセッサ上でのみ実行できます。

オペランド

次のオペランドを指定できます。

filename アセンブリ言語のソースファイル

環境

TMPDIR asコマンドは通常、ディレクトリ /tmp に一時ファイルを作成します。環境変数 TMPDIR に選択したディレクトリを設定することにより、別のディレクトリを指定することができます。(TMPDIRが有効なディレクトリではない場合、as は /tmp を使用します。)

ファイル

デフォルトでは、as は一時ファイルを /tmp に作成します。

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWsprot

関連項目

cc(1B), cpp(1),ld(1), m4(1), nm(1), strip(1), tmpnam(3C), a.out(4), attributes(5)

Sun Workshop では、dbx のマニュアルページを利用できます。

注意事項

m4(1) マクロプロセッサを起動する -m オプションを指定した場合、m4 はどのキーワードがアセンブラシンボルで、どのキーワードが実際の m4 マクロかを判断できません。そのため、 m4 のキーワードを入力ファイルのシンボル (変数、関数、ラベル) として使用することはできません。

可能であれば、cc(1B) などのコンパイルシステムインタフェースプログラムを介して、アセンブラにアクセスしてください。

すべての未定義シンボルは、グローバルシンボルとして扱われます。

名前 | asa – FORTRAN キャリッジ制御出力の印刷可能形式への変換

形式 | **asa** [-f] [*file...*]

機能説明 | asa ユーティリティは、その入力ファイルを標準出力に書き込んで、テキストファイル内のキャリッジ制御文字をラインプリンタ制御シーケンスにマッピングします。

各行の最初の文字が入力から削除され、次の処理が実行されます。

次に示すように、削除される文字によって実行される処理が異なります。

スペース文字　残りの行が変更なしで出力されます。

`0`　復帰改行制御シーケンスと残りの入力行で置き換えられます。

`1`　改ページ制御シーケンスと残りの入力行で置き換えられます。

`+`　前の行の第1カラムに戻って印刷を行う制御シーケンスで置き換えられます。残りの入力行は、その行の第1カラムから印刷されます。

入力行の第1カラムに上記以外の文字がある場合、asa はその文字をスキップして、残りの行を変更せずに印刷します。

ファイル名を指定しないで asa を呼び出すと、標準入力が使用されます。

オプション | 次のオプションを指定できます。

`-f`　各ファイルを新しいページから開始します。

オペランド | 次のオペランドを指定できます。

file　入力に使用されるテキストファイルのパス名。file オペランドを指定しないか、または「 – 」を指定すると、標準入力が使用されます。

使用例 | **例 1** asa コマンドの使用例

次のコマンドは、従来のプリンタに準拠するように、a.out からの出力を変換します。

```
a.out | asa | lp
```

また、出力をパイプを介してプリンタに送ります。

次のコマンドは、ファイル *output* の内容を、プリンタ出力と同じように端末上に表示します。

```
asa output
```

次のプログラムは、以下の 2 つの例で使用されます。

```
write(*,'(" Blank")')
write(*,'("0Zero ")')
write(*,'("+        Plus ")')
write(*,'("1One  ")')
end
```

例 1 asa コマンドの使用例　(続き)

例 1. 実際のファイルを使用した場合

```
a.out >  MyOutputFile
asa < MyOutputFile | lp
```

例 2. パイプだけを使用した場合

```
a.out | asa | lp
```

上記の 2つの例ではどちらも、2 ページの出力が作成されます。

ページ 1:

```
Blank

ZeroPlus
```

ページ 2:

```
One
```

環境

asa の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH については、environ(5) のマニュアルページを参照してください。

終了ステータス

次の終了ステータスが返されます。

0　　すべての入力ファイルが正常に出力された

>0　　エラーが発生した

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

lp(1), attributes(5), environ(5)

名前 | at, batch – 指定した時刻にコマンドを実行

形式 | **at** [-c | -k | -s] [-m] [-f *file*] [-p *project*] [-q *queuename*] -t *time*

at [-c | -k | -s] [-m] [-f *file*] [-p *project*] [-q *queuename*] *timespec...*

at -l [-p *project*] [-q *queuename*] [*at_job_id. ..*]

at -r *at_job_id. ..*

batch [-p *project*]

at | at ユーティリティは、一群のコマンドを標準入力から読み込み、それを 1 つの *at–job* として統合し、指定された時刻に実行します。

この *at–job* は、あとでシェルを別途呼び出して実行します。このシェルは、別のプロセスグループで、制御端末なしで 動作しているものです。ただし環境変数、現作業用ディレクトリ、ファイル生成マスク (umask(1))、システム資源の限界 (sh と ksh だけに適用。詳しくは ulimit(1) を参照) については、at を実行した時点のものが保持されて、*at–job* 実行時に使用されます。

at–job が投入されると、*at_job_id* と実行開始予定時刻が標準エラー出力に書き出されます。*at_job_id* は、*at–job* の識別子で、英数字とピリオドだけで構成される文字列です。投入時点で、そのジョブが一意に識別できるような名前を *at_job_id* としてシステムが割り当てます。

ユーザーへの通知やジョブの標準出力の処理方法に関しては、-m オプションの項で説明します。

at や batch (後述) を使用できるのは、ファイル /usr/lib/cron/at.allow 中に名前が登録されているユーザーだけです。このファイルが存在していない場合は、ファイル /usr/lib/cron/at.deny をアクセスして、そのユーザーの at 使用を拒否すべきかどうかを決定します。どちらのファイルも存在しないときは、solaris.jobs.user の承認を受けたユーザーだけがジョブを投入できます。at.deny だけが存在しその内容が空の場合には、どのユーザーもジョブを投入できます。at.allow と at.deny の両ファイルは、どちらも 1 行に 1 つのユーザー名という形式です。

ユーザーアカウントがロックされていると、cron ジョブおよび at ジョブは実行されません。shadow(4) で定義されているように、ロックされていないアカウントだけが、ジョブまたはプロセスを実行します。

batch | batch ユーティリティは、あとで実行すべきコマンド群を読み込みます。以下のコマンドと同じ意味を持ちます。

```
at -q b -m now
```

このうち b は at の特殊な待ち行列で、バッチジョブ専用に使用します。バッチジョブは、バッチ待ち行列に投入されるとただちに実行されます。

オプション　以下のオプションを指定できます。シェルの種類を指定するオプション -c、-k、-s がすべて省略された場合、デフォルトとして SHELL 環境変数によりシェルが決定されます。

-c	C シェル。csh(1) を使って *at–job* を実行します。
-k	Korn シェル。ksh(1) を使って *at–job* を実行します。
-s	Bourne シェル。sh(1) を使って *at–job* を実行します。
-f *file*	at–job の元になるファイルとして標準入力以外のファイルを使用するとき、そのファイルのパスを指定します。
-l	(英字のエル) *at_job_id* が指定されなかったときは、コマンドを呼び出したユーザー用にスケジュールした ジョブをすべて報告します。*at_job_id* が指定されていれば、そのジョブに関するジョブだけを出力します。
-m	*at–job* の実行が終了したら、その旨をメールで当該ユーザーに通知します。at–job が生成した標準出力と標準エラー出力の内容も、他の出力先が指定されない限り、ユーザーにメールで送られます。なおメールは、ジョブが何の出力も生成しなかった場合でも送付されます。 -m オプションを省略すると、標準出力と標準エラー出力の 内容は、他の出力先が指定されない限り、メールで当該ユーザーに通知されます。そのような出力が生成されなければ、ジョブの終了は通知されません。
-p *project*	どのプロジェクトで at ジョブまたは batch ジョブを実行するかを指定します。-l オプションと共に使用すると、指定した特定のプロジェクトだけを検索します。*project* の値全体が数値である場合は、まずプロジェクト名として解釈され、次にプロジェクト ID として解釈されます。デフォルトでは、ユーザーの現在のプロジェクトが使用されます。
-q *queuename*	*queuename* で示す待ち行列にジョブをスケジュールします。-l オプションも一緒に指定すると、その待ち行列だけが検索の対象となります。*queuename* として指定できるのは、a から z までの英小文字です。デフォルトでは、*at–job* は待ち行列 a にスケジュールされます。また待ち行列 b はバッチジョブ用に予約されています。待ち行列 c は cron ジョブ用に予約されているので、-q オプションの引数として使うことはできません。
-r *at_job_id*	以前の at ユーティリティでスケジュールされたジョブのうち、*at_job_id* で示す識別子を持ったジョブを削除します。
-t *time*	time 引数が示す時刻に、ジョブを投入します。引数の形式は、touch(1) ユーティリティが規定する形式と同一です。

オペランド　以下のオペランドを指定できます。

at_job_id
以前の at ユーティリティによりジョブがスケジュールされたときに報告された名前。

timespec
ジョブを投入し実行する日時を指定します。すべての *timespec* の値は、空白文字で区切られて連結されていると見なされます。日付と時刻の値は、そのユーザーのタイムゾーン (TZ 変数が決定) で表されていると見なされます。ただし後述する *time* オペランドでタイムゾーン名を指定した場合を除きます。

C ロケールの場合、日時を指定する文字列は以下に述べる 3 つの部分で構成されます。C ロケールの LC_TIME カテゴリから得られる値は大文字と小文字の区別はなく、たとえば A と a は同じと見なされます。

time
これは時刻を表す部分で、1、2、または 4 桁の数で指定します。1 桁または 2 桁の場合は「何時」を示す値として、4 桁の 場合は「何時何分」を示す値として解釈されます。2 つの数をコロンで区切り、*hour*:*minute* の形式で「何時何分」を指定することもできます。時刻指定の直後に AM または PM (LC_TIME ロケールカテゴリの am_pm キーワードから得られる値) 表示を付加することもできます。この付加表示がなければ、24 時間制で記述された時刻と見なされます。GMT 、UCT 、または ZULU (あえて使用する場合) のタイムゾーン名を、ユニバーサル時間を調整する時間の指定に 続けることもできます。その他のタイムゾーンは TZ 環境変数を使用して指定できます。また *time* の部分に、C ロケールの以下のトークンのいずれかを記述することも可能です。

`midnight`
12:00 am (真夜中) を表します (00:00) 。

`noon`
12:00 pm (正午) を表します。

`now`
現在の日時を表します。つまり at now という指定は、ただちに at–job を投入するという意味ですが、すぐに実行されるかどうかはその時点でのジョブの スケジューリング状況に依存します。

date
日付を示す *date* の指定は任意で、「月」の名前 (LC_TIME ロケールカテゴリの mon または abmon キーワードから得られる値) の後

に「日」を表す数値を記述する (さらにその後にコンマと「年」を表す数値があってもよい) 方法と、曜日 (LC_TIME ロケールカテゴリの day または abday キーワードから得られる値) を記述する方法があります。さらに C ロケールには以下の 2 つの特殊な日付が定義されています。

today	現在の日付が示す日、つまり当日を表します。
tomorrow	現在の日付が示す日の次の日、つまり翌日を表します。

date を省略すると、指定された時刻が現時刻より後であれば当日、現時刻より前であれば翌日とみなされます。*date* として「月」を指定し「年」を省略した場合、月の値が当月よりも前であれば翌年とみなされます。

increment — *increment* 引数は任意指定で、正の符号 (+) 数値の後に minutes、hours、days、weeks、months、years の文字列のいずれかを付加したものです。複数形を示す s は省略できます。また + 1 と同じ意味を持つキーワード next も使用できます。たとえば次の 2 つのコマンドは同じ意味となります。

```
at 2pm + 1 week at 2pm next week
```

使用法

この項で述べる at コマンド行の形式は、C ロケールに対してだけ保証されています。その他のロケールでは、midnight、noon、now、mon、abmon、day、abday、today、tomorrow、minutes、hours、days、weeks、months、years、next の各指定はサポートされていません。

コマンドの実行は、別のプロセスグループで制御端末なしで動作しているシェルを別途呼び出して行うので、コマンドを呼び出した環境でのオープンファイル記述子やトラップ、優先順位などは失われてしまいます。

at

例 1 端末での指定例

端末でのコマンドシーケンスの例を示します。

```
$ at −m 0730 tomorrow
sort < file >outfile
<EOT>
```

例 1 端末での指定例　(続き)

例 2 出力先のリダイレクト

次のシーケンスは、出力先を標準エラー出力からパイプに変更するもので、コマンドプロシージャの中で使用すると便利です。なお、出力先変更指定の記述順序は重要なので注意してください。

```
$ at now + 1 hour <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
```

例 3 ジョブによる再スケジュール

ジョブ自身に再スケジュールさせるため、at–job の中から `at` を呼び出すことができます。次の例では、`my.daily` という名の日常業務用スクリプトは毎日実行されます。ただし、`crontab` を使う方法のほうが一般的です。

```
# my.daily runs every day
at now tomorrow < my.daily
daily-processing
```

例 4 時刻、オペランド指定

C ロケールの *timespec* の 3 つの部分は、明示的に記述してあれば高い自由度で使用できます。時間やオペランド指定の例を以下に示します。

```
at 0815am Jan 24
at 8 :15amjan24
at now "+ 1day"
at 5 pm FRIday
at '17
    utc+
    30minutes'
```

batch

例 5 端末での指定例

端末でのコマンドシーケンスの例を示します。

```
$ batch
sort <file >outfile
<EOT>
```

例 6 出力先のリダイレクト

次のシーケンスは、出力先を標準エラー出力からパイプに変更するもので、コマンドプロシージャの中で使用すると便利です。なお、出力先変更指定の記述順序は重要なので注意してください。

```
$ batch <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

例 **6** 出力先のリダイレクト　*(続き)*

環境

at と batch の両ユーティリティの実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES 、NLSPATH 、LC_TIME の詳細については、environ(5) を参照してください。

SHELL　at–job を呼び出すのに用いるコマンドインタプリタの名前を表します。この変数が設定されていないか値が NULL の場合には、sh が使用されます。sh 以外の値に設定されていれば、そのシェルを使用します。このとき、どのシェルを使うかを表す 警告メッセージが出力されます。

TZ　タイムゾーンを表します。ジョブは、*timespec* または -t *time* が示す時刻に実行するために投入されますが、この時刻は TZ 変数が示すタイムゾーンに対応した値です。*timespec* の値にタイムゾーン指定が含まれていれば、TZ が示すゾーンに代わってそちらが使用されます。*timespec* にタイムゾーン指定が含まれておらず、TZ も未設定か NULL の場合、デフォルトのタイムゾーンが用いられます。

DATEMSK　環境変数 DATEMSK が設定されていれば、at はその値を、書式文字列を含んでいるテンプレートファイルの完全パス名として使用します。この文字列は書式記述子とテキスト文字から構成され、環境変数 LANG または LC_TIME の設定値に従って各国の言語固有の日付表示形式をサポートするために 使用されます。利用可能な書式記述子の一覧については、getdate(3C) を参照してください。なお「オペランド」の項で説明している time と date 引数、特殊名の noon、midnight、now、next、today、tomorrow、さらに *increment* 引数の書式は、DATEMSK が設定されている場合には認識されません。

終了ステータス

以下の終了ステータスが返されます。

0　at ユーティリティによるジョブの投入、削除、または一覧表示が正常終了した

>0　エラーが発生したので、ジョブはスケジュールされない

ファイル

/usr/lib/cron/at.allow　at と batch の両ユーティリティへのアクセス権を持つユーザーの一覧。1 行に 1 ユーザー名の形式

/usr/lib/cron/at.deny　at と batch の両ユーティリティへのアクセスを許可しないユーザーの一覧。1 行に 1 ユーザー名の形式

属性

次の属性については attributes(5) のマニュアルページを参照してください。

at

属性タイプ	属性値

使用条件	SUNWcsu
CSI	未対応

batch

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 `auths`(1)、`crontab`(1)、`csh`(1)、`date`(1)、`ksh`(1)、`sh`(1)、`touch`(1)、`ulimit`(1)、`umask`(1)、`cron`(1M)、`getdate`(3C)、`auth_attr`(4)、`attributes`(5)、`environ`(5)

注意事項 待ち行列を使用しているかどうかにかかわらず、`cron`(1M) の実行は 100 ジョブに限られています。

`cron` ではジョブの実行に遅れの出る場合があります。これらの遅れによって `cron` ジョブの処理がハングしたように見えることがあります。すべてのジョブは最後には実行されますが、極端に遅れが生じた場合には、`cron` を終了させてから再起動させることが 唯一の回避策です。

名前 | atq – 指定時刻に実行するために待ち行列に入っているジョブの表示

形式 | **atq** [-c] [-n] [*username*...]

機能説明 | atq ユーティリティは、現在のユーザーについて、待ち行列に入っている at ジョブを表示します。at(1) は、ユーザーがコマンドを後日実行できるようにするユーティリティです。solaris.jobs.admin 承認されたユーザーが atq を実行すると、待ち行列内のすべてのジョブが表示されます。

オプションを指定しない場合は、ジョブは実行される時間順に表示されます。

承認されたユーザーが *username* を指定しないで atq を実行した場合は、待ち行列全体が表示されます。*username* を指定すると、指定したユーザーに属するジョブだけが表示されます。

オプション | 次のオプションを指定できます。

-c　待ち行列に入っているジョブが作成された順序 (つまり、at コマンドが実行された時刻順) で表示します。

-n　現在待ち行列内にあるジョブの合計数だけを表示します。

ファイル | /var/spool/cron/atjobs　at ジョブ用のスプール領域

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | at(1), atrm(1), auths(1), cron(1M), auth_attr(4), attributes(5)

名前 | atrm – at または batch によってスプールされたジョブの削除

形式 | **atrm** [-afi] [[*job* #] [*user*...]]

機能説明 | atrm ユーティリティは、at(1) コマンドによって作成されたがまだ実行されていない遅延実行ジョブを削除します。これらのジョブと関連ジョブ番号のリストは、atq(1) を使用して表示できます。

ユーザーが指定したジョブを所有している場合、atrm は、そのユーザーが指定した各ジョブ番号や、そのユーザーが指定したユーザーに属するすべてのジョブを削除します。

solaris.jobs.admin 権限を持つ場合のみ、他のユーザーに属するジョブを削除できます。

オプション | 次のオプションを指定できます。

-a　すべて。現在のユーザーによって作成されたすべての未実行ジョブを削除します。特権ユーザーによって実行された場合は、待ち行列全体がフラッシュされます。

-f　強制。指定されたジョブの削除に関するすべての情報が抑制されます。

-i　対話形式。atrm は、ジョブを削除するかどうかを尋ねます。y と応答すると、ジョブが削除されます。

ファイル | /var/spool/cron/atjobs　at ジョブ用のスプール領域

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | at(1), atq(1), auths(1), cron(1M), auth_attr(4), attributes(5)

audioconvert(1)

名前 | audioconvert – オーディオファイル形式の変換

形式 | **audioconvert** [-pF] [-f *outfmt*] [-o *outfile*] [[-i *infmt*] [*file...*]] ...

機能説明 | audioconvert は、サポートされている一連のオーディオコードとファイル形式の間でオーディオデータを変換します。このユーティリティは、オーディオデータの圧縮と圧縮解除、raw オーディオデータファイルへのオーディオファイルヘッダーの追加、-law やリニア PCM などの標準データコード間の変換を行うために使用できます。

ファイル名が指定されていない場合、audioconvert は、標準入力ストリームからデータを読み取って、標準出力にオーディオファイルを書き込みます。ファイル名が指定されていれば、入力ファイルは順番に処理 、連結されて、出力ファイルに書き込まれます。

入力ファイルには、オーディオデータ形式を識別するオーディオファイルヘッダーが含まれています。オーディオデータに認識可能なヘッダーが含まれていない場合は、rate、encoding、および channels の各キーワードを使用して入力データ形式を識別し、-i オプションで形式を指定する必要があります。

出力ファイル形式は、-f 指定の形式オプションを用いて、最初の入力ファイルの形式を更新することによって決定されます。-p を指定しないと、後続のすべての入力ファイルが、この結果決定された形式に変換されて連結されます。format=*raw* が出力形式オプションに指定されていない限り、出力ファイルには、オーディオファイルヘッダーが含まれます。

入力ファイルは、-p オプションを使用して適宜変換できます。-p が有効な場合、各入力ファイルの形式は、出力形式を決定する -f オプションに従って変更されます。既存のファイルは、変換されたデータによって上書きされます。

file(1) コマンドは、Sun オーディオファイルのオーディオデータ形式をデコードして出力します。

オプション | 次のオプションを指定できます。

-p — *In Place*：入力ファイルは、-f オプションで指定された形式に個々に変換されて、再度書き込まれます。ターゲットファイルがシンボリックリンクの場合、元のファイルが再度書き込まれます。-o オプションは、 -p と同時に指定することはできません。

-F — *Force*：このオプションは、audioconvert が -i オプションによって形式が指定されている入力ファイルのファイルヘッダーをすべて無視するようにします。F を指定しないと、audioconvert は、有効なオーディオファイルヘッダーを含む入力ファイルに対する i オプションを無視します。

-f *outfmt* — *Output Format*：このオプションは、出力ファイルのファイル形式とデータコードを指定するために使用されます。指定されていないフィールドのデフォルト値は、入力ファイル形式から決定されます。有効なキーワードと値については、以下の項で説明します。

`-o` *outfile*
: *Output File*：すべての入力ファイルが連結され、出力形式に変換されて、指定の出力ファイルに書き込まれます。`-o` と `-p` のいずれも指定しないと、連結された出力は標準出力に書き込まれます。`-p` オプションは、 `-o` と同時に指定することはできません。

`-i` *infmt*
: *Input Format*：このオプションは、raw 入力ファイルのデータコードを指定するために使用されます。通常、入力データ形式は、オーディオファイルヘッダーから決定されます。このオプションは、有効なオーディオファイルヘッダーが付いていないオーディオデータを変換するときに必要です。オーディオファイルヘッダーを含む入力ファイルに `-i` を指定すると、`-F` が指定されている場合を除き、入力形式文字列は無視されます。形式指定構文は、`-f` 出力ファイル形式と同じです。

  複数の入力形式を指定できます。入力形式は、新しい入力形式が指定されるまで、その指定に続くすべての入力ファイルに適用されます。

file
: *File Specification*：指定のオーディオファイルは連結され、出力形式に変換されて、書き出されます。ファイル名が指定されていないか、または特殊ファイル名 _ が指定されている場合、オーディオデータは標準入力から読み取られます。

`-?`
: *Help*：コマンド行の使用法に関するメッセージを出力します。

形式指定

入力および出力形式指定の構文は、次のとおりです。

keyword=value[,*keyword=value* . . .]

間に空白は入りません。一意の値は *keyword=* を前に付けなくても指定できます。

`rate`
: オーディオサンプリング率は、1秒あたりのサンプル数で指定されます。数値の後に続く文字 `k` は、その数値に 1000 を掛けた値を意味します。 (たとえば、44.1k = 44100)。一般的に使用されるサンプル率の標準は、8k、16k、32k、44.1k、および 48k です。

`channels`
: インタリーブされたチャネルの数は整数で指定されます。1 つまたは 2 つのチャネルデータを指定するために、それぞれ `mono` または `stereo` という語を使用することもできます。

`encoding`
: このオプションは、デジタルオーディオデータ表示を指定します。コードは、精度を暗黙に (`ulaw` は暗黙に 8 ビット精度を示す)、または名前の一部として明示的に (たとえば、`linear16`) 指定します。有効なコード値は次のとおりです。

  `ulaw`
  : CCITT G.711 -law コード。これは、電話レベルの音声に主に使用される 8 ビット形式です。

`alaw`	CCITT G.711 A-law コード。これは、ヨーロッパで電話レベルの音声に主に使用される 8 ビット形式です。
`linear8,linear16,linear32`	リニアパルスコード変調 (PCM) コード。名前は、精度のビット数を示します。`linear16` は通常、高品質オーディオデータに使用されます。
`pcm`	`linear16` と同じです。
`g721`	acronym CCITT G.721 圧縮形式。このコードは、4 ビット精度の適応デルタパルスコード変調 (ADPCM) を使用します。これは、主に圧縮 -law 音声データに使用されます (2:1 の圧縮率を達成)。
`g723`	CCITT G.723 圧縮形式。このコードは、3 ビット精度の適応デルタパルスコード変調 (ADPCM) を使用します。これは、主に圧縮 -law 音声データに使用されます (8:3 の圧縮率を達成)。オーディオ品質は、G.721 と同等ですが、非スピーチデータに使用した場合、品質が低下することがあります。

次のコード値も、サンプル率、チャネル、およびコードを設定するための短縮形として使用できます。

`voice`	`encoding=ulaw,rate=8k,channels=mono` と同じ
`cd`	`encoding=linear16,rate=44.1k,channels=stereo` と同じ

dat　encoding=linear16,rate=48k,channels=stereo と同じ

format　このオプションは、オーディオファイル形式を指定します。有効な形式は、次のとおりです。

sun　Sun 互換ファイル形式 (デフォルト)。

raw　この形式は、raw オーディオデータ (オーディオヘッダーなし) を読み取るか書き込む場合に使用します。あるいは、外部オーディオファイル形式をインポートするために offset と組み合わせて使用します。

offset　(-i のみ) オーディオデータの開始を見つけるためのバイトオフセットを指定します。このオプションは、認識不能なファイルヘッダーを含むオーディオデータをインポートするために使用できます。

使用法

ファイルのサイズが 2G バイト (2^{31} バイト) 以上ある場合の audioconvert の動作については、largefile(5) のマニュアルページを参照してください。

使用例

例 1 音声データを記録および圧縮してから格納する

音声データを記録し、圧縮してからファイルに保存します。

```
example% audiorecord | audioconvert -f g721 > mydata.au
```

例 2 2 つの音声ファイルを連結する

データ形式に関係なく、 2 つの Sun 形式のオーディオファイルを連結して、8 ビットの -law、16 KHz のモノラルファイルを出力します。

```
example% audioconvert -f ulaw,rate=16k,mono -o outfile.au infile1 infile2
```

例 3 ディレクトリを Sun 形式に変換する

raw 音声データファイルを含むディレクトリを、適宜 Sun 形式に変換します (各ファイルにファイルヘッダーを追加します)。

```
example% audioconvert -p -i voice -f sun *.au
```

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
アーキテクチャ	SPARC, IA
使用条件	SUNWauda

属性タイプ	属性値
インタフェースの安定性	開発中

関連項目 audioplay(1), audiorecord(1), file(1), attributes(5), largefile(5)

注意事項 マルチチャネルデータをモノラルに変換するために使用されるアルゴリズムは、単にチャネルを足し合わせることにより実装されます。入力データが完全に同相している場合 (モノラルファイルがステレオに変換されて再びモノラルに変換される場合のように)、結果のデータには若干の変形が含まれることがあります。

名前 | audioplay – オーディオファイルの再生

形式 | **audioplay** [-iV] [-v *vol*] [-b *bal*] [-p speaker | headphone | line] [-d *dev*] [*file...*]

機能説明 | audioplay ユーティリティは、指定されたオーディオファイル (ファイル名が指定されていない場合は標準入力) をオーディオデバイスにコピーします。入力ファイルが指定されておらず、標準入力が tty の場合、コマンド行に指定されたポート、ボリューム、およびバランス設定が適用されて、プログラムは終了します。

入力ファイルには、有効なオーディオファイルヘッダーが含まれていなければなりません。このヘッダーのコード情報はオーディオデバイスの機能に対応しており、データ形式に互換性がない場合は、エラーメッセージが出力されて、その入力ファイルはスキップされます。圧縮された ADPCM (G.721) モノラルオーディオデータは、自動的に圧縮解除されてから再生されます。

サンプリング周波数のわずかな違い (1% 未満) は、通常無視されます。これにより、たとえば、8000 Hz の周波数だけしかサポートしていないオーディオデバイスでも、8012 Hz でサンプリングされたデータを再生することができます。-V オプションが指定されている場合、周波数の違いには、警告メッセージによってフラグが付けられます。

オプション | 次のオプションを指定できます。

-i
Immediate：オーディオデバイスを利用できない場合 (つまり、別のプロセスが現在書き込み権を持つ場合)、audioplay は通常、デバイスへのアクセス権を取得するまで待機します。-i オプションが指定されている場合、audioplay はエラーメッセージを出力して、デバイスが使用中であれば即座に終了します。

-V
Verbose：オーディオデバイスへのアクセスを待機している場合、またはサンプル率のずれが検出された場合、標準エラー出力にメッセージを書き込みます。

-v *vol*
Volume：出力ボリュームが指定の値に設定されてから再生が開始され、audioplay が終了すると元のレベルにリセットされます。*vol* 引数は、0 ～ 100 (0 と 100 を含む) の間の整数値です。この引数を指定しないと、出力ボリュームは、いずれかのプロセスによって最後に設定されたレベルのままになります。

-b *bal*
Balance：出力バランスが、指定の値に設定されてから再生が開始され、audioplay が終了すると元のレベルにリセットされます。*bal* 引数は、-100 ～ 100 (-100 と 100 を含む) の間の整数値です。-100 の値は、左バランスを示し、0 は中央、100 は右バランスを示します。この引数を指定しないと、出力バランスは、いずれかのプロセスによって最後に設定されたレベルのままになります。

-p speaker | headphone | line
Output Port：オーディオ出力シグナルの宛先として、ビルトイン(組み込み)スピーカー (デフォルト)、ヘッドフォンジャック、またはラインを選択します。この引数を指定しないと、出力ポートはそのままの状態になります。注*:* オーディオアダプ

タによっては、一部の出力ポートをサポートしていないことがあります。指定されたポートが存在しない場合は、適切な代替ポートが使用されます。

`-d` *dev*
Device：*dev* 引数は、出力が送信される代替オーディオデバイスを指定します。`-d` オプションを指定しないと、`AUDIODEV` 環境変数が参照されます (下記参照)。あるいは、`/dev/audio` がデフォルトのオーディオデバイスとして使用されます。

`–?`
Help: コマンド行の使用法に関するメッセージを出力します。

オペランド

次のオペランドを指定できます。

file — *File Specification*：コマンド行に指定されたオーディオファイルは、順番に再生されます。ファイル名が指定されていない場合は、標準入力ストリーム (tty ではない場合) が再生されます (このストリームには、オーディオファイルヘッダーも含まれていなければなりません)。特殊なファイル名 – を使用すると、ファイルの代わりに標準入力ストリームを読み取ることができます。相対パス名を指定すると、`AUDIOPATH` 環境変数が参照されます (下記参照)。

使用法

ファイルのサイズが 2G バイト (2^{31} バイト) 以上ある場合の `audioplay` の動作については、`largefile`(5) のマニュアルページを参照してください。

環境

`AUDIODEV` — `-d` 引数が指定されていない場合の、書き込み先オーディオデバイスの完全パス名。`AUDIODEV` 変数が設定されていない場合は、`/dev/audio` が使用されます。

`AUDIOPATH` — 相対パス名で名前が指定されているオーディオファイルの検索先ディレクトリをコロンで区切ったリスト。現在のディレクトリ (`.`) は、検索パスに明示的に指定できます。`AUDIOPATH` 変数が設定されていない場合は、現在のディレクトリだけが検索されます。

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
アーキテクチャ	SPARC
使用条件	SUNWaudio
インタフェース安定性	開発中

関連項目

`audioconvert`(1), `audiorecord`(1), `mixerctl`(1), `attributes`(5), `largefile`(5), `usb_ac`(7D), `audio`(7I), `mixer`(7I)

使用上の留意点

audioplay が現在サポートしているオーディオ形式変換は限定されています。オーディオファイルの形式がオーディオデバイスによってサポートされていない場合は、まずこのファイルを変換する必要があります。たとえば、進行状況に合わせて音声形式に変換するには、次のコマンドを使用します。

```
example% audioconvert -f voice myfile | audioplay
```

形式変換がオーディオ出力に追いつけない場合があります。その場合は、一時ファイルに変換してから、データを再生する必要があります。

名前 | audiorecord – オーディオファイルの記録

形式 | **audiorecord** [-af] [-v *vol*] [-b *bal*] [-m *monvol*] [-p mic | line | internal-cd] [-c *channels*] [-s *rate*] [-e *encoding*] [-t *time*] [-i *info*] [-d *dev*] [*file*]

機能説明 | audiorecord ユーティリティは、オーディオデータをオーディオデバイスから指定されたオーディオファイル (ファイル名が指定されていない場合は標準出力) にコピーします。出力ファイルが指定されておらず、標準出力が tty の場合は、コマンド行に指定されたボリューム、バランス、モニターボリューム、ポート、およびオーディオ形式の設定が適用されて、プログラムは終了します。

デフォルトでは、モノラルオーディオデータは 8 kHz で記録されて、-law 形式でコード化されます。オーディオデバイスが追加構成をサポートしている場合は、-c、-s、および -e オプションを使用して、データ形式を指定することができます。出力ファイルの先頭には、出力ファイル中のコード化されたデータの形式を識別するオーディオファイルヘッダーが付けられます。

記録はただちに開始され、SIGINT シグナル (たとえば、Ctrl-C) が受信されるまで継続されます。-t オプションを指定した場合、audiorecord は、指定されたデータ量が記録されると停止します。

オーディオデバイスを利用できない (つまり、別のプロセスが読み取り権を持つ) 場合、audiorecord はエラーメッセージを出力して、ただちに終了します。

オプション | 次のオプションを指定できます。

-a
Append：指定のオーディオファイルの最後にデータを追加します。オーディオデバイスは、既存ファイルのオーディオデータ形式をサポートしていなければなりません。

-f
Force：-a フラグが指定されている場合、オーディオデバイスのサンプル率は、元のファイルが記録されたサンプル率と一致しなければなりません。-f フラグも指定されている場合、サンプル率の違いは無視されて、警告メッセージが標準エラー出力に書き込まれます。

-v *vol*
Volume： 記録が開始される前に記録の取得が指定の値に設定され、audiorecordが終了すると元のレベルにリセットされます。*vol* 引数は、0 ～ 100 (0 と 100 を含む) の間の整数値です。この引数を指定しないと、入力ボリュームは、いずれかのプロセスによって最後に設定されたレベルのままになります。

-b *bal*
Balance：記録バランスが、指定の値に設定されてから再生が開始され、audiorecord が終了すると元のレベルにリセットされます。*bal* 引数は、-100 ～ 100 (-100 と 100 を含む) の間の整数値です。-100 の値は、左バランスを示し、0 は中央、100 は右バランスを示します。この引数を指定しないと、入力バランスは、いずれかのプロセスによって最後に設定されたレベルのままになります。

`-m` *monvol*
Monitor Volume：入力モニターボリュームが、指定の値に設定されてから再生が開始され、`audiorecord` が終了すると元のレベルにリセットされます。*monval* 引数は、0 〜 100 (0 と 100 を含む) の間の整数値です。`0` 以外の値を指定すると、記録の進行中に、直接接続された入力ソースを出力スピーカで聞くことができます。この引数を指定しないと、モニターボリュームは、いずれかのプロセスによって最後に設定されたレベルのままになります。

`-p mic | line | internal-cd`
Input Port：`mic`、`line`、または `internal-cd` 入力をオーディオ出力シグナルの送信元として選択します。この引数を指定しないと、入力ポートはそのままの状態になります。注*:* システムによっては、一部の入力ポートをサポートしていないことがあります。指定されたポートが存在しない場合、このオプションは無視されます。

`-c` *channels*
Channels：オーディオチャネル (1 または 2) の番号を指定します。値は整数、または `mono` か `stereo` で指定します。デフォルトは `mono` です。

`-s` *rate*
Sample Rate: サンプルレートを1秒あたりのサンプル数で指定します。数値の後に続く文字 `k` は、その数値に 1000 を掛けた値を意味します。 (たとえば、44.1k = 44100 です)。デフォルトのサンプル率は 8 kHz です。

`-e` *encoding*
Encoding：オーディオデータコードを指定します。 `ulaw`、`alaw`、または `linear` のいずれかの値を指定できます。デフォルトコードは `ulaw` です。

`-t` *time*
Time： 最長記録時間を指定します。時間は、秒数を示す浮動小数点値として指定するか、または *hh:mm:ss.dd* の形式で指定することができます。この場合、時間と分の指定は任意です。

`-i` *info*
Information：出力ファイルヘッダーの 'information' フィールドには、*info* 引数で指定された文字列が設定されます。このオプションは、`-a` 引数と同時に指定することはできません。

`-d` *dev*
Device：*dev* 引数は、入力を受け取る代替オーディオデバイスを指定します。`-d`オプションを指定しないと、`AUDIODEV` 環境変数が参照されます (下記参照)。あるいは、`/dev/audio` がデフォルトのオーディオデバイスとして使用されます。

`-?`
Help: コマンド行の使用法に関するメッセージを出力します。

オペランド

file　*File Specification*：指定されたオーディオファイルが再度書き込まれます (または追加されます)。ファイル名が指定されていない場合 (そして標準入力ストリームが tty ではない場合)、あるいは特殊なファイル名 - が指定されている場合は、出力は標準出力に送られます。

使用法 | ファイルのサイズが 2G バイト (2^{31} バイト) 以上ある場合の `audiorecord` の動作については、`largefile`(5) のマニュアルページを参照してください。

環境 | `AUDIODEV` `-d` 引数が指定されていない場合の、記録元オーディオデバイスの完全パス名。`AUDIODEV` 変数が設定されていない場合は、`/dev/audio` が使用されます。

属性 | 次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
アーキテクチャ	SPARC
使用条件	SUNWaudio
インタフェース安定性	開発中

関連項目 | `audioconvert`(1), `audioplay`(1), `mixerctl`(1), `attributes`(5), `largefile`(5), `usb_ac`(7D), `audio`(7I), `mixer`(7I)

名前

awk – パターン走査およびパターン処理の言語

形式

/usr/bin/awk [-f *progfile*] [-F *c*] [' *prog* '] [*parameters*] [*filename...*]

/usr/xpg4/bin/awk [-F *ERE*] [-v *assignment...*] *'program'* -f *progfile...* [*argument...*]

機能説明

/usr/xpg4/bin/awk ユーティリティについては nawk(1) のマニュアルページで説明しています。

/usr/bin/awk ユーティリティは、*filename* で指定したファイルを走査し、*prog* で指定したパターンのいずれかと一致する行を探します。*prog* は、シェルと区別できるように単一引用符 (') で囲みます。 *prog* で示される各パターンとともに、*filename* 内に一致する行が見つかったときに実行される アクション (動作) を記述することができます。 一連のパターン–アクション文は *prog* に直接書いても良く、-f *progfile* オプションで示されるファイル内に指定してもかまいません。 ファイルは順番に読み込まれます。ファイルが指定されないときは、標準入力が使用されます。ファイル名 '-' は標準入力を意味します。

オプション

次のオプションを使用できます。

-f *progfile*　awk は *progfile* から読み込んだ一連のパターンを使用します。

-F*c*　フィールドセパレータとして *c* を使用します。下記の FS の説明を参照してください。

入力行

各行は、各パターン–アクション文のパターン部分と 比較されます。一致すると、関連するアクションが実行されます。 *var=value* の形式の *filename* はファイル名ではなく変数の割り当てとみなされ、 その場合には、そのファイルがオープンされるであろう 時点で行われます。 この方法によって割り当てられた変数は BEGIN パターンのルール内では使用できず、以前に 指定されたファイルがすべて読み込まれた後で割り当てられます。

入力行は通常、 空白で区切られたフィールドで構成されています (このデフォルト値は -F*c* オプションまたは FS 組み込み変数を使用して変更できます。下記参照) 。 デフォルトでは先行する空白文字を無視し、 空白文字またはタブ、あるいはその両方でフィールドを区切るようになっています。 なお、変数 FS に空白を含まない値が代入されていると、 先行空白文字を無視しません。 フィールドは $1 、 $2 、 ... のように表され、 $0 は全行を意味します。

パターン–アクション文

パターン–アクション文は次の形式をとります。

pattern { *action* }

パターンかアクションのどちらかを省略することができます。アクションが指定されていないときは、一致した行を印刷します。パターンが指定されていないときは、全行に対してアクションが 実行されます。パターン–アクション文は復帰改行またはセミコロンで区切られます。

パターンは、正規表現および関係式を任意の論理演算子 (`!` 、 `||` 、 `&&` 、および括弧) によって組み合わせたものです。 パターンは任意のブール型の組み合わせです。 関係式は次のいずれかです。

expression relop expression
expression matchop regular_expression

ここで、*relop* とは C 言語の 6 つの関係演算子のうちのどれかで、*matchop* とは `~` (含む) または `!~` (含まない) のいずれかを示します。*expression* とは、数値演算式、関係演算式、配列中の変数 (*var* `in` *array*)、またはこれらを論理演算で組み合わせたものです。

var `in` *array*

`egrep`(1) と同様、正規表現はスラッシュで囲まれていなければなりません。パターン内の独立した正規表現はその行すべてに適用されます。正規表現は関係式の中にも現われます。パターンは、コンマで区切られた 2 つのパターンからなることもあります。この場合、関連するアクションは、最初のパターンが一致した行と 2 番目のパターンが一致した行の間にあるすべての行に対して実行されます。

`BEGIN` と `END` は特殊なパターンで、最初の行を読む前と最後の行を読んだ後に制御を確保するのに使用します。この 2 つの特殊パターンは他のパターンと組み合わせて使うことはできません。

組み込み変数

組み込み変数には次のものがあります。

`FILENAME`	現入力ファイル名
`FS`	入力フィールドセパレータの正規表現 (デフォルトは空白文字とタブ)
`NF`	現レコード中のフィールド数
`NR`	現レコード番号
`OFMT`	数値の出力形式 (デフォルトは `%.6g`)
`OFS`	出力フィールドセパレータ (デフォルトは空白文字)
`ORS`	出力レコードセパレータ (デフォルトは復帰改行文字)
`RS`	入力レコードセパレータ (デフォルトは復帰改行文字)

アクションは一連の文です。 使用できる文は次のうちのいずれかです。

```
if ( expression ) statement [ else statement ]
while ( expression ) statement
do statement while ( expression )
for ( expression ; expression ; expression ) statement
for ( var in array ) statement
break
continue
{ [ statement ] . . . }
```

```
expression      # 一般に variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next            # この行の残りのパターンをスキップする
exit [expr]     # 残りの入力をスキップする。終了ステータスは expr
```

上記において、 *statement* は文を、 *expression* は式を、 *expression-list* は式のリストを、 *variable* は変数を、 *conditional* は条件を、 `format` は書式をそれぞれ表します。

文は、セミコロン、復帰改行、 右かっこのうちのいずれかで終了します。 式のリストが空の場合は入力行全体を意味します。 式は、文字列または数字と `+` 、 `-` 、 `*` 、 `/` 、 `%` 、 `^` 、および連結 (空白文字で示される) の各演算子で構成されます。 C の演算子 `++` 、 `--` 、 `+=` 、 `-=` 、 `*=` 、 `/=` 、 `%=` 、 `^=` 、 `>,` 、 `>=` 、 `<` 、 `<=` 、 `==` 、 `!=` 、 `?:` も式の中に記述できます。 変数は、スカラー、配列要素 (x[i] で表される) またはフィールドです。変数は NULL 値または 0 で初期化されます。 配列の添字は、必ずしも数字である必要はなく、文字列でもかまいません。これによって、ある種の連想記憶形式を使用できます。 文字列定数は、二重引用符 (`""`) で囲みます。C のエスケープ文字はエスケープとして認識します。

`print` 文はその引数を標準出力に出力します。*>expression* が指定されたときはファイルへ、*'|cmd'* が指定されたときはパイプへ出力します。出力は、現在の出力フィールドセパレータで区切られた各引数を持つ、出力レコードセパレータで終了します。 `printf` 文は、その書式に従って式のリストの書式を定めます。(`printf`(3C)参照)。

組み込み関数

演算関数は次の通りです。

`cos`(*x*)	*x* をラジアン単位として *x* の余弦を返します。 (`/usr/xpg4/bin/awk` のみ。`nawk`(1) のマニュアルページを参照)
`sin`(*x*)	*x* をラジアン単位として *x* の正弦を返します。 (`/usr/xpg4/bin/awk` のみ。`nawk`(1) のマニュアルページを参照)
`exp`(*x*)	*x* の指数関数を返します。
`log`(*x*)	*x* の自然対数を返します。
`sqrt`(*x*)	*x* の平方根を返します。
`int`(*x*)	引数を切り捨てて、整数にします。 つまり *x* が 0 よりも大きい場合は、 0 に向かって切り捨てます。

文字列関数は次の通りです。

`index`(*s*, *t*)
: 文字列 *s* の中で 文字列 *t* が最初に出現する位置を返します。 出現しなければ `0` を返します。

`int`(*s*)
: 整数値になるよう、 *s* を切り捨てます。 *s* が指定されていなければ、 $0 が使われます。

`length`(*s*)
引数を文字列として解釈しその長さを返します。 引数がない場合は行全体を返します。

`split`(*s*, *a*, *fs*)
文字列 *s* を *a*[*1*] 、 *a*[2] 、 . . . *a*[*n*] の配列要素に分割し、値 *n* を返します。この分割は、正規表現 *fs* によって行われ、 *fs* が指定されていない場合は フィールドセパレータ `FS` によって行われます。

`sprintf` (*fmt*, *expr*, *expr*, . . .)
`fmt` で指定した `printf`(3C) 形式に従って式の書式を定め、 その結果得られた文字列を返します。

`substr`(*s*, *m*, *n*)
文字列 *s* 内の *m* 番目から始まる長さ *n* の部分文字列を返します。

出入力用の関数は次の通りです。

`getline` `$0` に、現入力ファイルの次のレコードを 設定します。 `getline` 関数は正常終了時には `1` を、ファイルの終わりに達すると `0` を、またエラー発生時には `-1` を返します。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `awk` の動作については、`largefile`(5) を参照してください。

使用例

例 **1** 72 文字以上の行の出力：

```
length > 72
```

例 **2** 最初の 2 フィールドを逆順に出力：

```
{ print $2, $1 }
```

例 **3** 最初の 2 フィールドを逆順に出力 (入力フィールドをコンマまたは空白文字とタブ、あるいはそのすべてで区切る) :

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

例 **4** 最初のカラムを合計して、合計値と平均値の出力：

```
     { s += $1 }
END  { print "sum is", s, " average is", s/NR }
```

例 **5** フィールドを逆順に出力：

```
{ for (i = NF; i > 0; —i) print $i }
```

例 **6** start/stop の間にあるすべての行の出力：

```
/start/, /stop/
```

例 **6** start/stop の間にあるすべての行の出力： *(続き)*

例 **7** 最初のフィールドが前行と異なるすべての行の出力：

```
$1 != prev { print; prev = $1 }
```

例 **8** ページ番号付きでのファイルの出力 (5 ページから) :

```
  /Page/    { $2 = n++; }
            { print }
```

例 **9** 5 ページから始まるページ番号でファイルを出力：

このプログラムが `prog` のファイルに記録されている場合、 以下のコマンドは 5 ページから始まるページ番号で `input` ファイルを出力します。

```
awk f prog n=5 input
```

環境

awk の実行に影響を与える環境変数 `LC_CTYPE`、 `LC_MESSAGES` についての詳細は、`environ`(5) を参照してください。

`LC_NUMERIC` 数値入力の解釈、数値と文字列との変換、数値出力のフォーマットに用いる、小数点文字を決定します。`awk` プログラム (コマンド行引数で指定される代入も含む) の処理で認識される小数点文字は、ロケールに関係なくピリオド (POSIX ロケールの小数点文字) です。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/awk

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

/usr/xpg4/bin/awk

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

`egrep`(1), `grep`(1), `nawk`(1), `sed`(1), `printf`(3C), `attributes`(5), `environ`(5), `largefile`(5), `XPG4`(5)

注意事項

入力行に空白が含まれる場合、出力時に保証されません。 数値と文字の間の明示的な変換は行われません。 式を数値として扱いたい場合は 0 を加え、文字として扱いたい場合は NULL 文字列 (`""`) を連結してください。

名前 | banner – ポスターの作成

形式 | **banner** *strings*

機能説明 | `banner` は、その引数 (それぞれの長さは最大 10 文字まで) を大きな文字で標準出力に出力します。

属性 | 次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | `echo`(1), `attributes`(5)

名前 | basename, dirname – パス名の部分的な抽出

形式 | **/usr/bin/basename** *string* [*suffix*]

/usr/xpg4/bin/basename *string* [*suffix*]

dirname *string*

機能説明 | basename ユーティリティは、/ で終わるすべてのプレフィックスと suffix (*string* 中にある場合) を *string* から削除して、その結果を標準出力に出力します。このユーティリティは通常、シェルプロシージャ内の置換マーク (‘’) の中で使用されます。

/usr/bin | *suffix* は、expr(1) のマニュアルページに定義されているパターンです。

/usr/xpg4/bin | *suffix* は、含まれるどの文字にも特殊な意味が付加されていない文字列です。

dirname ユーティリティは、*string* 内のパス名の最後のレベルを除く、すべてのレベルを出力します。

使用例 | 例 **1** 環境変数を設定する

次の例は、引数 /home/sms/personal/mail を指定して呼び出しを行なった場合で、環境変数 NAME に mail というファイルを設定し、環境変数 MYMAILPATH に文字列 /home/sms/personal を設定します。

```
example% NAME=`basename  $HOME/personal/mail`
example% MYMAILPATH=`dirname  $HOME/personal/mail`
```

例 **2** ファイルをコンパイルして出力を移動する

次のシェルプロシージャは、引数 /usr/src/bin/cat.c を指定して呼び出しを行なった場合で、指定したファイルをコンパイルし、出力を現在のディレクトリ内の cat というファイルに移動します。

```
example% cc $1
example% mv a.out `basename $1 .c`
```

環境 | basename と dirname の実行に影響を与える環境変数 LC_CTYPE, LC_MESSAGES、 および NLSPATH については、environ(5) のマニュアルページを参照してください。

終了ステータス | 次の終了ステータスが返されます。

0　　正常終了

>0　　エラーが発生した

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

/usr/bin

属性タイプ	属性値
使用条件	SUNWcsu

/usr/xpg4/bin

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目 `expr`(1), `attributes`(5), `environ`(5), `XPG4`(5)

名前 | at, batch – 指定した時刻にコマンドを実行

形式 | **at** [-c | -k | -s] [-m] [-f *file*] [-p *project*] [-q *queuename*] -t *time*

at [-c | -k | -s] [-m] [-f *file*] [-p *project*] [-q *queuename*] *timespec...*

at -l [-p *project*] [-q *queuename*] [*at_job_id. ..*]

at -r *at_job_id. ..*

batch [-p *project*]

at | at ユーティリティは、一群のコマンドを標準入力から読み込み、それを 1 つの *at–job* として統合し、指定された時刻に実行します。

この *at–job* は、あとでシェルを別途呼び出して実行します。このシェルは、別のプロセスグループで、制御端末なしで 動作しているものです。ただし環境変数、現作業用ディレクトリ、ファイル生成マスク (umask(1))、システム資源の限界 (sh と ksh だけに適用。詳しくは ulimit(1) を参照) については、at を実行した時点のものが保持されて、*at–job* 実行時に使用されます。

at–job が投入されると、*at_job_id* と実行開始予定時刻が標準エラー出力に書き出されます。*at_job_id* は、*at–job* の識別子で、英数字とピリオドだけで構成される文字列です。投入時点で、そのジョブが一意に識別できるような名前を *at_job_id* としてシステムが割り当てます。

ユーザーへの通知やジョブの標準出力の処理方法に関しては、-m オプションの項で説明します。

at や batch (後述) を使用できるのは、ファイル /usr/lib/cron/at.allow 中に名前が登録されているユーザーだけです。このファイルが存在していない場合は、ファイル /usr/lib/cron/at.deny をアクセスして、そのユーザーの at 使用を拒否すべきかどうかを決定します。どちらのファイルも存在しないときは、solaris.jobs.user の承認を受けたユーザーだけがジョブを投入できます。at.deny だけが存在しその内容が空の場合には、どのユーザーもジョブを投入できます。at.allow と at.deny の両ファイルは、どちらも 1 行に 1 つのユーザー名という形式です。

ユーザーアカウントがロックされていると、cron ジョブおよび at ジョブは実行されません。shadow(4) で定義されているように、ロックされていないアカウントだけが、ジョブまたはプロセスを実行します。

batch | batch ユーティリティは、あとで実行すべきコマンド群を読み込みます。以下のコマンドと同じ意味を持ちます。

```
at -q b -m now
```

このうち b は at の特殊な待ち行列で、バッチジョブ専用に使用します。バッチジョブは、バッチ待ち行列に投入されるとただちに実行されます。

オプション　以下のオプションを指定できます。シェルの種類を指定するオプション -c、-k、-s がすべて省略された場合、デフォルトとして SHELL 環境変数によりシェルが決定されます。

-c	C シェル。csh(1) を使って *at–job* を実行します。
-k	Korn シェル。ksh(1) を使って *at–job* を実行します。
-s	Bourne シェル。sh(1) を使って *at–job* を実行します。
-f *file*	at–job の元になるファイルとして標準入力以外のファイルを使用するとき、そのファイルのパスを指定します。
-l	(英字のエル) *at_job_id* が指定されなかったときは、コマンドを呼び出したユーザー用にスケジュールした ジョブをすべて報告します。*at_job_id* が指定されていれば、そのジョブに関するジョブだけを出力します。
-m	*at–job* の実行が終了したら、その旨をメールで当該ユーザーに通知します。at–job が生成した標準出力と標準エラー出力の内容も、他の出力先が指定されない限り、ユーザーにメールで送られます。なおメールは、ジョブが何の出力も生成しなかった場合でも送付されます。 -m オプションを省略すると、標準出力と標準エラー出力の 内容は、他の出力先が指定されない限り、メールで当該ユーザーに通知されます。そのような出力が生成されなければ、ジョブの終了は通知されません。
-p *project*	どのプロジェクトで at ジョブまたは batch ジョブを実行するかを指定します。-l オプションと共に使用すると、指定した特定のプロジェクトだけを検索します。*project* の値全体が数値である場合は、まずプロジェクト名として解釈され、次にプロジェクト ID として解釈されます。デフォルトでは、ユーザーの現在のプロジェクトが使用されます。
-q *queuename*	*queuename* で示す待ち行列にジョブをスケジュールします。-l オプションも一緒に指定すると、その待ち行列だけが検索の対象となります。*queuename* として指定できるのは、a から z までの英小文字です。デフォルトでは、*at–job* は待ち行列 a にスケジュールされます。また待ち行列 b はバッチジョブ用に予約されています。待ち行列 c は cron ジョブ用に予約されているので、-q オプションの引数として使うことはできません。
-r *at_job_id*	以前の at ユーティリティでスケジュールされたジョブのうち、*at_job_id* で示す識別子を持ったジョブを削除します。
-t *time*	time 引数が示す時刻に、ジョブを投入します。引数の形式は、touch(1) ユーティリティが規定する形式と同一です。

オペランド　以下のオペランドを指定できます。

at_job_id
以前の at ユーティリティによりジョブがスケジュールされたときに報告された名前。

timespec
ジョブを投入し実行する日時を指定します。すべての *timespec* の値は、空白文字で区切られて連結されていると見なされます。日付と時刻の値は、そのユーザーのタイムゾーン (TZ 変数が決定) で表されていると見なされます。ただし後述する *time* オペランドでタイムゾーン名を指定した場合を除きます。

C ロケールの場合、日時を指定する文字列は以下に述べる 3 つの部分で構成されます。C ロケールの LC_TIME カテゴリから得られる値は大文字と小文字の区別はなく、たとえば A と a は同じと見なされます。

time
これは時刻を表す部分で、1、2、または 4 桁の数で指定します。1 桁または 2 桁の場合は「何時」を示す値として、4 桁の 場合は「何時何分」を示す値として解釈されます。2 つの数をコロンで区切り、*hour*:*minute* の形式で「何時何分」を指定することもできます。時刻指定の直後に AM または PM (LC_TIME ロケールカテゴリの am_pm キーワードから得られる値) 表示を付加することもできます。この付加表示がなければ、24 時間制で記述された時刻と見なされます。GMT 、UCT 、または ZULU (あえて使用する場合) のタイムゾーン名を、ユニバーサル時間を調整する時間の指定に 続けることもできます。その他のタイムゾーンは TZ 環境変数を使用して指定できます。また *time* の部分に、C ロケールの以下のトークンのいずれかを記述することも可能です。

`midnight`
12:00 am (真夜中) を表します (00:00) 。

`noon`
12:00 pm (正午) を表します。

`now`
現在の日時を表します。つまり at now という指定は、ただちに at–job を投入するという意味ですが、すぐに実行されるかどうかはその時点でのジョブの スケジューリング状況に依存します。

date
日付を示す *date* の指定は任意で、「月」の名前 (LC_TIME ロケールカテゴリの mon または abmon キーワードから得られる値) の後

に「日」を表す数値を記述する (さらにその後にコンマと「年」を表す数値があってもよい) 方法と、曜日 (LC_TIME ロケールカテゴリの day または abday キーワードから得られる値) を記述する方法があります。さらに C ロケールには以下の 2 つの特殊な日付が定義されています。

today	現在の日付が示す日、つまり当日を表します。
tomorrow	現在の日付が示す日の次の日、つまり翌日を表します。

date を省略すると、指定された時刻が現時刻より後であれば当日、現時刻より前であれば翌日とみなされます。*date* として「月」を指定し「年」を省略した場合、月の値が当月よりも前であれば翌年とみなされます。

increment — *increment* 引数は任意指定で、正の符号 (+) 数値の後に minutes、hours、days、weeks、months、years の文字列のいずれかを付加したものです。複数形を示す s は省略できます。また + 1 と同じ意味を持つキーワード next も使用できます。たとえば次の 2 つのコマンドは同じ意味となります。

```
at 2pm + 1 week at 2pm next week
```

使用法

この項で述べる at コマンド行の形式は、C ロケールに対してだけ保証されています。その他のロケールでは、midnight、noon、now、mon、abmon、day、abday、today、tomorrow、minutes、hours、days、weeks、months、years、next の各指定はサポートされていません。

コマンドの実行は、別のプロセスグループで制御端末なしで動作しているシェルを別途呼び出して行うので、コマンドを呼び出した環境でのオープンファイル記述子やトラップ、優先順位などは失われてしまいます。

at

例 1 端末での指定例

端末でのコマンドシーケンスの例を示します。

```
$ at −m 0730 tomorrow
sort < file >outfile
<EOT>
```

例 1 端末での指定例　(続き)

例 2 出力先のリダイレクト

次のシーケンスは、出力先を標準エラー出力からパイプに変更するもので、コマンドプロシージャの中で使用すると便利です。なお、出力先変更指定の記述順序は重要なので注意してください。

```
$ at now + 1 hour <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
```

例 3 ジョブによる再スケジュール

ジョブ自身に再スケジュールさせるため、at–job の中から `at` を呼び出すことができます。次の例では、`my.daily` という名の日常業務用スクリプトは毎日実行されます。ただし、`crontab` を使う方法のほうが一般的です。

```
# my.daily runs every day
at now tomorrow < my.daily
daily-processing
```

例 4 時刻、オペランド指定

C ロケールの *timespec* の 3 つの部分は、明示的に記述してあれば高い自由度で使用できます。時間やオペランド指定の例を以下に示します。

```
at 0815am Jan 24
at 8 :15amjan24
at now "+ 1day"
at 5 pm FRIday
at '17
    utc+
    30minutes'
```

batch

例 5 端末での指定例

端末でのコマンドシーケンスの例を示します。

```
$ batch
sort <file >outfile
<EOT>
```

例 6 出力先のリダイレクト

次のシーケンスは、出力先を標準エラー出力からパイプに変更するもので、コマンドプロシージャの中で使用すると便利です。なお、出力先変更指定の記述順序は重要なので注意してください。

```
$ batch <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

例 **6** 出力先のリダイレクト　*(続き)*

環境

at と batch の両ユーティリティの実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES 、NLSPATH 、LC_TIME の詳細については、environ(5) を参照してください。

SHELL　at–job を呼び出すのに用いるコマンドインタプリタの名前を表します。この変数が設定されていないか値が NULL の場合には、sh が使用されます。sh 以外の値に設定されていれば、そのシェルを使用します。このとき、どのシェルを使うかを表す 警告メッセージが出力されます。

TZ　タイムゾーンを表します。ジョブは、*timespec* または -t *time* が示す時刻に実行するために投入されますが、この時刻は TZ 変数が示すタイムゾーンに対応した値です。*timespec* の値にタイムゾーン指定が含まれていれば、TZ が示すゾーンに代わってそちらが使用されます。*timespec* にタイムゾーン指定が含まれておらず、TZ も未設定か NULL の場合、デフォルトのタイムゾーンが用いられます。

DATEMSK　環境変数 DATEMSK が設定されていれば、at はその値を、書式文字列を含んでいるテンプレートファイルの完全パス名として使用します。この文字列は書式記述子とテキスト文字から構成され、環境変数 LANG または LC_TIME の設定値に従って各国の言語固有の日付表示形式をサポートするために 使用されます。利用可能な書式記述子の一覧については、getdate(3C) を参照してください。なお「オペランド」の項で説明している time と date 引数、特殊名の noon、midnight、now、next、today、tomorrow、さらに *increment* 引数の書式は、DATEMSK が設定されている場合には認識されません。

終了ステータス

以下の終了ステータスが返されます。

0　at ユーティリティによるジョブの投入、削除、または一覧表示が正常終了した

>0　エラーが発生したので、ジョブはスケジュールされない

ファイル

/usr/lib/cron/at.allow　at と batch の両ユーティリティへのアクセス権を持つユーザーの一覧。1 行に 1 ユーザー名の形式

/usr/lib/cron/at.deny　at と batch の両ユーティリティへのアクセスを許可しないユーザーの一覧。1 行に 1 ユーザー名の形式

属性

次の属性については attributes(5) のマニュアルページを参照してください。

at

属性タイプ	属性値

使用条件	SUNWcsu
CSI	未対応

batch

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 `auths`(1)、`crontab`(1)、`csh`(1)、`date`(1)、`ksh`(1)、`sh`(1)、`touch`(1)、`ulimit`(1)、`umask`(1)、`cron`(1M)、`getdate`(3C)、`auth_attr`(4)、`attributes`(5)、`environ`(5)

注意事項 待ち行列を使用しているかどうかにかかわらず、`cron`(1M) の実行は 100 ジョブに限られています。

`cron` ではジョブの実行に遅れの出る場合があります。これらの遅れによって `cron` ジョブの処理がハングしたように見えることがあります。すべてのジョブは最後には実行されますが、極端に遅れが生じた場合には、`cron` を終了させてから再起動させることが 唯一の回避策です。

bc(1)

名前 | bc – 任意精度の演算言語

形式 | **bc** [-c] [-l] [*file...*]

機能説明 | bc ユーティリティは、任意の精度の電卓機能を実装します。このユーティリティは、指定したファイルから入力を読み取り、次に標準入力から読み取ります。bc の標準入力と標準出力が端末に接続されている場合、bc は対話形式で実行され、以下の項で説明する動作上の制約を受けます。bc は、C に似た言語を処理します。bc は、計算プログラム dc のプリプロセッサであり、-c オプションが指定されていない場合は自動的に dc を呼び出します。この場合、dc への入力は、標準出力に送られます。

使用法 | bc プログラムの構文は次のとおりです。

L a ～ z の文字を表します。

E 式を表します。算術値または論理値、値をとるオペランド、または値に評価されるオペランドと演算子の組み合わせです。

S 文を表します。

注釈 | /* と */で囲みます。

名前 **(**オペランド**)** | 単純変数： *L*
配列要素: *L* [*E*] (最大 BC_DIM_MAX 次元まで)
ワード：ibase、obase (最大 BC_BASE_MAX まで)、および scale (最大 BC_SCALE_MAX まで)

その他のオペランド | オプションの符号および小数点付きの任意の桁数の数字。BC_STRING_MAX 文字より少ない、二重引用符 (") で囲まれた文字列 (*E*)

sqrt (*E*)	立方根
length (*E*)	10 進数の有効桁数
scale (*E*)	小数点の右側の桁数

L (*E* , ... , *E*)

演算子 | + − * / % ^
(% は剰余、 ^ はべき乗)

++ −−
(前置と後置、名前に適用される)

== <= >= != < >

= =+ =− =* =/ =% =^

文 | *E*
{ *S* ; . . . ; *S* }
if (*E*) *S*
while (*E*) *S*
for (*E* ; *E* ; *E*) *S*

NULL 文
`break`
`quit`

.string

関数定義

`define` *L* (*L* ,... , *L*) {
`auto` *L* ,... , *L*
S ;... *S*
`return` (*E*)
}

-l 数字ライブラリの関数

`s`(*x*)	正弦関数
`c`(*x*)	余弦関数
`e`(*x*)	指数関数
`l`(*x*)	対数関数
`a`(*x*)	逆正接関数
`j`(*n*,*x*)	ベッセル関数

すべての関数の引数は、値で渡されます。

式である文の値は、主演算子が代入演算子でない限り出力されます。セミコロンまたは復帰改行のいずれかで文を区切ることができます。`scale` への数の代入は、`dc` 方式の演算において保存される桁数に影響を与えます。`ibase` または `obase` への代入により、入力と出力の基数がそれぞれ設定されます。

同じ文字を、同時に配列、関数、および単純変数に使用することができます。すべての変数はプログラムに対してグローバルです。`auto` 変数は、関数呼び出し中はスタックされます。配列を関数の引数、または自動変数の定義に使用する場合は、空の角括弧を配列名の後に続ける必要があります。

オプション

次のオプションを指定できます。

`-c`	コンパイルだけを行います。出力は、標準出力に送られる `dc` コマンドです。
`-l`	数字関数を定義して、`scale` を `0` (デフォルト) ではなく `20` に初期化します。

オペランド

次のオペランドを指定できます。

file	`bc` プログラムの文を含むテキストファイルのパス名。`bc` はすべての *file* オペランドを読み取ってから、標準入力を読み取ります。

使用例

例 1 変数の精度の設定

シェルにおいて、次のコードは、n の最初の 10 桁の近似値を変数 *x* に代入します。

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

例 2 計算関数の定義

関数を定義して、指数関数の近似値を計算します。

```
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1; 1==1; i++){
        a = a*x
        b = b*i
        c = a/b
        if(c == 0) return(s)
        s = s+c
    }
}
```

例 3 関数の近似値の出力

最初の 10個の整数の、指数関数の近似値を出力します。

```
for(i=1; i<=10; i++) e(i)
```

または

```
for (i = 1; i <= 10; ++i) {         e(i) }
```

環境

bc の実行に影響を与える環境変数 LC_CTYPE、 LC_CTYPE、 LC_CTYPE については、environ(5) のマニュアルページを参照してください。

終了ステータス

次の終了ステータスが返されます。

0	すべての入力ファイルが正常に処理された
unspecified	エラーが発生した

ファイル

/usr/lib/lib.b	数字ライブラリ
/usr/include/limits.h	BC_ パラメータを定義します

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | dc(1), awk(1), attributes(5)

注意事項 | bc コマンドは、論理演算子 && および || を認識しません。

for 文には 3 つの式 (*E*) のすべてが必要です。

名前 | bdiff – 大規模ファイルの diff

形式 | **bdiff** *filename1 filename2* [*n*] [-s]

機能説明 | bdiff は diff と同様の機能を持ち、 *filename1* と *filename2* 引数で示した2 つのファイルの相違箇所を見つけ出します。 本コマンドは、 diff が処理可能な限度を超えた大きなファイルを比較するために提供されています。 どちらかの引数にファイル名の代わりに – を指定すると、標準入力が比較対象となります。

bdiff は両ファイルの先頭にある共通部分を無視し、残りの部分を *n* 行ずつのセグメントに分割し、個々のセグメントを比較するために diff を呼び出します。ファイル名以外の2 つのオプションを両方とも 指定する場合、上記の「形式」で示した順序で記述しなければなりません。

bdiff の出力は diff の出力とまったく同じ形式です。 行番号もファイルの分割を考慮して連続したものが割り当てられます。 したがって出力リストは、ファイル全体が一度に比較されたかのように 見えます。ただし分割により比較が行われているため、bdiff は、全体を一度に比較した場合に比べて、 必ずしもファイルの微妙な差を探し出せるわけではないことに注意してください。

オプション

n	分割の単位とする行数を指定します。デフォルトは 3500 です。 3 番目の引数が記述されていてその値が数値の場合、 自動的に行数指定と見なされます。本引数は、デフォルトの行数である 3500 行が diff にとって多すぎて処理できない、といった場合に使用すると便利です。
-s	サイレントオプション。 bdiff が通常出力する診断メッセージを抑止します。ただし bdiff が呼び出す diff からの診断メッセージは、本オプションが指定されていても出力されます。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の bdiff の動作については、largefile(5) を参照してください。

ファイル | /tmp/bd?????

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 | diff(1), attributes(5), largefile(5)

診断 | 診断メッセージの意味については help を使用してください。

名前 | bfs – 大型ファイル用のスキャナ

形式 | **/usr/bin/bfs** [-] *file*

機能説明 | bfs コマンドは、ほとんど ed(1) と同じですが、読み取り専用で、より大きなファイルを処理する点が異なります。ファイルは最大 1024K バイト、32K 行で、1 行あたりに復帰改行を含めて最大 512 文字 (16 ビットマシンの場合は 255 文字) を含めることができます。bfs ではファイルがバッファにコピーされないため、ファイルの走査は一般に ed(1) よりも効率的です。csplit(1) を使用して、大きなファイルを編集のために扱いやすい大きさのいくつかのファイルに分割できる場合、bfs は、大きなファイルの各セクションを識別するのに役立ちます。

通常、w (write) コマンドで書き込んだファイルのサイズが出力されるのと同様に、走査しているファイルのサイズが出力されます。オプションの – は、サイズの出力を抑制します。P とキャリッジリターンを入力すると、ed(1) の場合と同様に入力を求めるプロンプトとして * が出力されます。プロンプトは、再度 P とキャリッジリターンを入力するとオフにできます。メッセージは、プロンプトがオンになっている場合には、エラーに対してメッセージが出力されることに注意してください。

ed(1) で説明されているアドレス表現式はすべて使用できます。また、正規表現を、/ と ? 以外の 2 つの記号で囲むことができます。

> 折り返しのない下方検索を示します。

< 折り返しのない上方検索を示します。

マーク名には多少の違いがあります。つまり、a ～ z の文字だけが使用可能であり、26 のマークすべてが記憶されます。

bfs コマンド | e、g、v、k、p、q、w、=、!、および NULL の各コマンドは、ed(1) の説明どおりに動作します。——、+++−、+++=、−12、+4p などのコマンドは受け入れられます。1、10p、および1,10 は、いずれも最初の 10 行を出力します。f コマンドは、走査中のファイルの名前だけを出力します。ファイル名の記憶機能はありません。w コマンドは、出力先の変更、切り捨て、または圧縮機能には影響されません (以下の xo、xt、および xc の各コマンドを参照)。次の追加コマンドを使用できます。

xf *file*
指定した *file* からコマンドを取り出します。ファイルの終わり、割り込みシグナルの受信、またはエラーの発生により、コマンドの読み取りは、xf を含むファイルに戻ります。xf コマンドは、10 の深さまでネストすることができます。

xn
現在使用中のマークをリストします (マークは、k コマンドで設定されます)。

xo *[file]*
p および NULL コマンドからの出力を、指定した *file* に送ります。このファイルは、umask の設定 (umask(1) を参照) で別の指定がされているのでなければ、必要に応じてモード 666 (だれでも読み書きが可能) で作成されます。*file* を指定しないと、出力は標準出力に送られます。出力先の変更を行うたびに、ファイルの切り捨てや作成が行われることに注意してください。

: *label*
コマンドファイル内に *label* 定義します。*label* は復帰改行で終了し、: (コロン) と *label* の先頭の間にあるブランクは無視されます。ラベルは参照されなくてもよいので、このコマンドは、コマンドファイル内に注釈を挿入するためにも使用できます。

(. , .)xb/*regular expression*/*label*
このコマンドが正常に実行すると、*label* への (上方または下方) ジャンプが行われます。次の条件のいずれかが成立すると、ジャンプは失敗します。

1. アドレスが 1 と $ の間にない。
2. 2 番目のアドレスが 1 番目のアドレスよりも小さい。
3. 最初と最後の行を含む指定した範囲内に、正規表現と一致する行がない。

正常終了すると、一致した行に . (ドット) が設定されて、*label* へのジャンプが行われます。このコマンドは、アドレスが間違っているときにもエラーメッセージを出さない唯一のコマンドですので、他のコマンドを実行する前にアドレスが間違っていないかを検査するために使用できます。コマンド、xb/^/ label は無条件ジャンプであることに注意してください。

xb コマンドは、端末以外から読み取られた場合にのみ使用できます。パイプから読み取られた場合は、下方ジャンプだけしか実行できません。

xt *number*
p および NULL コマンドからの出力を、*number* で指定した文字数まで切り捨てます。初期値は 255 です。

xv[*digit*] [*spaces*] [*value*]
xv に続いて指定された *digit* が変数名になります。コマンド xv5100 と xv5 100 は、どちらも値 100 を変数 5 に割り当てます。コマンド xv61、100p は、値 1、100p を変数 6 に割り当てます。変数を参照するには、変数名の前に % を付けします。たとえば次の例は、上記の割り当てを変数 5 と 6 に使用します。

```
1,%5p
1,%5
%6
```

すべてで、最初の 100 行が出力されます。

```
g/%5/p
```

上記のコマンドは、文字 100 をグローバルに検索して、一致した各行を出力します。% の特殊な意味をエスケープするには、その前に \ を付ける必要があります。

```
g/".*\%[cds]/p
```

上記のコマンドは、文字、10 進整数、または文字列の %c、%d、または %s 形式 (たとえば、printf のような文) を照合してリストするために使用できます。xv コマンドの機能には、このほかに UNIX コマンドの出力の最初の行を変数に格納できるというものがあります。そのための条件は、*value* の最初の文字が ! でなければならないということだけです。次に例を示します。

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

上記のコマンドは、現在の行を変数 35 に入れ、それを出力して、変数 36 を 1 増分します。value の最初の文字としての ! の特殊な意味をエスケープするには、その前に \ を付けます。

```
xv7\!date
```

上記のコマンドは、値 !date を変数 7 に格納します。

xbz *label*
xbn *label*

これらの 2 つのコマンドは、UNIX コマンド (! コマンド) の実行で最後に保存されたリターンコード、または 0 以外の値をそれぞれテストして、指定された *label* にジャンプします。次の 2 つの例はいずれも、文字列 size を含む次の 5 行を検索するものです。

例 1 :

```
xv55
: l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
```

例 2 :

```
xv45
: l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz l
```

xc [switch]

switch が 1 の場合、p および NULL コマンドからの出力は圧縮されます。switch が 0 の場合は圧縮されません。引数がない場合、xc は switch を反転します。switch は、最初は圧縮を行わないように設定されています。圧縮された出力では、タブとブランクからなる文字列は 1 つのブランクに置き換えられ、空白行は出力されません。

オペランド　次のオペランドを指定できます。

file 最大 1024K バイト、32K 行、1 行あたりに復帰改行を含め最大 512 文字 (16 ビットマシンの場合は 255 文字) を収納するすべてのファイル。*filename* には、`csplit`(1) によって編集用に扱いやすい大きさに分割された大きなファイルの 1 つのセクションを指定することができます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了 (ファイルエラーおよびコマンドエラーなし)

`>0` エラーが発生した

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目

`csplit`(1), `ed`(1), `umask`(1), `attributes`(5)

診断

プロンプトがオフの場合、コマンドにエラーがあると ? だけが表示されます。プロンプトがオン場合は、詳しいエラーメッセージが表示されます。

名前 | jobs, fg, bg, stop, notify – プロセスの実行の制御

sh **jobs** [-p | -l] [% *job_id*...]

jobs -x *command* [*arguments*]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

csh **jobs** [-l]

fg [% *job_id*]

bg [% *job_id*...]

notify [% *job_id*...]

stop % *job_id*...

stop *pid*...

ksh **jobs** [-lnp] [% *job_id*...]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

sh ジョブ制御が有効なとき、Bourne シェルに組み込まれた jobs は、停止中またはバックグラウンドで実行中の すべてのジョブを表示します。 %*job_id* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが表示されます。次のオプションを使って、ジョブに関する表示を変更できます。

-l ジョブのプロセスグループ ID および作業ディレクトリを表示します。

-p ジョブのプロセスグループ ID のみを表示します。

-x *command* または *argument* 中に見つかった *job_id* を、対応するプロセスグループ ID に置き換え、*command* に *argument* を渡して実行します。

シェルを jsh として呼び出すと、sh の説明で述べたすべての機能に加えて、ジョブ制御が可能になります。通常、ジョブ制御は対話型シェルに対してだけ可能です。通常、非対話型シェルは、ジョブ制御の機能 を使用しません。

ジョブ制御が可能なとき、ユーザーが端末から入力したコマンドまたはパイプラインは、すべて *job_id* と呼ばれます。どのジョブも、必ずフォアグラウンド、バックグラウンド、または停止のいずれかの状態にあります。 これらの用語の定義を次に示します。

1. フォアグラウンド状態にあるジョブは、 制御している端末への読み取りおよび 書き込みアクセス権を持っています。
2. バックグラウンド状態にあるジョブは、 制御している端末への読み取りアクセスを拒否されていますが、 条件付き書き込みアクセス権は持っています (`stty`(1) を参照)。
3. 停止ジョブは保留状態に置かれたジョブであり、通常は `SIGTSTP` シグナルにより、この状態になります (`signal`(3HEAD) を参照)。

シェルが起動するすべてのジョブには、 ジョブ番号 (*job-id number*) と呼ばれる正の整数が割り当てられます。シェルはこの番号を把握し、 特定のジョブを示す識別子として使用します。また、シェルは現在 (*current*) および前回 (*previous*) のジョブも把握しています。現在のジョブとは、最後に起動または再起動されたジョブです。 前回のジョブとは、その直前のジョブです。

ジョブ識別子の正しい構文は次のような形式です。

`%`*job_id*

job_id は、次のいずれかの形式で指定できます。

`%` または `+`	現在のジョブ
–	前回のジョブ
`?`<*string*>	*string* を含むコマンド行 (一意に表す) に対応したジョブ
n	ジョブ番号が *n* のジョブ
pref	コマンド名の先頭が *pref* のコマンド。 たとえば `ls -l` *name* がバックグラウンドで実行中だった場合、`% ls` と指定すればこのコマンドを示すことができます。*pref* は、引用符で囲まない限り、空白文字を含めることができません。

ジョブ制御が有効なとき、`fg` は中断しているジョブの実行を フォアグラウンドで再開します。 またバックグラウンドで実行中のジョブを フォアグラウンドに移動します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

ジョブ制御が有効なとき、`bg` は中断されているジョブの実行を バックグラウンドで再開します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

`stop` は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (`ps`(1) を参照) 。

csh C シェルに組み込まれた `jobs` は、引数なしでジョブ制御下で活動中のジョブを一覧表示します。

`-l` 通常の情報の他に、プロセス ID を表示します。

シェルは、番号の付いた *job_id* を各コマンドシーケンスと対応付けて、バックグラウンドで動作中のコマンド、または TSTP シグナル (通常は Control-Z) によって停止したコマンドの動作を追跡します。コマンドまたはコマンドシーケンス (セミコロンで区切られたリスト) をメタキャラクタ & を使用してバックグラウンドで起動した場合、 シェルは角括弧で囲まれたジョブ番号と 関連するプロセス番号のリストを表示します。以下に例を示します。

[1] 1234現在のジョブリストを見るには、組み込みコマンド jobs を使用します。最後に停止したジョブ (停止したジョブがない場合は、 最後にバックグラウンドに投入されたジョブ) を「現在のジョブ」といい、 '+' で示します。前のジョブは '-' で示します。 現在のジョブが終了したりフォアグラウンドに移された場合、 前のジョブが新しく現在のジョブになります。

ジョブの操作方法については、組み込みコマンド bg、 fg、 kill、 stop、 % の説明を参照してください。

ジョブの参照は '%' で始まります。 パーセント記号だけの指定は、現在のジョブを示します。

% %+ %%	現在のジョブ
%-	前のジョブ
%*j*	'kill -9 %*j*' のようにジョブ *j* を参照します。 *j* はジョブ番号、またはジョブを起動した コマンド行を一意に表す文字列です。 たとえば 'fg %vi' は、停止した vi ジョブをフォアグラウンドに移します。
%?*string*	*string* を含むコマンド行 (一意に表す) に対応したジョブを指定します。

バックグラウンドで動作中のジョブは、端末からの読み取り時に停止します。 バックグラウンドジョブは、通常出力を生成しますが、'stty tostop' コマンドを使用して抑止することも可能です。

fg は現在のジョブまたは指定された *job_id* をフォアグラウンドへ移します。

bg はバックグラウンドで、現在のジョブ または指定されたジョブを実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

notify は現在のジョブまたは指定されたジョブの状態が変わったとき、 その旨非同期にユーザーに知らせます。

ksh jobs は、現在のシェル環境で開始されたジョブの状況を表示します。 jobs がジョブの終了を報告したとき、シェルはそのジョブのプロセス ID を、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除します。

特定のジョブの報告だけが必要なら、*job_id* を使ってジョブを指定します。*job_id* を 1 つも指定しないと、全ジョブに関する情報が出力されます。

以下のオプションは、 `jobs` の出力を変更または拡張するために使用します。

`-l` (文字のエル) 個々のジョブに関して詳細な情報を出力します。具体的には、 ジョブ番号、現在のジョブ、プロセスグループ ID 、状態、ジョブを生成したコマンドを出力します。

`-n` 前回通知を受けた後に停止または終了した ジョブだけを表示します。

`-p` 選択されたジョブのプロセスグループリーダのプロセスグループ ID だけを出力します。

デフォルトでは、`jobs` は、停止しているすべてのジョブの状態、実行中のバックグラウンドジョブの 状態、そして状態が変わったのにシェルによりまだ報告されていない すべてのジョブの状態を表示します。

`set` コマンドの `monitor` オプションを有効にすると、対話型シェルが `job` を各パイプラインと関連付けます。 このオプションは、`jobs` コマンドが表示する現在のジョブのテーブルを維持し、これらのジョブに整数番号を割り当てます。ジョブを `&` で非同期に起動すると、シェルは、`[1] 1234` という形式の行を表示します。 非同期に起動されたジョブはジョブ番号 1 であり、プロセス ID が 1234 であるプロセスが 1 つ (トップレベル) あることを示します。実行中のジョブがあるが、 別に実行したいジョブがある場合、`^Z` (Control-Z) キーを押せば、 現在のジョブに `STOP` シグナルが送信されます。 そうするとシェルは通常、 ジョブが「停止」されたことを示し (後述の「出力」の項を参照)、 プロンプトを表示します。これで、このジョブの状態を `bg` コマンドでバックグラウンドで処理するか、または他のコマンドを実行してから、 `fg` というコマンドで ジョブをフォアグラウンドに移すことができます。`^Z` は直ちに有効になります。つまり `^Z` は、 保留中の出力や読み取られていない入力が 直ちに中止されるという点で、割り込みに似ています。

シェル内のジョブを参照する方法はいくつかあります。そのジョブのいずれかのプロセスの ID を使っても、また以下のいずれかを使っても参照できます。

`%`*number* *number* が示す番号のジョブ

`%`*string* コマンド行が *string* で始まっていたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%?`*string* コマンド行が *string* を含んでいたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%%` 現在のジョブ

`%+` `%%` と同じ

`%-` 直前のジョブ

シェルは、 プロセスの状態が変更すると、 直ちにそれを検出します。 ジョブがブロックされてそれ以上進めない状態になると、 シェルはその旨をユーザーに通知します。 ただし、ユーザーの作業の妨げにならないよう、 この通知はプロンプトを発行

する直前にだけ行います。 モニタモードが有効なとき、 完了した各バックグラウンドジョブは、 CHLD に設定されているトラップを起こします。 ジョブの実行中または停止中にシェルを終了しようとすると、 「停止中 (実行中) のジョブがある (‘You have stopped (running) jobs.’) 」旨の警告を受けます。 jobs コマンドを使用すれば、 どのジョブが該当するのかを確認できます。これを実行するか、 または直ちにシェルを再終了しようとすると、 シェルは 2 度目の警告は出さず、 停止中のジョブは終了します。

fg は、バックグラウンドジョブを、現在の環境からフォアグラウンドへ移します。 fg を使ってジョブをフォアグラウンドへ移した場合、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除されます。 fg コマンドが使えるのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をフォアグラウンドで実行します。*job_id* が指定されないと、 現在のジョブをフォアグラウンドで実行します。

bg は、現在の環境で中断されたジョブを、バックグラウンドジョブとして 実行することにより再開します。 *job_id* が示すジョブがすでにバックグラウンドジョブを実行している場合、bg は何も行わず正常に終了します。bg を使ってジョブをバックグラウンドへ移した場合、あたかも非同期リストから起動されたかのように、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID 」の 1 つとなります。 bg コマンドが使えるのは、 ジョブ制御をサポートするシステム上だけです。*job_id* が省略された場合は、 現在のジョブをバックグラウンドで実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

出力

-p オプションを指定すると、各プロセス ID に対して次に示す 1 行の情報が出力されます。

"%d\n", *"process ID"*

-p を省略すると、 -l オプションも省略されていれば、以下の形式の一連の行が出力されます。

"[%d] %c %s %s\n", *job-number*, *current*, *state*, *command*

各フィールドの意味を以下に説明します。

current

文字 + は、 fg および bg コマンド用のデフォルトとして使用するジョブを表します。 このデフォルトジョブは、 *job_id* %+ または %% を使って指定することもできます。文字 – は、現在のデフォルトジョブが終了してしまった場合に デフォルトとなるジョブを表します。このジョブは、 *job_id* %– を使って指定することもできます。 その他のジョブに関しては、このフィールドは空白文字として出力されます。 + や – を使って表せるジョブの数は、どちらも最大 1 つです。停止中の

ジョブがあれば、現在のジョブも停止ジョブとなります。 停止中のジョブが 2 つ以上あれば、以前のジョブも停止ジョブとなります。

job-number

`wait` 、 `fg` 、 `bg` 、 `kill` の各ユーティリティ用にプロセスグループを識別するのに使用する番号。 これらのユーティリティを使うと、ジョブはジョブ番号の後に `%` を付加することにより識別できます。

state

以下の文字列 (POSIX ロケールにある) のいずれかです。

`Running`	ジョブはシグナルによって中断されておらず、 終了もしていないことを表す。
`Done`	ジョブは終了して、ゼロの終了ステータスを返したことを表す。
`Done`(*code*)	ジョブは正常に終了し、指定されたゼロ以外の終了ステータス (*code* が示す 10 進数) を返したことを表す。
`Stopped`	
`Stopped (SIGTSTP)`	`SIGTSTP` シグナルがジョブを停止したことを表す。
`Stopped (SIGSTOP)`	`SIGSTOP` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTIN)`	`SIGTTIN` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTOU)`	`SIGTTOU` シグナルがジョブを停止したことを表す。

利用者側で、文字列 `Stopped` の代わりに `Suspended` を使うよう定義することができます。 ジョブをシグナルが終了した場合、 `state` の形式は不定ですが、ここに示した他の `state` 形式とは明確に区別できるものです。その出力上で、 ジョブを終了させたシグナルの名前または説明が与えられます。

command

シェルに与えられた関連コマンド。

`-l` オプションを指定すると、プロセスグループ ID を示すフィールドが `state` フィールドの前に挿入されます。さらに、プロセスグループ内の より多くのプロセスが別の行に出力されることがあります。 その内容は、プロセス ID と `command` フィールドだけです。

環境

jobs 、 fg 、 bg の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

jobs 、 fg 、 bg は、以下の終了ステータスを返します。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), kill(1), ksh(1), ps(1), sh(1), stop(1), shell_builtins(1), stty(1), wait(1), signal(3HEAD), attributes(5), environ(5)

名前 | break, continue – while、for、foreach、until の各ループ制御から抜け出す、または続行するためのシェル組み込み関数

sh | **break** [*n*]

continue [*n*]

csh | **break**

continue

ksh | ***break** [*n*]

***continue** [*n*]

sh | break は for ループまたは while ループ中であれば、ループを終了します。 *n* を指定すると、 *n* レベル分だけ ループを終了します。

continue は for ループまたは while ループの次の繰り返しを実行します。 *n* を指定すると、 *n* 番目のループから実行します。

csh | break は foreach または while 内の最も内側にあるループの end の次から実行を再開します。 現在の行の残りのコマンドは実行されます。 これによって、複数レベルのループから抜けるには、 break コマンドを 1 行に複数記述します。

continue は while または foreach 内の最も内側にあるループの、次の繰り返しから実行します。

ksh | break は for ループ、 while ループ、 until ループ、または select ループがあれば終了します。 *n* を指定すると、 *n* レベル分だけ ループを終了します。

continue は for ループ、 while ループ、 until ループ、 または select ループの次の繰り返しを実行します。 *n* を指定すると、 *n* 番目のループから実行します。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | csh(1), exit(1), ksh(1), sh(1), attributes(5)

名前 | cal – カレンダの表示

形式 | **cal** [[*month*] *year*]

機能説明 | cal ユーティリティは、グレゴリ暦のカレンダを標準出力に書き込みます。*year* オペランドを指定すると、その年のカレンダが書き込まれます。オペランドを指定しないと、現在の月のカレンダが書き込まれます。

オペランド | 次のオペランドを指定できます。

month 表示する月を、1 (1 月) 〜 12 (12 月) の 10 進数で指定します。デフォルト値は、現在の月です。

year カレンダを表示する年を、1 〜 9999 の 10 進数で指定します。デフォルト値は、現在の年です。

環境 | cal の実行に影響を与える環境変数 LC_TIME, LC_MESSAGES, , NLSPATH については、environ(5) のマニュアルページを参照してください。

終了ステータス | 次の終了ステータスが返されます。

0 正常終了

>0 エラーが発生した

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | calendar(1), attributes(5), environ(5)

注意事項 | 1752 年の 9 月のカレンダ出力は、通常のものと異なります。この月は、うるう年の調整の欠落を補うために 11 日分とばされています。このカレンダを見るには、次のように入力します。

```
cal 9 1752
```

コマンド cal 83 は、西暦 1983 年ではなく、西暦 83 年を出力します。

年は、常に 1 月から始まるものと見なされます。

名前 | calendar – リマインダーサービス

形式 | **calendar** [-]

機能説明 | calendar ユーティリティは現在のディレクトリにあるファイル calendar を参照して、今日または明日の日付を行中のどこかに含んだ行を 標準出力に書き出します。 Aug. 24 、 august 24 、 8/24 などの最も一般的な月日の日付は認識されますが、 24 August や 24/8 は認識されません。 金曜日と週末に"明日"と言う場合は、月曜日を指します。 calendar は、 crontab(1) や at(1) コマンドを使用して、定期的に起動できます。

オプションの引数 - がある場合、 calendar は自分のログインディレクトリにファイル calendar を持つすべてのユーザーに対し処理を実行し、 mail(1) によって結果を送ります。 通常、この仕事は UNIX オペレーティングシステムの機能として毎日実行されています (cron(1M) 参照)。

環境変数 DATEMSK が設定されている場合、 calendar はその値をフォーマット文字列を含む テンプレートファイルの完全なパス名として使用します。 この文字列は、変換仕様とテキスト文字からなり、 環境変数 LANG や LC_TIME を適切に設定することによって、 異なる言語で許容される日付フォーマットをより豊富に提供するために使用します (environ(5) 参照) 。 可能な変換仕様のリストについては strftime(3C) を参照。

使用例 | **例 1** テンプレートの内容

以下に、テンプレートファイルの内容の例を示します。

```
%B %eth of the year %Y
```

%B は完全な月名、 %e は日付、 %Y は年 (4 桁)を表します。

DATEMSK がこのテンプレートファイルを指していれば、次のような calendar ファイルが有効になります。

```
March 7th of the year 1989 < Reminder>
```

環境 | calendar の実行に影響を与える環境変数 LC_CTYPE、 LC_TIME、 LC_MESSAGES、 NLSPATH、 TZ についての詳細は、environ(5) を参照してください。

終了ステータス

0	正常終了
>0	エラーが発生した

ファイル

/etc/passwd	システムパスワードファイル
/tmp/cal*	calendar が使用するテンポラリファイル
/usr/lib/calprog	今日または明日の日付を計算するのに使用するプログラム

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 `at`(1), `crontab`(1), `mail`(1), `cron`(1M), `ypbind`(1M), `strftime`(3C), `attributes`(5), `environ`(5)

注意事項 空白で始まる適切な行は印刷されません。

本サービスを受けるには、カレンダーが公的に認められたものでなければなりません。

`calendar` は"明日"を拡張解釈し、休日は数えません。

引数 – は、マシンに対してローカルなファイルシステム上の カレンダーファイルに対してのみ有効です。 つまり `NFS` を使ってリモートでマウントされたファイルシステム上の ファイルは処理対象とはなりません。 したがって ‘`calendar -`’ 指定は、ホームディレクトリが存在する ディスクフルクライアント上で実行しなければなりません。 ディスクレスクライアント上で実行しても効果はありません。

`calendar` は、現在ではデフォルトの root の crontab 内には存在していません。 ‘`calendar -` はネットワークに負荷をもたらすので、 `ypbind`(1M) を使用して、大きな passwd.byname マップ を管理している環境では、使用しない方がよいでしょう。 ただし、ネットワークへの影響をしのぐだけの便利さが `calendar` にある、と判断した場合には、スーパーユーザーは ‘`crontab -e`’ を使用して root の crontab を編集してもかまいません。 ネットワークへの影響を避けたければ、個々のユーザーが ‘`crontab -e`’ を使って、 – 引数なしの `calendar` を `cron` が呼び出すよう、つまり出力を自分宛のメールにパイプするように 自身の crontab を編集することができます。

名前 | cancel – 印刷要求の取消し

形式 | **cancel** [*request-ID*...] [*destination*...]

cancel -u *user*... [*destination*...]

機能説明 | cancel コマンドは印刷要求を取り消します。 cancel コマンドには 2 つの形式があります。

第 1 の形式では、 印刷要求 (*request–ID*) と宛先 (*destination*) の 2 つの引数がオプションとして指定できます。 *request-ID* を指定するか、または *destination* を指定して、 *request-ID* 、または *destination* で示される要求を取り消します。 *destination* だけを指定した場合、 *destination* 上に現在ある印刷要求を取り消します。 *destination* が省略された場合には cancel はすべての宛先にリクエストされた印刷要求を取り消します。

第 2 の形式は、 特定の宛先にあるユーザーの印刷要求を取り消します。

ユーザーは自身のユーザー名に対応する印刷要求だけを取り消すことができます。 デフォルトでは、 ユーザーが印刷要求を送信したホスト上でのみ 印刷要求を取り消すことができます。 スーパーユーザーが 印刷サーバー上の /etc/printers.conf 内に user-equivalence=true を設定した場合、 ユーザーは自身のユーザー名に対応する印刷要求であれば、どのホスト上でも 取り消すことができます。 スーパーユーザーは印刷要求が送信されたホスト上で印刷要求を取り消すことができ、 また、印刷サーバーからでも取り消すことができます。

宛先の情報を決定するとき、印刷クライアントに関係するコマンドはネームサービススイッチ内にある printers データベースを使用します。詳細については、nsswitch.conf(4)、および printers.conf(4) のマニュアルページを参照してください。

オプション | 次のオプションを指定できます。

-u *user* — 印刷要求を取り消すユーザー名。 *user* にはユーザー名を指定します。

オペランド | 以下のオペランドを指定できます。

destination — 印刷要求を取り消す宛先。 *destination* にはプリンタまたはプリンタのクラスのどちらかを 指定できます (lpadmin(1M) を参照)。 *destination* が省略された場合、 cancel はすべての宛先にリクエストされた印刷要求を取り消します。 名前、 POSIX スタイル名 (*server*:*destination*) 、 またはフェデレーテッド・ネーミング・サービス (FNS) 名 (.../service/printer/...) を使用して、 *destination* を指定します。 cancel に POSIX スタイルの宛先名を用いる場合は、 「注意事項」を参照してください。 名前や FNS 名の命名規約については printers.conf(4) を参照してください。

request–ID — 取り消しする印刷要求。 LP スタイルのリクエスト ID (*destination-number*) を使用して *request–ID* を指定します。

user 印刷要求を取り消すユーザー名。 *user* にはユーザー名を指定します。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
0 以外	エラーが発生した。

ファイル

/var/spool/print/*	LP 印刷待ち行列
$HOME/.printers	ユーザーが構成可能なプリンタデータベース
/etc/printers.conf	システムのプリンタ構成データベース
printers.conf.byname	/etc/printers.conf の NIS バージョン
printers.org_dir	/etc/printers.conf の NIS+ バージョン
fns.ctx_dir.*domain*	/etc/printers.conf の FNS バージョン

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpcu

関連項目

lp(1), lpq(1B), lpr(1B), lprm(1B), lpstat(1), lpadmin(1M), nsswitch.conf(4), printers(4), printers.conf(4), attributes(5), standards(5)

注意事項

destination が LP スタイルの *request–ID* と同じ形式である場合、 POSIX スタイルの宛先名 (*server:destination*) は、印刷要求として扱われます。 standards(5) のマニュアルページを参照してください。

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | cat – ファイルの連結と表示

形式 | **cat** [-nbsuvet] [*file...*]

機能説明 | cat は *file* を指定された順に読み込み、標準出力に出力します。 次の例は、*file* を端末に出力します。

```
example% cat file
```

また、次の例では *file1* と *file2* を連結して、結果を *file3* に出力します。

```
example% cat file1 file2 > file3
```

入力ファイルが指定されなかったときは、 cat は標準入力ファイルから読み込みます。

オプション | 以下のオプションを指定できます。

-n 各出力行の前に、行番号を振ります。

-b -n と同様に行番号を振ります。ただし、空行は数えません。

-u 出力はバッファリングされません (デフォルトではバッファリングされます)。

-s cat は、ファイルが存在しないとき何もしません。

-v 非印字文字 (タブ、復帰改行文字、用紙送り文字 (フォームフィード)は除く) を印刷します。ASCII 制御文字 (8 進数の 000 から 037) は ^*n* として印刷されます。ここで *n* は対応する ASCII 文字で、 8 進数で 100 から 137 (@ 、 A 、 B 、 C 、. . . 、 X 、 Y 、 Z 、 [、 \ 、] 、 ^ 、 _) の範囲の文字です。 DEL 文字 (8 進数で 0177)は ^? として印刷されます。 その他の非印字文字は M-*x* として印刷されます。ここで、 *x* は ASCII の下位 7 ビットで示される文字です。

-v オプションを使用した場合、以下のオプションも使用できます。

-e $ 記号が各行の最後 (復帰改行文字の前) に印刷されます。

-t タブは ^I 、用紙送り文字は ^L として印刷されます。

-e オプションおよび -t オプションは、 -v オプションが指定されていなければ無視されます。

オペランド | 以下のオペランドを指定できます。

file 入力ファイルのパス名。 このオペランドを 1 つも指定しないと、標準入力と見なされます。 file として ' - ' が指定されると、 cat はその時点で標準入力を読み込みます。 このように ' - ' をいくつでも file として指定できますが、 cat はそのたびに標準入力をクローズしたり再オープンすることは ありません。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の cat の動作については、largefile(5) を参照してください。

使用例 | 例 1 1 つのファイルの書き出し

次のコマンドは、 myfile というファイルの内容を標準出力に書き出します。

```
example% cat myfile
```

例 2 2 つのファイルを 1 つのファイルに連結

次のコマンドは、2 つのファイル doc1 と doc2 を連結してその結果を doc.all に書き出します。

```
example% cat doc1 doc2 > doc.all
```

例 3 cat の呼び出し 1 回で 2 組の入力を連結

次の例は、標準入力が端末の場合には、1 回の cat の呼び出しで、端末から 2 組の入力データを得るものです。

```
example% cat start   - middle   - end > file
```

標準入力が通常ファイルの場合には、このコマンドは以下のコマンドと同じ意味を持ちます。

```
cat start   - middle /dev/null end > file
```

なぜなら、1 つ目の ' - ' 指定に対して cat は標準入力ファイルの内容をすべて読み取ってしまい、2 つ目の ' - ' 指定に対してはただちにファイルの終わり (EOF) が検出されるためです。

環境 | cat の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 入力ファイルはすべて正常に出力された。

>0 エラーが発生した。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | touch(1), attributes(5), environ(5), largefile(5)

注意事項

cat の出力先を入力中のファイルに変更すると、 入力ファイルのデータが失われます。たとえば、次の例では filename1 の元データが失われます。

```
example% cat filename1 filename2 > filename1
```

名前 | cd, chdir, pushd, popd, dirs – 現在の作業用ディレクトリの変更

形式 | **/usr/bin/cd** [*directory*]

sh | **cd** [*argument*]

chdir [*argument*]

csh | **cd** [*dir*]

chdir [*dir*]

pushd [+ | *n* | *dir*]

popd [+ *n*]

dirs [-l]

ksh | **cd** [*arg*]

cd *old new*

/usr/bin/cd | /usr/bin/cd ユーティリティは、cd ユーティリティ自身だけの現在のディレクトリを変更します。これは、後述するシェル組み込みの cd とは対照的です。/usr/bin/cd はプロセスの呼び出しには影響しませんが、あるディレクトリを現在のディレクトリとして設定できるかどうかを決定するのに使用できます。

sh | Bourne シェルに組み込まれている cd は、現在のディレクトリを *argument* で指定されたディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *argument* になります。 シェル変数 CDPATH は、 *argument* を含むディレクトリの検索パスを定義します。代替ディレクトリ名は、 コロン (:) で区切ります。 デフォルトのパスは空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは空のパス名で指定します。 このパス名は、等号の直後か、 パスリスト内にある区切り文字のコロンの間に指定します。*argument* の先頭文字が / 、 . 、または .. の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *argument* を検索します。 cd は、 *argument* 中で実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は効率が悪くなります。 そのため、 cd コマンドは、シェルに組み込まれています。 (pwd(1) 、 sh(1) 、 chdir(2) を参照)

chdir は、cd を呼び出すもうひとつの方法です。

csh | *dir* 引数を省略すると、 C シェルに組み込まれている cd は、シェル変数 HOME の値を新たな作業用ディレクトリとして 使用します。 *dir* を指定した場合、それが / 、 . 、または .. で始まる完全なパス名であれば、その *dir* が新たな作業用ディレクトリとなります。 それ以外の場合は、シェル変数 CDPATH が指定するパスと相対関係を持つディレクトリの中から 該当するものを探し出します。 CDPATH の構文は PATH シェル変数と同一で、セマンティクスも似ています。 cd は *dir* に対する実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、C シェルに組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

chdir はシェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。 引数を指定しないと、 そのユーザーのホームディレクトリに変更します。 *dir* が現在のディレクトリからは見つからない相対パス名の場合、変数 cdpath 内のディレクトリリストを検索します。 *dir* が / で始まる値を持つシェル変数の名前である場合、 その変数の値が示すディレクトリに変更します。

pushd はディレクトリスタックにディレクトリをプッシュ (押し込む) します。 引数を指定しないと、スタックにある先頭の 2 つの構成要素を交換します。

+*n* *n* 番目のエントリがスタックの先頭になるよう回転し、 そのディレクトリに移ります。

dir 現在の作業用ディレクトリをスタックにプッシュし、 そのディレクトリに移ります。

popd はディレクトリスタックからポップして (取り出して)、新たに先頭となったディレクトリへ cd します。ディレクトリスタックの構成要素の先頭番号は、 0 となります。

+*n* スタック内の *n* 番目のエントリを破棄します。

dirs はディレクトリスタックを出力します。 現在のディレクトリが最も左に現れるように 時間順に出力されます。 -l 引数を指定すると、 ~ を使った省略形ではなく、完全な形式で出力されます。

ksh Korn シェルに組み込まれた cd コマンドは、上記 2 つの形式の いずれかで入力します。第 1 の形式は、現在のディレクトリを *arg* に変更します。 *arg* が – の場合、ディレクトリを直前のディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *arg* になります。PWD 変数は、現在のディレクトリに設定されます。シェル変数 CDPATH は、*arg* を含むディレクトリの検索パスを定義します。 ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは NULL のパス名で指定します。 このパス名は、 等号の直後か、パスリスト内にある区切り文字のコロンの間に指定します。 *arg* の先頭文字が / 、 . 、 または . . の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *arg* を検索します。 cd の第 2 の形式は、 PWD 中の現在のディレクトリ名における *old* という文字列を *new* という文字列に置換し、 この新規のディレクトリへ変更しようとします。 cd コマンドは rksh では実行できません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、 ksh に組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

オペランド 以下のオペランドを指定できます。

directory 新たな作業用ディレクトリとなるディレクトリの絶対または相対パス名。 cd が相対パス名をどのように解釈するかは、環境変数 CDPATH の設定により異なります。

出力 CDPATH に設定されている空でないディレクトリ名が用いられる場合、 新たな作業用ディレクトリの絶対パス名が 以下のような形式で標準出力に出力されます。

"%s\n", <*new directory*>

それ以外の場合には、何も出力されません。

環境

cd の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

CDPATH　コロンで区切られた、ディレクトリを示すパス名のリスト。 *directory* オペランドの先頭文字がスラッシュ (/) でなく、 先頭部分が . でも .. でもない場合には、 cd はこのリスト内のパス名を順番に検索し、環境変数 CDPATH に指定されている名前のディレクトリから *directory* を探します。その結果、最初に見つかったディレクトリ名が 新たな作業用ディレクトリとなります。 ディレクトリのパス名として空の文字列を指定すると、 それは現在のディレクトリと見なされます。 CDPATH は、設定されていないときには空の文字列として扱われます。

HOME　*directory* オペランドが省略されたときに用いるホームディレクトリの名前

PWD　現在の作業用ディレクトリのパス名。 この変数は、そのディレクトリに移った後に cd により設定されます。

終了ステータス

以下の終了ステータスが返されます。

0　ディレクトリが正常に変更された。

>0　エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), pwd(1), sh(1), chdir(2), attributes(5), environ(5)

名前 | cdrw – CD の読み取りと書き込み

形式 | **cdrw** -i [-vSCO] [-d *device*] [-p *speed*] [*image-file*]

cdrw -a [-vSCO] [-d *device*] [-p *speed*] [-T *audio-type*] *audio-file1* [*audio-file2...*]

cdrw -x [-v] [-d *device*] [-T *audio-type*] *track-number out-file*

cdrw -c [-vSC] [-d *device*] [-p *speed*] [-m *tmp-dir*] [-s *src-device*]

cdrw -b [-v] [-d *device*]all | session

cdrw -M [-v] [-d *device*]

cdrw -l [-v]

cdrw -h

機能説明 | cdrw コマンドを使用すると、データ CD および音楽 CD を作成することができます。また音楽 CD から音楽トラックを取り出すこともできます。MMC 準拠の CD-R/CD-RW ドライブであれば、どれでも cdrw で使用することができます。

-d オプションによりデバイスを指定しない場合は、cdrw はシステムに接続された CD ライターを探します。システム上に 1 つのライターデバイスを発見したときは、それをデフォルト CD ライターとして扱います。

2 つ以上の CD ライターがシステムに接続されている場合は、-d オプションによりデバイスを指定してください。CD ライターのデバイス名は、/dev/rdsk/cNtNdNsN、cNtNdNsN、cNtNdN のように指定されます。また、ボリュームマネージャで使用する cdrom や cdrom1 などのシンボル名でも指定できます。-l オプションを使用すると、CD ライターのリストを表示できます。

USB 外部記憶装置クラスに準拠した CD-RW をシステムに追加する方法については、scsa2usb(7D) を参照してください。

データ **CD** の作成 | データ CD を作成する場合、cdrw はトラック書き込みモード (track-at-once) を使用します。CD メディアに書き込むデータのファイルを指定するには、-i オプションを使用します。ファイルが指定されていない場合、cdrw は標準入力からデータを読み込みます。

どちらの場合でも、データはあらかじめ mkisofs(1M) コマンドを使用してファイルおよびファイル情報を CD で使用される High Sierra フォーマットに変換されます。コマンドの使用方法については「使用例」を参照してください。

音楽 **CD** の作成 | 音楽 CD を作成する場合、-a オプションを使用すると、1 つまたは複数の音楽ファイルを指定できます。すべての音楽ファイルはサポートされた音楽形式でなければなりません。現在対応している形式は以下のものです。

sun　　レッドブック CD-DA 形式の Sun .au ファイル

wav　　レッドブック CD-DA 形式の RIFF (.wav) ファイル

cda　生の CD 音楽データを含む .cda ファイル (リトルエンディアン 16 bit PCM ステレオ 44.1 KHz サンプリング)

aur　生の CD データを含む .aur ファイル (ビッグエンディアン)

音楽形式が指定されていない場合は、cdrw はファイルの拡張子により音楽形式を認識しようとします。拡張子の大文字小文字は無視されます。-T オプションにより音楽形式を指定した場合は、指定されたすべてのファイルの音楽形式として扱われます。また、-cdrw は音楽トラックを書き込んだ後セッションをクローズします。したがって、書き込まれるトラックは 1 つのコマンド行で指定する必要があります。

音楽の取り出し

cdrw に -x オプションを指定すると、音楽 CD から音楽データを取り出すこともできます。CD にはレッドブック CD-DA 形式のトラックが存在する必要があります。デフォルトでは、出力の形式はファイルの拡張子に応じたものになります。-T オプションを使用すると、出力形式を sun、wav、cda、aur のいずれかに指定できます。

CD のコピー

cdrw はシングルセッションのデータ CD-ROM やレッドブックの音楽 CD のコピーに使用できます。CD をコピーする場合、cdrw は指定されたソースデバイスを探します。-c オプション使用時にソースデバイスが指定されていない場合は、現在の書き込み用 CD デバイスがソースデバイスとみなされます。cdrw は一時ファイルに音楽トラックを取り出し、現在の書き込み用 CD デバイスにブランクの書き込み可能 CD-R/CD-RW メディアが入っているかどうかを調べます。メディアが見つからない場合は、現在の書き込み用 CD デバイスにブランクの書き込み可能 CD メディアを入れるようユーザーに要求します。デフォルトの一時ディレクトリに十分な領域がない場合は -m オプションで代わりのディレクトリを指定できます。

CD-RW メディアの消去

ユーザーは再書き込みする前に CD-RW メディアを消去する必要があります。-b オプションは以下の消去方法をサポートしています。

session　最後のセッションを消去する

all　メディア全体を消去する.

session 消去では cdrw は最後のセッションを消去します。セッションが 1 つしか書き込まれていない CD-RW (例: cdrw によって作成されたデータ/音楽 CD-RW) の場合は、記録された部分を消去するだけでブランクディスクにすることができます。これはメディア全体を消去するよりも速くできます。

all 消去は、マルチセッションの CD 、最後のセッションがクローズされていない場合、CD の状態が不明な場合、ユーザーが CD 全体を消去したい場合に使用してください。この場合、cdrw はディスク全体を消去します。

デバイスリストおよびメディアの状態のチェック

-l オプションによって現在システムに接続されている CD ライターの一覧が表示されます。また、メディアによっては -M オプションによって、消去の状態や内容リスト (TOC) が表示されます。また、-M オプションは、最後のセッションの開始アドレスや次の書き込み可能アドレスも表示しま す。この情報は、-O オプションを指定してマルチセッション CD を作成するときに一緒に使用されます。詳細は mkisofs(1M) を参照してください。

オプション　以下のオプションを指定できます。

`-a`　音楽 CD を作成します。少なくとも 1 つの音楽ファイル (*audio-file*) を指定する必要があります。最大 99 までの音楽トラックを指定することができます。また最長の音楽データは、`-c` オプションを指定しない場合、デフォルトで 74 分になります。

`-b`　CD-RW メディアを消去します。消去する方法として、`all` と `session` のどちらかを指定します。

`-c`　CD をコピーします。他に引き数の指定がない場合、`cdrw` はデフォルトの CD 書き込みデバイスをソースデバイスとみなします。その場合、コピー処理はソースメディアから一時ディ レクトリに読み出し、ユーザーにブランクメディアをドライブに入れるよう要求します。

`-C`　メディアの容量を指定します。このオプションがない場合、`cdrw` で書き込み可能な CD メディアの容量のデフォルト値は、音楽 CD では 74 分、データ CD では 681984000 (約 650M) バイトになります。

`-d`　CD 書き込みを行うデバイスを指定します。

`-h`　ヘルプ。使用法を表示します。

`-i`　データ CD を作成するイメージファイルを指定します。書き込めるファイルサイズは CD-R/CD-RW メディアのいずれの場合でもデフォルトの 681984000 バイト、または `-c` オプションによって指定された容量以下になります。CD 書き込み処理は、途切れない連続したデータの供給を必要とするため、イメージファイルは NFS マウントされたファイルシステムではなくローカルなファイルシステムに置くようにしてください。

`-l`　システム上のすべての CD ライターを表示します。

`-m`　CD をコピーする際にトラックデータを置く一時ディレクトリを (システムのデフォルトの一時ディレクトリの代替として) 指定します。代替の一時ディレクトリが必要になる理由は、CD 上のデータ量は膨大になる可能性がある (80 分の音楽 CD の場合で約 800M バイト) のに対して、システムのデフォルト一時ディレクトリに十分な容量がない場合があるためです。

`-M`　メディアの状態を表示します。`cdrw` はメディアがブランクかどうか、内容リスト (TOC)、最後のセッションの開始アドレス、および、ディスクがオープンな場合は次の書き込み可能アドレスを表示します。

`-O`　ディスクをオープンのままにします。`cdrw` は通常セッションをクローズしますが、マルチセッション CD を作成する場合は次のセッションを追加するために、オープンのままにしておきます。

`-p`　CD ライターの書き込み速度を設定します。たとえば、`-p 4` は 4 倍速で書き込みます。オプションの指定がない場合、`cdrw` は CD ライターのデフォルトの書き込み速度を使用します。オプションが指定された場合、`cdrw` はドライブに指定された速度での書き込みを設定しようとしますが、実際にドライブがその速度で書き込めるという保証はありません。

`-s`　CD をコピーするソースデバイスを指定します。

`-S` 疑似書き込みモードです。このコードでは、`cdrw` は CD ライターのレーザーをオフにして書き込みを行います。したがってメディアにはデータは書き込まれません。システムが CD ライターに指定された書き込み速度で、十分にデータを提供できるかどうか調べる場合に使用します。

`-T` 音楽 CD を作成するために読み出す音楽ファイルまたは取り出す音楽ファイルの形式を指定します。音楽形式 (*audio-type*) には、`sun`、`wav`、`cda`、`aur` のいずれかを指定できます。

`-v` 冗長モードです。

`-x` 音楽トラックから音楽データを取り出します。

使用例

例 1 データ CD の作成

```
example% cdrw -i /local/iso_image
```

例 2 ディレクトリからの CD の作成

以下は、`/home/foo` ディレクトリ以下を CD に書き込む例です。

```
example% mkisofs -r /home/foo 2>/dev/null | cdrw -i -p 1
```

例 3 トラック番号を使用した音楽データの取り出し

以下は、トラック番号 `1` の音楽データを `/home/foo/song1.wav` に取り出す例です。

```
example% cdrw -x -T wav 1 /home/foo/song1.wav
```

例 4 wav ファイルの使用

以下は、ディスク上の `wav` ファイルから音楽 CD を作成する例です。

```
example% cdrw -a song1.wav song2.wav song3.wav song4.wav
```

例 5 CD-RW メディアの消去

以下は、CD-RW ドライブ内の CD-RW メディアのデータを消去する例です。

```
example% cdrw -b all
```

例 6 複数のドライブでのデータ CD の作成

以下は、複数の CD-R/RW ドライブが接続されたシステムでデータ CD を作成する例です。

```
example% cdrw -d c1t6d0s2 -i /home/foo/iso-image
```

例 7 データ書き込み速度のチェック

以下は、システムが書き込み処理に対して十分な速度でデータを CD-RW ドライブに提供できるかどうかチェックする例です。

```
example% cdrw -S -i /home/foo/iso-image
```

例 8 高優先度での実行

以下は、cdrw を優先度を上げて実行する例です (スーパーユーザーのみ実行可能)。

```
example# priocntl -e -p 60 cdrw -i /home/foo/iso-image
```

例 9 マルチセッションディスクの作成

以下は、1 つめのセッションイメージを mkisofs(1M) を使用して作成し、その内容をオープンにしたままのディスクに記録する例です。

```
example% cdrw -O -i /home/foo/iso-image
```

オープンしたままのディスクには、mkisofs(1M) で作成したイメージと、cdrw が報告するセッション開始アドレスおよび次の書き込み可能なアドレスを使用して、ソフトウェアを追加することができます。

```
example% cdrw -M

Track No. |Type    |Start address
----------+--------+-------------
 1        |Data    | 0
Leadout   |Data    | 166564

Last session start address: 162140
Next writable address: 173464

example% mkisofs -o /tmp/image2 -r -C 0,173464 -M \
   /dev/rdsk/c0t2d0s2 /home/foo
```

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcdrw

関連項目

audioconvert(1), mkisofs(1M), priocntl(1), attributes(5), rbac(5), scsa2usb(7D), sd(7D)

注意事項

CD 書き込み中、システムは一定の転送速度でドライブにデータを供給し続ける必要があります。CD 作成中は I/O 処理を最小限にし、不要なアプリケーションは終了しておくことをお勧めします。

コピーの作成や音楽トラックの取り出しには MMC 準拠のソース CD-ROM ドライブの使用をお勧めします。CD ライターはこの目的のために使用できます。

CD に書き込む前に、`-M` オプションでメディアがブランクであることを 確認し、`-S` オプションの疑似モードを使用して、システムが要求された書き込み速度でデータを供給できるかどうかを調べてください。システムが指定された書き込み速度でデータを供給できない場合は、`-p` オプションを使用して書き込み速度を遅くしてください。`priocntl`(1) コマンドを使用して `cdrw` を高い優先度で実行することもできます。

`-p` オプションは CD-R/RW ドライブに詳しく、書き込み速度を変更して使用できるユーザーのために提供されています。一般に普及しているドライブでは、ドライブによって書き込み速度の設定コマンドの扱いが異なるため、注意してこのオプションを使用してください。

一般に普及しているドライブでは、 74 分を超えてメディアの容量の上限まで書き込むことができます (80 分メディアの場合など)。しかし、使用しているドライブでは 74 分以上の書き込みをサポートしていないかもしれません。使用しているドライブが 74 分以上書き込める場合は、`-C` オプションを使用してメディアの容量を指定するようにしてください。

`cdrw` コマンドは、`rbac`(5) を使用してデバイスに対するユーザーのアクセスを制御しています。デフォルトでは、`cdrw` にアクセスできるユーザーに制限はありませんが、特定のユーザーしかアクセスできないように設定することもできます。詳細については『Solaris のシステム管理 (基本編)』の CD-R および CD-RW デバイスの管理に関する説明を参照してください。

名前 cd, chdir, pushd, popd, dirs – 現在の作業用ディレクトリの変更

形式 **/usr/bin/cd** [*directory*]

sh **cd** [*argument*]

chdir [*argument*]

csh **cd** [*dir*]

chdir [*dir*]

pushd [+ | *n* | *dir*]

popd [+ *n*]

dirs [-l]

ksh **cd** [*arg*]

cd *old new*

/usr/bin/cd /usr/bin/cd ユーティリティは、cd ユーティリティ自身だけの現在のディレクトリを変更します。これは、後述するシェル組み込みの cd とは対照的です。/usr/bin/cd はプロセスの呼び出しには影響しませんが、あるディレクトリを現在のディレクトリとして設定できるかどうかを決定するのに使用できます。

sh Bourne シェルに組み込まれている cd は、現在のディレクトリを *argument* で指定されたディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *argument* になります。 シェル変数 CDPATH は、 *argument* を含むディレクトリの検索パスを定義します。代替ディレクトリ名は、 コロン (:) で区切ります。 デフォルトのパスは空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは空のパス名で指定します。 このパス名は、等号の直後か、 パスリスト内にある区切り文字のコロンの間に指定します。*argument* の先頭文字が / 、 . 、または . . の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *argument* を検索します。 cd は、 *argument* 中で実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は効率が悪くなります。 そのため、 cd コマンドは、シェルに組み込まれています。 (pwd(1) 、 sh(1) 、 chdir(2) を参照)

chdir は、cd を呼び出すもうひとつの方法です。

csh *dir* 引数を省略すると、 C シェルに組み込まれている cd は、シェル変数 HOME の値を新たな作業用ディレクトリとして 使用します。 *dir* を指定した場合、それが / 、 . 、 または . . で始まる完全なパス名であれば、その *dir* が新たな作業用ディレクトリとなります。 それ以外の場合は、シェル変数 CDPATH が指定するパスと相対関係を持つディレクトリの中から 該当するものを探し出します。 CDPATH の構文は PATH シェル変数と同一で、セマンティクスも似ています。 cd は *dir* に対する実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、C シェルに組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

chdir はシェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。 引数を指定しないと、 そのユーザーのホームディレクトリに変更します。 *dir* が現在のディレクトリからは見つからない相対パス名の場合、変数 cdpath 内のディレクトリリストを検索します。 *dir* が / で始まる値を持つシェル変数の名前である場合、 その変数の値が示すディレクトリに変更します。

pushd はディレクトリスタックにディレクトリをプッシュ (押し込む) します。 引数を指定しないと、スタックにある先頭の 2 つの構成要素を交換します。

+*n* *n* 番目のエントリがスタックの先頭になるよう回転し、 そのディレクトリに移ります。

dir 現在の作業用ディレクトリをスタックにプッシュし、 そのディレクトリに移ります。

popd はディレクトリスタックからポップして (取り出して)、新たに先頭となったディレクトリへ cd します。ディレクトリスタックの構成要素の先頭番号は、 0 となります。

+*n* スタック内の *n* 番目のエントリを破棄します。

dirs はディレクトリスタックを出力します。 現在のディレクトリが最も左に現れるように 時間順に出力されます。 -l 引数を指定すると、 ~ を使った省略形ではなく、完全な形式で出力されます。

ksh Korn シェルに組み込まれた cd コマンドは、上記 2 つの形式の いずれかで入力します。第 1 の形式は、現在のディレクトリを *arg* に変更します。 *arg* が – の場合、ディレクトリを直前のディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *arg* になります。PWD 変数は、現在のディレクトリに設定されます。シェル変数 CDPATH は、*arg* を含むディレクトリの検索パスを定義します。 ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは NULL のパス名で指定します。 このパス名は、 等号の直後か、パスリスト内にある区切り文字のコロンの間に指定します。 *arg* の先頭文字が / 、 . 、または . . の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *arg* を検索します。 cd の第 2 の形式は、 PWD 中の現在のディレクトリ名における *old* という文字列を *new* という文字列に置換し、 この新規のディレクトリへ変更しようとします。 cd コマンドは rksh では実行できません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、 ksh に組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

オペランド 以下のオペランドを指定できます。

directory 新たな作業用ディレクトリとなるディレクトリの絶対または相対パス名。 cd が相対パス名をどのように解釈するかは、環境変数 CDPATH の設定により異なります。

出力 CDPATH に設定されている空でないディレクトリ名が用いられる場合、 新たな作業用ディレクトリの絶対パス名が 以下のような形式で標準出力に出力されます。

"%s\n", <*new directory*>

それ以外の場合には、何も出力されません。

環境

cd の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

CDPATH　コロンで区切られた、ディレクトリを示すパス名のリスト。 *directory* オペランドの先頭文字がスラッシュ (/) でなく、 先頭部分が . でも .. でもない場合には、 cd はこのリスト内のパス名を順番に検索し、環境変数 CDPATH に指定されている名前のディレクトリから *directory* を探します。その結果、最初に見つかったディレクトリ名が 新たな作業用ディレクトリとなります。 ディレクトリのパス名として空の文字列を指定すると、 それは現在のディレクトリと見なされます。 CDPATH は、設定されていないときには空の文字列として扱われます。

HOME　*directory* オペランドが省略されたときに用いるホームディレクトリの名前

PWD　現在の作業用ディレクトリのパス名。 この変数は、そのディレクトリに移った後に cd により設定されます。

終了ステータス

以下の終了ステータスが返されます。

0　ディレクトリが正常に変更された。

>0　エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), pwd(1), sh(1), chdir(2), attributes(5), environ(5)

名前 | eqn, neqn, checkeq – 数学的記述のタイプセット

形式 | **eqn** [-d *xy*] [-f *n*] [-p *n*] [-s *n*] [*file*...]

neqn [*file*...]

checkeq [*file*...]

機能説明 | eqn および neqn は、数式を記述するのに便利な言語プロセッサです。eqn は troff(1) 用のプリプロセッサで、 troff の出力を印刷できる装置用に提供されています。一方、neqn は nroff(1) 用のプリプロセッサで、端末での出力用に提供されています。通常この 2 つのコマンドは、以下の形式で指定します。

```
example% eqn file . . . | troff
example% neqn file . . . | nroff
```

ファイル名を表す *file*引数を省略すると、eqn または neqn は標準入力から読み込みます。数式の開始を示すには、行の先頭に .EQ を記述します。同様に数式の終了は、行の先頭に .EN を記述して表します。この 2 つの行は変換されないので、センタリングや番号付けなどを行うマクロパッケージ中に定義しておくこともできます。また、一対の文字を「区切り記号」として設定し、区切り記号に囲まれたテキストを eqn 入力として処理させることもできます。

区切り記号や .EQ/.EN が存在しない、または対で指定されていない場合、checkeq はメッセージを出力します。

オプション | 以下のオプションが指定できます。

-d*xy* コマンド行引数で指定される数式の区切り記号として、文字 *x* と *y* を指定します。ただしこの方法よりも、.EQ と .EN の間で delim *xy* を使って区切り記号を指定する方法がより一般的です。*x* と *y* には同じ文字を指定することも可能です。テキスト中に delim off と記述すると、区切り記号は有効でなくなります。区切り記号にも .EQ と .EN にも囲まれていないテキストは、すべてそのまま渡されます。

-f*n* ドキュメント全体を通じて使用するフォントとして *n* を指定します。このグローバルフォントの設定は、ドキュメントの本文中に gfont *n* 命令を指定して変更することもできます。*n* はフォント指定です。

-p*n* 下付きおよび上付きの添字のサイズを、直前の文字サイズより *n* ポイントだけ小さくします。 -p オプションを省略すると、添字のサイズは 3 ポイント小さくなります。

-s*n* ドキュメント全体を通じて使用する文字サイズとして *n* を指定します。このグローバルサイズの設定は、ドキュメントの本文中に gsize *n* 命令を指定して変更することもできます。*n* はポイントサイズです。

オペランド | 以下のオペランドが指定できます。

file eqn または neqn によって処理される nroff のファイルまたは troff のファイル。

EQN 言語

この説明を nroff を使って端末画面に表示した場合、端末画面の制限から neqn による出力箇所は正確には表示できません。出力の正確な表示を確認するために、このページを印刷してご覧ください。

eqn 中のトークンは、中括弧、二重引用符、チルド、山型記号、空白文字、タブ、または復帰改行文字で区切られます。中括弧 { } は、グループ分けに用いられます。一般的には、たとえば *x* のような 1 つの文字が記述できる箇所であれば、中括弧で囲んだ複雑な記述を代わりに指定できます。チルド (~) は出力中における 1 文字分の空白を、山型記号 (^) は半文字分の空白を表します。

下付きおよび上付きの添字:
これらは、キーワード sub と sup を使って生成できます。

`x sub i` という記述の出力結果は x_i になります。

`a sub i sup 2` の出力は $a_i^{\ 2}$ になります。

`e sup {x sup 2 + y sup 2}` の出力は $e^{x^2+y^2}$ になります。

分数:
分数は、キーワード over で指定します。

```
a over b
```

の出力結果は、

$$\frac{a}{b}$$

となります。

平方根の式:
平方根の式は、キーワード sqrt で指定します。

```
1 over sqrt {ax sup 2 +bx+c}
```

の出力結果は、

$$\frac{1}{\sqrt{ax^2+bx+c}}$$

となります。

制限値:
キーワード from と to は、種々の指定における最小値と最大値を表します。

```
lim from {n→ inf } sum from 0 to n x sub i
```

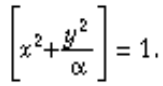
の出力結果は、

となります。

括弧:
大括弧、中括弧などを適切な高さで出力するには、左括弧に `left` を、右括弧には `right` をそれぞれ使用します。

```
left [ x sup 2 + y sup 2 over alpha right ] ~=~1
```

と記述すれば



という出力が得られます。なお、`right` 文節は省略することができます。キーワード `left` と `right` の直後に指定できる文字は、大括弧、中括弧、縦棒、上端と下端を表す `c` と `f`、何もない旨を示す `""` (対になるべき括弧のうち右括弧だけを使う場合に便利) です。

分数を縦に重ねる:
分数を縦に重ねるには、`pile`、`lpile`、`cpile`、または `rpile` を使用します。

```
pile {a above b above c}
```

という記述により

が出力されます。何重に積み重ねてもかまいません。文字を合わせる位置は、`lpile` は左詰め、`pile` と `cpile` はともにセンタリング (ただし縦方向の間隔が異なる)、そして `rpile` は右詰めとなります。

行列:
行列は `matrix`というキーワードで生成されます。

```
matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }
```

と記述すると、

が出力されます。カラムを右詰めにするには rcol を使用します。

文字の上下に付加する記号:
発音符記号のように文字の上下に付加する記号は、dot、dotdot、hat、tilde、bar、vec、dyad、underを使って指定できます。

```
x dot = f(t) bar
```

という記述の出力結果は、

```
y dotdot bar ~=~ n under
```

の結果は、

```
x vec ~=~ y dyad
```

の結果は、

文字のサイズとフォント:
文字のサイズやフォントの変更は、size *n* または size ±*n*、roman、italic、bold、font *n* で指定します。ドキュメント全体を通じてグローバルに使用する文字サイズとフォントは、gsize *n* と gfont *n* をドキュメント中に指定するか、またはコマンド行引数の -s*n* と -f*n* を使って変更できます。

表示引数の位置:
一連の表示引数の位置をそろえることもできます。先頭の数式において、そろえたい表示引数の直前に mark と記述します。さらに後続の数式において、それと合わせたい表示引数の直前に lineup と記述します。

短縮形:
入力の短縮形を定義したり既存のキーワードを再定義するには、define を使用します。次に例を示します。

`define` *thing* `%` *replacement* `%`
これにより *thing* というトークンが新たに定義され、その後このトークンが現れるたびに *replacement* に置き換えられます。なお `%` の位置には、任意の文字 (ただし *replacement* に含まれていないもの) を指定できます。

キーワードと短縮形:
`sum int inf` のようなキーワード、および `>=` → や `!=` のような短縮形も処理されます。

ギリシャ文字:
ギリシャ文字は `alpha` または `GAMMA` のように、大文字・小文字のうち希望する方のつづりで出力できます。

数学用語:
`sin`、`cos`、`log` のような数学用語は自動的にローマン字体で出力されます。

\(bu (•) のような 4 文字からなる `troff`(1) のエスケープコードは、どこでも記述できます。二重引用符に囲まれた文字列 `"..."` は、そのまま渡されます。これによりキーワードをテキストとして入力でき、また (他の方法が使えないとき) `troff` との通信用に使うことができます。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目

`nroff`(1), `tbl`(1), `troff`(1), `attributes`(5), `ms`(5)

使用上の留意点

数字や括弧をボールドで出力したい場合、`bold "12.3"` のように引用符で囲んでください。

名前 | checknr – nroff と troff 用の入力ファイルをチェックして誤りを報告

形式 | **checknr** [-fs] [-a . *x1* . *y1* . *x2* . *y2* . . . . *xn* . *ym*] [-c . *x1* . *x2* . *x3* . . . . *xn*] [*filename...*]

機能説明 | checknr は nroff(1) または troff(1) 用の入力ファイルの内容をチェックして、 起こりうるエラーを見つけ出します。 見つけられるエラーの種類は、 区切り記号が対になっていない、 未定義のコマンドが記述されている、などです。 ファイル名を指定しないと、 checknr は標準入力の内容をチェックします。 対になっているかのチェックは、 以下の区切り記号を対象として行われます。

- \f*x* . . . \fP で記述するフォント変更指定
- \s*x* . . . \s0 で記述するサイズ変更指定
- たとえば、 .TS と .TE のような、常に開始と終了とが対になって 指定されるべきマクロ

checknr は、 ms(5) と me(5) マクロパッケージに対してもチェックできます。

checknr コマンドは、作者が checknr でのチェックを想定して記述したドキュメント の検査に適しています。特に \f と \s の両コマンドに関しては、特定のスタイルで 記述されていなければなりません。 具体的には、個々の \f*x* 指定は必ず対応する \fP 指定で終了し、同様に \s*x* 指定は必ず \s0 指定で終了していなければなりません。 終了指定の代わりに 元のフォントまたはサイズ指定を明示的に記述しても、 指定としては有効ですし、実際に多くのドキュメントは そのように書かれていることでしょう。 しかしそのようなドキュメントを checknr でチェックすると、大量のメッセージが出力されることになります。 いずれにせよ、 \fP や \s0 を使った形式の方が優れていますので、 checknr に処理させるかどうかに関わらず この形式を使うようお勧めします。

オプション

-f　　\f によるフォント変更指定を無視します。

-s　　\s によるサイズ変更指定を無視します。

-a .*x1* . *y1*. . .　　マクロの対を、対になっているかを検査する対象のマクロ (たとえば DS と DE) のリストに追加します。 -a の後には、 6 文字一組でマクロの対を記述します。 その 6 文字の内訳は、先頭がピリオド、次の 2 文字が開始マクロ名、 次がピリオド、最後の 2 文字が終了マクロ名です。 たとえば .BS と .ES の対を定義したければ、 '-a.BS.ES' と指定します。

-c . *x1* . . .　　たとえ未定義であっても無視すべき コマンドを指定します。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目 | eqn(1), nroff(1), troff(1), attributes(5), me(5), ms(5)

使用上の留意点 | 1 文字からなるマクロ名は、 -a オプションを使って指定することができません。

chgrp(1)

名前 | chgrp – ファイルのグループ所有権の変更

形式 | **chgrp** [-fhR] *group file...*

機能説明 | chgrp ユーティリティは、 *file* で指定された各ファイルのグループ ID を、 *group* で示す値に設定します。

file の個々のファイルに対し、 chgrp は、 chown(2) 関数を以下の引数とともに呼び出した場合と同じ動作を行います。

- *file* の値を *path* 引数に指定
- そのファイルのユーザー ID を *owner* 引数に指定
- 指定されたグループ ID を *group* 引数に指定

適切な特権を持ったプロセスが chgrp を呼び出さない限り、正常終了時に通常ファイルのセットユーザー ID ビットとセットグループ ID はクリアされます。 他の種類のファイルのセットユーザー ID ビットとセットグループ ID もクリアされることがあります。

オペレーティングシステムは、所有者変更を制限するコンフィギュレーションオプション {_POSIX_CHOWN_RESTRICTED} を持っています。このオプションが有効になっていると、 ファイルの所有者は自分自身が属するグループへしか ファイルのグループを変更できません。 このオプションが有効かどうかに関わらず、 スーパーユーザーだけが所有者 ID を変更できます。 コンフィギュレーションオプションを 設定する場合は、 /etc/system ファイルに次の行を挿入してください。

```
set rstchown = 1
```

このオプションを無効にする場合は、 /etc/system ファイルに次の行を挿入してください。

```
set rstchown = 0
```

デフォルトでは {_POSIX_CHOWN_RESTRICTED} は有効です。 system(4) と fpathconf(2) を参照してください。

オプション

-f	強制実行。エラーは報告しません。
-h	ファイルがシンボリックリンクであるとき、 そのシンボリックリンクのグループを変更します。 このオプションが指定されていない場合は、 そのシンボリックリンクによって参照される ファイルのグループが変更されます。
-R	再帰。 chgrp はディレクトリおよびすべてのサブディレクトリを検索し、指定されたグループ ID を設定していきます。 シンボリックリンクに出合うと、 (-h オプションが指定されていなければ) 対象となるファイルのグループが変更されます が、再帰は発生しません。

オペランド | 以下のオペランドを指定できます。

group グループデータベースから得られるグループ名、 または数値のグループ ID 。どちらの場合でも、 `file` で指定した各ファイルに与えるグループ ID を表します。 *group* が数値で、それがグループ名として グループデータベースに存在していると、 そのグループ名に対応したグループ ID 番号がグループ ID として用いられます。

file グループ ID を変更するファイルのパス名

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `chgrp` の動作については、 `largefile`(5) を参照してください。

環境 `chgrp` の実行に影響を与える環境変数 `LC_CTYPE`、 `LC_MESSAGES`、 `NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス 以下の終了ステータスが返されます。

`0` ユーティリティの実行が正常終了し、要求されたすべての変更が行われた。

`>0` エラーが発生した。

ファイル `/etc/group` グループファイル

属性 次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み (「注意事項」参照)

関連項目 `chmod`(1), `chown`(1), `id`(1M), `chown`(2), `fpathconf`(2), `group`(4), `passwd`(4), `system`(4), `attributes`(5), `environ`(5), `largefile`(5)

注意事項 `chgrp` は *group* 名を除いて CSI に対応しています。

名前 | chkey – ユーザーの Secure RPC 鍵ペアの変更

形式 | **chkey** [-p] [-s nisplus | nis | files | ldap] [-m <*mechanism*>]

機能説明 | chkey は、ユーザーの Secure RPC 公開鍵と秘密鍵のペアを変更するために使用されます。chkey は、古い Secure RPC パスワードを要求するプロンプトを出力し、秘密鍵を解読することによりそのパスワードが正しいことを検証します。ユーザーが、keylogin(1) を使用して秘密鍵を解読し、keyserv(1M) を使用して秘密鍵を保存する処理をまだ行なっていない場合、chkey は、ローカルの keyserv(1M) デーモンを使用して秘密鍵を登録します。Secure RPC パスワードがログインパスワードと一致しない場合、chkey はログインパスワードの入力を求めるプロンプトを出力します。chkey は、ログインパスワードを使用して、ユーザーの Diffie-Hellman (192 ビット) 秘密鍵を暗号化します。chkey は、nisauthconf(1M) を使用して構成された認証機構用に他の Diffie-Hellman 鍵を暗号化することもできます。

chkey は、ログインパスワードと Secure RPC パスワードが同一であることを保証し、パスワードのシャドウ化を可能にします。shadow(4) を参照してください。

鍵ペアは、/etc/publickey ファイル (publickey(4) を参照)、NIS の publickey マップ、または NIS+ の cred.org_dir テーブルに格納できます。新たに生成された秘密鍵は、ローカルの keyserv(1M) デーモンによって登録されます。ただし、192 ビット以外の Diffie-Hellman 鍵を格納できるのは、NIS+ だけです。

特別な機構のための鍵は、-m オプションに認証機構名を付けて指定することにより、変更または再暗号化することができます。複数の -m オプションを使用すると、1 つまたは複数の鍵を変更できます。ただし、chkey では nisauthconf(1M) を使用して構成された機構以外は、変更できません。

publickey のソースが -s オプションで指定されていない場合、chkey は、ネームサービススイッチ構成ファイルの publickey エントリを参照します。nsswitch.conf(4) を参照してください。publickey エントリがソースを 1 つだけ指定する場合、chkey は、指定されたネームサービスの鍵を変更します。ただし、複数のネームサービスがリストされている場合、chkey は更新するソースを決定できず、エラーメッセージを表示します。ユーザーは、-s オプションにを使用してソースを明示的に指定する必要があります。

root 以外のユーザーは、files データベース内の各自の鍵ペアを変更できません

オプション | 次のオプションを指定できます。

-p	ユーザーのログインパスワードを使用して既存の秘密鍵を再暗号化します。
-s nisplus	NIS+ データベースを更新します。
-s nis	NIS データベースを更新します。
-s files	files データベースを更新します。
-s ldap	LDAP データベースを更新します。
-m <*mechanism*>	指定した機構の秘密鍵を変更または再暗号化します。

ファイル

/etc/nsswitch.conf

/etc/publickey

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

keylogin(1), keylogout(1), keyserv(1M), newkey(1M), nisaddcred(1M), nisauthconf(1M), nsswitch.conf(4), publickey(4), shadow(4), attributes(5)

NIS+ は、Solaris™ オペレーティング環境の将来のリリースではサポートされなくなる可能性があります。Solaris 9 オペレーティング環境には、NIS+ から LDAP への移行を支援するツールが含まれています。詳細については、http://www.sun.com/directory/nisplus/transition.html を参照してください。

名前 | chmod – ファイルのアクセス権モードの変更

形式 | **chmod** [-fR] <*absolute-mode*> *file...*

chmod [-fR] <*symbolic-mode-list*> *file...*

機能説明 | chmod ユーティリティはファイルモードの変更や割り当てを行います。 ファイルモードは、アクセス権、およびその他の属性を指定するものです。 モードには、絶対モード (absolute mode) とシンボリックモード (symbolic mode) があります。

絶対モード | 絶対モードは次の形式で指定します。

chmod [*options*] *absolute-mode file* . . .ここで、*absolute-mode* は、以下のように定義されている 8 進数 *nnnn* を使用して指定します。

n 0 から 7 の数字を表します。 絶対モードは、以下のモードのいくつかの論理和をとったものです。

4000	4000 実行時にユーザー ID を設定します。
20#0	# が 7 、 5 、 3 または 1 のとき、 実行時にグループ ID を設定します。 # が 6 、 4 、 2 または 0 のとき、 強制ロックを許可します。 ディレクトリに対して使用されると、グループ ID の継承については BSD のセマンティクスに基づいてファイルが作成されます。 このオプションを使用すると、そのディレクトリ中に作成される ファイルやサブディレクトリは、カレントプロセスのグループ ID ではなく、ディレクトリのグループ ID を引き継ぎます。ディレクトリの場合、set-gid ビットをセットまたはクリアできるのはシンボリックモードを使用する場合だけです。
1000	スティッキビットをオンにします chmod(2) 参照。
0400	所有者による読み取りを許可します。
0200	所有者による書き込みを許可します。
0100	所有者による実行 (ディレクトリ内での検索) を許可します。
0700	所有者による読み取り、書き込み、実行 (検索) を許可します。
0040	グループによる読み取りを許可します。
0020	グループによる書き込みを許可します。

0010	グループによる実行 (ディレクトリ内での検索) を許可します。
0070	グループによる読み取り、書き込み、実行 (検索) を許可します。
0004	その他のユーザーによる読み取りを許可します。
0002	その他のユーザーによる書き込みを許可します。
0001	その他のユーザーによる実行 (ディレクトリ内での検索) を許可します。
0007	その他のユーザーによる読み取り、書き込み、実行 (検索) を許可します。

ディレクトリに対して、絶対モードでは setgid のセットやクリアは行えません。 シンボリックモードで g+s (または g-s) を使って行う必要があります。

シンボリックモード

シンボリックモードは次の書式で指定します。

chmod [*options*] *symbolic-mode-list file . . .*

symbolic-mode-list は、いくつかのシンボリックモードの式をコンマで区切った (空白をはさまない) ものです。 各々の式は、次の形式で記述します。

[*who*] *operator* [*permissions*]

処理は、記述された順序で実行されます。 1 つの演算子 (*operator*) の後に *permissions* として複数のアクセス権文字を指定すると、 その演算子で示された処理が同時に行われます。

who

u、 g、 o、 a の文字の 1 つ以上の組み合わせ (または省略) で、誰のアクセス権が変更または割り当てられるのかを指定します。

u	ユーザーのアクセス権
g	グループのアクセス権
o	その他のアクセス権
a	全アクセス権 (ユーザー、グループ、その他)

who が省略されると、デフォルトとして a と解釈されますが、ファイルモード作成マスク (詳細は sh(1) または csh(1) の umask の項を参照) の設定値は考慮されます。 *who* が省略されると、chmod はユーザーマスクの制限を置き換えません。

operator

アクセス権をどのように変更するかを指定する演算子で、 +、 −、 = のいずれかです。

	`+`	アクセス権を付加します。*permissions* を省略すると、何も付加されません。*who* が省略されると、対応するビットがファイルモード生成マスク中に セットされている場合を除き、*permissions* が示すファイルモードビットが付加されます。*who* が指定されていれば、*permissions* が示すファイルモードビットが付加されます
	`-`	アクセス権を除去します。*permissions* を省略すると、何も行われません。*who* が省略されると、対応するビットがファイルモード生成マスク中に セットされている場合を除き、*permissions* が示すファイルモードビットをクリアします。*who* が指定されていれば、*permissions* が示すファイルモードビットをクリアします。
	`=`	指定されたアクセス権をそのまま割り当てます。*who* が省略されると、すべてのファイルモードビットをクリアします。 *who* が指定されていれば、 *who* が示すファイルモードビットをクリアします。 *permissions* を省略すると、他には何も行われません。*who* が省略されると、対応するビットがファイルモード生成マスク中に セットされている場合を除き、*permissions* が示すファイルモードビットが付加されます。*who* が指定されていれば、*permissions* が示すファイルモードビットが付加されます。 `=` は、他のシンボリック演算子と違って、 *who* が示す他のすべてのビットをリセットしてしまう という絶対的な効果があります。 *permission* の省略は、 `=` を使用してすべてのアクセス権を除去するときのみ便利です。
permission	以下の文字の適切な組み合わせで指定します。	
	`l`	強制ロッキング
	`r`	読み取り権
	`s`	ユーザーまたはセットグループ ID
	`t`	スティッキビット
	`w`	書き込み権
	`x`	実行権
	`X`	実行権 (ファイルがディレクトリの場合、または他のユーザークラスのいずれかに実行権がある場合)

u,g,o	*permission* がそれぞれ現在のユーザー、グループ、 またはその他のモードから除去されることを意味します。

ファイルへのアクセス権は、ユーザー ID 番号 (UID) およびグループ ID 番号 (GID) によって異なります。アクセス権は、 3 つのユーザーモードがそれぞれ 3 つの文字を持つように記述されます。

```
ユーザー      グループ      その他
rwx  rwx  rwx
```

この例 (ユーザー、グループ、その他が、当該ファイルの読み取り、書き込み、 実行権を持っている) は、アクセス権を与える際の 2 つのカテゴリを示しています。 1 つはアクセスクラス、もう 1 つはアクセス権そのものです。 文字 `s` は、 `u` または `g` と一緒に指定したときのみ有効です。 また、 `t` は `u` とのみ指定できます。

強制的なファイルおよびレコードのロック (`l`) は、プログラムがアクセスしている間、そのファイルの読み込み、 書き込み権がロックされることを意味します。

セットグループ ID ビットがオンになっているディレクトリは、 `'ls -ld'` で `-----s---` または `-----l---` が出力されることで判定できますが、このようなディレクトリ中で ファイルやサブディレクトリを生成する際には、 その親ディレクトリのセットグループ ID が与えられます。カレントプロセスの セットグループ ID ではありません。

グループ実行権を与えて、 同時に実行時にファイルをロックすることはできません。さらに、セットグループ ID ビットをオンにして同時に実行時にファイルをロックすることはできません。 したがって、以下の例は無効であり、エラーメッセージが出力されます。

```
chmod g+x,+l file
chmod g+s,+l file
```

ファイルまたはディレクトリの所有者 (またはスーパーユーザー) だけが そのファイルまたはディレクトリのモードを変更できます。 スーパーユーザーだけがディレクトリでないファイルの スティッキビットを設定できます。 スーパーユーザーでなければ、 `chmod` はスティッキビットをマスクしますが、エラーは返しません。 ファイルのセットグループ ID ビットをオンにするには、自分自身のグループ ID がファイルのそれと一致していて、 さらにグループ実行権が設定されていなければなりません。

オプション　以下のオプションを指定できます。

`-f`	強制実行。 `chmod` はファイルモードの変更に失敗しても何もしません。
`-R`	再帰的にディレクトリの階層をたどって、 上述のように各ファイルのモードを変更していきます。 シンボリックリンクに出合うと、対象のファイルのモードが変更されます。 しかし、再帰は発生しません。

オペランド

以下のオペランドを指定できます。

absolute-mode
symbolic-mode-list *file* オペランドのいずれかで指定された各ファイルのファイルモードビットに加えられる変更を示します。詳細については、前途の「絶対モード」と「シンボリックモード」の項を参照してください。

file ファイルモードビットを変更するファイルのパス名

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の chmod の動作については、largefile(5) を参照してください。

使用例

例 1 すべてのユーザーに対して実行を許可しない

```
example% chmod a-x file
```

例 2 すべてのユーザーに対して読み取り権だけを与える

```
example% chmod 444 file
```

例 3 グループおよびその他のユーザーに対して読み取り、書き込みを許可する

```
example% chmod go+rw file
example% chmod 066 file
```

例 4 アクセス中はファイルをロックする

```
example% chmod +l file
```

例 5 すべてのユーザーに対して読み取り、書き込み、実行を許可し、セットグループ ID をオンにする

```
example% chmod =rwx,g+s file
example% chmod 2777 file
```

環境

chmod の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 正常終了

>0 エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

CSI	対応済み

関連項目 getfacl(1), ls(1), setfacl(1), chmod(2), attributes(5), environ(5), largefile(5)

注意事項 ディレクトリのセットグループ ID ビットをそっくり置き換えることはできません。g+s または g-s を使用してください。

chmod は規定に反しないかぎり、無意味なモード作成でも 許可します (たとえば、テキストファイルを実行可能にする) 。 chmod は強制ロックが意味をなすか否かを見るために、 ファイルのタイプをチェックすることはありません。

ファイルシステムが *nosuid* オプションでマウントされている場合、 *setuid* の実行は許可されません。

chmod を使用して ACL エントリのファイルのアクセス権を変更すると、ファイルのグループ所有権と ACL マスクの両方が、新しいアクセス権に変わります。新しい ACL マスクのアクセス権は、ファイル上の ACL エントリを所有する追加ユーザーや追加グループの実効アクセス権を変更する場合があることに注意してください。getfacl(1) コマンドを使用して、すべての ACL エントリに対する適切なアクセス権を確認してください。

名前 | chown – ファイルの所有者の変更

形式 | **chown** [-fhR] *owner* [: *group*] *file...*

機能説明 | chown ユーティリティは、 *file* オペランドが示す各ファイルのユーザー ID を、 *owner* オペランドが示す値に変更し、さらに *group* が指定されていれば、グループ ID もその値に変更します。

chown がスーパーユーザー以外のユーザーによって起動された 場合、セットユーザー ID ビットはクリアされます。

ファイルの所有者 (またはスーパーユーザー) だけが、ファイルの所有者を変更できます。

オペレーティングシステムは所有者の変更を制限するコンフィギュレーションオプション {_POSIX_CHOWN_RESTRICTED} を持っています。このオプションが有効なときは、ファイルの所有者は そのファイルの所有者 ID を変更することができません。このオプションに関係なく、 スーパーユーザーだけが所有者 ID を変更できます。 コンフィギュレーションオプションを 設定する場合は、 /etc/system ファイルに次の行を挿入してください。 コンフィギュレーションオプションを 設定する場合は、 /etc/system ファイルに次の行を挿入してください。

```
set rstchown = 1
```

このオプションを無効にする場合は、 /etc/system ファイルに次の行を挿入してください。

```
set rstchown = 0
```

デフォルトでは {_POSIX_CHOWN_RESTRICTED} は有効です。 system(4) と fpathconf(2) を参照してください。

オプション | 以下のオプションを指定できます。

-f	エラーを報告しません。
-h	ファイルがシンボリックリンクであるときそのシンボリックリンクの所有者を変更します。 このオプションが指定されていない場合は、 そのシンボリックリンクによって参照される ファイルの所有者が変更されます。
-R	再帰。 chown はディレクトリおよびすべてのサブディレクトリを検索し、指定された所有権 ID を設定していきます。 シンボリックリンクに出合うと、 (-h オプションが指定されていなければ) 対象となるファイルの所有者が変更されます。 しかし、再帰は行われません。

オペランド | 以下のオペランドを指定できます。

owner[: *group*]	*file* が示すファイルに割り当てるユーザー ID とグループ ID を指定します。グループ ID は省略可能です。 *owner* 部分は、ユーザー

データベースから得られるユーザー名、 または数値のユーザー ID のどちらかでなければなりません。 どちらの場合でも、 *file* オペランドで指定した各ファイルに与えるユーザー ID を表します。 *owner* が数値で、それがユーザー名としてユーザーデータベースに存在していると、そのユーザー名に対応したユーザー ID 番号がユーザー ID として用いられます。 *group* も同様で、指定するのであれば、 グループデータベースから得られるグループ名、 または数値のグループ ID でなければなりません。どちらの場合でも、 *file* オペランドで指定した各ファイルに与えるグループ ID を表します。 *group* オペランドが数値で、それがグループ名として グループデータベースに存在していると、 そのグループ名に対応したグループ ID 番号がグループ ID として用いられます。

file ユーザー ID を変更するファイルのパス名。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の chown の動作については、largefile(5) を参照してください。

環境

chown の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 ユーティリティの実行が正常終了し、要求されたすべての変更が行われた。

>0 エラーが発生した。

ファイル

/etc/passwd システムパスワードファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み (「注意事項」参照)

関連項目

chgrp(1), chmod(1), chown(2), fpathconf(2), passwd(4), system(4), attributes(5), environ(5), largefile(5)

注意事項

chown は *owner* 名と *group* 名を除いて CSI に対応しています。

名前 | ckdate, errdate, helpdate, valdate – 日付の入力要求とその検証

形式 | **ckdate** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errdate [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helpdate [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valdate [-f *format*] *input*

機能説明 | ckdate ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに日付の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザーの応答は、date コマンドで定義されている書式に一致したものでなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckdate コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errdate (エラーメッセージを書式化して表示する)、helpdate (ヘルプメッセージを書式化して表示する)、および valdate (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errdate および helpdate の各モジュールに format が定義されている場合、メッセージには、指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は、どのような書式も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。指定できる書式とその定義は、次のとおりです。

- %b = 省略された月の名前 (jan、feb、mar)
- %B = 完全な月の名前 %d = 日 (01 ～ 31)
- %D = %m/%d/%y の形式の日付 (デフォルトの書式)
- %e = 日 (1 ～ 31、1 桁の数字の前にはブランクが挿入されます)

`%h` = 省略形の月の名前。`%b`%と同じ

`%m` = 月 (01 ～ 12)

`%y` = 西暦中の年 (たとえば、89)

`%Y` = CCYY の形式の年 (たとえば、1989)

`-h` *help* `help` をヘルプメッセージとして定義します。

`-k` *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

`-p` *prompt* *prompt* をプロンプトメッセージとして定義します。

`-Q` 有効な応答として終了 (quit) を使用できないように指定します。

`-s` *signal* 終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル `signal` を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-W` *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、`width` の行長に書式化します。

オペランド

次のオペランドを指定できます。

input 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0` 正常な終了

`1` 入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`3` ユーザー終了 (quit)

`4` 不正な書式引数

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckdate` のデフォルトのプロンプトは、次のとおりです。

```
Enter the date [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter a date.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter a date. Format is <format>.
```

終了 (quit) オプションを選択した場合 (かつ使用できる場合)、リターンコード 3 と共に q が返されます。valdate モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前 | ckgid, errgid, helpgid, valgid – グループ ID の入力要求とその検証

形式 | **ckgid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errgid [-W *width*] [-e *error*]

/usr/sadm/bin/helpgid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valgid *input*

機能説明 | ckgid は、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに既存のグループ ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckgid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errgid (エラーメッセージを書式化して表示する)、helpgid (ヘルプメッセージを書式化して表示する)、および valgid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような書式基準も満たす必要はありません。

-e *error* — *error* エラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するように指定します。

-m — ユーザーがヘルプを要求した場合、あるいはユーザーの入力が不正な場合は、グループ名の一覧を表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/group` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckgid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the name of an existing group [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

デフォルトのヘルプメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valgid` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckint, errint, helpint, valint – 整数値の入力要求とその検証

形式 | **ckint** [-Q] [-W *width*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errint [-W *width*] [-b *base*] [-e *error*]

/usr/sadm/bin/helpint [-W *width*] [-b *base*] [-h *help*]

/usr/sadm/bin/valint [-b *base*] *input*

機能説明 | ckint ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白（復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckint コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errint (エラーメッセージを書式化して表示する)、helpint (ヘルプメッセージを書式化して表示する)、および valint (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errint と helpint の各モジュールに *base* が定義されている場合、メッセージには入力する整数の基数が含まれます。

オプション | 次のオプションを指定できます。

-b *base* — 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。

-d *default* — *default* をフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

`-s` *signal*　終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-W` *width*　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　基数 *base* の基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　正常な終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckint` のデフォルトのプロンプトは、次のとおりです (基数は 10)。

```
Enter an integer [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
ERROR - Please enter an integer.
```

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
Please enter an integer.
```

基数を 10 以外の数に設定した場合は、上記のメッセージは "integer" から "base *base* integer" に変更されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valint` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckitem, erritem, helpitem – メニューの作成、およびメニュー項目の入力要求とその検証

形式 | **ckitem** [-Q] [-W *width*] [-uno] [-f *filename*] [-l *label*] [[-i *invis*] [,...]] [-m *max*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]] [*choice* [...]]

/usr/sadm/bin/erritem [-W *width*] [-e *error*] [*choice* [..]]

/usr/sadm/bin/helpitem [-W *width*] [-h *help*] [*choice* [..]]

機能説明 | ckitem ユーティリティは、メニューを作成し、ユーザーにプロンプトを出力してメニュー項目から 1 つの項目を選択するように促します。さらに、ユーザーの応答入力を検証します。このコマンドのオプションでは、プロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) を定義します。

デフォルトでは、メニューは書式化されており、各項目の前に番号が付けられ、端末上に複数の列で出力されます。列の長さは、選択した最大長の項目によって決まります。項目はアルファベット順に表示されます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckitem コマンドには、2 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erritem (エラーメッセージを書式化して表示する) と helpitem (ヘルプメッセージを書式化して表示する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。これらのモジュールに *choice* が定義されている場合、メッセージには利用可能なメニューの選択項目が含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *filename* — *filename* を表示されるメニュー項目のリストを含むファイルとして定義します。(このファイルの書式は、token<tab>description です。ポンド記号 (#) で始まる行は注釈として指定され、無視されます。)

-h *help* — *help* をヘルプメッセージとして定義します。

-i *invis* — 非表示メニュー項目 (メニューに表示されない選択項目) を定義します。(たとえば、非表示項目として使用される "all" は、有効なオ

プションですが、メニューには表示されません。任意の数の非表示項目を定義できます。) 非表示項目があることを、プロンプトメッセージまたはヘルプメッセージのどちらかで、ユーザーに知らせるようにします。

`-k` *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

`-l` *label* メニューの上に表示するラベル、*label* を定義します。

`-m` *max* ユーザーが選択できるメニュー項目の最大数を定義します。デフォルト値は 1 です。

`-n` メニュー項目をアルファベット順で表示しないようにします。

`-o` 1 つのメニュートークンだけが返されるようにします。

`-p` *prompt* *prompt* をプロンプトメッセージとして定義します。

`-Q` 有効な応答として終了 (quit) を使用できないようにします。

`-s` *signal* 終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-u` メニュー項目に番号を付けずに表示するようにします。

`-W` *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

choice メニュー項目を定義します。項目は空白か復帰改行で区切ります。

終了ステータス

次の終了ステータスが返されます。

`0` 正常な終了

`1` 入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、`-f` オプションでファイルが開けない、あるいは使用法に誤りがあった

`3` ユーザー終了 (quit)

`4` 選択すべき項目がない

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

ユーザーは、メニュー項目に番号が付いている場合はその項目の番号を、あるいは、その項目を一意に識別するのに必要な長さの文字列を入力できます。長いメニューはページに分割され、各ページには 10 個の項目が表示されます。

メニュー項目が、`-f` オプションで指定したファイルとコマンド行の両方に定義されている場合、メニュー項目は通常、アルファベット順に表示されます。ただし、アルファベット順での表示を抑制する `-n` オプションが使用されている場合は、ファイルに定義された項目が最初に表示され、次にコマンド行に定義されたオプションが表示されます。

`ckitem` のデフォルトのプロンプトは次のとおりです。

```
Enter selection [?,??,q]:
```

1 つの疑問符はヘルプメッセージを表示してから、プロンプトを再表示します。2 つの疑問符は、ヘルプメッセージを表示してから、メニューラベル、メニュー、およびプロンプトを再表示します。

番号を入力した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Bad numeric choice specification
```

文字列を入力した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Entry does not match available menu selection.
Enter the number of the menu item you wish to select,
the token which is associated with the menu item,
or a partial string which uniquely identifies the token
for the menu item. Enter ?? to reprint the menu.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Enter the number of the menu item you wish to select,
the token which is associated with the menu item,
or a partial string which uniquely identifies the token for
the menu item. Enter ? to reprint the menu.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。

ckkeywd(1)

名前 | ckkeywd – キーワードの入力要求とその検証

形式 | **ckkeywd** [-Q] [-W *width*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]] *keyword* [...]

機能説明 | ckkeywd は、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーにキーワードリストのいずれかの入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。このコマンドから返される応答は、定義済みキーワードリストのいずれかと一致しなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

-W *width* — プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド | 次のオペランドを指定できます。

keyword — 応答を照合するときの基準となるキーワード、またはキーワードのリストを定義します。

終了ステータス | 次の終了ステータスが返されます。

0 — 正常終了

1　入力で EOFが検出された、-W オプションで負の行長が指定された、選択すべきキーワードがない、あるいは、使用法に誤りがあった

3　ユーザー終了 (quit)

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

attributes(5)

注意事項

ckkeywd のデフォルトのプロンプトは、次のとおりです。

Enter appropriate value [*keyword*,[. . .],?,q]:

デフォルトのエラーメッセージは、次のとおりです。

ERROR: Please enter one of the following keywords: *keyword*,[. . .],q

デフォルトのヘルプメッセージは、次のとおりです。

keyword,[. . .],q

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード 3 と共に q が返されます。

名前 | ckpath, errpath, helppath, valpath – パス名の入力要求とその検証

形式 |

ckpath [-Q] [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errpath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-e *error*]

/usr/sadm/bin/helppath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-h *help*]

/usr/sadm/bin/valpath [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] *input*

機能説明 |

ckpath ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーにパス名の入力を促すプロンプトメッセージ、ヘルプメッセージおよびエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

パス名は、最初のオプショングループで指定した基準に従わなければなりません。基準が定義されていない場合、パス名はまだ存在していない通常のファイルに対するものでなければなりません。-a (絶対) と -l (相対) のいずれも指定されていない場合は、どちらかが有効であるとみなされます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckpath コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errpath (標準出力上にエラーメッセージを書式化して表示する)、helppath (標準出力上にヘルプメッセージを書式化して表示する)、および valpath (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-a	パス名は絶対パスでなければなりません。
-b	パス名はブロック型特殊ファイルでなければなりません。
-c	パス名は文字型特殊ファイルでなければなりません。
-d *default*	*default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

`-e` *error*	*error* をエラーメッセージとして定義します。
`-f`	パス名は通常ファイルでなければなりません。
`-h` *help*	*help* をヘルプメッセージとして定義します。
`-k` *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
`-l`	パス名は相対パスでなければなりません。
`-n`	パス名が存在していてはなりません (新規のものでなければなりません)。
`-o`	パス名が存在していなければなりません (既存のものでなければなりません)。
`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (`quit`) を使用できないようにします。
`-r`	パス名は読み取り可能でなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-t`	パス名は作成可能 (処理可能) でなければなりません。パス名が、まだ存在していない場合には作成されます。
`-w`	パス名は書き込み可能でなければなりません。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。
`-x`	パス名は実行可能でなければなりません。
`-y`	パス名はディレクトリでなければなりません。
`-z`	パス名には、`0` バイトを超えるサイズのファイルがなければなりません。

オペランド

次のオペランドを指定できます。

input	検証オプションと照合される入力

使用例

`ckpath` のデフォルトメッセージのテキストは、使用されている基準オプションによって異なります。

例 1 デフォルトのプロンプト

`ckpath` (`-a` オプションを使用) のデフォルトプロンプトの例は、次のとおりです。

```
example% ckpath -a
Enter an absolute pathname [?,q]
```

例 **1** デフォルトのプロンプト　(続き)

例 **2** デフォルトのエラーメッセージ

デフォルトのエラーメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/errpath -a
ERROR: A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 **3** デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/helppath -a
A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 **4** 終了オプション

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード 3 と共に q が返されます。終了を入力すると、後に復帰改行が続きます。

例 **5** valpath モジュールの使用

valpath モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valpath
usage: valpath [-[a|l][b|c|f|y][n|[o|z]]rtwx] input
    .
    .
    .
```

終了ステータス

次の終了ステータスが返されます。

0	正常終了
1	入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった
2	同時に指定できないオプション
3	ユーザー終了 (quit)
4	同時に指定できないオプション

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 face(1), signal(3HEAD), attributes(5)

名前 | ckrange, errange, helprange, valrange – 整数の入力要求とその検証

形式 | **ckrange** [-Q] [-W *width*] [-l *lower*] [-u *upper*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errange [-W *width*] [-e *error*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/helprange [-W *width*] [-h *help*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/valrange [-l *lower*] [-u *upper*] [-b *base*] *input*

機能説明 | ckrange ユーティリティは、ユーザーに指定範囲内の整数の入力を要求して、ユーザーの応答が有効かどうかを検証します。このユーティリティでは、ユーザーに指定範囲内の整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

また、このコマンドは、有効な入力範囲も定義します。下限または上限のいずれかが定義されていない場合、範囲は一方の限界だけに制限されます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckrange コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errrange (標準出力上にエラーメッセージを書式化して表示する)、helprange (標準出力上にヘルプメッセージを書式化して表示する)、および valrange (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

注：*input* 引数として負の値を指定すると、valrange 内の getopt で混乱が生じます。引数の前に - を付けると、getopt は処理を停止します。getopt のパラメータ処理については、getopt(1) および intro(1) のマニュアルページを参照してください。getopt は、位置指定パラメータを解析して、有効なオプションかどうかを検査するために使用されます。

オプション | 次のオプションを指定できます。

-b *base* | 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。基数を変換するには strtol(3C) を使用します。出力は常に、基数を 10 として実行されます。

-d *default*	*default* をデフォルト値として定義します。*default* は strtol(3C) を使用して指定した基数に変換できます。指定した基数に無効な文字があると、strtol はエラーを示すことなく変換を終了します。
-e *error*	*error* をエラーメッセージとして定義します。
-h *help*	*help* をヘルプメッセージとして定義します。
-k *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
-l *lower*	*lower* を範囲の下限として定義します。デフォルト値は、マシンの負の最大整数です。
-p *prompt*	*prompt* をプロンプトメッセージとして定義します。
-Q	有効な応答として終了 (quit) を使用できないようにします。
-s *signal*	終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。
-u *upper*	*upper* を範囲の上限として定義します。デフォルト値は、マシンの正の最大整数です。
-W *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定します。

input	上限、下限、および基数と照合される入力

使用例

例 1 デフォルトのプロンプト

ckrange のデフォルトのプロンプトは、次のとおりです (基数は 10)

```
example% ckrange Enter an integer between lower_bound and
upper_bound [lower_bound–upper_bound,?,q]:
```

例 2 デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/errange
ERROR: Please enter an integer between lower_bound \
     and upper_bound.
```

例 3 デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/helprange
Please enter an integer between lower_bound and upper_bound.
```

例 **3** デフォルトのヘルプメッセージ　*(続き)*

例 **4** 基数を 10 以外の数に変更した場合のメッセージ

基数を 10 以外の数に設定した場合、メッセージは "integer" から "base *base* integer" に変更されます。次に例を示します。

```
example% /usr/sadm/bin/helprange -b 36
```

例 **5** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **6** valrange モジュールの使用

`valrange` モジュールは、標準エラー出力に使用法に関するメッセージを作成します。正常終了したな場合は `0`、失敗した場合には `0` 以外の値を返します。

```
example% /usr/sadm/bin/valrange
usage: valrange [-l lower] [-u upper] [-b base] input
```

終了ステータス

次の終了ステータスが返されます。

`0`　正常終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`2`　使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`intro`(1), `face`(1), `getopt`(1), `strtol`(3C), `attributes`(5), `signal`(3HEAD)

名前

ckstr, errstr, helpstr, valstr – 文字列の入力要求とその検証

形式

ckstr [-Q] [-W *width*] [[-r *regexp*] [...]] [-l *length*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [- s *signal*]]

/usr/sadm/bin/errstr [-W *width*] [-e *error*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/helpstr [-W *width*] [-h *help*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/valstr [-l *length*] [[-r *regexp*] [...]] *input*

機能説明

ckstr ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、文字列の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

このコマンドから返される応答は、定義済みの正規表現に一致していなければならず、指定された長さ以内でなければなりません。正規表現が指定されていない場合、有効な入力は、内部に空白を含まず、また先行や後続する空白がない、指定された長さ以内の文字列でなければなりません。長さが指定されていない場合、長さは検査されません。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckstr コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errstr (標準出力上にエラーメッセージを書式化して表示する)、helpstr (標準出力上にヘルプメッセージを書式化して表示する)、および valstr (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション

次のオプションを指定できます。

-d *default*
: *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error*
: *error* をエラーメッセージとして定義します。

-h *help*
: *help* をヘルプメッセージとして定義します。

-k *pid*
: ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-l *length*
: 入力の最大長を指定します。

`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (quit) を使用できないようにします。
`-r` *regexp*	入力を検証するときの基準となる正規表現、 *regexp* を指定します。空白を含めることができます。複数の式が定義されている場合、応答はその内のいずれかに一致しなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを *width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input	書式の長さや正規表現の基準に対して検証される入力

使用例

例 **1** デフォルトのプロンプト

`ckstr` のデフォルトのプロンプトは、次のとおりです。

```
example% ckstrEnter an appropriate value [?,q]:
```

例 **2** デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、適用される妥当性検査のタイプによって異なります。ユーザーには、長さまたはパターン照合のいずれかが失敗したことが通知されます。デフォルトのエラーメッセージは、次のとおりです。

```
example% /usr/sadm/bin/errstr
ERROR: Please enter a string which contains no embedded,
leading or trailing spaces or tabs.
```

例 **3** デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージも、適用される妥当性検査のタイプによって異なります。正規表現が定義されている場合、メッセージは次のようになります。

```
example% /usr/sadm/bin/helpstr -r regexp
Please enter a string which matches the following pattern:
regexp
```

他のメッセージは、文字列の長さの要件と定義を指定します。

例 **4** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **5** valstr モジュールの使用

valstr モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valstr
usage: valstr [-l length] [[-r regexp] [ . . . ]] input
```

終了ステータス

次の終了ステータスが返されます。

0　正常終了

1　入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは、使用法に誤りがあった

2　無効な正規表現

3　ユーザー終了 (quit)

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

face(1), signal(3HEAD), attributes(5)

cksum(1)

名前 | cksum – ファイルのチェックサムとサイズの出力

形式 | **cksum** [*file...*]

機能説明 | cksum コマンドは、各入力ファイルの循環冗長性検査 (CRC) を算出し、その結果を各ファイルのオクテット数と共に標準出力に書き込みます。

正常に処理された各ファイルについて cksum は次の書式でチェックサム情報を出力します。

"%u %d %s\n" <*checksum*>, <*# of octets*>, <*path name*>

file オペランドが指定されていない場合、パス名とその前の空白は省略されます。

使用される CRC は、参照されるイーサネット標準における CRC エラーチェックで使用される多項式に基づいています。

CRC チェックサムのコード化は、生成多項式によって定義されます。

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

指定したファイルに対応する CRC 値は、数学的には次の手順で定義されます。

1. 評価される *n* ビットは、2 を法とした n-1 次多項式 $M(x)$ の係数と見なされます。これらの *n* ビットは、ファイル内のビットに対応し、最上位ビットはファイルの先頭のオクテットの最上位ビットであり、最後のビットはファイルの最後のオクテットの最下位ビットです。これらの n ビットには、オクテットの数が整数になるように、(必要であれば) 0 のビットが埋め込まれ、その後にファイルの長さを表す 1 つまたは複数のオクテットが、最下位オクテットを先頭にして 2 進数で続きます。この整数を表すことができる最小のオクテット数が使用されます。
2. $M(x)$ の値は、x^{32} で乗算されて (つまり、32 ビット左にシフト)、2 を法とした除算を使用して $G(x)$ で除算されます。結果として、次数 ≤ 31 の除余 $R(x)$ が生成されます。
3. $R(x)$ の係数は、32 ビットシーケンスと見なされます。
4. ビットシーケンスの補数がとられ、その結果が CRC になります。

オペランド | 次のオペランドを指定できます。

file — 検査するファイルのパス名。*file* オペランドを指定しないと、標準入力が使用されます。

使用法 | cksum コマンドは通常、疑いのあるファイルを同じファイルの信頼できるバージョンと簡単に比較して、ノイズの多い媒体を介して送信されたファイルが正しく受信できたかどうかを確認するのに使用します。ただし、この比較は、暗号化した場合ほど安全ではありません。損傷を受けたファイルが、元のファイルと同じ CRC を生成する確率は天文学的低い値です。ただし、意図的に同じ CRC を生成することは困難ではありますが、不可能ではありません。

cksum の入力ファイルはどんなタイプでもかまいませんが、文字型特殊デバイスファイルでは期待通りの結果が得られないことがあります。本書では、入力時に使用されるブロックサイズを規定していないため、文字型特殊ファイルのチェックサムでは、これらのファイル内のデータがすべて処理されるとはかぎりません。

アルゴリズムは、オクテット単位に分割されたビットストリームで表現されます。ファイルが 2 つのシステム間で送信されるときに、何らかのデータ変換 (8 ビット文字を 9 ビットバイトに移行したり、「リトルエンディアン」のバイト順序を「ビッグエンディアン」に変更するなど) が行われる場合、同じ CRC 値を得ることは期待できません。このような変換を実行する実装では、cksum を拡張して、そのような状況に対処していきます。

ファイルのサイズが 2G バイト (2^{31} バイト) 以上ある場合の cksum の動作については、largefile(5) のマニュアルページを参照してください。

環境

cksum の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH については、environ(5) のマニュアルページを参照してください。

終了ステータス

次の終了ステータスが返されます。

0　　すべてのファイルが正常に処理された

>0　　エラーが発生した

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

sum(1), attributes(5), environ(5), largefile(5)

名前 | cktime, errtime, helptime, valtime – 時刻の入力要求とその検証

形式 | **cktime** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errtime [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helptime [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valtime [-f *format*] *input*

機能説明 | cktime ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに時刻の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザー応答は、時刻について定義されている書式に一致しなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

cktime コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errtime (エラーメッセージを書式化して表示する) と helptime (ヘルプメッセージを書式化して表示する) と、valtime (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errtime および helptime の各モジュールに format が定義されている場合、メッセージには指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。次の書式と定義を指定できます。

```
%H  =  hour (00 - 23)
%I  =  hour (00 - 12)
%M  =  minute (00 - 59)
%p  =  ante meridian or post meridian
%r  =  time as %I:%M:%S %p
%R  =  time as %H:%M (the default format)
%S  =  seconds (00 - 59)
%T  =  time as %H:%M:%S
```

-h *help* *help* をヘルプメッセージとして定義します。

-k *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* *prompt* をプロンプトメッセージとして定義します。

-Q 有効な応答として終了 (quit)を使用できないようにします。

-s *signal* 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

-W *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった

3 ユーザー終了 (quit)

4 無効な書式引数

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

attributes(5)

注意事項

cktime のデフォルトのプロンプトは、次のとおりです。

```
Enter a time of day [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter the time of day.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the time of day.  Format is <format>.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード 3 と共に q が返されます。valtime モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前 | ckuid, erruid, helpuid, valuid – ユーザー ID の入力要求とその検証

形式 | **ckuid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erruid [-W *width*] [-e *error*]

/usr/sadm/bin/helpuid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valuid *input*

機能説明 | ckuid ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに既存のユーザー ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckuid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erruid (エラーメッセージを書式化して表示する) と helpuid (ヘルプメッセージを書式化して表示する) と、valuid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-m — ヘルプが要求された場合、またはユーザーがエラーを犯した場合は、すべてのログインのリストを表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/passwd` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、または使用法に誤りがあった

`2`　　使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、 `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckuid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the login name of an existing user [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR:  Please enter one of the following login names:  <List>
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのヘルプメッセージは、次のとおりです。

```
Please enter one of the following login names:  <List>
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。 `valuid` モジュールは出力を何も生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

ckyorn(1)

名前 | ckyorn, erryorn, helpyorn, valyorn – yes/no の入力要求とその検証

形式 | **ckyorn** [-Q] [-W *width*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erryorn [-W *width*] [-e *error*]

/usr/sadm/bin/helpyorn [-W *width*] [-h *help*]

/usr/sadm/bin/valyorn *input*

機能説明 | ckyorn は、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、「はい (yes)」または「いいえ (no)」の応答を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckyorn コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erryorn (エラーメッセージを書式化して表示する) と helpyorn (ヘルプメッセージを書式化して表示する) と、valyorn (応答を検証する) です。これらのモジュールは、FACE オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* デフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

-W *width* — プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input y、yes、または n、no (大文字小文字の任意の組み合わせ) に対して検証される入力

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった

2 使用法に誤りがあった

3 ユーザー終了 (quit)

属性

次の属性については、 attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

attributes(5)

注意事項

ckyorn のデフォルトのプロンプトは、次のとおりです。

```
Yes or No [y,n,?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter yes or no.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
To respond in the affirmative, enter y, yes, Y, or YES.
To respond in the negative, enter n, no, N, or NO.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード 3 と共に q が返されます。valyorn モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前 | clear – 端末画面の消去

形式 | **clear** [*term*]

機能説明 | `clear` は、可能であれば端末画面を消去します。*term* オペランドで端末タイプが指定されていない場合は、環境内で調べ、そのあとで terminfo データベースを調べ、画面を消去する方法を決定します。

オペランド | *term* 端末タイプを指定します。通常は、環境変数 `TERM` からデフォルト値を使用するので必要ありません。

属性 | 次の属性については、 `attributes`(5)のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | `tput`(1)、`attributes`(5)

名前 | cmp – 2 つのファイルの比較

形式 | **cmp** [-l] [-s] *file1 file2* [*skip1*] [*skip2*]

機能説明 | cmp ユーティリティは 2 つのファイルを比較します。 2 つのファイルの内容が同じである場合、 cmp は何も出力しません。 内容が異なる場合、 デフォルトの設定では、最初に差異が生じた箇所の行番号とバイト位置を標準出力に 書き出します。 バイト数と行数の初めをそれぞれ 1 とします。 一方のファイルが他方のファイルの全内容をファイルの先頭から持つ、 異なるファイルである場合、それを知らせます。 *skip1* と *skip2* はそれぞれ *file1* と *file2* の初期バイト位置を表します。 これは、 8 進数と 10 進数のいずれかになりますが、 0 から始まる数は 8 進数を表します。

オプション | `-l` 各差異部分について、バイト位置 (10 進数) と差異部分のバイト長 (8 進数) を書き出します。

`-s` ファイルの差異については何も書き出しません。終了ステータスのみを返します。

オペランド | 以下にオペランドを示します。

file1 比較する 1 つ目のファイルのパス名。 *file1* として – を指定すると、 標準入力とみなされます。

file2 比較する 2 つ目のファイルのパス名。 *file2* として – を指定すると、 標準入力とみなされます。

file1 と *file2* がどちらも標準入力を指す場合、 または同じ先入れ先出し型特殊ファイル、 ブロック型特殊ファイル、 あるいは文字型特殊ファイルを指す場合、 エラーで終了します。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の cmp の動作については、 largefile(5) を参照してください。

使用例 | **例 1** バイトごとのファイル比較

以下に例を示します。

```
example% cmp file1 file2 0 1024
```

これは file1 と file2 をバイトごとに比較しています。 比較を行う前に file2 を 1024 バイト分 とばしています。

環境 | cmp の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 2 つのファイルは同一だった。

1 2 つのファイルは異なっていた。 1 つのファイルがもう一方のファイルの先頭部分と同一の場合も、 このコードが返される。

>1 エラーが発生した。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | comm(1), diff(1), attributes(5), environ(5), largefile(5)

名前 col – 逆方向改行フィルタ

形式 **col** [-bfpx]

機能説明 col ユーティリティは、標準入力を読んで標準出力に書き出します。 col は、逆方向改行コードおよび順方向半改行と逆方向半改行が示す行の重ね打ちを実行します。 -x オプションが指定されない限り、入力中の空白文字を可能であればタブ文字に変換します。 col は nroff(1) の .rt コマンドによる複数カラム出力、および tbl(1) プリプロセッサを使用すると得られる出力にフィルタをかけるときに便利です。

col は ASCII 制御文字の SO および SI を、代替文字セットの開始および終了とみなします。 各入力文字が属する文字セットは記憶されていて、 出力上では各文字が正しい文字セットで印刷されるように必要に応じて SI と SO が生成されます。

入力データ中で認識される制御文字は、 空白文字、バックスペース、タブ、改行文字、復帰改行文字、SI、SO、VT、逆方向改行、順方向半改行、そして逆方向半改行です。 VT 文字は、逆方向全改行の代わりに使用できます。 これは、このプログラムの初期タイプと互換性を保つためです。 これら以外に出力上にコピーされる文字は、すべて印刷可能文字です。

上記の制御機能や改行文字の ASCII コードを以下の表に示します。 ESC は ASCII のエスケープ文字を表します。 8 進数コードでは 033 となります。 ESC- という表示は、 ESC の後に文字 x を伴った合計 2 文字の並びを示します。

逆方向改行	ESC-7
逆方向半改行	ESC-8
順方向半改行	ESC-9
仮想タブ (VT)	013
テキスト開始 (SO)	016
テキスト終了 (SI)	017

オプション

-b col 使用中の出力装置がバックスペースに対応できないとみなします。 この場合、同じ箇所に 2 文字以上が現われたときは最後に読んだ 1 文字が出力されます。

-f col は、入力時には 1/2 行単位の移動を認めますが、出力時には通常認めません。 その代わり、行間に現われたテキストは次の全改行の位置に移動します。 この処理は、 -f (fine) オプションによって抑制できます。 この場合、 col からの出力には順方向半改行 (ESC-9) が含まれる可能性がありますが、 逆方向改行または逆方向半改行が含まれることは決してありません。

-p 通常 col は、入力時に見つかった認識できないエスケープシーケンスをすべて無視します。 -p オプションは、逆方向改行による重ね打ちが発生することを前提として、 col がこれらのシーケンス

を通常の文字として出力できるようにします。 そのエスケープシーケンスのテキスト上の位置を完全に把握していないかぎり、このオプションは使用しないでください。

`-x` 出力上において空白文字を可能であればタブ文字に変換する、 という `col` の動作を抑止します。 タブ位置は、 8 カラムごとに設定されていると見なされます。

環境

`col` の実行に影響を与える環境変数 `LC_CTYPE`、 `LC_MESSAGES`、 `NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 正常終了

`>0` エラーが発生した。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

`nroff`(1), `tbl`(1), `ascii`(5), `attributes`(5), `environ`(5)

注意事項

`col` が使用する入力フォーマットは `nroff` に `-T37` または `-Tlp` オプションを付けた場合の 出力と一致します。 `col` が最終的に出力する装置が半改行を認識できないときは `-T37` オプション (および `col` の `-f` オプション) を、認識できるときは `-Tlp` をそれぞれ使用してください。

`col` は、128 行以上の行の逆戻りおよび 1 行当たり 800 字以上の処理はできません。

その文書の先頭行を越えて逆戻りする改行動作は無視されます。 したがって、先頭行にスーパースクリプトを使用しないでください。

名前 | comm – 2 つのファイルに共通な行または共通でない行の表示

形式 | **comm** [-123] *file1 file2*

機能説明 | comm ユーティリティは、現在有効な照合シーケンスでソートされた 2 つのファイル *file1* と *file2* を読んで、 *file1* だけに存在する行、 *file2* だけに存在する行、および両者に存在する行の計 3 つのカラムからなる出力を生成します。

入力ファイルが、現在のロケールの照合シーケンスで ソートされている場合、 出力でもその順序は保たれます。 そうでない場合、出力行の順序は不定です。

オプション | 以下のオプションを指定できます。

-1 *file1* だけに存在している行の出力を抑止します。

-2 *file2* だけに存在している行の出力を抑止します。

-3 *file1* と *file2* の両方に存在している行の出力を抑止します。

オペランド | 以下のオペランドを指定できます。

file1 比較する 1 つ目のファイルのパス名。 *file1* として – を指定すると、標準入力とみなされます。

file2 比較する 2 つ目のファイルのパス名。 *file2* として – を指定すると、標準入力とみなされます。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の comm の動作については、largefile(5) を参照してください。

使用例 | **例 1** ファイルによって指定された、ユーティリティのリストを出力する

file1、*file2*、*file3* の 3 つのファイルには、ユーティリティのリストが 正しくソートされて書かれているものとします。次のように実行すると、 *file1* だけに含まれているユーティリティだけが出力されます。

```
example% comm -23 file1 file2  | comm -23 - file3
```

次のように実行すると、 3 つのファイルすべてに含まれているユーティリティだけが出力されます。

```
example% comm -12 file1 file2 | comm -12 - file3
```

次のように実行すると、 *file2* と *file3* には含まれているが *file1* には含まれていないユーティリティだけが出力されます。

```
example% comm -12  file2 file3 | comm -23 -file1
```

環境 | comm の実行に影響を与える環境変数 LC_COLLATE、LC_CTYPE、 LC_MESSAGES、NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 入力ファイルはすべて、 (指定されたとおりに) 正常に出力された。

>0 エラーが発生した。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 | cmp(1), diff(1), sort(1), uniq(1), attributes(5), environ(5), largefile(5)

名前 | command – 単純コマンドの実行

形式 | **command** [-p] *command_name* [*argument...*]

command [-v | -V] *command_name*

機能説明 | command ユーティリティを使用すると、シェルは関数の検索を実行せずに、指定された引数を単純コマンドとして扱います。

command_name が特殊組み込みユーティリティのいずれかの名前である場合、特性が発生することはありません。*command_name* が関数の名前でない場合、通常、command の効果は、command を使用しないでユーティリティを呼び出した場合と同じ結果になります。

command ユーティリティは、シェルによるコマンド名の解釈方法についての情報も提供します。-v および -V を参照してください。

オプション | 次のオプションを指定できます。

-p　すべての標準ユーティリティの検索を可能にする PATH のデフォルト値を使用して、コマンド検索を実行します。

-v　現在のシェル実行環境で *command_name* を呼び出すためにシェルが使用するパスまたはコマンドを示す文字列を標準出力に書き込みます。

- ユーティリティ、正規の組み込みユーティリティ、スラッシュ文字を含む *command_names*、および PATH 変数を使用して検出される、実装で提供されている関数では、出力文字列に絶対パス名が含まれます。
- シェル関数、特殊組み込みユーティリティ、PATH 検索に関連しない正規の組み込みユーティリティ、およびシェル予約語では、出力文字列にはその名前だけが含まれます。
- 別名では、出力文字列は別名定義を表すコマンド行です。
- 上記以外では、何も出力されず、終了ステータスに名前が検出されなかったことが示されます。

-V　現在のシェル実行環境で、*command_name* オペランドで指定された名前をシェルが解釈する方法を示す文字列を、標準出力に書き込みます。この文字列の書式は規定されていませんが、出力文字列は指定した *command_name* が属するカテゴリを示し、次の情報を含みます。

- ユーティリティ、正規の組み込みユーティリティ、および PATH 変数を使用して検出される、実装で提供されている関数では、そのように識別され、文字列に絶対パス名が含まれます。
- これ以外のシェル関数は関数として識別されます。
- 別名は別名として識別されて、文字列には別名定義が含まれます。
- 特殊組み込みユーティリティは、特殊組み込みユーティリティとして識別されます。
- PATH 検索に関連しない正規の組み込みユーティリティは、正規の組み込みユーティリティとして識別されます。
- シェル予約語は、予約語として識別されます。

オペランド | 次のオペランドを指定できます。

argument *command_name* への引数として扱われる文字列の 1 つ。

command_name ユーティリティまたは特殊組み込みユーティリティの名前。

使用例

例 1 新しい作業ディレクトリを一度だけ出力する cd を作成する

```
cd() {
    command cd "$@" >/dev/null
    pwd
}
```

例 2 先に起動されたスクリプトに影響されない「安全なシェルスクリプト」の先頭部分

```
IFS='
'
#    The preceding value should be <space><tab><newline>.
#    Set IFS to its default value.
\unalias -a
#    Unset all possible aliases.
#    Note that unalias is escaped to prevent an alias
#    being used for unalias.
unset -f command
#    Ensure command is not a user function.
PATH="$(command -p getconf _CS_PATH):$PATH"
#    Put on a reliable PATH prefix.
#    . . .
```

PATH で参照されるディレクトリに適切な許可が与えられていれば、この時点でスクリプトは、呼び出すユーティリティがすべて目的のものであることを保証できます。このスクリプトは、実装システム上に拡張機能が用意されていることを前提としています。そのような拡張機能は、スクリプトが呼び出されるときに、ユーザー関数が存在できるようにします。このような拡張機能は本書では規定されていませんが、拡張機能としては禁止されていません。たとえば、ENV 環境変数が、このスクリプトの前に実行されるユーザー起動スクリプトを定義しているとします。その場合、ユーザー起動スクリプトは、アプリケーションを出し抜く (アプリケーションが呼び出す予定のユーティリティに代わる) 関数を定義できます。

環境

command の実行に影響を与える環境変数、LC_CTYPE、 LC_CTYPE、LC_MESSAGES、 NLSPATH については、environ(5) のマニュアルページを参照してください。

PATH コマンド検索中に使用される検索パスを決定します。-p オプションが指定されている場合は除きます。

終了ステータス

-v または -V オプションを指定すると、次の終了ステータスが返されます。

0 正常終了

>0 *command_name* が検出できなかった、あるいはエラーが発生した

上記以外の場合は、次の終了ステータスが返されます。

126 *command_name* で指定したユーティリティが見つかったが、呼び出せなかった

127 command ユーティリティでエラーが発生した、あるいは *command_name* で指定したユーティリティが見つからなかった。

上記以外の場合、command の終了ステータスは、command_name 引数で指定した単純コマンドの終了ステータスになります。

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

sh(1), type(1), attributes(5)

名前 | compress, uncompress, zcat – ファイルの圧縮、圧縮解除、または圧縮解除結果の表示

形式 | **compress** [-fv] [-b *bits*] [*file*...]

compress [-cfv] [-b *bits*] [*file*]

uncompress [-cfv] [*file*...]

zcat [*file*...]

compress | compress ユーティリティは、適応型レンペル・ジブ・コーディング法を利用して、指定されたファイルの圧縮を試みます。出力先が標準出力でないとき、各ファイルは、可能な限り、拡張子 .Z の付いたファイルに置き換えられます。所有モード、アクセス時刻、更新時刻は変わりません。なお .Z をファイルに付加したときパス名の長さが 1023 バイトを超えてしまう場合、コマンドの実行は失敗します。ファイル名が指定されていない場合には、標準入力が圧縮されて結果が標準出力に送られます。

どの程度圧縮されるかは、入力ファイルのサイズ、コードあたりのビット数、共通部分文字列の配置などによって異なります。通常、ソースコードや英語の文章などのテキストの場合、50% から 60% は圧縮されます。この圧縮処理は、ハフマンコーディング法 (pack(1) で使用されている) で行われる圧縮処理よりも優れており、計算時間も短くてすみます。このコマンド中に指定される *bits* パラメータの値は、圧縮結果のファイル内にコード化されます。さらにこのファイル内には、ランダムデータの圧縮解除および圧縮データの再圧縮を防止するためのマジックナンバーも書き込まれます。

uncompress | uncompress ユーティリティは、compress ユーティリティによって圧縮されたファイルを元の状態に復元します。ファイル名を省略すると、標準入力を圧縮解除して標準出力に書き出します。

このユーティリティは、compress が生成したどのようなファイルでも、圧縮解除します。ただし他のシステム上で compress が生成したファイルに関しては、9 および 16 ビット圧縮だけがサポートの対象となります。詳しくは -b の説明を参照してください。

zcat | zcat ユーティリティは、compress により圧縮されたファイルの圧縮解除した内容を、標準出力に出力します。これは uncompress -c と同じ結果となります。入力ファイルの内容は変わりません。

オプション | 以下のオプションを指定できます。

-c　結果を標準出力に書き出します。既存のファイルは変更されず、.Z ファイルも作られません。つまり uncompress -c という指定は zcat と同機能です。

-f　圧縮の場合、圧縮してもファイルが小さくならない場合や、対応する *file*.Z ファイルがすでに存在している場合でも、強制的に圧縮を行います。このオプションを指定しない場合は、プロセスがバックグラウンドで実行している場合を除き、既存の *file*.Z ファイルを上書きするかどうかの確認を求めるプロンプトが表示されます。圧縮解除の場合、このオプションは、上書きの確認用プロンプトを表示しないよう指定します。-f オプションを指定しない場合は、プロセスがバックグラウンドで実行してい

る場合を除き、既存のファイルを上書きするかどうかの確認を求めるプロンプトが表示されます。標準入力が端末ではないときに -f オプションが省略されると、標準エラー出力に診断メッセージを出力し、0 より大きな終了ステータスで終了します。

-v　冗長 (verbose) 。各ファイルが何パーセント圧縮または圧縮解除されたかを、標準エラー出力に書き出します。

-b *bits*　共通部分文字列コードの最大長 (ビット単位) を指定します。*bits* には、9 から 16 の値を指定します (デフォルトは 16 です)。この値を小さくすると、圧縮率が低くなり、結果ファイルは大きくなります。

オペランド

以下のオペランドを指定できます。

file　compress で圧縮または uncompress で圧縮解除するファイルのパス名、または圧縮解除された内容が zcat によって標準出力に書き込まれるファイルのパス名。– を指定するか、このオペランドを省略すると、標準入力とみなされます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の compress 、uncompress 、zcat の動作については、largefile(5) を参照してください。

環境

compress 、uncompress 、zcat の実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES 、NLSPATH の詳細については、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了した

1　エラーが発生した

2　(-f オプションが指定されていないため) サイズが大きくなってしまい、いくつかのファイルが圧縮されなかった

>2　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

ln(1)、pack(1)、attributes(5)、environ(5)、largefile(5)

関連項目

Usage: compress [-fvc] [-b maxbits] [*file*. . .]
　コマンド行に無効なオプションが指定されました。

Missing maxbits
　-b の後に最大ビット数が指定されていません、または不正な値か数値でない値が指定されました。

file: not in compressed format
uncompress に指定されたファイルは 圧縮されたファイルではありません。

file: compressed with *xx*bits, can only handle *yy*bits
file は、このマシンの圧縮コードよりも大きいビット数を扱うプログラムによって、圧縮されています。これより小さいビット数でファイルを再圧縮する必要があります。

file: already has .Z suffix -- no change
このファイルは、すでに圧縮されたものと思われます。ファイル名を変更して、再試行してください。

file: already exists; do you wish to overwrite (y or n)?
出力ファイルで置き換える場合は y を、置き換えない場合は n と応答してください。

uncompress: corrupt input
SIGSEGV 違反が検出されました。通常、入力ファイルが破損していることを示します。

Compression: *xx.xx*%
入力ファイルが圧縮によって縮小された割合です。(-v オプションが指定された場合のみ表示します。)

– – not a regular file: unchanged
入力ファイルが通常ファイルではない場合 (たとえばディレクトリ)、内容は変更されません。

– – has *xx* other links: unchanged
入力ファイルにリンクがあります。内容は変更されません。詳細は、ln(1) を参照してください。

– – file unchanged
圧縮しても、縮小されません。入力ファイルの内容は 変更されません。

filename too long to tack on .Z
パス名が長すぎて、接尾辞 .z を付加できません。

注意事項

圧縮されたファイルは、大容量メモリーを持つマシン間では互換性がありますが、プロセスあたりのデータ領域が小さい (64K バイト以下) アーキテクチャへのファイル転送には -b 12 を指定してください。

拡張子 . z の付いたファイルが存在する場合の compress の処理は、もう少し柔軟であるべきです。

名前 | break, continue – while、for、foreach、until の各ループ制御から抜け出す、または続行するためのシェル組み込み関数

sh | **break** [*n*]

continue [*n*]

csh | **break**

continue

ksh | ***break** [*n*]

***continue** [*n*]

sh | break は for ループまたは while ループ中であれば、ループを終了します。 *n* を指定すると、 *n* レベル分だけ ループを終了します。

continue は for ループまたは while ループの次の繰り返しを実行します。 *n* を指定すると、 *n* 番目のループから実行します。

csh | break は foreach または while 内の最も内側にあるループの end の次から実行を再開します。 現在の行の残りのコマンドは実行されます。 これによって、複数レベルのループから抜けるには、 break コマンドを 1 行に複数記述します。

continue は while または foreach 内の最も内側にあるループの、次の繰り返しから実行します。

ksh | break は for ループ、 while ループ、 until ループ、または select ループがあれば終了します。 *n* を指定すると、 *n* レベル分だけ ループを終了します。

continue は for ループ、 while ループ、 until ループ、 または select ループの次の繰り返しを実行します。 *n* を指定すると、 *n* 番目のループから実行します。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | csh(1), exit(1), ksh(1), sh(1), attributes(5)

名前 | cp – ファイルのコピー

形式 | **/usr/bin/cp** [-fip@] *source_file target_file*

/usr/bin/cp [-fip@] *source_file… target*

/usr/bin/cp -r | -R [-fip@] *source_dir… target*

/usr/xpg4/bin/cp [-fip@] *source_file target_file*

/usr/xpg4/bin/cp [-fip@] *source_file… target*

/usr/xpg4/bin/cp -r | -R [-fip@] *source_dir… target*

機能説明 | 上記「形式」の項で示す第 1 の形式では 、*source_file* および *target_file* はディレクトリファイルではありません。また、両者は同じ名前ではありません。cp ユーティリティは、*source_file* が示すファイルの内容を、*target_file* が示す宛先パスにコピーします。*target_file* がすでに存在していると、cp はその内容を上書きしますが 、ファイルのモード (さらに ACL が有効な場合は ACL)、所有者、およびグループは変更しません。*target_file* の最新更新時刻および *source_file* の最新アクセス時刻の値は、コピーが行われた時刻に設定されます。*target_file* が存在しなければ、cp は *target_file* という名のファイルを作成します。このファイルのモードは *source_file* と同じになります。ただしコマンドを発行したユーザーがスーパーユーザーでない限り、スティッキビットは設定されません。この場合、*target_file* の所有者とグループは (*target_file* を作成した) ユーザーのものと同じになります。ただし、新たに作成されたファイル (*target_file*) を含むディレクトリに setgid ビットがセットされている場合を除きます。このディレクトリに setgid ビットがセットされている場合、新たに作成されたファイル (*target_file*) のグループは、そのファイルを作成したユーザーのグループではなく、そのファイルを含むディレクトリのグループになります。*target_file* が別のファイルへのリンクである場合、リンク先のファイルに対して *source_file* の内容を上書きします。*target_file* からのリンクはそのまま残ります。

第 2 の形式の cp は、*source_file* が示す 1 つ以上のファイルを、*target* が示すディレクトリにコピーします。個々の *source_file* に対して、同じモード (さらに ACL が有効な場合は同じ ACL) の新たなファイルが *target* 中に作成されます。所有者とグループは、このコマンドを 発行したユーザーのものが用いられます。*source_file* がディレクトリの場合 、*target* が存在しない場合、および *target* がディレクトリでない場合は、いずれもエラーとなります。

第 3 の形式の cp は 、*source_dir* が示す 1 つ以上のディレクトリを、*target* が示すディレクトリにコピーします。-r または -R のどちらかを指定しなければなりません。個々の *source_dir* に関して 、cp はその中のすべてのファイルとサブディレクトリをコピーします。

オプション | 以下のオプションは 、/usr/bin/cp と /usr/xpg4/bin/cp で指定できます。

-f　リンク解除。宛先ファイルのファイル記述子が得られない場合、宛先ファイルのリンクを解除して処理を続けようとします。

-i　対話型。宛先ファイルとして指定した *target* がすでに存在している場合、上書きしてもよいかどうかを問い合わせる確認メッセージを出力します。

そのメッセージに対して y と応答すると、コピー処理が続行されます。他の文字で応答すると上書きは実行されません。

`-r` 再帰。cp はそのディレクトリ全体、つまりディレクトリ内の全ファイルをコピーするだけでなく、ディレクトリ にあるすべてのサブディレクトリとそのサブディレクトリ中の全ファイルも *target* にコピーします。

`-R` パイプが読まれるのではなく複写される、という点を除いて `-r` と同じです。

`-@` 拡張属性を保持します。cp は、ファイルのデータとともに、すべてのソースファイルの拡張属性を宛先ファイルにコピーしようとします。

/usr/bin/cp 以下のオプションは `/usr/bin/cp` でのみ指定できます。

`-p` 保持。cp は *source_file* で示したファイルの内容をコピーするだけでなく、出力側ファイルの 所有者とグループの ID、アクセス権モード、更新時刻とアクセス時刻、および、適用可能であれば ACL と拡張属性をそのまま設定します。ACL をサポートしていないファイルシステムに ACL をコピーすると、このコマンドは、失敗する可能性があります。このコマンドは、拡張属性、更新時刻およびアクセス時刻、またはアクセス権モードを設定することができない場合には、失敗しません。また、所有者とグループの ID を設定できない場合も 、cp は失敗しないで、出力側ファイルの `S_ISUID` と `S_ISGID` ビット をクリアします。cp は、これらのビットをクリアできない場合には 、標準エラー出力に診断メッセージを表示し、ゼロ以外の終了ステータスを返します。

所有者とグループの ID 、アクセス権モード、更新時刻およびアクセス時刻を保存するためには 、ユーザーが、適切なファイルのアクセス権を持っている必要があります。つまり、スーパーユーザーであることや、目的のファイルと同じ所有者 ID であることが必要です。

/usr/xpg4/bin/cp 以下のオプションは `/usr/xpg4/bin/cp` でのみ指定できます。

`-p` 保持。cp は *source_file* で示したファイルの内容をコピーするだけでなく、出力側ファイルの 所有者とグループの ID、アクセス権モード、更新時刻とアクセス時刻、および、適用可能であれば ACL と拡張属性をそのまま設定します。ACL または拡張属性をサポートしていないファイルシステムに ACL をコピーすると、このコマンドは、失敗する可能性があります。更新時刻とアクセス時刻、またはアクセス権モードを設定できない場合 、cp は 標準エラー出力に診断メッセージを表示し、ゼロ以外の終了ステータスを返します。所有者とグループの ID を設定できない場合は、cp は失敗しないで、出力側ファイルの `S_ISUID` と `S_ISGID` ビット をクリアします。cp はこれらのビットがクリアできなければ 、標準エラー出力に診断メッセージを表示し、ゼロ以外の終了ステータスを返します。

所有者とグループの ID 、アクセス権モード、更新時刻およびアクセス時刻を保存するためには、ユーザーが、適切なファイルのアクセス権を持っている必要があります。つまり、スーパーユーザーであることや、目的のファイルと同じ所有者 ID であることが必要です。

オペランド　以下のオペランドを指定できます。

source_file	コピーされる通常ファイルのパス名
source_dir	コピーされるディレクトリのパス名
target_file	1 つのファイルをコピーする際に出力として用いられるファイルのパス名。既存のファイルでも存在していないファイルでもよい
target	コピーしたファイルを出力するディレクトリのパス名

使用法　ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の cp の動作については、largefile(5) を参照してください。

使用例

例 1 1 つのファイルをコピーする

```
example% cp goodies goodies.old

example% ls goodies*
goodies goodies.old
```

例 2 一群のファイルをディレクトリにコピーする

```
example% cp ~/src/*   /tmp
```

例 3 あるディレクトリを、最初に新たなディレクトリへ、次に既存のディレクトリへコピーする

```
example% ls ~/bkup
/usr/example/fred/bkup not found

example% cp -r ~/src ~/bkup

example% ls -R ~/bkup
x.c y.c z.sh

example% cp -r ~/src ~/bkup

example% ls -R ~/bkup
src x.c y.c z.sh
src:
x.c y.c z.s
```

環境　cp の実行に影響を与える環境変数 LC_COLLATE 、LC_CTYPE 、LC_MESSAGES 、NLSPATH についての詳細は 、environ(5) を参照してください。

終了ステータス　以下の終了ステータスが返されます。

0	ファイルはすべて正常にコピーされた
>0	エラーが発生した

属性　次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/cp

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み
インタフェースの安定性	安定

/usr/xpg4/bin/cp

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み
インタフェースの安定性	標準

関連項目 chmod(1), chown(1), setfacl(1), utime(2), attributes(5), environ(5), fsattr(5), largefile(5), XPG4(5)

注意事項 ソースファイルのアクセス権モードは、コピー時にそのまま保存されます。

ユーザーはハイフンを 2 つ連続して記述 (–) することにより 、コマンド行オプションの終わりを明確に指定することができます。この方法を用いれば、ハイフンで始まるファイル名を *filename* 引数として指定したとき、それがファイル名であることを cp に伝えることができます。

名前

cpio – アーカイブからのファイルの抽出または復元

形式

cpio -i [-bBcdfkmPrsStuvV6@] [-C *bufsize*] [-E *file*] [-H *header*] [-I *file* [-M *message*]] [-R *id*] [*pattern...*]

cpio -o [-aABcLPvV@] [-C *bufsize*] [-H *header*] [-O *file* [-M *message*]]

cpio -p [-adlLmPuvV@] [-R *id*] *directory*

機能説明

cpio コマンドは、ファイルを cpio アーカイブへコピーしたり、cpio からコピーしたりします。cpio アーカイブは多数のファイルの容量を補います。-i 、 -o 、 -p の各オプションは、実行する処理を選択します。以下に、これらの各処理を説明します (これらの処理は相互に排他的です)。

復元モード

cpio -i (copy in、復元) は、以前に cpio -o コマンドが生成した出力を標準入力としてファイルを抽出します。*pattern* の 1 つに一致する名前を持つファイルのみが選択されます。*pattern* についての詳細は、後述する「オペランド」の項および sh(1) の説明を参照してください。抽出されたファイルは条件に応じて作成され、以下に述べるオプションに基づいて現在のディレクトリにコピーされます。ファイルのアクセス権は前の cpio -o コマンドの出力ファイルと同じです。現在のユーザーがスーパーユーザーでないかぎり、所有者およびグループは現在のユーザーと同じです。スーパーユーザーの場合は、前に実行した cpio -o コマンドの出力ファイルから得られる所有者およびグループと同じになります。cpio -i が作成しようとするファイルがすでに存在していて、さらに既存のファイルが同じか新しいバージョンのときは、メッセージが出力されファイルは置換されないので注意してください。-u オプションは既存のファイルを無条件に置換するときに使用します。

保管モード

cpio -o (copy out、保管) は、ファイルのパス名のリストを標準入力から読み込んで、それらのファイルをパス名およびステータス情報と共に、cpio アーカイブ形式で標準出力へコピーします。出力は、デフォルトでは 8192 バイト境界でブロック化されますが、(-B または -C オプションを使用して) ユーザーが指定したブロックサイズまたは (CTC テープのように) デバイスに依存したブロックサイズでの出力も可能です。

パスモード

cpio -p (pass、パス) は、ファイルのパス名のリストを標準入力から読み込んで、条件に応じてファイルを作成し、以下のオプションに基づいてコピー先の ディレクトリへコピーします。

注意 : cpio は 4 バイトワードを前提としています。

文字型デバイスへ出力 (-o) または文字型デバイスからの入力 (-i) 時に、媒体の終わり (たとえば、フロッピーディスクの終わり) に到達し、さらに -O オプションや -I オプションが使用されていないと cpio は次のメッセージを表示します。

```
To continue, type device/file name when ready.
```

継続するには、媒体を交換して文字型特殊デバイス名 (たとえば /dev/rdiskette) を入力して RETURN キーを押してください。これは、cpio に異なるデバイスを使用するように命令して、処理を続行したい場合などに使います。たとえば、2 台のフロッピーデバイスがあるなら、フロッピー交換の間に cpio が処理を継続できるよう他方のデバイスへ切り替えたいという場合です。RETURN キーを押すと cpio は処理を終了します。

オプション

以下のオプションを指定できます。

-i (copy in、復元) 標準入力からアーカイブを読み込んで、条件に応じて、アーカイブに含まれているファイルを抽出し、現在のディレクトリツリーに配置します。

-o (copy out、保管) ファイルのパス名のリストを標準入力から読み込んで、そのファイルを cpio 形式で標準出力へコピーします。

-p (pass、パス) ファイルのパス名のリストを標準入力から読み込んで、条件に応じて、読み込んだファイルを宛先のディレクトリツリーへコピーします。

上記の -i、-o、-p オプションのいずれかを指定した後に、以下のオプションを任意の順序で指定できます。

-a コピー後に入力ファイルのアクセス時間をリセットして、cpio によるアクセスの痕跡を消去します。cpio -pla が指定されていると、リンクされたファイルのアクセス時間はリセットされません。

-A ファイルをアーカイブへ追加します。-A オプションには -O が必要です。このオプションは、ファイル、フロッピーディスク、またはハードディスクパーティションのアーカイブにのみ有効です。アーカイブ内に以前から存在する、リンクされたファイルへの影響は予測不能です。

-b 各ワード内のバイト順を逆にします。-i オプションとのみ使用できます。

-B 入出力を 5120 バイトでブロック化します。このオプションと -C オプションを使用していないときのデフォルトのバッファサイズは 8192 バイトです。-B は -p (pass、パス) オプションには適用されません。

-c 可搬性のために、ASCII 文字形式でヘッダー情報を読み書きします。このヘッダー形式に関してユーザー ID またはグループ ID の制限はありません。SVR4 をベースとしたマシン間ではこのオプションを使用してください。また種類が不明なマシン間では -H odc オプションを使用してください。-c オプションは、SVR4 ベースのシステムでだけサポートされている拡張デバイス番号を使うことを意味しています。SunOS 4 または Interactive UNIX

と、Solaris 2.6 オペーレーティング環境またはその互換バージョンとの間でファイルを転送する場合は、`-H odc` を使用してください。

`-C` *bufsize*
: 入出力を *bufsize* 単位でブロック化します。*bufsize* は正の整数で置き換えられます。このオプションと `-B` オプションを使用していないときのデフォルトのバッファサイズは 8192 バイトです。`-C` は `-p` (pass、パス) オプションには適用されません。

`-d`
: 必要に応じてディレクトリを作成します。

`-E` *file*
: アーカイブから抽出するファイル名のリストを含む (1 行 1 ファイル名の) 入力ファイル (*file*) を指定します。

`-f`
: *pattern* で指定されたものを除くすべてのファイルを抽出します。*pattern* については「オペランド」の項を参照してください。

`-H` *header*
: *header* 形式でヘッダー情報を読み書きします。コピー先とコピー元のマシンが異なるタイプのときは常にこのオプション (または `-c` オプション) を使用してください。このオプションは、`-c` および `-6` と同時には使用できません。*header* として有効な値は以下のとおりです。

  `bar`
  : `bar` ヘッダーと形式。`-i` オプション (読み取り専用) とのみ使用されます。

  `crc` または `CRC`
  : 拡張デバイス番号とファイル単位のチェックサムを持った ASCII ヘッダー。このヘッダー形式に関して ユーザー ID またはグループ ID の制限はありません。

  `odc`
  : スモールデバイス番号を持った ASCII ヘッダー。これは IEEE/P1003 データ交換標準による cpio のヘッダーと形式です。他のヘッダー形式と比べて、最も幅広い可搬性を提供します。POSIX に準拠したシステム間でファイル転送を行う場合の公式の形式です (`standards`(5) を参照)。SunOS 4 または Interactive UNIX と通信する場合には、この形式を使用してください。このヘッダー形式では 262143 までのユーザー ID とグループ ID をヘッダーに格納できます。

  `tar` または `TAR`
  : `tar` ヘッダーと形式。これは 2097151 までのユーザー ID とグループ ID を格納できる、古い `tar` ヘッダー形式です。`-i` オプションと一緒に使用して、古い形式のアーカイブを読み込むためにのみ用意されています。

このアーカイブ形式を-o オプションと一緒に使用すると、"ustar" 形式を指定した場合と同じ効果が得られます。つまり、出力アーカイブは ustar 形式になり、アーカイブの読み込みには -H ustar を使用する必要があります。

ustar または USTAR	IEEE/P1003 データ交換標準の tar ヘッダーと形式。このヘッダー形式では 2097151 までのユーザー ID とグループ ID をヘッダーに格納できます。

上述の制限値を超えたユーザー ID とグループ ID を持つファイルは 60001 のユーザー ID とグループ ID で保存されます。大規模ファイル (8 Gバイト — 1 Gバイト) を転送するには、ヘッダー形式は tar または TAR、 ustar または USTAR、odc のいずれかを使用できます。

-I *file* 入力アーカイブとして、標準入力の代わりに *file* の内容を読み込みます。*file* が文字型特殊デバイスで、現在の媒体をすべて読み終えた場合、 処理を続けるために媒体を交換して RETURN キーを押してください。このオプションは -i オプションとのみ使用できます。

-k 破壊されたファイルヘッダーや I/O エラーを読み飛ばします。破壊されたりシーケンスが乱れたりした媒体からファイルをコピーしたい場合は、このオプションによって正常なヘッダーを持つファイルだけを読むことができます。他の cpio アーカイブを含む cpio アーカイブの場合、エラーが発生すると cpio は終了します。cpio は次の正常なヘッダーを検索し、より小さいアーカイブを見つけるとそのトレーラに到達するまで読み取って、終了します。-i オプションとのみ使用できます。

-l パスモードでは、可能な場合は必ずリンク元とリンク先の間にハードリンクを作成します。-L オプションを同時に指定した場合は、シンボリックリンクによって参照されているファイルに対してハードリンクを作成します。そうでない場合は、シンボリックリンク自体に対してハードリンクを作成します。このオプションは、-p オプションとのみ使用してください。

-L シンボリックリンクをたどります。 シンボリックリンクの宛先がディレクトリであった場合は、参照されているディレクトリを、そのリンクの名前で保存します。そうでない場合は、参照されているファイルを、そのリンクの名前で保存します。

-m 以前のファイル更新時間を保持します。このオプションは、コピー中のディレクトリには無効です (-a とは同時使用不可)。

-M *message* 媒体交換時の *message* を定義します。オプション -O または -I を使用して、文字型特殊デバイスを指定しているときに、媒体の終

わりに達したときに出力されるメッセージを定義します。次の媒体のシーケンス番号を表示するのに 1 つの `%d` を使用できます。

`-O` *file*
: `cpio` の出力先を、標準出力から *file* に変更します。*file* が文字型特殊デバイスで現在の媒体が一杯のとき処理を継続するには、媒体を交換してキャリッジリターンを押してください。`-o` オプションとのみ使用できます。

`-P`
: ACL を保持します。このオプションを出力用に使用した場合、既存の ACL が、拡張属性以外のその他の属性とともに標準出力に書き出されます。ACL は、特殊なファイルタイプを持つ特殊ファイルとして作成されます。このオプションを入力用に使用した場合、ACL は他の属性とともに標準入力から復元されます。このオプションは特殊ファイルタイプを認識できます。なお旧バージョンの `cpio` を使って ACL を持つ cpio アーカイブを復元しようとすると、エラーが発生します。ACL がすべてのシステムにサポートされているとは 限らず、可搬性がないため、このオプションは `-c` オプションとともに使用しないでください。可搬性を保つために ASCII ヘッダーを使用してください。

`-r`
: 対話形式でファイル名を変更します。キャリッジリターンだけを押すと、そのファイルを飛ばします。"." と入力すると、元のパス名を使用します。`cpio -p` とは使用できません。

`-R` *id*
: 各ファイルの所有者とグループ情報を *user ID* に変更します (*ID* は `/etc/passwd` ファイル内の有効なログイン ID でなければなりません)。このオプションはスーパーユーザーだけが使用できます。

`-s`
: 各ハーフワード毎にバイトを交換します。

`-S`
: 各ワード毎にハーフワードを交換します。

`-t`
: 入力の内容を表示します。入力したファイルに拡張属性が含まれている場合は、それらの属性も表示します。ファイルは生成されません。`-t` と `-V` は同時には使用できません。

`-u`
: 無条件にコピーを実行します。通常、古いファイルは同一名の新しいファイルを置換しません。

`-v`
: 詳細表示。ファイル名のリストと拡張属性の名前を出力します。`-t` オプションと一緒に使用すると `ls -l` コマンドの出力のようになります (`ls`(1) 参照)。

`-V`
: 詳細表示。入力または出力した各ファイルの内容を完全に表示します。すべてのファイル名を表示せずに `cpio` が動作していることをユーザーが確認するときに便利です。

`-6`
: UNIX System Sixth Edition 形式のアーカイブ形式・ファイルを処理します。`-i` オプションとのみ使用できます。このオプションは、`-c` および `-H` オプションと同時には使用できません。

`-@`
: 拡張属性をアーカイブに含めます。デフォルトでは、`cpio` は拡張属性をアーカイブに含めません。このフラグを指定すると `cpio`

は、アーカイブ内のファイルに拡張属性が存在するかどうかを検査し、存在する場合は、その拡張属性を通常のファイルと同様にアーカイブに含めます。拡張属性ファイルは、特殊なファイルタイプを持つ特殊ファイルとしてアーカイブに収められます。`-@` フラグをオプション `-i` または `-p` と一緒に使用すると、`cpio` は、拡張属性のデータを通常のファイルデータと一緒に復元します。拡張属性ファイルの抽出は、通常のファイル抽出の一部としてのみ実行できます。属性レコードだけを抽出しようとしても無視されます。

オペランド　以下のオペランドを指定できます。

directory　`cpio -p` の対象となる既存ディレクトリのパス名。

pattern　パターンマッチング用の表現方法を利用した式。これはシェルがファイル名のパターンマッチングに利用する式 (`sh`(1) を参照) や正規表現に似たものです。以下に示すメタキャラクタが定義されています。

`*`　空の文字列を含み、あらゆる文字列と一致します。

`?`　あらゆる 1 文字と一致します。

`[ . . . ]`　カッコ内のいずれかの文字と一致します。2 つの文字を – で区切って指定すると、システムのデフォルトの照合シーケンスに 従ってその 2 文字の間のすべての文字 (2 文字自身も含む) と一致します。なお最初のカッコ `[` の直後の文字が `!` のとき、結果は保証されません。

`!`　`!` (感嘆符) は否定を表します。たとえば `!abc*` と指定すると、文字列 `abc` で始まるファイル名とは一致しません。 *pattern* 指定において、メタキャラクタ `?`、 `*`、`[...]` はスラッシュ (/) と一致し、バックスラッシュ (\) はエスケープと一致します。複数 *pattern* も指定可能で、 *pattern* が何も指定されなければデフォルト値として `*` (すなわち、すべてのファイルを選択する) が採用されます。各パターンは二重引用符で囲む必要があります。

そうしないと、現在のディレクトリ内のファイルが使用されることがあります。

使用法　ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `cpio` の動作については、`largefile`(5) を参照してください。

使用例　以下に `cpio` の使用例を示します。

例 **1** 標準入力を使用する

```
example% ls | cpio -oc > ../newfile
```

例 1 標準入力を使用する　（続き）

上記の例のように標準入力をパイプ経由で cpio -o に渡すと、ファイルがグループ化されて 1 つのアーカイブファイル (../newfile) にまとめられます。-c オプションは (-H オプションと同様に)、アーカイブファイルの他のマシンとの互換性を保証します。ファイル名のリストをパイプ経由で cpio に渡すには、 ls(1) の代わりに find(1)、 echo(1)、 cat(1) などを使用できます。ファイルではなくデバイスへ出力先を変更することもできます。

例 2 ディレクトリにファイルを抽出する

```
example% cat newfile | cpio -icd "memo/a1" "memo/b*"
```

この例では、cpio -i は cpio -o の出力を使用して (cat を使用してパイプに渡している)、パターン (memo/a1、memo/b*) に一致するファイルを抽出します。さらに必要に応じて現在のディレクトリ下にディレクトリを作成し (-d オプション)、適切なディレクトリにファイルを格納します。-c オプションは互換性のあるヘッダーで入力ファイルが作成されているときに使用します。何もパターンを指定しないと、newfile 内のすべてのファイルをディレクトリに格納します。

例 3 別のディレクトリにファイルをコピーまたはリンクする

```
example% find . -depth -print | cpio -pdlmv newdir
```

この例では、cpio -p はパイプ経由でファイル名を読み込んで、それらのファイルを別のディレクトリ (newdir) へコピーまたはリンク (-l オプション) します。-d オプションは必要に応じてディレクトリを作成します。-m オプションは変更時間を保持します (cpio に渡すパス名を作成するには find(1) の -depth オプションを使用してください。これによって読み取りのみが許可されたディレクトリ下にファイルを作成しようとするときに発生する問題を排除できます)。コピー先のディレクトリ newdir が存在している必要があります。

find と一緒に cpio を使用する場合、cpio のオプションに -L を使用しているときは find のオプションに -follow を (逆の場合も同様) 使用してください。そうしない場合の結果は保証されません。

マルチリールアーカイブに対しては、古いボリュームのマウントを解除し、新しいボリュームをマウントし、さらに次の装置名を入力して (通常、最初のリールと同じ) 次のテープに引き継いでください。RETURN キーを押すと cpio が終了します。

環境

cpio の実行に影響を与える環境変数 LC_COLLATE、 LC_CTYPE、 LC_MESSAGES、LC_TIME、 TZ、 NLSPATH についての詳細は、 environ(5) を参照してください。

TMPDIR　　cpio の一時ファイルは、デフォルトでは /var/tmp に作成されます。デフォルト以外の場合は、TMPDIR で指定したディレクトリに作成されます。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み
インタフェース安定性	安定

関連項目　ar(1), cat(1), echo(1), find(1), ls(1), setfacl(1), sh(1), tar(1), vold(1M), archives(4), attributes(5), environ(5), fsattr(5), largefile(5), standards(5)

注意事項　cpio アーカイブで使用できるパス名の最大長は、そのアーカイブに含まれているヘッダータイプによって決まります。以下の表に、サポートされているヘッダータイプと、そのヘッダータイプで許可されている最大パス長を示します。

ヘッダータイプ	コマンド行オプション	最大パス長
BINARY	“-o”	256
POSIX	“-oH odc”	256
ASCII	“-oc”	1023
CRC	“-oH crc”	1023
USTAR	“-oH ustar”	255

コマンド行オプション“-o -H tar” を指定した場合は、作成されるアーカイブのタイプは USTAR になります。つまり、このアーカイブをコマンド行オプション “-i -H tar” を使用して読み込むとエラーが発生します。このアーカイブの読み込みには、コマンド行オプション “-i -H ustar” を使用してください。"-i -H tar" オプションは、古い tar アーカイブ形式を読み込むためのものです。

選択されたヘッダー形式に対してユーザー ID またはグループ ID が大きすぎるファイルについてはエラーメッセージが出力されます。ユーザー ID またはグループ ID のすべての値をサポートできるアーカイブを作成するには、-H crc あるいは -c を使用してください。

スーパーユーザーだけが特殊ファイルをコピーできます。

512 バイトを 1 ブロックとみなします。

ファイルのアクセス権が 000 で、なんらかの文字データを持ち、さらにユーザーが root でない場合、そのファイルは保管または復元されません。

ヘッダーに書かれている i ノード番号 (/usr/include/archives.h) は unsigned short (2 バイト) です。このため、この i ノード番号は 0 から 65535 までの値となります。ハードリンクされたファイルはこの i ノード番号の範囲には入りません。これはベンダーが異なるマシン間で cpio アーカイブを移動する場合に問題となります。

ボリューム管理デーモンが稼働している場合には、 /dev/rdiskette などの通常のデバイス名を使ってフロッピー装置にアクセスできないことがあります。詳しくは vold(1M) を参照してください。

テープからハードディスクへファイルを 取り出したりコピーしたりする場合は、ハードディスクからテープへコピーした時と同じブロック化因数を使用してください。したがって、-B オプションまたは -C オプションを指定してください。

-p および -o の処理中は、標準入力上のファイルリストは、cpio によって一時ファイルに保存されます。

名前 | cpp – C 言語プリプロセッサ

形式 | **/usr/lib/cpp** [-BCHMpPRT] [-undef] [-D*name*] [-D*name* = *def*]
[-I*directory*] [-U*name*] [-Y*directory*] [*input-file* [*output-file*]]

機能説明 | cpp は、C 言語プリプロセッサです。cpp は、cc(1B) コマンドで開始される C プログラムのコンパイルにおける最初のパスとして起動されます。ただし、cpp は、他の Sun コンパイラの第 1 パスプリプロセッサとしても使用できます。

cpp はマクロプロセッサとしても使用できますが、その出力がコンパイラの第 2 パスへの入力として使用できる形式となっているため、一般にマクロプロセッサとしての使用はお勧めできません。したがって、cpp を起動するときは、cc(1B) コマンドか、または他のコンパイルコマンドを使用するようにしてください。汎用のマクロ処理については、m4(1) のマニュアルページを参照してください。

cpp は、オプションとして 2 つのファイル名を引数として受け付けます。*input-file* および *output-file* は、それぞれ、プリプロセッサ用の入力ファイルと出力ファイルです。デフォルトでは、標準入力と標準出力が使用されます。

オプション | 次のオプションを指定できます。

`-B` C++ の注釈インジケータ / / を使用できるようにします。このインジケータを使用すると、/ / の後の行は注釈として扱われます。

`-C` すべての注釈 (cpp の指令行にあるものは除く) をそのまま渡します。デフォルトでは、cpp は C 言語スタイルの注釈を削除します。

`-H` インクルードファイルのパス名を、1 行に 1 つずつ標準エラー出力に出力します。

`-M` メイクファイル依存関係のリストを生成して、標準出力に書き込みます。このリストは、入力ファイルから生成されるオブジェクトファイルが、入力ファイルだけでなく、参照されるインクルードファイルにも依存することを示します。

`-p` 最初の 8 文字だけを使用してプリプロセッサシンボルを識別し、指令を含む行の終わりに余分なトークンがあれば警告を発行します。

`-P` C コンパイラの次のパスで使用される行制御情報を生成せずに、入力を前処理します。

`-R` 再帰マクロを可能にします。

`-T` 最初の 8 文字だけを使用して、異なるプリプロセッサシンボルを識別します。このオプションは、常に最初の 8 文字だけを使用するシステムとの下方互換性のために提供されています。

`-undef` 事前定義されているすべてのシンボル初期定義を削除します。

-D*name*

name を 1と定義します。これは、cpp コマンド行に -D*name*=1 オプションを指定した場合と同じです。

```
#define name 1
```

または、上記の行が cpp の処理するソースファイル内にある場合と同じです。

-D*name*=*def*

#define 指令と同様に *name* を定義します。これは、次の行が cpp の処理するソースファイル内にある場合と同じです。

```
#define name def
```

-D オプションは -U オプションよりも優先順位が低くなります。つまり、-U オプションと -D オプションの両方で同じ名前を使用した場合は、オプションの順序に関係なくその名前は定義されません。

-I*directory*

/ 以外で始まる名前を持つ #include ファイルの検索パスに *directory* を挿入します。*directory* は、標準の include ディレクトリリストの前に挿入されます。したがって、名前が二重引用符 (") で囲まれた #include ファイルは、まず #include 行を含むファイルのあるディレクトリで検索され、次に -I オプションで指定されたディレクトリで検索されて、最後に標準リストのディレクトリで検索されます。名前が山括弧 (< >) で囲まれた #include ファイルの場合、#include 行を含むファイルのあるディレクトリは検索されません。この検索順序の詳細については、以下の「詳細」を参照してください。

-U*name*

name の初期定義をすべて削除します。ここで、*name* は個々のプリプロセッサによって事前定義されたシンボルです。以下に、システムのアーキテクチャに応じて事前定義されているシンボルリストの一部を示します。

オペレーティングシステム：	ibm、 gcos、os、 tss、 unix
ハードウェア：	interdata、 pdp11、 u370、 u3b、 u3b2、 u3b5、u3b15、 u3b20d、 vax、ns32000、 iAPX286、 i386、 sparc、 sun
UNIX システムからの派生：	RESおよび RT
lint コマンド：	lint

シンボル sun、sparc、 unix は、すべての Sun システムで定義されています。

-Y*directory*

#include ファイルを検索する場合は、標準のディレクトリリストの代わりに、ディレクトリ *directory* を使用します。

指令

すべての cpp 指令行は、ハッシュシンボル (#) で始まります。適切なインデンテーションを行うために、空白 (SPACE または TAB 文字) を # の後に入れることができます。

`#define` *name token-string*
これ以降、*name* を *token-string* で置き換えます。

`#define` *name*`(`*argument* [`,` *argument*] . . . `)` *token-string*
name と `(` の間に空白を入れることはできません。括弧で囲んだ引数のリストが後に続く、以降の *name* を *token-string* で置き換えします。ここで、*token-string* 中の各 *argument* は、カンマで区切られたリスト内の対応するトークンで置き換えられます。引数を持つマクロが展開されると、引数は展開された *token-string* にそのまま入れられます。*token-string* 全体が展開されると、cpp は、新たに作成された *token-string* の先頭から、展開すべき名前の検索を再開します。

`#undef` *name*
シンボル *name* の定義をすべて削除します。*name* の後の指令行には、トークンを付加することはできません。

`#include "`*filename*`"`
`#include <`*filename*`>`
この位置に *filename* の内容を読み込みます。このデータは、現在のファイルの一部であるかのように cpp によって処理されます。<*filename*> の表記法を使用すると、*filename* は標準の `include` ディレクトリのみで検索されます。詳細については、上記の `-I` および `-Y` オプションを参照してください。最後の '`"`' または '`>`' の後の指令行にはトークンを付加することはできません。

`#line` *integer-constant* `"`*filename*`"`
C コンパイラの次のパスのための行制御情報を生成します。*integer-constant* は次の行の行番号として解釈され、*filename* はその行を含むのファイルとして解釈されます。*filename* が指定されていない場合、現在のファイル名は変更されません。オプションの *filename* の後の指令行には、トークンを付加することはできません。

`#if` *constant-expression*
対応する `#else`、`#elif`、または `#endif` 指令までの後続行は、*constant-expression* が `0` 以外の値に評価された場合にのみ出力されます。`&&`、`||`、および `,` を含む、C 言語における代入以外のすべての 2 項演算子を、*constant-expression* 中で使用できます。?: 演算子と単項演算子 – と `!`、および ~ も *constant-expression* 中で使用できます。

これらの演算子の優先順位は、C における優先順位と同じです。また、単項演算子 `defined` は、2 つの形式で、*constant-expression* 中で使用できます。つまり、`defined (` *name* `)` または `defined` *name* の形式です。これにより、`#ifdef` および `#ifndef` の各ディレクトリ (下記参照) は `#if` 指令で有効になります。cpp によって認識されている演算子、整数定数、および名前だけを *constant-expression* 内で使用する必要があります。特に、`sizeof` 演算子は使用できません。

`#ifdef` *name*
対応 `#else`、`#elif`、または `#endif` までの後続行は、*name* が`#define` 指令または -D オプションのいずれかによって定義されており、*name* が `#undef` 指令の対象となっていない場合にのみ出力されます。指令行上の *name* の後に付加されたトークンは無視されます。

`#ifndef` *name*
対応する `#else`、`#elif`、または `#endif` までの後続行は、*name* が定義されていないか、あるいは、その定義が `#undef` 指令の対象となっていない場合にのみ出力されます。*name* の後の指令行には、トークンを付加することはできません。

`#elif` *constant-expression*
`#if`、`#ifdef`、または `#ifndef` 指令と、対応する `#else` または `#endif` 指令の間には、任意の数の `#elif` 指令を置くことができます。`#elif` 指令に続く行は、次の条件がすべてあてはまる場合にのみ出力されます。

- 先行する `#if` 指令で *constant-expression* が `0` と評価され、先行する `#ifdef` で *name* が定義されていない、あるいは先行する `#ifndef` 指令で *name* が定義されていた。
- すべての `#elif` 指令の対象となっている *constant-expression* が `0` に評価された。
- 現在の *constant-expression* が `0` 以外に評価された。

constant-expression が `0` 以外に評価された場合、後続の `#elif` と `#else` 指令は、対応する `#endif` まで無視されます。`#if` 指令で使用できる *constant-expression* はすべて、#elif 指令でも使用できます。

`#else`
これは、条件指令の意味を反転します。つまり逆の意味になります。先行する条件指令が、行を含めることを示す場合、`#else` と対応する `#endif` の間の行は無視されます。先行する条件指令が、行を無視することを示す場合、後続の行は出力に含まれます。条件指令と対応する `#else` 指令はネストできます。

`#endif`
条件指令 `#if`、`#ifdef`、または `#ifndef` のいずれかによって開始した行セクションを終了します。このような各指令には、対応する `#endif` が必要です。

マクロ

マクロの正規パラメータは、文字定数や引用文字列内で発生する場合でも、`#define` 指令の本体で認識されます。たとえば、次の指令を見てください。

```
#define abc(a)| `|a|
abc(xyz)
```

上記の指令の出力は次のとおりです。

```
# 1 ""
| `|xyz |
```

2 番目の行は復帰改行です。最後の 7 文字は "| ヽ|xyz |" (縦線、逆引用符、縦線、x、y、z、縦線) です。マクロ名は、通常の走査中は、文字定数や引用文字列内では認識されません。したがって、次の指令は、2 番目の行で abc を展開しません。これは、abc が、#define マクロ定義の一部ではない引用文字列内にあるためです。

```
#define abc xyz
printf("abc");
```

マクロは、#define または #undef の処理中には展開されません。したがって、次の指令は abc を作成します。#ifdef または #ifndef のすぐ後にあるトークンは展開されません。

```
#define abc zingo
#define xyz abc
#undef abc
xyz
```

マクロは、別のマクロ呼び出しへの実際のパラメータを判別する走査中は展開されません。したがって、次の指令は、「 #define hello goodbye hello」を生成します。

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

出力

出力は入力ファイルのコピーからなり、変更が加えられており、次の形式の行が追加されています。

#*lineno* " *filename* " "*level* "

後続のの出力行の元のソース行番号とファイル名、および、これがインクルードファイルを入力した後の最初のそのような行であるか(レベル 1)、インクルードファイルが終了した後の最初のそのような行であるか (レベル 2)、あるいはそれ以外のそのような行であるか (*level* は空) を示します。

ディレクトリの検索順序

#include ファイルは、次の順序で検索されます。

1. #include 要求を含むファイルのディレクトリ (つまり #include は、要求が行われたときに、検索されているファイルに対応します)。
2. -I オプションで指定された複数のディレクトリ。左から右の順に検索されます。
3. 標準ディレクトリ (UNIX システム上の /usr/include)

特殊名

cpp は、2 つ特別な名前を認識します。名前__ _LINE_ __は、cpp が認識する現在の行番号 (10 進整数) として定義されます。__ _FILE_ __は、cpp が認識する現在のファイル名 (C 文字列) として定義されます。これらは、他の定義済み名と同様に、任意の場所 (マクロ内を含む) で使用できます。

改行文字

NEWLINE 文字は、文字定数または引用文字列を終了します。エスケープされた NEWLINE 文字 (つまり、バックスラッシュが先行する NEWLINE) を `#define` 文の本体で使用すると、定義を次の行に続けることができます。エスケープされた NEWLINE は、マクロ値には含まれません。

注釈

注釈は削除されます (コマンド行に `-C` オプションが指定されている場合を除く)。また、注釈がトークンを終了する場合をのぞき注釈は、無視されます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`0` 以外 エラーが発生した

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWsprot

関連項目

`cc`(1B), `m4`(1), `attributes`(5)

診断

cpp によって生成されるエラーメッセージは、読めばわかるものです。エラーが発生した行番号とファイル名が、診断と共に出力されます。

注意事項

NEWLINE 文字が、展開されるマクロの引数リストで検出された場合、以前のバージョンの cpp の中には、NEWLINE 文字が検出されて展開された場合のように、NEWLINE文字を出力するものがあります。現在のバージョンの cpp は、NEWLINE文字を SPACE 文字で置き換えします。

インクルードファイルの標準ディレクトリは環境によって異なる場合があるため、次の形式の `#include` 指令を使用する必要があります。

```
#include <file.h>
```

次のような絶対パスを使用する `#include` 指令は使用しないようにしてください。

```
#include "/usr/include/file.h"
```

cpp は、絶対パス名が使用されていれば警告を出します。

コンパイラでは 8 ビットの文字列と注釈を使用できますが、コンパイラ以外では 8 ビットは使用できません。

名前 | crontab – ユーザーの crontab ファイル

形式 | **crontab** [*filename*]

crontab [-elr] *username*

機能説明 | crontab ユーティリティは、crontab ファイルをコピー、作成、表示、削除することにより、cron (cron(1M) を参照) を使ってユーザーのアクセスを管理します。 オプションなしで crontab を実行すると、 指定したファイル (ファイルを指定しない場合は標準入力) を、 全ユーザーの crontab が登録されているディレクトリにコピーします。

crontab にファイル名 (*filename*) を指定して実行すると、コマンドを実行したユーザーの crontab エントリが上書きされます。

アクセス制御 | 次の場合、ユーザーは crontab にアクセスできます。

- /etc/cron.d/cron.allow に自分の名前が登録されている場合
- /etc/cron.d/cron.allow が存在せず、 /etc/cron.d/cron.deny に自分の名前が登録されていない場合

次の場合、ユーザーは crontab にアクセスできません。

- /etc/cron.d/cron.allow が存在し、ファイル内に自分の名前が登録されていない場合
- /etc/cron.d/cron.allow が存在せず、 /etc/cron.d/cron.deny に自分の名前が登録されている場合
- /etc/cron.d/cron.allow も /etc/cron.d/cron.deny も存在しない場合

  この場合、ジョブを送ることができるのは、solaris.jobs.user の承認を受けているユーザーだけです。
- BSM 監査が有効な場合、ユーザーのシェルは監査されず、ユーザーは crontab の所有者になりません。これは、ユーザーが SSH のいずれかのバージョンなど、監査パラメータを設定しないプログラムを介してログインする場合に起こります。

なお allow または deny の規則が root に適用されるのは、 allow または deny ファイルが存在している場合だけです。

allow と deny の両ファイルは、いずれも 1 行に 1 ユーザー名が記述される形式になっています。

入力形式 | crontab ファイル内の各行は、6 つのフィールドで構成されています。各フィールドの間は、 空白文字またはタブで区切られています。 最初の 5 つのフィールドは、次の内容を指定する整数パターンです。

```
分 (0-59)
時 (0-23)
日 (1-31)
月 (1-12)
曜日 (0-6、0 は日曜日)
```

整数値の代わりに、アスタリスク (有効な値全部を表す) や、コンマで区切った形式の要素リストを指定することもできます。要素は、数値 1 個で指定しますが、ある数値からある数値までの範囲を指定する場合は、数値 2 個をマイナス記号 (–) で区切って指定します。日付は、2 つのフィールド (日および曜日) で指定できます。どちらのフィールドも、要素のリストとして指定された場合に付加されます。詳しくは「使用例」を参照してください。

crontab ファイル内の行の 6 番目のフィールドは、 指定した時間にシェルによって実行されるコマンド文字列です。このフィールド内の % 文字 (\ によってエスケープされているものは除く) は、復帰改行 (NEWLINE) 文字に変換されます。

シェルによって実行されるのは、このコマンドフィールドの第 1 行 (' % ' または行の終わりまで) のみです。 その他の行は、標準入力として実行されます。空行、または ' # ' で始まる行はコメントとみなされ、無視されます。

シェルは、sh の arg0 によって、 ユーザーの $HOME ディレクトリから呼び出されます。自分の .profile を実行したいユーザーは、 crontab ファイルの中で明示的に指定する必要があります。cron は、ホーム、LOGNAME、SHELL (= /bin/sh)、TZ、パスを定義し、すべてのシェルのデフォルト環境を指定します。ユーザーの cron ジョブのデフォルトパスは /usr/bin で、 root の cron ジョブ の デフォルト パスは /usr/sbin:/usr/bin です。 デフォルトパスは /etc/default/cron 中に設定できます (cron(1M) を参照)。

コマンドの標準出力や標準エラー出力をリダイレクトするように指定していないと、生成されたすべての出力またはエラーが、ユーザーにメールで通知されることになります。

オプション　次のオプションを使用できます。

-e　現在のユーザーの crontab ファイルをコピーし、その内容を編集します。 このファイルが存在しない場合は、空の crontab ファイルを作成して編集します。 編集が終了すると、このファイルがユーザーの crontab ファイルとしてインストールされます。 *username* が指定された場合、 現在のユーザーの crontab ファイルではなく、指定のユーザーの crontab ファイルを編集します。 このようにユーザー名を指定できるのは、solaris.jobs.user の承認を受けているユーザーだけです。 -e オプションを指定してファイルを編集するときに起動するエディタは、環境変数 EDITOR によって決まります。 デフォルトエディタは、 ed(1) です。crontab のジョブを登録するときは、必ず crontab を使用してください。crontab ファイルを直接編集してジョブを追加してはなりません。cron はこの方法による変更を認識しません。

crontab ファイル内のすべての行を削除すると、古い crontab ファイルが復元されます。すべての行を削除するには、-r オプションを使用して crontab ファイルを削除するようにします。

-l　crontab を起動したユーザーの crontab ファイルの内容を表示します。-r または -l オプションの後にユーザー名 (*username*) を指定して、指定のユーザーの crontab ファイルを削除または表示できるのは、solaris.jobs.user の承認を受けているユーザーだけです。

`-r` crontab ディレクトリからユーザーの crontab を削除します。

使用例

例 1 core ファイルを削除する

平日 (月－金) の午前 3 時 15 分に core ファイルを削除する例です。

```
15 3 * * 1-5 find $HOME -name core 2>/dev/null | xargs rm -f
```

例 2 誕生日のお祝いを送る

誕生日のお祝いメールを送ります。

```
0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.
```

例 3 日付と曜日を同時に指定する

日付と曜日を同時に指定します。

```
0 0 1,15 * 1
```

この例では、毎月 1 日と 15 日、さらに毎週月曜日にコマンドが実行されます。曜日だけ、もしくは日付だけを指定する場合には、もう一方のフィールドに * を指定します。次の例を参照してください。

```
0 0 * * 1
```

毎週月曜日にコマンドが実行されます。

環境

crontab の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、 NLSPATH の詳細については、 environ(5) を参照してください。

`EDITOR` -e オプションを指定したときに呼び出すエディタを指定します。デフォルトのエディタは ed(1) です。 EDITOR 環境変数と VISUAL 環境変数の両方が設定された場合、 優先されるのは VISUAL 環境変数の値です。

終了ステータス

以下の終了ステータスが返されます。

`0` 正常終了

`>0` エラーが発生した

ファイル

/etc/cron.d	主 cron ディレクトリ
/etc/cron.d/cron.allow	許可されているユーザーのリスト
/etc/default/cron	cron のデフォルト設定を含む
/etc/cron.d/cron.deny	許可されていないユーザーのリスト
/var/cron/log	アカウンティング情報
/var/spool/cron/crontabs	crontab のスプール空間

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

atq(1)、atrm(1)、auths(1)、ed(1)、sh(1)、cron(1M)、su(1M)、auth_attr(4)、attributes(5)、environ(5)

注意事項

誤って引数なしで crontab コマンドを実行してしまった場合、CTRL–D を使って処理を中止しないでください。 CTRL–D を使用すると、 crontab ファイルからすべてのエントリが 削除されてしまいます。 処理の中止には、CTRL–C を使用してください。

承認されたユーザーが他のユーザーの crontab ファイルを編集する場合は、これによってどのような影響が生じるかを予測できません。このような事態を防ぐため、su(1M) コマンドを実行してスーパーユーザーになり、 crontab ファイルのユーザーとしてログインしてから編集作業に取りかかってください。

cron を更新するときには、まず、既存の crontab を調べて、更新時近くにイベントがスケジュールされているかどうかをチェックします。イベントが予定されていた時刻より後に更新プロセスが完了すると、当該エントリが失われる可能性があります。この現象は次のように発生します。ユーザーの crontab ファイルの内部ビューを更新するように crontab から通知されると、cron はまず、既存の crontab ファイルの内部ビューとスケジュールされている内部イベントを削除します。次に、新しい crontab ファイルを読み込んで、crontab の内部ビューと内部イベントを構築し直します。この最後の段階には (特に、crontab ファイルが大きいときには) 時間がかかります。既存の crontab エントリが更新時に極めて近くに予定されていると、予定された時刻の後に最後の段階が完了することがあり得ます。安全を期すために、新しいジョブは少なくとも現在の日付と時刻の 60 秒後に起動するようにしてください。

名前 | crypt – ファイルの暗号化または復号化

形式 | **crypt** [*password*]

機能説明 | crypt ユーティリティは、ファイルの内容を暗号化および復号化します。crypt は標準入力から読み取って、標準出力に書き込みます。*password* は、特定の変換を選択する鍵です。*password* を指定しないと、crypt は端末から鍵の入力を求め、鍵が入力される間、出力を停止します。crypt は、暗号化と復号化を行います。

```
example% crypt key<clear.file> encrypted.file
example% crypt key<encrypted.file | pr
```

上記の例は、clear.*file* の内容を出力します。

crypt によって暗号化されたファイルは、暗号化モードでエディタ ed(1)、ex(1)、および vi(1) が処理するファイルと互換性があります。

暗号化されたファイルのセキュリティは、3 つの要因によって決まります。第 1 に基本方式が解決困難なものでなければなりません。第 2 に鍵空間の直接検索が不可能でなければなりません。第 3 に鍵または暗号化されていないテキストが表示される可能性を最小限に抑えなければなりません。

crypt は、German Enigma の考え方に沿って設計された 1 ロータマシンを実装していますが、256 要素のロータを備えています。このようなマシンに対する攻撃方法は広く知られているため、crypt の提供するセキュリティは最低限必要なレベルにすぎません。

1 つの鍵をマシンの内部設定に変換する処理には意図的に大きな負荷がかかるように設計されており、計算には1 秒の数分の 1 を要します。ただし、鍵がたとえば 3 文字の小文字に制限されている場合、暗号化されたファイルは、5 分弱のマシン時間を費やすだけで読み取ることができます。

鍵は crypt コマンドへの引数であるため、ps(1) またはその派生型コマンドを実行することによって、鍵を表示することが可能です。この可能性を最小限に抑えるため、crypt は入力と同時にすべての鍵レコードを破棄します。鍵と鍵セキュリティの選択が、crypt のセキュリティ攻撃に対する強度を左右すると言えます。

ファイル | /dev/tty　　入力される鍵用

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | des(1), ed(1), ex(1), makekey(1), ps(1), vi(1), attributes(5)

名前 | csh – C に似た構文を持つシェル・コマンドインタプリタ

形式 | **csh** [-bcefinstvVxX] [*argument*.....]

機能説明 | csh は、C シェルの略で、C 言語に似た構文を持つ コマンドインタプリタです。Bourne シェルにはない多くの便利な 対話型の機能を備えています。たとえば、ファイル名の補完、コマンド別名指定、履歴置換、ジョブ制御、多くの組み込みコマンドなどの機能です。Bourne シェルと同様に、C シェルは変数、コマンド、およびファイル名置換機能を提供します。

初期化と終了 | C シェルを起動すると、まず通常ホームディレクトリにある .cshrc が読み取り可能であるか、当該ユーザーがその所有者であるか、または当該ユーザーのグループ ID がそのグループ ID と一致するかチェックし、一致していれば .cshrc ファイルを実行します。もし、C シェルを '-' で始まる名前で起動すると、そのシェルはログインシェルとして実行されます (login(1) を起動するときの動作と同じ)。

シェルがログインシェルであった場合、実行は連続して行われます。最初に /etc/.login 中のコマンドが実行されます。次に ユーザーのホームディレクトリにある .cshrc ファイルのコマンドが実行されます。その後、シェルはユーザーのホームディレクトリにある .login ファイルのコマンドを実行します。このとき、このファイルにも .cshrc と同様のアクセス権チェックが実施されます。通常 .login ファイルには、端末の種類や環境を指定する コマンドが入っています。(ファイルのインタプリタの説明については以下の「コマンドの実行」と exec(2) を参照。)

ログインシェルが終了したとき、シェルはユーザーのホームディレクトリにある .logout ファイルのコマンドを実行します。このとき、このファイルにも .cshrc と同様のアクセス権チェックが実施されます。

対話型動作 | 起動処理が終了すると、C シェルはプロンプト hostname% (特権ユーザーなら hostname#) を表示して、端末からコマンドの読み取りを開始します。以降、シェルは以下の処理を繰り返します。まず入力コマンドを 1 行読み取り、複数のワードに分解します。次にこれらのワードを履歴に記憶し、構文解析します (下記の「使用法」を参照)。最後に、現在の行にある各コマンドを実行します。

非対話型動作 | 非対話形式で動作しているとき、シェルは端末からの入力を促すプロンプトを表示しません。非対話形式の C シェルは、コマンド行で引数として与えられたコマンドを実行するか、スクリプトと呼ばれるファイルからコマンドを読み取り、実行します。

オプション | 以下のオプションを指定できます。

-b　オプションの処理を強制的に中断します。以降のコマンド行引数は、C シェルのオプションとしては解釈されません。これによって、オプションを明確にスクリプトに渡すことができます。このオプションが指定されていなければ、シェルは set-user-ID または set-group-ID スクリプトを実行しません。

-c　最初の引数 (必ず指定する) を実行します。残りの引数は、引数リスト変数 argv に格納され、csh に直接渡されます。

-e　コマンドが異常終了したり、0 以外の終了状態を返したとき、処理を終了します。

`-f`	高速起動。開始時に、`.cshrc` ファイルおよび `.login` ファイル (ログインシェルのとき) を読みません。
`-i`	強制的に対話型にします。標準入力が端末 (文字特殊装置) でなくても、コマンド行入力を促すプロンプトを表示します。
`-n`	構文解析はしますが、コマンドは実行しません。このオプションは、C シェルスクリプトの構文エラーを チェックするのに使用できます。
`-s`	標準入力からコマンドを受け取ります。
`-t`	コマンド行を 1 行読み取って実行します。後続の入力行へコマンド行が継続することを示すには、復帰改行 (NEWLINE) を '\' (バックスラッシュ) でエスケープします。
`-v`	冗長表示。定義済み変数 `verbose` を設定します。コマンド入力のエコーを、履歴置換後 (他の置換の前) かつ実行前に行います。
`-V`	`.cshrc` を読み取る前に `verbose` 変数を設定します。
`-x`	エコー。`echo` 変数を設定します。すべての置換後、実行の前にコマンドをエコーします。
`-X`	`.cshrc` を読み取る前に `echo` 変数を設定します。

`-c`、`-i`、`-s`、および `-t` の各オプションを除き、最初の必須 *argument* はコマンド名またはスクリプト名とみなされます。そして 0 番目の引数として渡され、後続の引数はそのコマンドまたは スクリプトの引数リストへ追加されます。

ファイル名の補完

`filec` 変数を設定することによって、対話型 C シェルは、部分的に入力されたファイル名またはユーザー名を補完することができます。ファイル名を部分的にあいまいに入力して ESC 文字を入力すると、シェルは 作業用ディレクトリから一致するファイルを検索して、残りの文字を埋めます。

ファイル名を部分的に入力して EOF 文字 (通常は CTRL-D) を入力すると、シェルは一致するすべてのファイル名を一覧表示します。そして、入力された不完全なコマンド行の内容を表示して、再度プロンプトを表示します。

最後のワード (入力値の一部) がチルド (~) で始まる場合、シェルは現在のディレクトリのファイル名ではなく、ユーザー名を補完しようとします。

端末は、ビープ音を鳴らすことによって、エラーまたは複数のファイル名が一致したことを知らせます。この音は、`nobeep` 変数を設定することによって抑止できます。`fignore` 変数に特定の接尾辞のリストを設定しておけば、その接尾辞を持つファイルを除外することができます。ただし、その接尾辞を持つファイル名以外に 一致するものが見つからなければ、この変数は無視されます。なお EOF 文字によるファイル名のリスト出力には、`fignore` が示す除外対象ファイル名も含まれます。

字句構造

シェルは、入力行を空白文字およびタブでワードに分解します。ただし、以下に説明する例外を除きます。`&`、`|`、`;` 、`<` 、`>` 、`(` 、および `)` の各文字は、それぞれ別個のワードを形成します。ただし対で使用すると、1 対で 1 つのワードを形成しま

す。これらのシェルメタキャラクタは、他のワードの一部になることもできます。その場合、'\' (バックスラッシュ) を前に付けると、メタキャラクタの持つ特別な意味を抑制できます。\ の付いた復帰改行文字は、空白文字と同じです。

さらに、単一引用符 (') 、二重引用符 (") 、または逆引用符 (` ) で囲まれた文字列は、ワードの一部を生成します。このような文字列の中に含まれる空白文字またはタブを含めた メタキャラクタは、別個のワードを形成しません。逆引用符 ( `) または二重引用符 (") の対で囲まれた場合、\ (バックスラッシュ) の付いた復帰改行は 本当の意味での復帰改行文字です。各引用符のその他の機能については、後述の 「変数置換」、「コマンド置換」、「ファイル名置換」 の各項で説明しています。

シェルの入力が端末ではない場合、# 文字は、入力行が最後までコメント行であることを表します。\ が前に付いたり、一対の引用符で囲まれると、その特別な意味は抑制されます。

コマンド行構文解析

単純コマンド (*simple command*) は、一連のワードで構成されます。最初のワード (リダイレクションの一部ではない) は、実行するコマンドを指定します。1 つの単純コマンド、または、| や |& 文字で区切られた一連の単純コマンド群は、パイプライン (*pipeline*) を形成します。| を使用すると、直前のコマンドの標準出力が、それに続くコマンドの標準入力に変更 (リダイレクション) されます。| & を使用すると、標準エラー出力および標準出力の両方の出力先が、パイプラインを介してリダイレクションされます。

パイプラインはセミコロン (;) で区切ることができます。この場合、それらは順次実行されます。&& または | | で区切られたパイプラインは、条件に従って実行されます。つまり左側のパイプラインの実行が成功するか失敗するかによって、右側のパイプラインが実行されるかどうかが決まります。

1 つのパイプラインまたはパイプラインの並びを、括弧 '()' で囲むことができます。囲まれた全体が単純コマンドとなり、他のパイプラインまたはパイプラインの並びの構成要素となることができます。

パイプラインの並びは、'&' を付けることによって非同期的にもしくは 「バックグラウンドで」実行できます。この場合シェルは、プロンプトの出力をパイプラインの並びが終了するまで待ちません。ただちにジョブ番号 (後述の「ジョブ制御」を参照) と関連するプロセス ID を表示してから プロンプトを表示します。

履歴置換

履歴置換を使用すれば、以前入力したコマンド行のワードを、これから入力するコマンド行で使用できます。これにより、綴りの訂正、複雑なコマンドや引数の繰り返し入力が簡単になります。コマンド行は履歴リストに保存されます。履歴リストのサイズは history 変数によって変更できます。どのような場合でも、最後に入力されたコマンドは必ず保持されます。履歴置換の指定は ! で始まり (histchars 変数により他の文字に変更可能) 、コマンド行のどこに現われてもかまいません。ただし履歴置換のネストはできません。! を \ でエスケープすれば、その特別な意味を抑制できます。

履歴置換を含んだ入力行は、展開された後で、他の置換が起こる前または コマンドが実行される前に、端末上にエコーバックされます。

イベント指示子

イベント指示子は、履歴リスト内の コマンド行エントリを参照するものです。

`!`	次の文字が空白文字、タブ、復帰改行、=、または (でなければ、履歴置換を開始します。
`!!`	直前のコマンドを指します。他の文字を加えずにこれだけを入力すると、直前のコマンドを繰り返します。
`!`*n*	*n* 番のコマンド行を指します。
`!-n`	入力中のコマンドから *n* 個前のコマンドを指します。
`!`*str*	*str* で始まる最新のコマンドを指します。
`!?`*str*`?`	*str* を含む最新のコマンドを指します。
`!?`*str*`?` *additional*	*str* を含む最新のコマンドを指し、その参照コマンドに *additional* を追加します。
`!{`*command*`}` *additional*	*command* で始まる最新のコマンドを指し、その参照コマンドに *additional* を追加します。
`^`*previous_word*`^`*replacement*`^`	文字列 *previous_word* を文字列 *replacement* に置き換えて、直前のコマンド行を繰り返します。これは以下の履歴置換と同じ動作です。 `!:s/`*previous_word*`/`*replacement*`/` . 以前の特定のコマンドを再実行し、さらに同じような置換を行う場合、たとえば 6 番目のコマンドを再実行するには、次のようにします。 `!:6s/`*previous_word*`/`*replacement*`/` .

ワード指示子

‘`:`’ (コロン) はイベント指示子とワード指示子とを区切ります。ワード指示子が `^`、`$`、`*`、`-`、または `%` で始まるときは省略できます。直前のコマンドからワードを選択する場合、イベント指示子の 2 番目の `!` は省略できます。たとえば `!!:1` と `!:1` は、両方とも直前のコマンドの最初のワードを指します。また `!!$` と `!$` は、両方とも直前のコマンドの最後のワードを指します。ワード指示子には以下のものがあります。

`#`	今までに入力したすべてのコマンド行
`0`	最初に入力したワード (コマンド)
n	*n* 番目の引数
`^`	最初の引数。すなわち `1` と同じ
`$`	最後の引数
`%`	最新の `?`*s* による検索で一致したワード
x`-`*y*	ワードの範囲。`-`*y* は `0-`*y* の省略形
`*`	すべての引数。イベント中に 1 ワードしか存在しないときは NULL
x`*`	*x*`-$`の省略形

x-　　*x** と同じだが、ワード $ を省略

修飾子　オプションのワード指示子の後に、: で始まる修飾子のうちの 1 つを追加できます。

h　　パス名の後部分の構成要素を除去して、前部分を残します。

r　　'.*xxx*' の形の接尾辞を除去して、ベース名を残します。

e　　接尾辞以外はすべて除去して、拡張部分を残します。

s/*l*/*r*/　　*l* を *r* で置き換えます。

t　　パス名の前部分の構成要素を除去して、後部分を残します。

&　　前の置換を繰り返します。

g　　各ワード内の最初の一致が発生した箇所を、上記のオプションに接頭辞を付けて変更します (たとえば g&) 。

p　　新しいコマンドを表示するだけで、実行はしません。

q　　置換されたワードをクォートして、それ以上の置換をエスケープします。

x　　q と同じですが、空白文字、タブ、復帰改行文字ごとにワードに分割します。

g を先頭に付加しないと、*l* に一致する最初の文字列だけが変更されます。一致する文字列がなければ、エラーとなります。

置換部分の左側は正規表現ではなく文字列です。/ の箇所には、区切り文字としてどのような文字でも使用できます。区切り文字用の文字はバックスラッシュで囲まれます。右側の & 文字は、左側のテキストで置換されます。& はバックスラッシュでクォートすることができます。*l* が NULL のとき、直前の文字列における *l* 、または !?*s* における文脈検索文字列 *s* を使用します。同様に *r* の直後に復帰改行がある場合、文脈検索の最右にある ? は省略できます。

イベントが指定されないと、履歴リファレンスは前のコマンドか、(もしあれば) そのコマンド行上での 前の履歴リファレンスを参照します。

高速置換　^*l*^*r*^　　履歴置換 !:s/*l*/*r*/ と同じです。

別名　C シェルが持っている別名のリストは、alias および unalias コマンドを使用してユーザーが作成、表示、および変更できます。シェルは、各コマンドの最初のワードが 既存の別名に一致するかどうかをチェックします。一致すれば、そのワードを別名に置き換えて 再度コマンド処理を実行します。履歴置換のメカニズムは、そのコマンドが 前の入力行であったかのように行うことが可能です。これによって履歴置換は (定義中ではバックスラッシュでエスケープされる) 、別名が使用されているとき、実際のコマンド行引数で置き換えることができます。履歴置換が 1 度も呼ばれていなければ、引数は変更されません。

別名はネストできます。すなわち、別名の定義の中に別の別名を入れてもかまいません。ネストされた別名は履歴置換が実行される前に展開されます。これは次のようなパイプラインに役立ちます。

```
alias lm 'ls -l \!* | more'
```

これは呼び出されたときに、ls(1) の出力を more(1) にパイプするものです。

最初のワードを除いて、別名を自分自身の定義内、また定義が参照している他のいかなる別名内にも書いてはいけません。このようなループが見つかると、エラーメッセージが表示されます。

I/O リダイレクション

以下のメタキャラクタは、それに続くワードが、コマンドの標準入力、標準出力、標準エラーのリダイレクション先のファイル名であることを示します。このワードは、コマンドの他の部分とは別個に展開された ファイル名、変数、またはコマンドです。

`<`	標準入力をリダイレクションします。
`<` `<` *word*	標準入力を *word* と一致する行まで読み取り、それらの行を一時ファイルに格納します。*word* がエスケープされるかクォートされていなければ、格納された行に対して、変数およびコマンド置換が行われます。そして、一時ファイルを標準入力としてパイプラインが起動されます。*word* は変数、ファイル名、およびコマンド置換の対象にはなりません。また、各行はシェルによって置換が実行される前に *word* と比較されます。
`>` `>!` `>&` `>&!`	ファイルへの標準出力をリダイレクションします。ファイルが存在しなければ作成します。存在すれば上書きします。このとき、以前の内容は失われます。 変数 noclobber が設定されていれば、既存のファイルを破壊しません。この変数は、! 形式のいずれかが使用されていないかぎり、端末および /dev/null へのリダイレクションも防ぎます。& 形式は、標準出力および標準エラー (診断出力) の両方の出力先を、指定ファイルに変更します。
`>` `>` `>` `>&` `>` `>!` `>` `>&!`	標準出力へ追加します。> と似ていますが、上書きするのではなくファイルの最後に追加します。noclobber が設定されていると、! 形式のうちの 1 つを使用しないかぎり、存在しないファイルに対してはエラーになります。& 形式は、標準出力および標準エラーの両方をファイルへ追加します。

変数置換

C シェルには一群の変数が備わっており、その各々は 名前と値の対で構成されています。名前つまり変数名は、英文字 (下線も含む) で始まり、最大 20 個の英数字からなります。変数の値は、空白で区切られた 0 個以上のワードです。

値を参照するには、変数名の前に '$' を付けます。(以下に述べる) ある種の参照は、値から特定のワードを選択したり、変数に関する別の情報を表示したりするのに使用できます。括弧は、入力行の他の文字から参照を区別するために使用できます。

変数置換は、入力行の解析、別名の展開、入出力先のリダイレクションが行われた後に実行されます。例外は、入出力先のリダイレクションの変数参照 (リダイレクション中に置換される) および逆引用符で囲まれた文字列 (「コマンド置換」の項を参照) です。

$ の前に \ を付加すれば、変数の置換を抑止できます。ただし、二重引用符で囲まれた中では常に置換されます。単一引用符に囲まれていれば、変数置換は抑止されます。空白文字、タブ、または復帰改行が続く場合は、$ はエスケープされます。変数を生成、表示、および破壊するためには、set または unset コマンドを使用します。いくつかの変数はシェルによって管理または使用されます。たとえば argv 変数には、シェルの引数リストのイメージが入っています。

シェルに使用される変数の多くはトグルです。つまり、シェルはその変数の値を知る必要はなく、単にその変数が設定されているか否かを知っているだけです。

数値の値は、数字として ((@) 組み込みコマンドで扱うように) 処理できます。数値処理では、空の値は 0 とみなされます。複数ワードからなる値の 2 番目以降のワードは無視されます。たとえば verbose 変数が何らかの値 (空の値も含む) に設定されていれば、コマンド入力は端末にエコーされます。

コマンドおよびファイル名の置換は、変数置換の結果生成されたワードに適用されます。ただし、二重引用符で抑止されている場合、noglob が設定されている場合 (ファイル名置換の抑止) 、および :q 修飾子で参照がクォートされている場合を除きます。二重引用符の中では、参照が展開され、クォートされた文字列 (の一部) を形成します。つまり、複数ワードからなる値は、空白文字を埋め込んだ 1 つの文字列に展開されます。:q 修飾子を参照に使用すると、空白で区切られたワードに展開され、各ワードにはそれ以降のコマンドまたはファイル名の置換を防ぐために、クォートされます。

後述する例外を除き、設定されていない変数を参照するとエラーになります。

$*var*

${*var*}　　*var* が示す変数の値から得られる、空白文字で区切られたワード群に置き換えられます。*var* が環境変数の場合、その値が返されます (ただし ':' 修飾子と以下に述べる他の形式は使用できない)。

$*var* [*index*]

${*var* [*index*] }　　*var* が示す変数の値から、特定のワードだけを選択します。*index* は 1 つの数字、'-' で区切られた 2 つの数字、またはアスタリスクで、これに対して変数置換が行われます。ワード群には 1 から始まるインデックス値が付けられています。'*' はすべてのワードを選択します。もし ($argv[-2] のように) 範囲の最初の数字を省略すると、デフォルトとして 1 とみなされます。もし ($argv[1-] のように) 範囲の最後の数字を省略すると、デフォルトとして $#*var* (ワード数) とみなされます。第 2 引数が省略されている (または範囲内の) ときは、範囲が空であってもエラーにはなりません。

$#*name*

${#*name*}　　変数内のワード数を指定します。

`$0` setuid シェルスクリプトを除いては コマンドを読み取っているファイル名に置き換わります。ファイルの名前がわからないときはエラーになります。

`$`*n*

`${`*n*`}` `$argv[`*n*`]` と同じです。

`$*` `$argv[*]` と同じです。

`:gh`、`:gt`、および `:gr` のように、修飾子 `:e`、`:h`、`:q`、`:r`、`:t`、および `:x` が適用できます (「履歴置換」の項を参照) 。中括弧 { } を使用するときは、修飾子は中括弧内に書きます。現在のインプリメントでは、このような修飾子を 1 展開あたり 1 つだけ認めています。

以下の参照は、修飾子 : を使って修飾することはできません。

`$?`*var*

`${?`*var*`}` *var* が設定されていれば文字列 1 に、設定されていなければ文字列 0 に置換します。

`$?0` 現在の入力ファイル名がわかっているときは文字列 1 に、わかっていなければ文字列 0 に置換します。

`$$` (親) シェルのプロセス番号に置換します。

`$<` 標準入力から得る行の内容に置換し、それ以降は何も解釈しません。キーボードからの入力を C シェルスクリプトで 読み取るときに使用できます。

コマンドおよびファイル名置換

コマンドおよびファイル名置換は、組み込みコマンドの引数に選択的に適用されます。評価されていない表現は、展開されません。非組み込みコマンドについては、コマンド名のファイル名展開と、引数リストのファイル名展開は別々に実行されます。展開は、入出力先のリダイレクションが終わった後、サブシェル内で行われます。

コマンド置換

逆引用符で囲まれた (`` `...` ``) コマンドはサブシェルによって実行されます。その標準出力は空白文字、タブ、復帰改行の箇所でワードに分解されます。空のワードは捨てられます。この分解されたテキストは、現在のコマンド行上の逆引用符で囲まれた文字列を置き換えます。二重引用符で囲まれると、復帰改行文字の箇所だけでワードに分解され、空白文字、タブはそのまま保存されます。ただし、最後の復帰改行は無視されます。したがって、コマンド置換がワードの一部を生成することも可能です。

ファイル名置換

`*`、`?`、`[`、または `{` のうちのいずれかの文字を含むクォートされていないワード、または `~` で始まるワードは、以下のようにアルファベット順に ソートされたファイル名のリストに展開 (グロビングとも呼ばれる) されます。

`*` (0 個以上の) どの文字とも一致します。

`?` どの単一文字とも一致します。

`[...]`	括弧内の文字リストまたは範囲内のどの単一文字とも一致します。リストは文字列です。範囲とは、ダッシュ (–) で区切られた 2 つの文字を指し、ASCII の順で (`ascii`(5) を参照) その文字間にあるすべての文字も含みます。
{*str*, *str*, ...}	コンマで区切られたリスト内の各文字列 (またはファイル名マッチングパターン) に展開します。この場合、上記のパターンマッチング表現とは違って、この構造の展開はソートされません。たとえば {b,a} は、'a' 'b' ではなく、'b' 'a' に展開されます。特別な例として、文字 { および } は、文字列 { } と共に、展開されずに引き渡されます。
~[*user*]	変数 `home` の値によって示されるホームディレクトリ、または *user* のパスワードエントリによって示される *user* のホームディレクトリを意味します。

*、?、および [...] だけがパターンマッチングを意味します。パターンに一致するファイル名が存在しないときはエラーになります。'.'(ドット文字) は、それがファイル名またはパス名の構成要素の最初の文字である場合、明示的に一致しなければなりません。/ (スラッシュ) も同様に明示的に一致させる必要があります。

式と演算子

多くの C シェルの組み込みコマンドは、式を受け付けます。この式の演算子は C の演算子と類似していて、その優先度も C での優先度と同じです。通常これらの式は `@`、`exit`、`if`、`set`、および `while` コマンド内に現われ、コマンド実行用のフロー制御の規定に使用します。式の構成要素は空白で区切られます。

式の値が NULL または値がないと `0` とみなされます。式の結果はすべて文字列で、10 進数データも文字列で表現します。

C シェルの演算子を、以下に優先度の高いものから順にグループ分けして示します。

`( ... )`	グループ化
`~`	1 の補数
`!`	論理否定
`* / %`	乗算、除算、剰余 (これらは右結合なので、予想した結果が得られない場合があります。その場合、括弧を使って明示的にグループ化してください) 。
`+ -`	加算、減算 (これらも右結合)
`<< >>`	ビット単位の左シフト、右シフト
`< > <= >=`	より小さい、より大きい、以下、以上
`= = != =~ !~`	等しい、等しくない、ファイル名置換パターン一致 (下記参照) 、ファイル名置換パターン不一致
`&`	ビット単位の AND (論理積)
`^`	ビット単位の XOR (排他論理和)

`|`	ビット単位の OR (包括論理和)
`&&`	AND (論理積)
`| |`	OR (論理和)

演算子 `==`、`!=`、`=~`、および `!~` は、引数を文字列として比較します。他の演算子は数字を使用します。演算子 `=~` および `!~` は、それぞれ左側の文字列が 右側のファイル名置換パターンと一致するか否かをチェックします。これを使えば、文字列間のパターンマッチングだけ実行したいとき、`switch` 文を使用する必要がなくなります。

以下に示すようなファイルに関する照会も可能です。

`-r` *filename*	ユーザーが読み取り権を持っていれば真すなわち 1 を、そうでなければ偽すなわち 0 を返します。
`-w` *filename*	ユーザーが書き込み権を持っていれば、真です。
`-x` *filename*	ユーザーが実行権 (またはディレクトリの検索権) を持っていれば、真です。
`-e` *filename*	*filename* が存在していれば、真です。
`-o` *filename*	ユーザーが *filename* を所有していれば、真です。
`-z` *filename*	*filename* のサイズが 0 のとき、真です。
`-f` *filename*	*filename* がプレーンファイルなら、真です。
`-d` *filename*	*filename* がディレクトリなら、真です。

filename が存在しない、またはアクセスできない場合、すべての照会に対して偽を返します。

コマンドが成功したか否かの照会もできます。

{ *command* }	*command* が正常に動作していれば、この式は真すなわち 1 として、そうでなければ偽すなわち 0 として評価されます。逆に、*command* 自体は正常に動作していれば通常 0 を、問題が発生すれば別の値を返します。ステータスを直接知りたければ、この式ではなく `status` 変数の値を使用します。

制御フロー

シェルはスクリプト内の制御フロー、および制限付きで端末からの制御フローを規定する 多くのコマンドを持っています。これらのコマンドはシェルに入力を再度読み込ませるか (ループさせる)、ある条件下で入力をスキップさせる (分岐させる) ことによって動作します。

`foreach`、`switch`、`while`、`if ... then`、および `else` の各組み込みコマンドは、入力行の最初のワードとして記述しなければなりません。

もし、シェルの入力が見つからないのにループが読み込まれているときは、その入力はバッファリングされます。シェルは、ループによって示される再読み込みを実行するために、内部バッファを検索します (この範囲において、逆方向の `goto` コマンドは、入力が見つからない場合でも成功します)。

コマンドの実行

コマンドが C シェルの組み込みコマンドの場合、シェルはそれを直接実行します。そうでない場合、シェルはその名前で実行可能なファイルを検索します。コマンド名が / を含んでいる場合、シェルはそれをパス名とみなし検索します。コマンド名が / を含んでいない場合、変数 path 内の各ディレクトリ内でコマンドを検索することによって、パス名として扱えるようにします。シェルは、検索を高速にするためにハッシュテーブルを使用して (rehash 組み込みコマンドの説明を参照)、該当するファイルが存在しないディレクトリを除外します。このハッシングは -c または -t オプション、または unhash 組み込みコマンドによって使用不能にできます。

特別な例として、スクリプト名に / が含まれず、ワード shell の別名が存在するときは、それの式をコマンド行に (何も変更を加えずに) 付加します。システムは、この特別な別名の最初のワードを実行します。このワードは、完全なパス名でなければなりません。別名定義中の残りのワードは、入力行のテキストと共に引数として扱われます。

適切な実行権を持つパス名が見つかると、シェルは新しいプロセスをフォークし、execve( ) システムコール (exec(2) を参照) を使用して、そのパス名を引数と共にカーネルに引き渡します。 カーネルは、希望するプログラムで新しいプロセスをオーバレイします。ファイルが実行可能なバイナリ (a.out(4) 形式) なら、カーネルは新しいプロセスを引き継ぎ、実行を開始します。ファイルがテキストファイルで、先頭行が #! で始まる場合、次のワードが、そのスクリプトを解釈するためのシェル (またはコマンド) のパス名とみなされます。先頭行の後続のワードは、そのシェルのオプションとみなされます。カーネルは、引数としてそのスクリプト名を使用し、指定されたシェルを起動 (オーバレイ) します。

上記のいずれの条件にもあてはまらない場合は、カーネルはそのファイルをオーバレイできず、execve( ) コールは失敗します (exec(2) を参照)。C シェルは、以下に述べるように新しいシェルを起動して、そのファイルを実行しようとします。

- ファイルの先頭文字が # の場合、C シェルを起動
- それ以外の場合は Bourne シェルを起動

シグナル方式

シェルは通常 QUIT シグナルを無視します。バックグラウンドジョブは、ハングアップ (HUP) などキーボードから生成されたシグナルを感知しません。他のシグナルは C シェルがその環境から引き継いだ値を持っています。シェルのスクリプト内における割り込みおよび終了シグナルの処理は、組み込みコマンド onintr で制御できます。TERM (終了) シグナルは、ログインシェルが受け取るか、あるいは子プロセスへ引き渡されます。ログインシェルが .logout ファイルを読み込んでいる間は、いかなる場合でも割り込みは許されません。

ジョブ制御

シェルは、番号の付いたジョブを各コマンドの並びと関連付けて、バックグラウンドで動作中のコマンド、または TSTP シグナル (通常は CTRL-Z) によって停止したコマンドの動作を追跡します。コマンドまたはコマンドの並び (セミコロンで区切られたリスト) をメタキャラクタ & を使用してバックグラウンドで起動した場合、シェルは角括弧で囲まれたジョブ番号が付いた行、および関連するプロセス番号のリストを表示します。

```
[1] 1234
```

現在のジョブリストを見るには、組み込みコマンド jobs を使用します。最後に停止したジョブ (停止したジョブがない場合は、最後にバックグラウンドに投入されたジョブ) を 「現在のジョブ」といい、'+' で示します。前のジョブは '-' で示します。現在のジョブが終了したり、フォアグラウンドに移された場合、前のジョブが新たな現在のジョブになります。

ジョブを操作するには、組み込みコマンド bg、fg、kill、stop、および % を使用します。

ジョブの参照は '%' で始まります。パーセント記号だけの指定は、現在のジョブを示します。

% %+ %%	現在のジョブ
%-	前のジョブ
%*j*	'kill -9 %*j*' のようにジョブ *j* を参照します。*j* はジョブ番号、またはジョブを起動した コマンド行を一意に表す文字列です。たとえば 'fg %vi' は、停止した vi ジョブをフォアグラウンドに移します。
%?*string*	*string* を含むコマンド行 (一意に表す) に対応したジョブを指定します。

バックグラウンドで動作中のジョブは、端末からの読み取り時に停止します。バックグラウンドジョブは、通常出力を生成しますが 'stty tostop' コマンドを使用して抑止することも可能です。

ステータスレポート

対話型で動作している場合、シェルは各ジョブのステータスを追跡し、ジョブが終了したり停止したりするとレポートを出力します。通常、入力表示を乱さないように、プロンプト出力時にメッセージを表示します。変数 notify が設定されているときは、シェルはステータスの変更を即座に報告します。デフォルトでは、notify コマンドは現在のプロセスをマークします。バックグラウンドのジョブの起動後に notify と入力すれば、そのジョブをマークできます。

コマンド

組み込みコマンドは C シェル内で実行されます。組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、それはサブシェル内で実行されます。

:	NULL コマンド。このコマンドは解釈されますが、実行はされません。
alias [*name* [*def*]]	別名 *name* に *def* を割り当てます。*def* は一群のワードで、エスケープされた 履歴置換のメタシンタックスを含んでいてもかまいません。*name* に alias または unalias を使用することはできません。*def* を省略すると、現在の別名の定義 *name* が表示されます。*name* と *def* の両方を省略すると、すべての別名と定義が表示されます。
bg [%*job* . . .]	バックグラウンドで、現在のジョブ または指定されたジョブを実行します。

`break`	`foreach` または `while` の最も内側のループの `end` の次から実行を再開します。現在の行の残りのコマンドが実行されます。これによって、複数レベルのブレークを `break` コマンドのリストとして 1 行にすべて書き込めます。
`breaksw`	`switch` からブレークして、`endsw` の直後から再開します。
`case` *label*`:`	`switch` 文のラベル。
`cd` [*dir*]	
`chdir` [*dir*]	シェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。引数を指定しないと、そのユーザーのホームディレクトリに変更します。*dir* が現在のディレクトリには見つからない相対パス名の場合、変数 `cdpath` 内のディレクトリリストを検索します。*dir* が / で始まる値を持つシェル変数の名前である場合、その変数の値が示すディレクトリに変更します。
`continue`	`while` または `foreach` の最も内側のループの、次の繰り返しから実行します。
`default:`	`switch` 文のデフォルトケースにラベルを付けます。デフォルトはすべての `case` ラベルの後に現れなければなりません。コマンド行の残りのコマンドが、最初に実行されます。
`dirs` [`-l`]	ディレクトリスタックを出力します。現在のディレクトリが最も左に現れるように 時間順に出力されます。`-l` 引数を指定すると、~ を使った省略形ではなく、完全な形式で出力されます。
`echo` [`-n`] *list*	*list* 内のワードを空白文字で区切って、シェルの標準出力へ書き出します。オプション `-n` を指定しないと、出力は復帰改行で終了します。`echo` が UNIX コマンドのフルパス名なしで 呼び出された場合、ユーザーの `PATH` の構成に関係なく、デフォルトでは `csh` は、組み込みの `echo` を実行します (`echo`(1) 参照)。
`eval` *argument* . . .	引数をシェルへの入力として読み取り、それをコマンドとして実行します。通常この指定は、コマンドまたは 変数置換の結果として生成されたコマンドを 実行するために使用します。`eval` の使用例については、`tset`(1B) を参照してください。
`exec` *command*	現在のシェルの代わりに *command* を実行します。シェルは終了します。
`exit` [(*expr*)]	呼び出し元のシェルまたは シェルスクリプトが終了し、状態変数の値、または式 *expr* で指定された値が返されます。
`fg` [`%`*job*]	現在のジョブ、または指定された *job* をフォアグラウンドへ移します。

`foreach` *var* (*wordlist*)

...

`end`

変数 *var* を、*wordlist* の各メンバーに連続して設定します。*var* の値が変わるたびに、`foreach` と `end` との間にあるコマンドの並びを実行します。`foreach` と `end` は異なる行に、単独で現われなければなりません。

ループの現在の繰り返しを終了するには、組み込みコマンド `continue` を使用します。またコマンド `foreach` の実行を終了するには、組み込みコマンド `break` を使用します。このコマンドが端末から読み取られるときは、ループ内の文が実行される前に、`?` をプロンプトしてから、ループが読み取られます。

`glob` *wordlist*

wordlist 内のファイル名を展開します。エスケープ \ を認識しない点を除いて、`echo` と同様です。出力のワードは `NULL` 文字によって区切られます。

`goto` *label*

指定する *label* は、展開されるとラベルを生成する コマンドとファイル名です。シェルは可能なかぎり入力をさかのぼり、*label*`:` という形式の行を探します。*label*`:` の前には空白文字またはタブ文字がある可能性もあります。指定された行の次から実行が再開します。`while` または `for` 組み込みコマンドと、対応する `end` の間にあるラベルへジャンプするとエラーになります。

`hashstat`

path 変数の内部ハッシュテーブルがコマンドの検索 (実行を伴わない) にどの程度有効に働いて いるかを示す統計情報行を出力します。ハッシュ関数がヒットの可能性を示しているパス の各構成要素と、'/' で始まらない各構成要素について、実行が試みられます。この統計は *cdpath* 変数ではなく *path* 変数だけの有効性を示します。

`history` [`-hr`] [*n*]

履歴リストを表示します。*n* を指定すると、*n* 個の最新のイベントを表示します。

`-r`

出力を、古いイベントからではなく 最近のイベントから順に並べ変えます。

`-h`

先頭の番号なしで履歴リストを出力します。これは、`source` に対する `-h` オプションを使用して、ソーシングに適したファイルを作成するのに使用します。

`if` (*expr*) *command*

指定された式が真ならば、引数付きの単一の *command* を実行します。*command* 上での変数置換を最初に実行し、同時に `if` コマンドの残りの部分を実行します。*command* は単純コマンドでなければならず、パイプライン、コマンドリスト、括弧付きコマンドリストは指定できません。なお *expr* が偽で *command* が実行されなくても、入出力先のリダイレクションが実行されてしまうので注意してください (これはバグです)。

```
if
(expr)
then

. . .

else if
(expr2)
then

. . .

else

. . .

endif
```

expr が真ならば、最初の `else` までのコマンドを実行します。*expr2* が真ならば、`else if` と 2 番目の `else` の間にあるコマンドを実行します。上記のいずれでもなければ、`else` と `endif` の間にあるコマンドを実行します。 `else if` の組はいくつでもかまいませんが `else` は 1 つしか許されません。`endif` は 1 つだけ必要で、これは必須です。`else` および `endif` というワードは、各行における空白を除く最初の文字でなければなりません。`if` は、その行に単独で現われるか、`else` の後に現われるかのいずれかです。

`jobs[-l]` ジョブ制御下で活動中のジョブをリストします。

`-l` 通常の情報の他に、プロセス ID を表示します。

`kill` [*–sig*] [*pid*] [`%`*job*] . . .

`kill -l` `TERM` (終了) シグナル (デフォルトの場合) または指定されたシグナルを、指定されたプロセス ID、指定されたジョブ、または現在のジョブへ送信します。シグナルは番号または名前で指定します。デフォルトはありません。`kill` と入力した場合、現在のジョブにはシグナルを送信しません。送信中のシグナルが `TERM` (終了) または `HUP` (ハングアップ) の場合、そのジョブまたはプロセスには `CONT` (継続) シグナルも送られます。

`-l` 送信可能なシグナル名の一覧を表示します。

`limit [ -h ] [` *resource* `[` *max-use* `] ]` 現在のプロセス、またはそれが生成したすべてのプロセスについて、各プロセスが指定された *resource* を *max-use* 以上消費しないよう制限します。*max-use* を省略すると、現在の上限値を表示します。*resource* を省略すると、すべての上限値を表示します。(システムで利用可能な最大上限値を調べるには `sysdef`(1M) コマンドを実行してください。 表示される値は 16 進数ですが、`bc`(1) コマンドを使って 10 進数に変換できます。)

`-h` 現在の上限値ではなく 強い上限値を使用します。強い上限値は現在の上限値を制限します。強い上限値を上げることができるのは 特権ユーザーだけです。

resource として指定できるものは次のとおりです。

`cputime` プロセス当たりの最大 CPU 使用時間 (秒)

filesize	ファイルシステムのサイズに制限された 最大の単一ファイル容量 (df(1M) 参照)
datasize (ヒープサイズ)	プロセスの (スタックも含めた) 最大データサイズ。システムの仮想記憶サイズ。swap(1M) を参照。
stacksize	プロセスの最大スタックサイズ。swap(1M) を参照。
coredumpsize	最大コアダンプのファイルサイズ。ファイルシステムのサイズに制限する。
descriptors	ファイル記述子の最大数 sysdef() を実行する。
memorysize	仮想記憶の最大サイズ

max-use は数値で、以下の単位を付加して指定することもできます。

*n*h	(cputime の) 時間
*n*k	*n* キロバイト。これは cputime を除くすべての値のデフォルト単位です。
*n*m	*n* メガバイトまたは (cputime の) 分
mm:*ss*	(cputime の) 分と秒

たとえば、0 メガバイトに コアファイルダンプのサイズを 制限するには次のように入力します。

```
limit coredumpsize 0M
```

login [*username* | -p] ログインシェルを終了して login(1) を起動します。.logout ファイルは処理しません。*username* を省略すると、login がユーザー名をプロンプトしてきます。

-p	現在の環境変数を保存します。

logout ログインシェルを終了します。

nice [+*n* | –*n*] [*command*] シェルまたは *command* のプロセス優先度の値を *n* だけ増減させます。 優先度の値が大きいほど、プロセスの優先度は低くなり、実行速度は遅くなります。*command* を指定すると、そのコマンドは常にサブシェルで実行され、指定された値は 1 つの if コマンドの範囲内でのみ有効です。*command* を省略すると、nice は現在のシェルの値を増やします。増やす値を省略すると、nice はプロセス優先度の値を 4 に設定します。 プロセス優先度の値の範囲は、–20 から 20 までです。*n* の値がこの範囲外の場合、最低値または最高値に設定されます。

+*n*	プロセス優先度の値を *n* だけ増やします。
–*n*	*n* だけ減らします。この引数は特権ユーザーだけが使用できます。

`nohup [` *command* `]`
HUP を無視して *command* を実行します。*command* 引数を省略すると、その後のスクリプト全体において HUP を無視します。*command* を指定すると、そのコマンドは常にサブシェル内で実行されます。このとき、単純な `if` 文中のコマンドに対する制限事項が適用されます。`&` を使って切り離したすべてのプロセスに対し、このコマンドが有効になります。

`notify [ %`*job* `] . . .`
現在のジョブまたは指定されたジョブのステータスが変わったとき、非同期的にユーザーに知らせます。

`onintr [ - | `*label*`]`
割り込み時のシェルの動作を制御します。引数を指定しないと、`onintr` はデフォルトの動作を復元します (すなわち、シェルはシェルスクリプトを終了して、端末のコマンド入力レベルに戻ります) 。`-` 引数を指定すると、シェルはすべての割り込みを無視します。*label* 引数を指定すると、割り込みを受信するか 割り込みのために子プロセスが終了したときに、シェルは `goto` *label* を実行します。

`popd [ +`*n* `]`
ディレクトリスタックをポップして、新しいトップディレクトリへ `cd` します。ディレクトリスタックの構成要素は、トップディレクトリを 0 として番号付けられます。

+n　スタック内の *n* 番目のエントリを破棄します。

`pushd [ +`*n* `|` *dir*`]`
ディレクトリスタックにディレクトリをプッシュします。引数を指定しないと、トップの 2 つの構成要素を交換します。

+n　*n* 番目のエントリとトップスタックを交換しそれに `cd` します。

dir　現在の作業用ディレクトリをスタックにプッシュし、*dir* を新たな作業用ディレクトリとします。

`rehash`
新しく追加されたコマンドに合わせて、*path* 変数内にリストされたディレクトリの内容の 内部ハッシュテーブルを再計算します。新しく追加されたディレクトリに合わせて、*cdpath* 変数内にリストされたディレクトリの内容の 内部ハッシュテーブルを再計算します。

`repeat` *count command*
command を *count* 回繰り返します。*command* は 1 行の `if` 文と同様の制限に従います。

`set [`*var* `[ = `*value* `] ]`

`set` *var*`[`*n*`] = `*word*
引数を指定しないと `set` はすべてのシェル変数の値を表示します。複数ワードからなる値は括弧付きのリストで表示されます。引数 *var* のみを指定すると、`set` は空の (NULL) 値を *var* が示す変数に割り当てます。引数を *var* = *value* の形式で指定すると、`set` は *var* が示す変数に *value* が示す値を割り当てます。*value* は次のいずれかです。

word　単一ワード (もしくはクォートされた文字列)

(*wordlist*)　空白で区切られた、括弧付きワードリスト

値は、割り当てられる前に、コマンドおよびファイル名展開されます。 `set` *var*`[`*n*`]` `=` *word* 形式は、複数ワードからなる値の *n* 番目のワードを *word* に置き換えます。

`setenv [` *VAR* `[` *word* `] ]` 引数を指定しないと `setenv` はすべての環境変数を表示します。引数 *VAR* を指定すると、`setenv` は環境変数 *VAR* に空の値 (NULL) を設定します (慣例上、環境変数名は大文字で指定されるのが通常) 。 *VAR* と *word* の両引数を指定すると、`setenv` は、環境変数 `NAME` に単一ワードまたはクォートされた文字列である値 *word* を設定します。 最もよく使用される環境変数 `USER`、`TERM`、および `PATH` は、自動的に `csh` 変数 *user*、*term*、および *path* から (へ) インポート (エクスポート) されます。 したがって、これらの変数に `setenv` を使用する必要はありません。さらにシェルは、`csh` 変数 `cwd` が変更されるたびに、その値を環境変数 `PWD` へ設定します。

環境変数 `LC_CTYPE`、 `LC_MESSAGES`、 `LC_TIME`、 `LC_COLLATE`、 `LC_NUMERIC`、 `LC_MONETARY` は、C シェル内で変更されると新しい値が即座に有効になります。

`LC_*` 変数 (`LC_CTYPE`、 `LC_MESSAGES`、 `LC_TIME`、 `LC_COLLATE`、 `LC_NUMERIC`、 `LC_MONETARY`) のいずれも環境に設定されていなければ (`environ`(5) を参照) 、それぞれ対応するロケールのカテゴリにおける `csh` の動作は、環境変数 LANG によって決定されます。 もし `LC_ALL` が設定されていれば、その内容が `LANG` 変数やその他の `LC_*` 変数より優先されます。上記の変数が環境にまったく設定されていなければ、C ロケール (米国スタイル) が `csh` の動作を決定します。

`LC_CTYPE` `csh` の文字の処理方法を決定します。`LC_CTYPE` に有効な値が設定されていると、`csh` は、そのロケールにあった文字を含むテキストやファイル名を表示および処理できます。

`LC_MESSAGES` 診断メッセージや情報メッセージの表示方法を決定します。また、メッセージの言語とスタイル、そして肯定応答および否定応答の正しい形も決定します。C ロケールにおいては、メッセージはプログラム自身が 使用しているデフォルトの形で表示されます (通常は米語) 。

`LC_NUMERIC` 基数文字 (C ロケールでは小数点 ".") および 3 桁ごとのセパレータ (C ロケールでは空文字列) の値を決定します。

`shift [` *variable* `]` `argv` の構成要素 (または *variable* が指定されればその変数の構成要素) を、左へシフトして最初の構成要素を切り捨てます。未設定の変数および NULL 値に対してはエラーとなります。

`source [ -h ]` *name* *name* からコマンドを読み取ります。`source` コマンドはネストできますが、あまり深くネストするとシェルのファイル記述子が 不足する可能性があります。ソースファイル中のエラーは、それがいかなるレベルであろうと、ネストされたすべての `source` コマンドを終了させます。

-h　*name* が示す、履歴リスト上のファイル からコマンドを持ってきますが、実行はしません。

`stop %`*jobid . . .*　現在のジョブまたは 指定されたバックグラウンドジョブを停止します。

`stop` *pid . . .*　指定された *pid* (プロセス ID 番号) の実行を停止します (ps(1) 参照) 。

`suspend`　`^Z` を使用して停止シグナルが送信されたときと同様に、トラック内でシェルを停止します。このコマンドは、`su` コマンドによって開始されたシェルを停止するときによく使用します。

`switch` (*string*)

`case` *label*`:`

`. . .`

`breaksw`

`. . .`

`default:`

`. . .`

`breaksw:`

`endsw`　各 *label* を、指定された *string* (初めにコマンドおよびファイル名展開される) とマッチングします。ファイルメタキャラクタ `*`、`?`、および `[...]` は、case ラベルとして使用できます。このラベルは変数展開されます。一致するラベルが見つかる前にデフォルトラベルを検出すると、そのデフォルトのラベルの次から実行が開始されます。各 `case` 文および `default` 文は、行の先頭になければなりません。コマンド `breaksw` は `endsw` の次から実行を続けます。 それ以外は、C の場合と同様に、後続の `case` および `default` 文に制御が移ります。一致するラベルが見つからずデフォルトも指定されていない場合、`endsw` の次から実行されます。

`time [` *command* `]`　引数を省略すると、現在の C シェルとその子プロセスが 消費した時間についての情報を出力します。`command` を指定すると、*command* を実行し、その実行時間の情報を出力します 現在 `time` 組み込みコマンドは出力の最後 6 フィールドに対する値を計算しません。これらのフィールドに "0" の値を表示します。

```
example% time ls -R
        9.0u 11.0s 3:32 10% 0+0k 0+0io 0pf+0w
```

(以下の「環境変数と定義済みシェル変数」を参照)。

`umask [` *value* `]`　ファイル生成マスクを表示します。*value* を指定すると、ファイル生成マスクが設定されます。*value* は 8 進数で指定され、どんなビットもオフにできますが、新たにアクセス権を追加するようなビット をオンにできません。

よく使われる値として 077 、002 、または 022 があります。077 は、自分以外の誰に対しても何のアクセス権も 与えません。002 は、グループユーザーに対しては全アクセス権、グループ外ユーザーに対しては読み取り (およびディレクトリ検索) 権を与えます。022 は、グループおよびグループ外ユーザーに対して読み取り (およびディレクトリ検索) 権を与えますが、書き込み権を与えません。

`unalias` *pattern*
: *pattern* が示すファイル名置換パターン に一致する別名を破棄します。'`unalias *`' と指定すると、すべての別名が破棄されます。

`unhash`
: *path* と *cdpath* 変数の内部ハッシュテーブルを使用不能にします。

`unlimit` [`-h`] [*resource*]
: *resource* に関する上限値を削除します。*resource* が指定されないと、すべての資源の上限値が削除されます。資源名の一覧については、前述の `limit` コマンドの説明を参照してください。

  `-h`
  : 対応する強い 上限値を削除します。これは特権ユーザーだけしか実行できません。

`unset` *pattern*
: *pattern* が示すファイル名置換パターンに一致する名の変数を削除します。'`unset *`' と指定すると、すべての変数が削除されます。ただしこれによって、たいへんな副作用を引き起こします。

`unsetenv` *variable*
: 環境から *variable* が示す変数を削除します。`unset` のようなパターンマッチングは行いません。

`wait`
: プロンプトする前に、バックグラウンドジョブの終了 (または割り込み) を待ちます。

`while` (*expr*)

`...`

`end`
: *expr* が真 (0 以外) であるかぎり、`while` と、対応する `end` 文の間のコマンドを繰り返します。ループを途中で終了させるには `break` を、または先頭から再開するには `continue` を使用します。`while` と `end` は入力行に単独で現われなければなりません。シェルの入力が端末の場合、疑問符のプロンプトを表示してコマンドを受け付け、`end` コマンドが入力されるとループ内でコマンドを実行します。

`%` [*job*] [`&`]
: 現在のジョブまたは *job* で指定されたジョブをフォアグラウンドへ移します。アンパサンドを付加すると、ジョブをバックグラウンドで実行し続けます。

`@` [*var* `=`*expr*]

`@` [*var* [*n*] `=`*expr*]
: 引数を指定しないと、すべてのシェル変数の値を表示します。引数を指定すると、*var* が示す変数の値、またはその値の *n* 番目のワードを、*expr* が示す式の評価値に設定します。[*n*] を指定するときは、*var* が示す変数およびその *n* 番目の構成要素が既に存在していなければなりません。

式が文字 >、<、&、または | を含んでいる場合、少なくとも その部分は括弧で囲む必要があります。

演算子 *= や += などは、C と同様に使用できます。名前と代入演算子とを区切っている空白は、指定してもしなくてもかまいません。ただし、指定しないと 1 ワードとみなされる式 の要素を区切る場合は、空白は必須です。

特別な接尾演算子の + + と - - は、*name* をそれぞれ増加・減少させます。

環境変数と定義済みシェル変数

C シェルは、起動したプロセスに自動的に値がエクスポートされる環境変数と、そうでないシェル変数を区別します。この点は Bourne シェルとは異なります。両タイプの変数とも変数置換においては同等に扱われます。 初期化の際、シェルは、変数 argv、cwd、home、path、 prompt、shell、および status を設定します。シェルは、環境変数 USER、TERM、HOMEをそれぞれシェル変数 user、term、home へコピーします。 そして、上記のシェル変数が再設定されるたびに、対応する環境変数へその値を書き戻します。PATH と path は同様に扱われます。path は .cshrc または .login ファイル内で 1 度設定するだけです。環境変数 PWD は、cwd が変更されるたびに設定されます。以下のシェル変数の意味は既に定義済みです。

argv
: 引数リスト。シェルの今回の呼び出しの際に渡されたコマンド行引数リストを含んでいます。この変数は、$1 や $2 などの定位置パラメタの値を決定します。

cdpath
: cd、chdir、および popd コマンドが検索するディレクトリのリスト。これらのコマンドが受け取ったディレクトリ引数が 現在のディレクトリのサブディレクトリでないとき、この変数が示すディレクトリ群が検索されます。

cwd
: 現在のディレクトリの完全なパス名

echo
: 実行前に置換後のコマンドをエコーします。

fignore
: ファイル名を補完させるときに無視する ファイル名接尾辞のリスト。典型的な例は '.o' の 1 ワードだけ。

filec
: ファイル名補完を可能にします。その際、CTRL-d 文字の EOT および ESC 文字が端末の入力行の最後に入力されると特別な意味を持ちます。

  EOT　前に指定した文字列で始まる ファイル名のリストをすべて出力します。

  ESC　前に指定した文字列を 最も長い明確な拡張子に置き換えます。

hardpaths
: 設定すると、ディレクトリスタック内のパス名は、シンボリックリンクの構成要素を含まないように展開されます。

histchars	2 文字の文字列。最初の文字は、履歴置換文字として ! を置き換えます。2 番目の文字は、高速置換のためのキャレット (^) を置き換えます。
history	履歴リストに保存される行数。この数が大きすぎると、C シェルのメモリすべてを消費してしまいます。設定されないと、C シェルは最新のコマンドのみを保存します。
home	ユーザーのホームディレクトリ。ファイル名展開の ~ は、この変数の値を指します。
ignoreeof	設定すると、シェルは端末からの EOF を無視します。これにより、CTRL-D を誤って入力して C シェルを終了させてしまう ことが防げます。
mail	C シェルがメールの有無をチェックするファイルのリスト。値の最初のワードが数字の場合、メールをチェックする間隔を秒数 (デフォルトは 5 秒) で表します。
nobeep	あいまいなファイル名を C シェルに拡張させる際、一致したものを見つけたときに発生するビープ音を抑止します。
noclobber	既存のファイルが間違って破壊されないように 出力先のリダイレクションを制限します。> は新規のファイルへのみ出力先をリダイレクションできます。>> は既存のファイルへのみ出力先をリダイレクションできます。
noglob	ファイル名置換を禁止します。これは、シェルスクリプト内で一度ファイル名 (もしあれば) を取得した後、これ以上拡張したくないときに便利です。
nonomatch	パターンが一致しなかったとき、エラーではなくファイル名置換パターンを返します。パターンが間違っているときは、エラーを返します。
notify	設定すると、シェルはプロンプトの発行まで待つことなく、ジョブの終了を即座に報告します。
path	コマンドを検索するディレクトリのリストです。path は環境変数 PATH から初期化されます。この環境変数は、path を変更するたびに C シェルによって更新されます。NULL ワードは現在のディレクトリを指定します。デフォルトは通常（/usr/bin .）です。.cshrc ファイルまたは .login ファイル (ログインシェルのみ) 中の設定は、csh 起動時にこの初期検索パスを上書きします、path が設定されないと、完全なパス名だけが実行されます。対話型の C シェルは、通常 .cshrc の読み取り後と path の再設定時には、必ず

リストされたディレクトリの内容をハッシングします。新しいコマンドが追加されたときは、rehash コマンドを使用してテーブルを更新します。

prompt　対話型 C シェルのプロンプト文字列。非対話型シェルの prompt 変数は未設定のままです。対話型でのみ有効な .cshrc ファイル内の別名および他のコマンドは、'if ($?prompt == 0) exit' という記述の後におくことができ、これによって、非対話型シェルの起動時間が短縮できます。prompt 文字列内の ! は、現在のイベント番号で置換されます。デフォルトプロンプトは通常の *hostname*% または特権ユーザー用の *hostname*# です。

$prompt の設定には次の 3 つの方法があります。

変数 $prompt が設定されていない -- 非対話型シェル、$?prompt の値によって判断する

設定されているが、"" である -- which(1) コマンドが .cshrc を呼び出した場合

設定されていて、"" ではない -- 通常の対話型シェル

savehist　ユーザーがログアウトしたときに ~/.history に保管される履歴リストの行数。 savehist の値が大きすぎると、C シェルの起動処理が遅くなります。

shell　C シェルが存在するファイル。実行ビットがセットされているが システムにより実行できないようなファイルを解釈するために、シェルをフォークする際に使用します。

status　最新のコマンドによって返されたステータス。そのコマンドが異常終了した場合、ステータスに 0200 が加算されます。異常終了した組み込みコマンドは、終了状態 1 を返します。他のすべての組み込みコマンドは、ステータスを 0 に設定します。

time　コマンドの自動タイミングを制御します。1 つまたは 2 つの値が与えられます。最初の値は、報告するしきい値の CPU 時間 (秒) です。2 番目の値は、どの資源について報告するかを示すタグとテキストの文字列です。タグは、先頭にパーセント記号 (%) を付加した 1 つの英大文字で表します。認識できないタグはテキストとして表示します。

- %D　使用された、共有でないデータ領域の平均サイズ (キロバイト)
- %E　コマンド実行時間
- %F　ページフォルト回数
- %I　ブロック入力操作の数
- %K　使用された、共有でないスタック領域の平均サイズ (キロバイト)

%M	プロセスの実行中に使用された最大実記憶領域
%O	ブロック出力操作の数
%P	エラップス時間 (E) に占める CPU 時間 (ユーザー時間 (U) とシステム時間 (S) の合計) のパーセンテージ
%S	ユーザープロセス内で、カーネルによって消費された CPU 時間 (秒)
%U	ユーザープロセスに使用された CPU 時間 (秒)
%W	スワップの回数
%X	使用されたの共有メモリ領域の平均サイズ (キロバイト)

デフォルトの出力は、%U、%S、%E、%P、%X、%D、%I、%O、%F、%W の順です。

verbose 履歴置換の後で各コマンドを表示します。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の csh の動作については、largefile(5) を参照してください。

ファイル

~/.cshrc	各シェルの起動時に読み取られる
~/.login	ログイン時、ログインシェルによって .cshrc の後に読み取られる
~/.logout	ログアウト時、ログインシェルによって読み取られる
~/.history	次回のログイン時に使用できるよう履歴を保管する
/usr/bin/sh	'#' で始まらないシェルスクリプト用の Bourne シェル
/tmp/sh*	'<<' 用の一時ファイル
/etc/passwd	'~*name*' 用のホームディレクトリを指定したソース

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

bc(1), echo(1), login(1), ls(1), more(1), ps(1), sh(1), shell_builtins(1), tset(1B), df(1M), swap(1M), sysdef(1M), access(2), exec(2), fork(2), pipe(2), a.out(4), environ(4), ascii(5), attributes(5), environ(5), largefile(5), termio(7I)

診断

`You have stopped jobs.` ジョブ制御下で停止しているジョブ上で C シェルを終了しようとしました。すぐに再度 C シェルを終了すれば正常に終了しますが、停止しているジョブも終了します。

警告

シェルスクリプトを `setuid` して使用することは避けてください。

注意事項

ワードは 1024 バイト以下でなければなりません。引数リストはシステムによって 1,048,576 バイト以下に制限されています。しかし、ファイル名の展開が適用できるコマンドの引数の最大数は 1706 です。コマンド置換は、引数リストに許されているのと同じ文字数までしか展開できません。ループを検出するために、シェルは 別名置換を 1 行あたり 20 に制限しています。

コマンドが停止後再開されたとき、最初にコマンドを開始したディレクトリが 現在のディレクトリと異なる場合、シェルは現在のディレクトリを表示します。これは、ジョブがディレクトリを内部で変更している可能性があるので、誤解を招きます (すなわち、間違っています) 。

シェルの組み込み関数を停止または再開することはできません。*a ; b ; c* のような形式のコマンドの並びも、停止するとうまく処理できません。*b* で停止するとシェルは *c* を実行しません。このことは、別名から展開された場合に特に注意が必要です。この現象は、コマンドの並びを括弧で囲んで サブシェルに実行させると避けられます。

プロセス開始後の端末出力制御は原始的です。より高度な出力の制御が必要な場合は、Sun Window システムを使用してください。

ループ内のコマンドは ? でプロンプトしてきますが、履歴リストには格納されません。

制御構造は組み込みコマンドと みなすより構文解析をするべきです。これによって、制御コマンドはどこにでも置くことができ、| と組み合わせたり、& および ; などのメタシンタックスと組み合わせることが できるようになるはずです。

コマンド置換の出力に、修飾子 : を使えるようにすべきです。この : 修飾子を変数置換に使用するには 2 つの問題があります。1 つはすべての修飾子が使用できるわけではないこと、もう 1 つは一度の置換に 1 つの修飾子しか許されないことです。

履歴置換内の g (グローバル) フラグは、すべてのワード内のすべての一致箇所ではなく、各ワード内の最初の一致箇所にしか適用されません。通常のテキストエディタは、置換コマンドに g が指定されると、すべての一致箇所の指定とみなすのが一般的です。

クォートの規則は複雑です。二重引用符内で変数置換を強要するエスケープ文字を 無効にするということは、混乱を招いて理解しにくくなり、Bourne シェルとの一貫性も保たれません。

シェルが、シンボリックリンクを無視してしまうことがあります。`hardpaths` 変数を設定すればこれを緩和できます。

次のような組み込みコマンドを 複数回使ってできた重複したパス名は、すべてユーザーが手動で除かなければなりません。set path = *pathnames* または setenv PATH *pathnames* シェルスクリプトまたは .cshrc ファイルが 'set path=(/usr/local /usr/hosts $path)' のようなコマンドを発行して、指定したディレクトリがパス名リスト内にあることを保証する場合に、よくパス名が重複します。

標準出力と標準エラーを分けて出力する唯一の方法は、以下のようにサブシェルを起動することです。

```
example% ( command > outfile ) > & errorfile
```

一般的な使用には十分耐えられますが、C シェルを少し複雑に使うと予期しない結果を招くことがあります。

csh をログインシェルとして起動しても、.login がホームディレクトリに存在しない場合は、csh は /etc/.login を読み取ります。

存在しないコマンドのインタプリタを実行しようとするシェルスクリプトを、シェルが処理した場合、シェルは、シェルスクリプトが存在しないという間違った診断メッセージを 返します。

使用上の留意点

現在 time 組み込みコマンドは 出力の最後 6 フィールドに対する値を計算しません。これらのフィールドに "0" の値を表示します。

```
example% time ls   -R
        9.0u 11.0s 3:32 10% 0+0k 0+0io 0pf+0w
```

名前 | csplit – 指定した文脈に従ってファイルを分割

形式 | **csplit** [-ks] [-f *prefix*] [-n *number*] *file arg1… argn*

機能説明 | csplit ユーティリティは、 *file* オペランドが示すファイルを読み込み、その内容のすべてまたは一部を *arg* オペランドの指示に従って他のいくつかのファイルに書き出し、その出力ファイルのサイズを表示します。

オプション | 以下のオプションを指定できます。

-f *prefix* — 生成されるファイルを、順番に *prefix*00、 *prefix*01、 . . . 、 *prefixn* と名付けます。 デフォルトは xx00 . . . xx*n* です。*prefix* 引数が長すぎて、生成されるファイル名の長さが 14 バイトを超えてしまう場合はエラーとなります。 その場合、ファイルは 1 つも生成されず、csplit は診断メッセージを伴って終了します。

-k — エラー発生時、デフォルトでは csplit は作成したファイルを削除します。 -k オプションを指定すると、csplit は、作成済みのファイルをそのままにしておきます。

-n *number* — 生成するファイル名の可変部分の桁数を *number* で指定します。デフォルトは 2 です。

-s — 生成するファイルのサイズに関する情報を出力しません。

オペランド | 以下のオペランドを指定できます。

file — 分割対象ファイルのパス名。 - を指定すると標準入力とみなされます。

オペランド *arg1* . . . *argn* は、以下の組み合わせで指定できます。

/*rexp*/[*offset*] — 現在の行から、 *rexp* で示す正規表現の評価 (*offset* 指定があればそれも考慮) により得られる行の直前の行までを 内容とするファイルを生成します。*rexp* は、基本的な正規表現の規則に従っていなければなりません。*offset* は行数を表す正または負の整数で、指定は任意です。指定するときは、符号 + または - を先頭に付加しなければなりません。 なお、この指定に従って行を選択したとき、生成されるファイルの行数がゼロになってしまう、あるいは 入力ファイル中の行数を超えてしまう場合、処理結果は予測できません。ファイル生成後、現在の行は、正規表現の評価により得られた行に設定されます。*rexp* のパターンマッチングは、常に現在の行から ファイルの終わりまでが対象となります。

%*rexp*%[*offset*] — このオペランドは、上記の /*rexp*/[*offset*] 指定と同じ意味ですが、入力ファイルから選択した行によるファイルの 生成は行われません。

line_no — 現在の行から、行番号 *line_no* で示す行の直前の行までを内容とするファイルを生成します。 生成したファイル中では、各行に 1 から始まる番号が振られます。現在の行は *line_no* で示す行に移ります。

{*num*} 直前のオペランドを繰り返します。この指定は、上記のどのオペランド の後でも指定できます。 *rexp* のタイプのオペランドの後に指定すれば、 そのオペランドは *num* 回繰り返されます。 *line_no* オペランドの後に指定すれば、ファイルはその位置から *line_no* 行ごとに *num* 回だけ分割されます。

オペランドが示す行が、現在の位置からファイルの終わりまでの 間に存在していない場合、エラーが報告されます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `csplit` の動作については、`largefile`(5) を参照してください。

使用例

例 **1** ファイルを分割および結合する

次の例は、4つのファイル `cobol00 . . . cobol03` を作成します。

```
example% csplit -f cobol filename '/procedure division/' /par5./ /par16./
```

分割されたファイルを編集した後、 次のようにすると、再度 1 つにまとめることができます。

```
example% cat cobol0[0-3] > filename
```

注意 : この例は、元のファイルを上書きします。

例 **2** ファイルを等分に分割する

次の例では、100 行毎に 10,000 行までファイルを分割します。 `-k` オプションは 10,000 行未満しかなくても、作成されたファイルを保存します。ただし、エラーメッセージはそのまま出力されます。

```
example% csplit -k filename 100 {99}
```

例 **3** 各 C ルーチンごとにファイルを作成する

`prog.c` が通常の C のコード化規則 (ルーチンの最後の行の1文字目が } だけである) に従っていれば、 この例では、 `prog.c` 中の C ルーチン (21個まで) に対し、それぞれファイルを作成します。

```
example% csplit -k prog.c '%main(%' '/^}/+1' {20}
```

環境

`csplit` の実行に影響を与える環境変数 `LC_COLLATE`、`LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 正常終了

`>0` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目　sed(1), split(1), attributes(5), environ(5), largefile(5)

診断　診断メッセージは、次に示すもの以外は、メッセージテキストを 読めば意味がわかるようになっています。

arg – out of range　指定された引数が、現在の位置とファイルの終わりまでの間に存在する行を指してしないことを意味します。

名前 | ctags – ex および vi で使用するタグファイルの作成

形式 | **/usr/bin/ctags** [-aBFtuvwx] [-f *tagsfile*] *file...*

/usr/xpg4/bin/ctags [-aBFuvwx] [-f *tagsfile*] *file...*

機能説明 | ctags ユーティリティは、C、C++、Pascal、FORTRAN、yacc(1)、 lex(1) の指定されたソースから ex(1) 用のタグファイルを作成します。タグファイルは、ファイルのグループにおける指定されたオブジェクト (この場合は関数と型定義) の位置を示します。タグファイルの各行には、オブジェクト名、それが定義されているファイル、およびオブジェクト定義のアドレス指定が含まれます。関数はパターンによって検索され、型定義は行番号によって検索されます。指定子は、行の個別のフィールドに指定され、空白または TAB 文字で区切られます。タグファイルを使用すると、ex はこれらのオブジェクト定義を簡単に検索できます。

通常 ctags は、tags というファイルにタグの記述を入れます。これは、-f オプションによって変更にできます。

名前が .c または .h で終わるファイルは、C または C++ のソースファイルとみなされ、C または C++ ルーチンとマクロ定義が含まれていないか調べられます。名前が .cc、.C、または .cxx で終わるファイルは、C++ ソースファイルとみなされます。名前が .y で終わるファイルは、yacc ソースファイルとみなされます。名前が .l で終わるファイルは、lex ファイルとみなされます。これ以外のファイルは、まず Pascal または FORTRAN のルーチン定義が含まれていないか調べられます。これらの定義が含まれていないファイルは、再度 C 定義について検索されます。

タグ main は、C または C++ プログラムでは特別に処理されます。タグ formed は、*file* の先頭に M を付け、最後に .c 、.cc、.C、または .cxx があれば削除し、先頭のパス名構成要素も削除して、書式が作成されます。これにより、複数のプログラムが置かれているディレクトリでも ctags を利用できます。

オプション | 出力に関するオプションの優先順位は、-x、-v、そして残りのオプションという順になります。次のオプションを指定できます。

-a	既存の tags ファイルに出力を追加します。
-B	後方検索パターン (?...?) を使用します。
-f *tagsfile*	tags ではなく、ファイル *tagsfile* にタグ記述を入れます。
-F	前方検索パターン (/.../) を使用します (デフォルト)。
-t	型定義のタグを作成します。/usr/xpg4/bin/ctags は、デフォルトで、型定義のタグを作成します。
-u	タグ内の指定したファイルを更新します。つまり、これらのファイルへの参照はすべて削除されて、ファイルに新しい値が追加されます。このオプションを指定すると実行速度が遅くなるので注意してください。通常、単に tags ファイルを作成し直す方が、速く実行できます。

`-v`	標準出力上に関数名、ファイル名、およびページ番号をリストする索引を出力します (64 行のページを想定)。出力は、辞書編集方式の順にソートされるため、出力を `sort -f` に通しておくのがよいでしょう。
`-w`	警告診断を抑制します。
`-x`	オブジェクト名、各オブジェクトが定義されている行番号とファイル名、およびその行のテキストを作成して、標準出力に出力します。これは、オフラインで読み取ることのできる関数索引として出力可能な単純な索引です。

オペランド

次の *file* オペランドを指定できます。

file`.c`	`.c` 接尾辞で終了するベース名を持つファイルは、C 言語のソースコードとみなされます。
file`.h`	`.h` 接尾辞で終了するベース名を持つファイルは、C++ 言語のソースコードとみなされます。
file`.f`	`.f` 接尾辞で終了するベース名を持つファイルは、FORTRAN 言語のソースコードとみなされます。

使用法

`-v` オプションは、オプションとして提供される BSD 互換性パッケージに含まれている `vgrind` で主に使用されます。

使用例

例 1 アルファベット順にエントリを生成する

`-v` オプションを付けて `ctags` を使用すると、`vgrind` に対して必ずしも適切とはいえない順序でエントリが作成されます。結果をアルファベット順で生成するには、出力を `sort -f` に送るようにします。

```
example% ctags -v filename.c filename.h | sort -f  >  index
example% vgrind -x index
```

例 2 タグファイルを作成する

sourcedir をルートとするディレクトリ階層に C ソースのタグファイルを作成するには、まず空のタグファイルを作成してから、`find`(1) を実行します。

```
example% cd sourcedir  ; rm -f tags ; touch tags
example% find . \( -name SCCS -prune -name \\
       '*.c' -o -name '*.h' \) -exec ctags -u {} \;
```

空白は、ここで示されたとおりに正確に入力してください。

環境

`ctags`: の実行に影響を与える環境変数 `LC_COLLATE`、 `LC_CTYPE`、`LC_MESSAGES`、 `NLSPATH` については、`environ`(5) のマニュアルページを参照してください。

終了ステータス

次の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した

ファイル

tags	出力タグファイル

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

/usr/bin/ctags

属性タイプ	属性値
使用条件	SUNWtoo

/usr/xpg4/bin/ctags

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目

ex(1), lex(1), vgrind(1), vi(1), yacc(1), attributes(5), environ(5), XPG4(5)

注意事項

FORTRAN と Pascal の 関数、サブルーチン、およびプロシージャは、非常に単純な方法で認識されます。ブロック構造とみなして処理されることはありません。異なるブロックに同じ名前の 2 つの Pascal プロシージャがある場合、一方しか判別されません。

C または Pasca 関数l、および FORTRAN 関数を検索するかどうかを決定する手法は、あまり出来の良いものではありません。

ctags ユーティリティは、#ifdefs を認識しません。

ctags ユーティリティは、Pascal の型を把握している必要があります。型定義の検出は、入力の形式が適切かどうかによります。-tx を使用すると、型定義の最後の行だけが示されます。

名前 | cut – ファイルの各行から選択されたフィールドをカット

形式 | **cut** -b *list* [-n] [*file...*]

cut -c *list* [*file...*]

cut -f *list* [-d *delim*] [-s] [*file...*]

機能説明 | cut ユーティリティは、テーブルからカラムを、またはファイルの各行からフィールドを切り出す場合に使用します。データベースの用語でいえば、リレーションの投影を実現します。*list* で指定されるフィールドは、パンチカード上の文字位置のように固定長でも構いません (-c オプション) 。また、行毎に長さが異なっていても、行が TAB などのフィールド区切り文字で区切られていてもかまいません (-f オプション) 。cut はフィルタとして使用されます。

オプション -b 、 -c 、または -f のうちいずれかを指定しなければなりません。

ファイルを水平方向に (文脈によって) カットするには grep(1) を使用します。また、ファイルをカラム方向 (すなわち水平方向) に結合するには paste(1) を使用します。テーブル内のカラムを並べ変えるには cut と paste を使います。

オプション | 次のオプションを指定できます。

list — コンマで区切られるか、または空白文字で区切られた整数値フィールド番号のリスト (昇順) 。- (オプション) は範囲を示します (たとえば、 1,4,7 、 1-3,8 、 -5,10 (1-5,10 の短縮形) 、または 3- (3 番目のフィールドから最後のフィールドまでを示す短縮形) など) 。

-b*list* — -b の後に続く *list* はバイト位置を指定します (たとえば、 -b1-72 は各行の最初の 72 バイトを引き渡します) 。-b と -n を一緒に使用すると、複数バイト文字が分割されないように *list* が調整されます。

-c*list* — -c に続く *list* は文字位置を指定します (たとえば、 -c1-72 は各行の最初の 72 文字を引き渡します) 。

-d*delim* — -d に続く *delim* はフィールド区切り文字です (-f オプションの場合のみ) 。デフォルトは *tab* です。シェルにとって意味のある空白や他の文字は引用符で囲まなければなりません。*delim* は複数バイト文字でもかまいません。

-f*list* — -f に続く *list* は区切り文字によってファイル内で区切られるフィールドのリストです (-d 参照)。たとえば、-f1,7 は最初と 7 番目のフィールドだけをコピーします。フィールド区切り文字のない行は、-s が指定されていないかぎりそのまま (テーブルのサブヘディングに有効) です。

-n — 文字を分割しません。-b*list* と -n を一緒に指定すると *list* は複数バイト文字が分割されないように調整されます。

-s — -f オプションが指定されたとき、区切り文字のない行を抑止します。指定されないと区切り文字のない行は、そのまま引き渡され

ます。

オペランド

以下のオペランドを指定できます。

file 入力するファイルのパス名。このオペランドを省略するかまたは – を指定すると、標準入力とみなされます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の cut の動作については、largefile(5) を参照してください。

使用例

例 1 ユーザー ID を割り当てる

ユーザー ID とユーザー名を組み合わせて出力するには次のようにします。

```
example% cut -d: -f1,5 /etc/passwd
```

例 2 現在のログイン名を設定する

name に現在のログイン名を設定するには次のようにします。

```
example$ name=`who am i | cut -f1 -d' '`
```

環境

cut の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 入力ファイルはすべて、正常に出力された

>0 エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

grep(1), paste(1), attributes(5) environ(5), largefile(5)

診断

cut: -n may only be used with -b

cut: -d may only be used with -f

cut: -s may only be used with -f

cut: cannot open <*file*>
file が読み取れない、または存在しない。複数のファイル名が存在するときは処理が続行される。

cut: no delimiter specified
-d オプションに区切り文字 *delim* が指定されていない。

cut: invalid delimiter

cut: no *list* specified
オプション -b 、 -c 、または -f に *list* が指定されていない。

```
cut:
invalid
range
specifier

cut:
too
many
ranges
specified

cut:
range
must be
increasing

cut:
invalid
character
in
range

cut:
internal
error
processing
input

cut:
invalid
multibyte
character

cut:
unable
to
allocate
enough
memory
```

名前 | date – 日付と時刻の出力と設定

形式 | **/usr/bin/date** [-u] [+ *format*]

/usr/bin/date [-a [-] *sss.fff*]

/usr/bin/date [-u] [[*mmdd*] | *HHMM* | *mmddHHMM* | [*cc*] | *yy*] [.*SS*]

/usr/xpg4/bin/date [-u] [+ *format*]

/usr/xpg4/bin/date [-a [-] *sss.fff*]

/usr/xpg4/bin/date [-u] [[*mmdd*] | *HHMM* | *mmddHHMM* | [*cc*] | *yy*] [.*SS*]

機能説明 | date ユーティリティは、日付と時刻を標準出力に出力するほか、システムの日付と時刻の設定も行います。 デフォルトでは、現在の日付と時刻を出力します。

月や曜日の名前を各国語に変換する仕様がサポートされています。使用される月や曜日の名前は、環境変数 LC_TIME で指定されたロケールに基づいて決定します (environ(5) 参照) 。

C ロケールのデフォルト形式は次のようになっています。

%a %b %e %T %Z %Y

この形式で日付と時刻を出力すると、次のようになります。

```
Fri Dec 23 10:10:42 EST 1988
```

オプション | 次のオプションを指定できます。

-a [-] *sss.fff* — システムクロックをゆっくり調整します。調整の単位は *sss.fff* 秒です (*fff* は秒の小数部)。 調整は、プラスとともマイナスの両方が可能です。システムクロックは、指定された秒だけ速くまたは遅くなります。日付の調整を行うことができるのはスーパーユーザーだけです。

-u — 日付の表示や設定を行う際に、通常のローカル時刻に変換しないでグリニッジ標準時刻 (GMT — 標準時刻) を使用します。

オペランド | 次のオペランドを指定できます。

+*format* — 引数が + で始まる場合、date の出力は、*format* と現時刻を strftime() に渡すことによって得られる値になります。date は、strftime(3C) のマニュアルページにある変換規則と、 %C の変換規則を使用します。%C の変換規則は、 /usr/bin/date と /usr/xpg4/bin/date のどちらが使われているかにより異なります。

/usr/bin/date	ロケールの日付と時刻の表現。date のデフォルト出力

`/usr/xpg4/bin/date`	世紀を 00 から 99 までの数値で表現 (年を 100 で割って整数に切り捨て)

文字列は常に復帰改行文字で終わります。引数に空白文字が含まれるときは、引用符で囲んでください。詳細については「使用例」の項を参照してください。

mm	月
dd	日
HH	時 (24 時 (間) 表示)
MM	分
SS	秒
cc	西暦 (年の上 2 桁)。つまり、年を 100 で割って端数を切り捨てた数字 (00-99)。たとえば、1988 年の場合、*cc* は `19` で、2007 年の場合、*cc* は `20` です。
yy	年の下 2 桁。*cc* を指定しない場合、「`69-99`」という範囲は「1969 年から 1999 年まで (1969 年と 1999 年を含む)」を示し、「`00-68`」の範囲は「2000 年から 2068 年まで (2000 年と 2068 年を含む)」を示します。

月 (*mm*) 、日 (`dd`) 、年 (*yy*)、世紀 (*cc*) は省略できます。この場合は、現在の値がデフォルト値になります。例を示します。

```
example% date 10080045
```

この例では、日付と時刻が 10 月 8 日午前 12 時 45 分に設定されます。年を指定していないので、現在の年がデフォルトで使用されます。 システムは GMT で動作していますが、`date` はローカルの標準時刻の変換を処理します。日付を変更することができるのは、スーパーユーザーだけです。日付と時刻を正しく設定し終わると、`date` は、デフォルトのフォーマットで新しい日付を表示します。また、`date` コマンドは正しいタイムゾーン情報を決定するために `TZ` を使用しています (`environ`(5) 参照)。

使用例

例 1 出力の生成

実行するコマンドと、その出力例です。

```
example% date '+DATE: %m/%d/%y%nTIME:%H:%M:%S'
DATE: 08/01/76 TIME: 14:45:05
```

例 1 出力の生成　(続き)

例 2 現在の時間の設定

次のコマンドは現在の時間を 12:34:56 に設定します。

```
example# date 1234.56
```

例 3 グリニッジ平均時による現在の時間の設定

```
example# date -u 010100302000
```

次のコマンドは現在の時刻を 2000 年 1 月 1 日、午前 12 時 30 分に設定します。コマンドを実行すると、次のように表示されます。

```
Thu Jan 01 00:30:00 GMT 2000
```

環境

date の実行に影響を与える環境変数 LC_CTYPE 、 LC_TIME 、 LC_MESSAGES 、 NLSPATH の詳細については、 environ(5) を参照してください。

TZ　　-u オプションが省略されたときに、時刻と日付の出力に使用するタイムゾーンを指定します。この TZ 変数が設定されず、 -u も省略されている場合は、システムのデフォルトのタイムゾーンが 使用されます。

終了ステータス

次の終了ステータスが返されます。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/date

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/date

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

strftime(3C)、 attributes(5)、 environ(5)、 XPG4(5)

診断

no permission　　スーパーユーザーではないので日付を変更できません。

bad conversion　　日付設定の構文が正しくありません。

注意事項

標準タイムゾーンから別のタイムゾーンに切り替わる日 (たとえば、夏時間が始まる日または終わる日) を現在の日付とし、標準タイムゾーンの時刻から別のタイムゾーンの時刻に切り替わるときを現在の時刻にしようとした場合の結果は、保証されません。

システムの日時の変更を行うために、ウィンドウ環境で `date` コマンドを使用すると、不具合が発生する可能性があります。この場合の結果は保証されません。また、ウィンドウ環境以外のマルチユーザーモードでも、システムの日時を大幅に変更したりすると、結果が不安定になることがあります。システムの日時を変更するには、コマンド `date -a` を使用することをお勧めします。

名前 | dc – 電卓機能

形式 | **dc** [*filename*]

機能説明 | dc は、任意の精度の算術演算パッケージです。通常 dc は、10 進整数を処理しますが、入力の基数、出力の基数、および少数の桁数を指定することもできます。dc の全体構造は、スタック式 (逆ポーランド式) 計算機です。引数を指定すると、入力は指定したファイルから読み取られ、ファイルの終わりに致達すると標準入力から読み取られます。

bc は、中置記法と、機能を実装するための C 言語スタイルの構文を備えた、dc のプリプロセッサです。bc は、プログラムに対する合理的な制御構造も規定しています。詳細については、bc(1) のマニュアルページを参照してください。

使用法 | 次の構文が認識されます。

数値	数値が、スタックにプッシュされます。数は、0 ～ 9 の数字からなる連続した文字列です。負の数を入力するには、下線を前に付けます。数値には、小数点が含まれる場合があります。
`+ – / * % ^`	スタックの先頭の 2 つの値が、加算 (+)、減算 (–)、乗算 (*)、除算 (/)、剰余計算 (%)、または累乗 (^) されます。2 つの値は、スタックからポップされます。結果は、スタックのその場所にプッシュされます。指数の小数部分は無視されます。
s*x*	スタックの先頭の値がポップされ、レジスタ *x* に格納されます。ここで *x* は任意の文字です。s を大文字にすると、*x* はスタックとみなされ、値はそこにプッシュされます。
l*x*	レジスタ *x* の値がスタックにプッシュされます。レジスタ *x* は変更されません。すべてのレジスタは、最初は 0 に設定されます。l を大文字にすると、レジスタ *x* はスタックとみなされ、その先頭の値がポップされ、メインスタックにプッシュされます。
d	スタックの先頭の値が複製されます。
p	スタックの先頭の値が出力されます。先頭の値はそのままです。
P	スタックの先頭の値を ASCII 文字列として解釈し、それをスタックから削除し、出力します。

`f`	スタック上のすべての値が出力されます。
`q`	プログラムを終了します。文字列を処理している場合、再帰レベルは 2 レベルだけポップします。
`Q`	プログラムを終了します。スタックの先頭の値がポップされ、文字列実行レベルはその値の分ポップします。
`x`	スタックの先頭要素を文字列とみなし、それを `dc` コマンドの文字列として実行します。
`X`	スタックの先頭の値をそのスケールファクタで置き換えます。
`[ ... ]`	角括弧で囲まれた ASCII 文字列をスタックの先頭に入れます。
<x *>x* *=x*	スタックの先頭の要素 2個がポップされ、比較されます。レジスタ *x* は、指定された関係が成立していれば評価されます。
`v`	スタックの先頭の要素をその平方根で置き換えます。引数に小数部分があるときは、それが平方根に反映されますが、スケールファクタは無視されます。
`!`	残りの行をシェルコマンドとして解釈します。
`c`	スタック上のすべての値がポップされます。
`i`	スタックの先頭の値がポップされ、以後の入力の基数として使用されます。
`I`	入力の基数をスタックの先頭にプッシュします。
`o`	スタックの先頭の値がポップされ、以後の出力の基数として使用されます。
`O`	スタックの先頭に出力の基数をプッシュします。
`k`	スタックの先頭の値がポップされ、その値が負ではないスケールファクタとして使用されます。出力の際には、適切な桁数の保持に、また乗算、除算、および累乗の際には、位置取りの保持に使用されます。スケールファクタ、入力の基数、および出力

	の基数の相互作用は、これらすべてを一度に変更した場合には妥当なものになります。
K	現在のスケールファクタをスタックの先頭にプッシュします。
z	スタックレベルをスタックにプッシュします。
Z	スタックの先頭の数をその長さで置き換えます。
?	入力行が入力ソース (通常は端末) から取られて実行されます。
Y	dc デバッグ情報を表示します。
; :	bc(1) が配列演算に使用します。

使用例

例 1 n! の最初の 10 個の値を出力

この例では、n! の最初の 10 個の値を出力します。

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目

bc(1), attributes(5)

診断

x is unimplemented	*x* は 8 進数です。
out of space	空きリストを使い果たしました (桁数が多すぎます)。
out of stack space	スタックへのプッシュが多すぎます (スタックのオーバーフロー)。
empty stack	スタックからのポップが多すぎます (スタックのアンダーフロー)。
nesting depth	ネストが深すぎます。
divide by 0	0 による除算です。
sqrt of neg number	負の数の平方根は定義されていません (虚数は扱えません)。

`exp not an integer`	dc は整数の累乗だけを処理します。
`exp too big`	最大許容指数は 999 です。
`input base is too large`	入力の基数 x: 2<= x <= 16。
`input base is too small`	入力の基数 x: 2<= x <= 16。
`output base is too large`	出力の基数は、`BC_BASE_MAX` 以下でなければなりません。
`invalid scale factor`	スケールファクタは 1 以上でなければなりません。
`scale factor is too large`	スケールファクタは `BC_SCALE_MAX` 以下でなければなりません。
`symbol table overflow`	指定された変数が多すぎます。
`invalid index`	インデックスは 1 以上でなければなりません。
`index is too large`	インデックスは、`BC_DIM_MAX` 以下でなければなりません。

名前 | deroff – nroff/troff、tbl、および eqn 構造体の削除

形式 | **deroff** [-m [m | s | l]] [-w] [-i] [*filename...*]

機能説明 | deroff は、各 *filenames* を順に読み取り、すべての troff(1) 要求、マクロ呼び出し、バックスラッシュ構文、eqn(1) 構文 (.EQ 行と .EN 行の間、および区切り文字の間にある)、および tbl(1) 記述を削除し、それらを空白 (ブランクとブランク行) で置き換えて、ファイルの残りを標準出力に書き込みます。deroff は、インクルードされたファイルのチェーン (.so および .nx troff コマンド) に従います。ファイルがすでにインクルードされている場合、そのファイルを指定する .so は無視されて、そのファイルを指定する .nx は実行を終了します。入力ファイルが指定されていない場合、deroff は標準入力を読み込みます。

オプション | 次のオプションを指定できます。

-m　-m オプションの後には、m、s、または l を続けることができます。-mm オプションを使用するとマクロが解釈されて、実行テキストだけが出力されます (つまり、マクロ行からのテキストは出力されません)。-ml オプションは -mm オプションを強制し、-mm マクロに対応するリストを削除します。

-w　-w オプションが指定されている場合、出力はワードリストです。1 行ごとに 1 ワードが示され、他の文字はすべて削除されます。指定されていない場合、出力は元のものに従い、上記のとおり削除されます。テキスト中の「ワード」は、少なくとも 2 つの文字を含み、英字、数字、アンパサント (&)、およびアポストロフィ (') からなる任意の文字列です。一方、マクロ呼び出しでは、「ワード」は少なくとも 2 文字から始まり、合計で 3 文字以上の文字列です。区切り文字は、英字、数字、アポストロフィ、およびアンパサント以外の任意の文字です。末尾のアポストロフィとアンパサントは、「ワード」から削除されます。

-i　deroff に .so および .nx コマンドを無視させます。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目 | eqn(1), nroff(1), tbl(1), troff(1), attributes(5)

注意事項 | deroff は、完全な troff インタプリタではないため、微妙な構文では混乱が生じることがあります。そのような場合、構文中の実際の問題を示すよりも多くのエラーメッセージ出力されます。

-ml オプションは、入れ子構造になっているリストを正しく処理できません。

dhcpinfo(1)

名前 | dhcpinfo – DHCP を介して受信されたパラメータ値の表示

形式 | **dhcpinfo** [-c] [-i *interface*] [-n *limit*] *code*

dhcpinfo [-c] [-i *interface*] [-n *limit*] *identifier*

機能説明 | dhcpinfo ユーティリティは、コマンド行で要求されたパラメータの DHCP 提供値を出力します。このパラメータは、DHCP 仕様の数値コードか、dhcp_inittab(4) にリストされているニーモニック識別子のどちらかによって識別されます。このコマンドは、システム起動時に init(1M) によって起動されるシェルスクリプトのコマンド置換で使用されます。このコマンドはまず DHCP クライアントデーモンの dhcpagent(1M) と通信して、要求されたインタフェースで DHCP が正常に終了したかどうかを確認します。DHCP が、要求されたインタフェースで正常に終了していれば、dhcpinfo は要求されたパラメータの値を取り出します。dhcpinfo が表示したパラメータ値は、その終了ステータスを検査してから使用します。「終了ステータス」の項を参照してください。

すべての DHCP パラメータのニーモニック識別コードの一覧は、dhcp_inittab(4) のマニュアルページを参照してください。詳細は、『*RFC 2132, DHCP Options and BOOTP Vendor Extensions*』を参照してください。

出力形式 | dhcpinfo からの出力は、1 行または複数行の ASCII テキストからなります。出力の書式は、要求されたパラメータによって異なります。行ごとに返される値の数と、特定のパラメータで出力される行の合計数は、dhcp_inittab(4) で規定されているように、それぞれ、パラメータ指定の精度と最大値によって決まります。

各値の書式は、dhcp_inittab(4) で規定されているオプションのデータ型によって決まります。使用できるデータ型と書式は次のとおりです。

データ型	書式	**dhcp_inittab(4)** 型
符号なし数値	1 つまたは複数の 10 進数	UNUMBER8, UNUMBER16, UNUMBER32, UNUMBER64
符号付き数値	1 つまたは複数の 10 進数、負符号を付けることができる	SNUMBER8, SNUMBER16, SNUMBER32, SNUMBER64
IP アドレス	小数点付きの表記	IP
オクテット	0x の後に 2 桁の 16 進数が続く文字列	*OCTET*
文字列	0 個またはそれ以上の ASCII 文字	ASCII

オプション | 次のオプションを指定できます。

-c　出力を標準の書式で表示します。これは、精度 1 の OCTET 書式と同じです。

`-i` *interface*　DHCP パラメータの値を取り出すインタフェースを指定します。このオプションを指定しないと、一次インタフェースが使用されます。

`-n` *limit*　表示される値のリストを *limit* で指定された行数に制限します。

オペランド

次のオペランドを指定できます。

code　DHCP 仕様で定義されている、要求された DHCP パラメータの数値コード。ベンダーオプションは、実際のベンダーコードに 256 を加算することによって指定されます。

identifier　`dhcp_inittab`(4) にリストされている、要求された DHCP パラメータのニーモニック記号

終了ステータス

次の終了ステータスが返されます。

`0`　正常終了

`2`　処理が正常に終了しなかった。DHCP クライアントデーモンが実行されていない、インタフェースが構成できなかった、あるいは十分な DHCP 応答が受信されなかった。

`3`　無効な引数。

`4`　処理がタイムアウトした。

`6`　起きるはずのないシステムエラーが発生した。

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsr
インタフェースの安定性	Evolving

関連項目

`dhcpagent`(1M), `ifconfig`(1M), `init`(1M), `dhcp_inittab`(4), `attributes`(5)

Alexander, S., and R. Droms, *RFC 2132, DHCP Options and BOOTP Vendor Extensions*, Silicon Graphics, Inc., Bucknell University, March 1997.

名前 | diff – 2つのファイルの比較

形式 | **diff** [-bitw] [-c | -e | -f | -h | -n | -u]*file1 file2*

diff [-bitw] [-C *number* | -U *number*]*file1 file2*

diff [-bitw] [-D *string*] *file1 file2*

diff [-bitw] [-c | -e | -f | -h | -n] [-l] [-r] [-s] [-S *name*] *directory1 directory2*

機能説明 | diff ユーティリティは *file1* と *file2* で指定された 2 つのファイルの内容を比較し、*file1* を *file2* に一致させるのに必要な変更リストを標準出力に書き込みます。変更リストの大きさは最小限に留められます。ごくまれな場合を除き、diff は、必要最低限の相違点を生成するようにします。2 つのファイルの内容が同一であれば、何も出力しません。

通常は、次のようなフォーマットの行を出力します。

n1 a *n3,n4*
n1,n2 d *n3*
n1,n2 c *n3,n4*

このうち *n1* と *n2* は *file1* 中の行を表し、 *n3* と *n4* は *file2* 中の行を表します。 これらの行は *file1* を *file2* へ変換するための ed(1) コマンドに似ています。 a と d とを入れ替えて、各出力行の内容を右から左へ逆向きに読めば、 *file2* を *file1* に変換する方法が確認できます。 ed の場合と同様に、一致するペアは *n1=n2* または *n3 = n4* のようになり、1 つの番号に省略されます。

最初のファイルで影響を受けた行には、'<' が付きます。 2 番目のファイルで影響を受けた行には、'>' が付きます。

オプション | 次のオプションを指定できます。

-b　末尾の空白 (空白文字とタブ文字) を無視し、 他の空白文字列は等価に扱います。

-i　大文字、小文字の区別を無視します。たとえば、 A と a は等しいとみなします。

-t　タブ文字を出力行上で展開します。 通常または -c による出力は、各行の先頭に文字を追加するので、 元の行のインデントに悪影響を与え、 結果として出力行を読みにくくします。 このオプションは、元の行のインデントを保持します。

-w　すべての空白 (空白文字とタブ文字) を無視し、他の空白文字は等価に扱います。 たとえば、'if (a = = b)' は'if(a= =b)' と同じとみなします。

次のオプションは相互に排他的です。 したがって同時に 2 つ以上指定しないでください。

-c　差異行の前後 3 行を追加して出力します。 このオプションを使用すると、出力フォーマットが多少変わります。 つまり、出力は

ファイルの ID と作成日で始まり、各差異部分は * からなる行で区切られます。 *file1* から除外された行には – が、 *file2* に追加された行には + が付きます。 1 つのファイルから他のファイルへ変更された行には、 双方のファイルに ! が付きます。

`-C` *number* 基本的に `-c` オプションと同じです。差異行の前後 *number* 行を追加して出力します。

`-D` *string* C のプリプロセッサの制御によって *file1* と *file2* のマージ版を作成します。 *string* を定義しないでコンパイルすると *file1* のコンパイル結果と同じになり、 *string* を定義すると、 *file2* を生成します。

`-e` *file1* から *file2* を再生するために、 エディタ `ed` の `a` 、 `c` 、 `d` コマンドのスクリプトを生成します。 以下のシェルプログラムは、この `-e` オプションを使用して、複数版のファイルを保守するものです。 元のファイル ($1) と `diff` によって作成される版から版への `ed` スクリプト($2, $3, ...) だけが必要です。最新版が標準出力へ表示されます。

```
(shift; cat $*; echo ´1,$p') | ed – $1
```

`-f` 類似したスクリプト(`ed` では使えない) を反対順に生成します。

`-h` 高速でおおまかな処理をします。 変更部分が短くてうまく区切られているときのみ動作しますが、 ファイル長には制限がありません。オプション `-c` 、 `-C`、`-e` 、 `-f` および `-n` は、このオプションと一緒に使用できません。 このオプションを指定した場合、 `diff` はディレクトリの下の階層を処理しません。

`-n` `-e` と同様のスクリプトを生成しますが、順序は反対で、 各挿入または削除コマンドに変更行数を出力します。

`-u` 差異行の前後 3 行を追加して出力します。`-c` オプションと似ていますが、`-u` オプションの場合、文脈が統合されて出力されます。つまり、*file1* で削除または変更された行には - が付き、*file2* で追加または変更された行には + が付きます。そして、変更された行は両方のファイルとも出力されますが、追加、削除、および同一の行は 1 度だけ出力されます。また、*file1* と *file2* の分別も異なります。-c オプションの場合は *** と ––– が出力されますが、`-u` オプションの場合は ––– と +++ が出力されます。変更された行はそれぞれ次のような行で分離されます。

```
@@ -n1,n2 +n3,n4 @@
```

`-U` *number* 出力形式は `-u` オプションと同じですが、出力する文脈の行数が *number* になります。

次のオプションは、ディレクトリを比較するのに使用します。

`-l` ロングフォーマットで出力を生成します。 `diff` で処理する前に、各テキストファイルは `pr`(1) によってページ付けされます。他の差異は記憶され、 すべてのテキストファイルの差異が報告された後で集計されます。

`-r` サブディレクトリに出合うと、 `diff` を再帰的に実行します。

`-s` 同一のファイルを報告します。 このオプションを指定しないと、同一のファイルは報告されません。

`-S` *name* ファイル *name* からディレクトリ `diff` を開始します。

オペランド

以下のオペランドを指定できます。

file1
file2 比較するファイルまたはディレクトリのパス名。 どちらかに – を指定すると、標準入力とみなされます。

directory1
directory2 比較するディレクトリのパス名。

file1 と *file2* のうちどちらか一方だけがディレクトリの場合、 `diff` が比較するのは、ディレクトリでない方のファイルの内容と、 当該ディレクトリ中のファイルのうち名前がディレクトリでない方の ファイル名の最終要素と同一のファイルの内容です。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `diff` の動作については、`largefile`(5) を参照してください。

使用例

例 **1** diff コマンドの典型的な出力

次のコマンドでは、`dir1` がディレクトリで `x` という名のディレクトリを含んでいて、 `dir2` もディレクトリで `x` という名のディレクトリを含んでいて、 `dir1/x` と `dir2/x` の両方が `date.out` というファイルを含んでいて、さらに `dir2/x` が `y` というファイルを含んでいると仮定します。

以下のような出力が生成されます。

```
example% diff -r dir1 dir2
Common subdirectories: dir1/x and dir2/x

Only in dir2/x: y

diff -r dir1/x/date.out dir2/x/date.out

1c1

< Mon Jul  2 13:12:16 PDT 1990

---

> Tue Jun 19 21:41:39 PDT 1990
```

環境

diff の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 LC_TIME 、 NLSPATH についての詳細は、 environ(5) を参照してください。

TZ　　-C または -c オプションで出力される時刻表示 のタイムゾーン (時間帯) に影響を与えるロケールを指定します。

終了ステータス

以下の終了ステータスが返されます。

0　　一致していた

1　　差異が見つかった

>1　　エラーが発生した

ファイル

/tmp/d?????　　比較に用いる一時ファイル

/usr/lib/diffh　　-h オプション用の実行可能ファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

bdiff(1), cmp(1), comm(1), dircmp(1), ed(1), pr(1), sdiff(1), attributes(5), environ(5), largefile(5)

注意事項

オプション -e または -f 指定で生成される編集用スクリプトは、 ピリオド (.) だけからなる行の作成に関しては慎重です。

Missing NEWLINE at end of file は、問題のファイルの最終行に復帰改行がないことを意味します。 最終行に差異がある場合、フラグが付けられて出力されます。しかし、出力には差異がないように見えます。

名前 | diff3 – 3 つのファイルの内容比較

形式 | **diff3** [-exEX3] *filename1 filename2 filename3*

機能説明 | diff3 はあるファイルの 3 つのバージョンの内容を比較し、異なった内容を持つ範囲を 以下のようなコードを使って通知します。

====	全ファイルが異なる
====1	*filename1* が他の 2 つと異なる
====2	*filename2* が他の 2 つと異なる
====3	*filename3* が他の 2 つと異なる

示された範囲の内容をどのように変更すれば比較対象の 他のファイルの内容と一致させられるかは、次のような方法で表されます。

f : *n1* a	ファイル *f* の行番号 *n1* の後にテキストを追加します。*f* は 1 、 2 、または 3
f : *n1* , *n2* c	行番号 *n1* から *n2* の範囲のテキストを変更します。 *n1* と *n2* が等しい場合には *n1* だけが表示されます。

示された範囲の元の内容が文字 c の後に出力されます。2 つのファイルの内容が同一のときは、 小さい方の番号のファイルの内容は省略されます。

次に示すコマンドは、その実行により生成されたスクリプトを *filename1* に反映させます。

```
(cat script; echo '1,$p') | ed   – filename1
```

オプション

-e	*filename2* と *filename3* の間のすべての相違箇所を *filename1* に取り込んだ、 ed(1) エディタ用スクリプトを生成します。この相違箇所は通常 ==== と ====3 のフラグにより示されます。
-x	==== フラグにより示される相違箇所だけを取り込んだスクリプトを生成します。
-3	====3 フラグにより示される相違箇所だけを取り込んだスクリプトを生成します。
-E	*filename2* と *filename3* の間のすべての相違箇所を取り込んだスクリプトを生成します。 ただし重複した変更箇所 (通常のリスト上では ==== フラグで示される) は、 <<<<<< と >>>>>> で囲んで挿入されます。
-X	==== フラグで示される変更箇所だけを、 -E オプションの場合と同様の方法で取り込んだスクリプトを生成します。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の diff3 の動作については、largefile(5) を参照してください。

ファイル

/tmp/d3*

/usr/lib/diff3prog

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

diff(1), attributes(5), largefile(5)

注意事項

1 つのピリオド '.' だけからなるテキスト行は -e オプションの処理対象とはなりません。

diff3 は 64 キロバイトを超える大きさのファイルは処理できません。

名前 | diffmk – troff 入力ファイル間の差分の表示

形式 | **diffmk** *oldfile newfile markedfile*

機能説明 | diffmk は、2 つのバージョンのファイルを比較して、nroff(1) および nroff(1) に対する変更マーク (.mc) コマンドを含む第 3 のバージョンを作成します。*oldfile* と *newfile* は、ファイルの新旧のバージョンを示します。diffmk は、*newfile* と *oldfile* が異なる場合に、troff(1) の変更マーク要求 (.mc) を挿入した *newfile* からテキスト形式の *markedfile* を生成します。*markedfile* が書式化されたときは、変更されたり挿入されたテキストの各行の右端に | が表示されます。削除されたテキストの位置には、1 つの * が表示されます。

使用法 | ファイルのサイズが 2G バイト (2^{31} バイト) 以上ある場合の diffmk の動作については、largefile(5) のマニュアルページを参照してください。

使用例 | 例 **1** diffmk コマンドの使用例

diffmk を適切な troff 要求と共に使用して、変更がマークされたプログラムリストを作成することもできます。次のコマンド行を見てください。

```
example% diffmk old.c new.c marked.c ; nroff reqs marked.c | pr
```

ファイル reqs には、次の troff 要求が記述されています。

```
.pl  1
.ll  77
.nf
.eo
.nh
```

これらの要求はそれぞれ、改ページを削除し、行の長さを調整し、「詰め込みなし」モードを設定し、エスケープ文字を無視して、ハイフネーションを無効にします。

文字 | と * が不適切な場合は、sed(1) を使用して *markedfile* を全体的に変更できます。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目 | diff(1), nroff(1), sed(1), troff(1), attributes(5), largefile(5)

使用上の留意点 | 体裁を整えるために、一部の出力を手動で調整しなければならない場合があります。書式に関する要求だけに差違がある場合、期待するような出力が得られないことがあります。たとえば、.sp が .sp 2 に変更されている場合、出力行の前後の行に変更マークが付けられることがあります。

名前

mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式

mailp [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明

`mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション

次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D`	出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。
`-F`	メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。
`-h`	バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。
`-l`	横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。
`-P` *printer*	`-d` オプションを指定した場合と同じです。
`-s` *subject*	*subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename	読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0`	正常終了
`1`	エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

名前 | dircmp – ディレクトリの比較

形式 | **dircmp** [-ds] [-w *n*] *dir1 dir2*

機能説明 | dircmp コマンドは、*dir1* と *dir2* を調べて、ディレクトリの内容についての情報を表形式で生成します。各ディレクトリに固有なファイルのリストが、すべてのオプションについて生成されます。オプションを指定しないと、両方のディレクトリに共通して存在するファイルの内容が同じかどうかを示すリストが出力されます。

オプション | 次のオプションを指定できます。

-d　両方のディレクトリで同じ名前を持つファイルの内容を比較して、2 つのファイルを一致させるために変更すべき箇所のリストを出力します。リストの形式については、diff(1) のマニュアルページを参照してください。

-s　同じファイルについてのメッセージを抑制します。

-w *n*　出力行の幅を *n* 文字に変更します。デフォルトの幅は 72 文字です。

オペランド | 次のオペランドを指定できます。

dir1　*dir2* と比較するディレクトリのパス名

dir2　*dir1* と比較するディレクトリのパス名

使用法 | ファイルのサイズが 2G バイト (2^{31} バイト) 以上ある場合の dircmp の動作については、largefile(5) のマニュアルページを参照してください。

環境 | dircmp の実行に影響を与える環境変数 LC_COLLATE、 LC_CTYPE、 LC_MESSAGES、 NLSPATHについては、environ(5) のマニュアルページを参照してください。

終了ステータス | 次の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した (ディレクトリの内容の違いはエラーとはみなされない)

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | cmp(1), diff(1), attributes(5), environ(5), largefile(5)

名前 | basename, dirname – パス名の部分的な抽出

形式 | **/usr/bin/basename** *string* ［*suffix*］

/usr/xpg4/bin/basename *string* ［*suffix*］

dirname *string*

機能説明 | basename ユーティリティは、/ で終わるすべてのプレフィックスと suffix (*string* 中にある場合) を *string* から削除して、その結果を標準出力に出力します。このユーティリティは通常、シェルプロシージャ内の置換マーク (‘’) の中で使用されます。

/usr/bin | *suffix* は、expr(1) のマニュアルページに定義されているパターンです。

/usr/xpg4/bin | *suffix* は、含まれるどの文字にも特殊な意味が付加されていない文字列です。

dirname ユーティリティは、*string* 内のパス名の最後のレベルを除く、すべてのレベルを出力します。

使用例 | 例 **1** 環境変数を設定する

次の例は、引数 /home/sms/personal/mail を指定して呼び出しを行なった場合で、環境変数 NAME に mail というファイルを設定し、環境変数 MYMAILPATH に文字列 /home/sms/personal を設定します。

```
example% NAME=`basename  $HOME/personal/mail`
example% MYMAILPATH=`dirname  $HOME/personal/mail`
```

例 **2** ファイルをコンパイルして出力を移動する

次のシェルプロシージャは、引数 /usr/src/bin/cat.c を指定して呼び出しを行なった場合で、指定したファイルをコンパイルし、出力を現在のディレクトリ内の cat というファイルに移動します。

```
example% cc $1
example% mv a.out `basename $1 .c`
```

環境 | basename と dirname の実行に影響を与える環境変数 LC_CTYPE, LC_MESSAGES、 および NLSPATH については、environ(5) のマニュアルページを参照してください。

終了ステータス | 次の終了ステータスが返されます。

0　　正常終了

>0　　エラーが発生した

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

/usr/bin

属性タイプ	属性値
使用条件	SUNWcsu

/usr/xpg4/bin

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目 `expr`(1), `attributes`(5), `environ`(5), `XPG4`(5)

名前 | cd, chdir, pushd, popd, dirs – 現在の作業用ディレクトリの変更

形式 | **/usr/bin/cd** [*directory*]

sh | **cd** [*argument*]

chdir [*argument*]

csh | **cd** [*dir*]

chdir [*dir*]

pushd [+ | *n* | *dir*]

popd [+ *n*]

dirs [-l]

ksh | **cd** [*arg*]

cd *old new*

/usr/bin/cd | /usr/bin/cd ユーティリティは、cd ユーティリティ自身だけの現在のディレクトリを変更します。これは、後述するシェル組み込みの cd とは対照的です。/usr/bin/cd はプロセスの呼び出しには影響しませんが、あるディレクトリを現在のディレクトリとして設定できるかどうかを決定するのに使用できます。

sh | Bourne シェルに組み込まれている cd は、現在のディレクトリを *argument* で指定されたディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *argument* になります。 シェル変数 CDPATH は、 *argument* を含むディレクトリの検索パスを定義します。代替ディレクトリ名は、 コロン (:) で区切ります。 デフォルトのパスは空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは空のパス名で指定します。 このパス名は、等号の直後か、 パスリスト内にある区切り文字のコロンの間に指定します。*argument* の先頭文字が / 、 . 、または .. の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *argument* を検索します。 cd は、 *argument* 中で実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は効率が悪くなります。 そのため、 cd コマンドは、シェルに組み込まれています。 (pwd(1) 、 sh(1) 、 chdir(2) を参照)

chdir は、cd を呼び出すもうひとつの方法です。

csh | *dir* 引数を省略すると、 C シェルに組み込まれている cd は、シェル変数 HOME の値を新たな作業用ディレクトリとして 使用します。 *dir* を指定した場合、それが / 、 . 、または .. で始まる完全なパス名であれば、その *dir* が新たな作業用ディレクトリとなります。 それ以外の場合は、シェル変数 CDPATH が指定するパスと相対関係を持つディレクトリの中から 該当するものを探し出します。 CDPATH の構文は PATH シェル変数と同一で、セマンティクスも似ています。 cd は *dir* に対する実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、C シェルに組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

chdir はシェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。 引数を指定しないと、 そのユーザーのホームディレクトリに変更します。 *dir* が現在のディレクトリからは見つからない相対パス名の場合、変数 cdpath 内のディレクトリリストを検索します。 *dir* が / で始まる値を持つシェル変数の名前である場合、 その変数の値が示すディレクトリに変更します。

pushd はディレクトリスタックにディレクトリをプッシュ (押し込む) します。 引数を指定しないと、スタックにある先頭の 2 つの構成要素を交換します。

+*n* *n* 番目のエントリがスタックの先頭になるよう回転し、 そのディレクトリに移ります。

dir 現在の作業用ディレクトリをスタックにプッシュし、 そのディレクトリに移ります。

popd はディレクトリスタックからポップして (取り出して)、新たに先頭となったディレクトリへ cd します。ディレクトリスタックの構成要素の先頭番号は、 0 となります。

+*n* スタック内の *n* 番目のエントリを破棄します。

dirs はディレクトリスタックを出力します。 現在のディレクトリが最も左に現れるように 時間順に出力されます。 -l 引数を指定すると、 ~ を使った省略形ではなく、完全な形式で出力されます。

ksh Korn シェルに組み込まれた cd コマンドは、上記 2 つの形式の いずれかで入力します。第 1 の形式は、現在のディレクトリを *arg* に変更します。 *arg* が – の場合、ディレクトリを直前のディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *arg* になります。PWD 変数は、現在のディレクトリに設定されます。シェル変数 CDPATH は、*arg* を含むディレクトリの検索パスを定義します。 ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは NULL のパス名で指定します。 このパス名は、 等号の直後か、パスリスト内にある区切り文字のコロンの間に指定します。 *arg* の先頭文字が / 、 . 、 または . . の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *arg* を検索します。 cd の第 2 の形式は、 PWD 中の現在のディレクトリ名における *old* という文字列を *new* という文字列に置換し、 この新規のディレクトリへ変更しようとします。 cd コマンドは rksh では実行できません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、 ksh に組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

オペランド 以下のオペランドを指定できます。

directory 新たな作業用ディレクトリとなるディレクトリの絶対または相対パス名。 cd が相対パス名をどのように解釈するかは、環境変数 CDPATH の設定により異なります。

出力 CDPATH に設定されている空でないディレクトリ名が用いられる場合、 新たな作業用ディレクトリの絶対パス名が 以下のような形式で標準出力に出力されます。

"%s\n", <*new directory*>

それ以外の場合には、何も出力されません。

環境

cd の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

CDPATH　コロンで区切られた、ディレクトリを示すパス名のリスト。 *directory* オペランドの先頭文字がスラッシュ (/) でなく、 先頭部分が . でも .. でもない場合には、 cd はこのリスト内のパス名を順番に検索し、環境変数 CDPATH に指定されている名前のディレクトリから *directory* を探します。その結果、最初に見つかったディレクトリ名が 新たな作業用ディレクトリとなります。 ディレクトリのパス名として空の文字列を指定すると、 それは現在のディレクトリと見なされます。 CDPATH は、設定されていないときには空の文字列として扱われます。

HOME　*directory* オペランドが省略されたときに用いるホームディレクトリの名前

PWD　現在の作業用ディレクトリのパス名。 この変数は、そのディレクトリに移った後に cd により設定されます。

終了ステータス

以下の終了ステータスが返されます。

0　ディレクトリが正常に変更された。

>0　エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), pwd(1), sh(1), chdir(2), attributes(5), environ(5)

名前 | dis – オブジェクトコードの逆アセンブラ

形式 | **/usr/ccs/bin/dis** [-C] [-o] [-V] [-L] [-d *sec*] [-D *sec*] [-F *function*] [-l *string*] [-t *sec*] *file...*

機能説明 | dis コマンドは、*file* のアセンブリ言語リストを作成します。*file* は、オブジェクトファイルでもオブジェクトファイルのアーカイブでもかまいません。このリストには、アセンブリ文と、それらの文を生成したバイナリコードの 8 進数表記または 16 進数表記が含まれます。ただし、IA64 の場合、リストにはアセンブリ文だけが含まれます。

オプション | 次のオプションは、逆アセンブラによって解釈され、任意の順序で指定できます。

-C　逆アセンブリで復号化された C++ シンボル名を表示します。

-d *sec*　指定されたセクションをデータとして逆アセンブルし、セクションの始めからのデータのオフセットを出力します。

-D *sec*　指定されたセクションをデータとして逆アセンブルし、データの実アドレスを出力します。

-F *function*　コマンド行に指定された各オブジェクトファイルの指定された関数だけを逆アセンブルします。-F オプションは、コマンド行で複数回指定できます。

-l *string*　*string* で指定されたアーカイブファイルを逆アセンブルします。たとえば、コマンド dis -l x -l z を発行して、LIBDIR にあると想定される libx.a と libz.a を逆アセンブルします。

-L　以降の出力を標準出力へ書き込むために、シンボルテーブル内の C 言語ソースラベルの検索を行います。

-o　数字を 8 進数で出力します。デフォルトでは 16 進数です。

-t *sec*　指定されたセクションをテキストとして逆アセンブルします。

-V　実行中の逆アセンブラのバージョン番号を標準エラー出力に出力します。

-d、-D、または -t オプションを指定した場合、ユーザーが指定した各ファイルの指定されたセクションだけが逆アセンブルされます。指定しないと、テキストを含むすべてのセクションが逆アセンブルされます。

出力において、[5] のように、行の始めにある角括弧で囲まれた数字は、ブレークポイント可能な行番号が次の命令から始まることを示します。これらの行番号は、ファイルがデバッグ情報を追加してコンパイルされた (たとえば cc(1B) の -g オプション) 場合にのみ出力されます。制御転送命令用の相対変位の後にある、オペランドフィールドまたはシンボリック逆アセンブリの <40> などの式は、セクション内の計算されたアドレスであり、ここに制御が渡されます。オブジェクトファイルにシンボルテーブルが含まれている場合、関数名は最初のカラムに表示され、その後に () が続きます。

オペランド | 次のオペランドを指定できます。

file オブジェクトファイルまたはオブジェクトファイルのアーカイブ (ar(1) のマニュアルページを参照) のパス名。

環境

dis の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH については、environ(5) のマニュアルページを参照してください。

LIBDIR この環境変数に値が設定されている場合は、それをライブラリ検索用のパスとして使用します。この環境変数に NULL の値が設定されている場合、あるいは、この環境変数が設定されていない場合は、デフォルトで /usr/lib にあるライブラリが検索されます。

終了ステータス

次の終了ステータスが返されます。

0 正常終了

>0 エラーが発生した。

ファイル

/usr/lib デフォルトの LIBDIR

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWbtool

関連項目

ar(1), as(1), cc(1B), ld(1), a.out(4), attributes(5), environ(5)

診断

コマンド行中のエラー、または指定したファイルで検出された問題は、診断メッセージに示されます。

名前 | enable, disable – LP プリンタを使用可能または不可能に変更

形式 | **/usr/bin/enable** *printer*...

/usr/bin/disable [-c | -W] [-r [*reason*]] *printer*...

機能説明 | enable コマンドは、 プリンタを起動して、 lp コマンドで投入される要求を印刷できるようにします。 enable はプリンタサーバー上で動作します。

disable コマンドは、 プリンタを使用不可能にして、 lp コマンドによる要求を印刷できないようにします。 デフォルトでは、 *printer* 上で disable が起動されているとき、印刷を実行していた要求は、 使用可能になると *printer* または同じクラスに属する 他のプリンタ上ですべて再印刷されます。 disable はプリンタサーバー上で動作します。

なお、プリンタの状況を調べるには、 lpstat -p コマンドを実行してください。

enable と disable コマンドは、プリンタサーバーのスプール用システム上の 待ち行列に対してのみ有効です。 これらのコマンドをクライアントシステムから 実行しても、サーバー上には何も起こりません。

オプション | 以下のオプションを指定できます。

-c *printer* 上で現在印刷されている要求も キャンセルします。このオプションは -W と一緒に指定することはできません。 プリンタがリモートの場合には、 -c オプションは無視されますが、 それに関するメッセージは出力されません。

-W *printer* 上で現在印刷されている要求が 終了するのを待って、 *printer* を使用不可能にします。 このオプションは、 -c と一緒に指定することはできません。 プリンタがリモートの場合には、 -W オプションは無視されますが、 それに関するメッセージは出力されません。

-r [*reason*] プリンタを使用不可能にする理由を *reason* 引数で文字列として記述します。 複数のプリンタを指定した場合、そのすべてにこの理由が適用されます。 ここで指定した理由は、プリンタの状況を調べる lpstat -p コマンドの出力中に表示されます。 理由を示す文字列は、 空白文字を含んでいる場合には引用符で囲まなければなりません。 理由のデフォルト値は、既存の宛先に関しては unknown reason 、システムに追加されてまだ使用可能になったことのない宛先に関しては new destination となります。

オペランド | 以下のオペランドを指定できます。

printer 使用可能または不可能にするプリンタ名。 名前を使用して *printer* を指定します。 名前の命名規約については printers.conf(4) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 正常終了

0 以外　　エラーが発生した

ファイル　/var/spool/lp/*　　LP 印刷待ち行列

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpcu
CSI	対応済み

関連項目　lp(1), lpstat(1), printers.conf(4), attributes(5)

名前 | dispgid – すべての有効なグループ名リストの表示

形式 | **dispgid**

機能説明 | `dispgid` は、システム上のすべてのグループ名のリストを表示します (行ごとに 1 つのグループ)。

終了ステータス | 次の終了ステータスが返されます。

`0` 正常終了

`1` グループファイルが読み取れなかった。

属性 | 次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | `attributes`(5)

dispuid(1)

名前 | dispuid – すべての有効なユーザー名リストの表示

形式 | **dispuid**

機能説明 | `dispuid` は、システム上のすべてのユーザー名のリストを表示します (名前ごとに 1 行)。

終了ステータス | 次の終了ステータスが返されます。

`0` 正常終了

`1` パスワードファイルが読み取れなかった。

属性 | 次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | `attributes`(5)

名前 dos2unix – DOS 形式から ISO 形式へのテキストファイルの変換

形式 **dos2unix** [-ascii] [-iso] [-7] [-437 | -850 | -860 | -863 | -865] *originalfile convertedfile*

機能説明 dos2unix ユーティリティは DOS 拡張文字セットの文字を、対応する ISO 標準文字に変換します。

このコマンドは、DOS と SunOS のどちらからでも起動できます。ただしファイル名は、コマンドが起動された環境の命名規則を満たさなければなりません。

元のファイルと変換されたファイルが同じである場合、dos2unix は、変換後のファイルで元のファイルを上書きします。

オプション 次のオプションを指定できます。

-ascii 余分なキャリッジリターンを削除して、DOS 形式のテキストファイル内のファイルの終わり文字を、SunOS の要件に合うように変換します。

-iso デフォルト値です。DOS 拡張文字セットの文字を、対応する ISO 標準文字に変換します。

-7 8 ビットの DOS グラフィックス文字を 7 ビット空間文字に変換して、SunOS がファイルを読み取れるようにします。

非 i386 システム上では、dos2unix はキーボードの種類を取得して、使用するコードページを決定します。そうでない場合、キーボードの種類はデフォルトで US です。使用するコードページを無効にするには、次のオプションを使用します。

-437 米国英語のコードページを使用します。

-850 複数言語のコードページを使用します。

-860 ポルトガル語のコードページを使用します。

-863 フランス系カナダ語のコードページを使用します。

-865 デンマーク語のコードページを使用します。

オペランド 次のオペランドは必須です。

originalfile ISO 形式に変換される DOS 形式の元のファイル

convertedfile DOS 形式の元のファイルから変換された ISO 形式の新しいファイル

属性 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | `unix2dos`(1), `ls`(1), `attributes`(5)

診断 | `File` *filename* `not found, or no read permission`
指定した入力ファイルが存在しないか、読み取り権が与えられていません。SunOS の `ls -l` コマンド (`ls`(1) のマニュアルページを参照) で検査します。

`Bad output filename` *filename*`, or no write permission`
指定された出力ファイルが無効であるか、あるいは、そのファイルまたはそのファイルを含むディレクトリの書き込み権がありません。また、ドライブまたはフロッピーディスクが書き込み保護されていないか検査してください。

`Error while writing to temporary file`
現在のドライブに十分な領域がないために、ファイルの変換中にエラーが発生しました。DIR コマンドを使用して、現在のドライブ上の領域を検査してください。また、デフォルトのフロッピーディスクまたはドライブが書き込み可能であることを (書き込み保護されていないことを) 確認してください。このエラーが発生した場合、元のファイルは変更されません。

`Translated temporary file name =` *filename*.
`Could not rename temporary file to` *filename*`.`
プログラムが、ファイル変換の最終段階を実行できませんでした。変換されたファイルは、上記のメッセージの 2 行目に示された名前で保存されます。

名前 | download – ホスト上の PostScript フォントのダウンローダ

形式 | **download** [-f] [-p *printer*] [-m *name*] [-H *directory*] [*files*...]

/usr/lib/lp/postscript/download

機能説明 | download は、入力ファイル (*files*) の先頭にホスト上のフォントを追加して、結果を標準出力に書き込みます。*files* を指定しないか、または – が *files* の 1 つである場合、標準入力が読み取られます。download は、入力ファイルが 1 つの PostScript ジョブを構成し、要求されたフォントを各入力ファイルの先頭に含めることができるものと想定します。

要求されたフォントは、入力ファイル (*files*) 中の注釈 (%%DocumentFonts: によってマークされます) で指定されます。使用可能なフォントは、-m オプションを使用して選択されたマップテーブルにリストされているものです。

マップテーブルは、フォント名とファイルのペアからなります。フォント名は、%%DocumentFonts: 注釈に示されたとおりの PostScript フォントの完全名です。ファイルは、ホスト上のフォントのパス名です。/ で始まるファイルはそのまま使用されます。それ以外の場合、パス名はホストフォントディレクトリからの相対パスです。注釈は (PostScript の場合と同様に) % で始まり、行の終りまで続きます。

ダウンロードできるのは、download に読み取り可能なファイルを指し示すマップテーブルにリストされているフォントだけです。フォントは 1 度だけダウンロードされます。リストされていないフォントまたはアクセス不可能なファイルに対する要求は無視されます。マップテーブルを読み取ることができない場合は、すべての要求が無視されます。

オプション

-f	各入力ファイルの完全な検索を強制します。ファイルの終わりを示す明示的な注釈がない場合、デフォルトの検索は PostScript ヘッダー注釈のすぐ後で停止します。
-p *printer*	ダウンロードの前に、/etc/lp/printers/*printer*/residentfonts 内のプリンタ常駐フォントのリストを検査します。
-m *name*	フォントマップテーブルとして *name* を使用します。/ で始まる *name* は、マップテーブルの完全パス名であり、そのまま使用されます。そうでなければ、*name* は、ホストフォントディレクトリのパス名に追加されます。
-H *directory*	ホストフォントディレクトリとして *dir* を使用します。デフォルト値は /usr/lib/lp/postscript です。

使用例 | **例 1** download コマンドの使用例

次のマップテーブルは、Bookman フォントファミリのダウンロードを制御するために使用できます。

```
%
% The first string is the full PostScript font name. The second string
% is the file name - relative to the host font directory unless it begins
% with a /.
```

例 1 download コマンドの使用例　(続き)

```
  %

    Bookman-Light               bookman/light
    Bookman-LightItalic      bookman/lightitalic
    Bookman-Demi                bookman/demi
    Bookman-DemiItalic       bookman/demiitalic
```

ファイル myprinter/map (デフォルトのホストのフォントディレクトリにある) をマップテーブルとして使用すると、次のコマンドを発行してフォントをダウンロードできます。

```
example% download -m myprinter/map file
```

終了ステータス

次の終了ステータスが返されます。

0	正常終了
0 以外の値	エラーが発生した。

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpsf

関連項目

dpost(1), postdaisy(1), postdmd(1), postio(1), postmd(1), postprint(1), posttek(1), attributes(5)

注意事項

download プログラムは、より一般的なプログラムの一部として使用しなければなりません。

download は %%PageFonts: 注釈を検索しないため、特定フォントを複数回ダウンロードさせる方法はありません。

マップテーブルまたはマップテーブルの名前のどちらかで完全パス名を使用することはお勧めできません。

名前 | dpost – PostScript プリンタ用の troff ポストプロセッサ

形式 | **dpost** [-c *num*] [-e *num*] [-m *num*] [-n *num*] [-o *list*] [-w *num*] [-x *num*] [-y *num*] [-F *dir*] [-H *dir*] [-L *file*] [-O] [-T *name*] [*files*...]

/usr/lib/lp/postscript/dpost

機能説明 | dpost は、troff(1) によって作成された入力ファイル (*files*) を PostScript に変換して、結果を標準出力に書き込みます。入力ファイル (*files*) を指定しないか、または – が *files* の 1 つである場合は、標準入力が読み取られます。

files は、troff によって作成する必要があります。/usr/lib/font/devpost にあるデフォルトのフォントファイルは、最も適切で効率的な出力を作成します。これらのフォントファイルは、720 dpi の解像度を前提としており、-Tpost オプションを付けて troff を呼び出すことによってファイルのフォーマットに使用できます。以前のバージョンの eqn および pic プリプロセッサには、troff が *files* のフォーマットに使用する解像度を認識させる必要があります。使用しているシステムにこれらのバージョンがインストールされている場合は、eqn では -r720 オプションを、pic では -T720 を使用してください。

dpost は、解像度を想定していません。最初の x res コマンドは、入力ファイル DESC.out ファイルの変換に使用する解像度を設定します。一般に、/usr/lib/font/devpost/DESC.out は、バイナリフォントファイルに使用する解像度を定義します。また、PostScript プロローグは、適切なユーザー座標システムの設定を行います。

オプション

-c *num* — 各ページのコピーを *num* 部ずつ出力します。デフォルトでは、コピーは 1 部だけ出力されます。

-e *num* — テキスト符号化レベルを *num* に設定します。認識される選択肢は 0、1、および 2 です。出力ファイルのサイズと出力時間は、*num* の値が増加すると減少します。レベル 2 の符号化は、一般にレベル 0 よりも約 20 パーセント高速です。レベル 0 がデフォルトであり、前のバージョンの dpost と基本的に同じ出力を作成します。

-m *num* — 各論理ページを、*num* で指定した係数で拡大します。ページは、各ページの左上端付近にある原点に対して均等にスケール変更されます。デフォルトの拡大率は 1.0 です。

-n *num* — 用紙 1 枚ごとに *num* で指定した数の論理ページを出力します。ここで、*num* には任意の正の整数を指定できます。デフォルトでは、*num* は 1 に設定されます。

-o *list* — カンマで区切られた *list* 内で番号が与えられているページを出力します。このリストには、単一の番号 *N* と *N1* ～ *N2* の範囲が含まれます。*N1* がなければ最小番号のページを意味し、*N2* がなければ最大番号のページを意味します。ページ範囲は、物理的な用紙ではなく、論理ページを表現したものです。たとえば、2 つの論理ページを用紙に出力するときに、範囲 4 を指定した場合は、4 ページのレイアウトを含む 2 枚の用紙が出力されます。ページ範

囲を 3 ～ 4 に指定して、2 つの論理ページを 1 枚の用紙に要求すると、ページ 3 と 4 のレイアウトだけが一枚の用紙に出力されます。

`-p` *mode*
: portrait (縦方向) または landscape (横方向) のどちらかの *mode* でファイルを出力します。*mode* の最初の文字だけが有効です。デフォルトのモードは portrait です。

`-w` *num*
: `troff` グラフィックスコマンドを *num* ポイントに導入するために使用するライン幅を設定します。ポイントは、1 インチの約 1/72 です。デフォルトでは、*num* は `0.3` ポイントに設定されます。

`-x` *num*
: 原点を正の x 軸に沿って *num* インチだけ変換します。デフォルトの座標システムでは、原点はページの左上端に固定されていて、正の x はページの右側、正の y はページの左側になります。正の *num* は、すべてを右に移動します。デフォルトのオフセットは `0` インチです。

`-y` *num*
: 原点を正の y 軸に沿って *num* インチだけ変換します。正の *num* はテキストをページ内で上に移動します。デフォルトのオフセットは `0` インチです。

`-F` *dir*
: *dir* をフォントディレクトリとして使用します。デフォルトの *dir* は `/usr/lib/font` であり、`dpost` はディレクトリ `/usr/lib/font/devpost` からバイナリフォントファイルを読み込みます。

`-H` *dir*
: *dir* をホスト上のフォントディレクトリとして使用します。このディレクトリのファイルは、完全な PostScript フォント記述でなければならず、適切な 2 文字の `troff` フォント名に対応する名前を割り当てる必要があります。各フォントファイルは、必要な場合にのみ、各ジョブの実行中に 1 回だけ出力ファイルにコピーされます。デフォルトのディレクトリはありません。

`-L` *file*
: *file* を PostScript プロローグとして使用します。デフォルトは `/usr/lib/lp/postscript/dpost.ps` です。

`-O`
: PostScript ピクチャーのインクルードを無効にします。ネットワーク化された環境でスプーラが `dpost` を実行する場合に推奨されるオプションです。

`-T` *name*
: デバイス *name* のフォントファイルを、使用可能な PostScript フォントの最善の記述として使用します。デフォルトでは、*name* は `post` に設定され、`dpost` は `/usr/lib/font/devpost` からバイナリファイルを読み込みます。

使用例

例 1 `dpost` コマンドの使用例

使用しているシステム上に旧バージョンの `eqn` および `pic` がインストールされている場合は、次のようなコマンド行によって、最善の体裁の出力を得ることができます。

例 1 dpost コマンドの使用例　(続き)

```
example% pic -T720 file | tbl | eqn -r720 | troff -mm -Tpost | dpost
```

あるいは、

```
example% pic file | tbl | eqn | troff -mm -Tpost | dpost
```

このコマンド行でも最善の結果が得られます。

終了ステータス

次の終了ステータスが返されます。

0　　正常終了

0 以外の値　　エラーが発生した

ファイル

/usr/lib/font/devpost/*.out

/usr/lib/font/devpost/charlib/*

/usr/lib/lp/postscript/color.ps

/usr/lib/lp/postscript/draw.ps

/usr/lib/lp/postscript/forms.ps

/usr/lib/lp/postscript/ps.requests

/usr/lib/macros/pictures

/usr/lib/macros/color

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpsf

関連項目

download(1), postdaisy(1), postdmd(1), postio(1), postmd(1), postprint(1), postreverse(1), posttek(1), troff(1), attributes(5)

注意事項

出力ファイルが、Adobe のファイル構造規則に準拠していないことがよくあります。dpost の出力をパイプを介して postreverse(1) へ渡すと、PostScript ファイルに最小限準拠するファイルが作成されます。

dpost は、どのデバイスに合わせてフォーマットされたファイルでも処理できますが、エミュレーションは負荷の重い処理であり、出力時間と出力ファイルのサイズは簡単に倍増してしまいます。そのため、troff がサポートするすべてのデバイスで使用できる文字セット、またはフォントを実装しようとする試みは、行われていません。欠落している文字は空白で置き換えられ、認識不能なフォントは通常、Times フォント (R、I、B、または BI) のいずれかにデフォルト設定されます。

`x res` コマンドは最初の `x init` コマンドの前になければならず、また、すべての入力ファイル (*files*) を同じ出力デバイスに合わせて用意する必要があります。

`-T` オプションは使用しないようにしてください。このオプションの唯一の目的は、異なる解像度、文字セット、またはフォントを使用する他の PostScript フォントとデバイス記述ファイルを使用できるようにすることです。

レベル 0 の符号化は完全にテストされている唯一のコード体系ですが、レベル 2 は高速であり、試してみる価値があります。

名前 | du – ディレクトリまたはファイルごとに使用されるディスクブロックの数の表示

形式 | **/usr/ucb/du**

/usr/ucb/du [-a] [-s] [*filename*]

機能説明 | du ユーティリティは、 *filename* で指定した各ディレクトリまたはファイルの中の すべてのファイルおよびディレクトリについて 、再帰的に、その容量をキロバイト数で表示します。 *filename* が省略されている場合は、 '.' (現在のディレクトリ) を使用します。複数のリンクを持つファイルは、 1 度しかカウントされません。

オプション | 次のオプションを指定できます。

-a　各ファイルに対してエントリを生成します。

-d　ファイルシステムの境界を越える検査は行いません。たとえば、du -d / はルートパーティション (/) の使用率だけを報告します。

-k　ファイルのサイズをデフォルトの 512 バイト単位ではなく 1024 バイト単位で書き出します。

-L　シンボリックリンクを処理するときに、シンボリックリンク自身ではなく、シンボリックリンクが参照するファイルまたはディレクトリを使用します。

-o　子ディレクトリの使用率を親ディレクトリの合計に追加しません。このオプションを指定しない場合、親ディレクトリの使用率には、親ディレクトリ直下にあるファイルに加え、親ディレクトリの下にあるすべての子ディレクトリのファイルが追加されます。-s オプションを指定すると、このオプションは無効になります。

-r　読み取れないディレクトリやオープンできないファイルについてメッセージを生成します。このオプションを指定しないと、メッセージは生成されません。

-s　指定した *filename* それぞれのについての総合計のみを表示します。

オプションがない場合には、 各ディレクトリに対してのみエントリを生成します。

使用例 | 例 **1** ディレクトリの中で du を使用

ここに示したのは、ディレクトリの中で du を使用した例です。 pwd(1) コマンドを使用してディレクトリを識別し、そのディレクトリにおけるすべてのサブディレクトリの使用を表示しました。 ディレクトリの総合計は、ディスプレイの最後のエントリです。

```
example% pwd
/usr/ralph/misc
example% du
5       ./jokes
33      ./squash
44      ./tech.papers/lpr.document
217     ./tech.papers/new.manager
401     ./tech.papers
```

例 1 ディレクトリの中で du を使用　(続き)

```
144     ./memos
80      ./letters
388     ./window
93      ./messages
15      ./useful.news
1211    .
example%
```

環境

LC_* 変数、つまり、LC_CTYPE、LC_MESSAGES、LC_TIME、LC_COLLATE、LC_NUMERIC、LC_MONETARY (environ(5) 参照) のいずれも環境に設定されていなければ、 それぞれ対応する ロケール の範疇における du の動作は、環境変数 LANG によって決定されます。もし、 LC_ALL が設定されていれば、その内容が LANG 変数やその他の LC_* 変数より優先されます。上記の変数が環境にまったく設定されていなければ、 C ロケール (米国スタイル)が du の動作を決定します。

LC_CTYPE　du の文字の処理方法を決定します。 LC_CTYPE に有効な値が設定されていると、 du は、その ロケール にあった文字を含むテキストやファイル名を表示および処理できます。 du は拡張 Unix コード(EUC)も表示および処理できます。 この場合、文字は 1 バイト幅、 2 バイト幅、 3 バイト幅のいずれも使用できます。 また、 du は 1 、 2 、 またはそれ以上のカラム幅の EUC 文字も処理することができます。 C ロケール においては、 ISO 8859-1 の文字だけが有効です。

LC_MESSAGES　診断メッセージや情報メッセージの表示方法を決定します。 また、メッセージの言語とスタイル、 そして肯定応答および否定応答の正しい形も決定します。 C ロケール においては、 メッセージはプログラム自身が使用しているデフォルトの形で表示されます(通常、米語)。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWscpu

関連項目

pwd(1), df(1M), du(1), quot(1M), attributes(5), environ(5)

注意事項

ユーザーが -a を使用しない限り、ディレクトリ名でないファイル名引数を無視します。

別にリンクされたファイルが過剰にある場合は、 du は余分にファイルを数えてしまうことがあります。

名前 | dump – オブジェクトファイルの選択部分のダンプ

形式 | **dump** [-aCcfghLorstV [-p]] [-T *index* [, *indexn*]] *filename...*

dump [-afhorstL [-p] [v]] *filename...*

dump [-hsr [-p] [-d *number* [, *numbern*]]] *filename...*

dump [-hsrt [-p] [-n *name*]] *filename...*

機能説明 | dump ユーティリティは、*filename* 引数で指定された各オブジェクトファイルの選択部分をダンプします。

dump ユーティリティはシェルスクリプトで使用するのに最適です。ただし、出力の読みやすさとしては、elfdump(1) コマンドを推奨します。

オプション | dump ユーティリティは、オブジェクトファイルとオブジェクトファイルのアーカイブの両方を受け入れます。dump ユーティリティは次のオプションに従い、*filename* 引数で指定された各ファイルを処理します。

-a
アーカイブの各メンバーのアーカイブヘッダーをダンプします。

-c
文字列テーブルをダンプします。

-C
復号化した C++ シンボルテーブル名をダンプします。

-f
各ファイルヘッダーをダンプします。

-g
アーカイブのシンボルテーブル内のグローバルシンボルをダンプします。

-h
セクションヘッダーをダンプします。

-L
可能であれば、動的リンク情報と静的共有ライブラリ情報をダンプします。

-o
各プログラムの実行ヘッダーをダンプします。

-r
再配置情報をダンプします。

-s
セクションの内容を 16 進数でダンプします。

-t
シンボルテーブルエントリをダンプします。

-T *index* または -T *index1*, *index2*
index で定義された索引付きシンボルテーブルエントリ、または *index1* と *index2* で定義された範囲のエントリだけをダンプします。

-V
バージョン情報を出力します。

次の修飾子を上記のオプションと組み合せて使用すると、オプションの機能が変わります。

-d *number*
-d *number1* , *number2*
number で示されたセクション番号、または *number1* から *number 2* までの範囲のセクションをダンプします。修飾子 -d はオプション -h、-s、および -r と共に使用できます。修飾子 -d をオプション -h または -s と共に使用すると、引数はセクション番号またはセクションの範囲として扱われます。修飾子 -d をオプション -r と共に使用すると、引数は、再配置が適用されるセクションの番号またはセクションの範囲として扱われます。たとえば、.text セクションに対応するすべての再配置エントリを出力するには、修飾子 -d の引数としてセクション番号を指定します。.text がファイル内のセクション番号 2 である場合、dump -r -d 2 を実行すると、対応するすべてのエントリが出力されます。特定の再配置セクションを出力するには、raw データを出力する場合は dump -s -n name を使用し、解釈付きのデータを出力する場合は dump -sv -n name を使用します。

-n *name*
指定されたエントリにのみ対応する情報をダンプします。修飾子 -n はオプション -h、-s、-r、および -t と共に使用できます。修飾子 -n をオプション -h または -s と共に使用すると、引数はセクション名として扱われます。修飾子 n をオプション -t または -r と共に使用すると、引数はシンボル名として扱われます。たとえば、dump -t -n .text を使用すると、名前が .text のシンボルに対応するシンボルテーブルエントリがダンプされます。また、dump -h -n .text を使用すると、.text 用のセクションのセクションヘッダー情報がダンプされます。

-p
ヘッダーの出力を抑制します。

-v
情報を、数値ではなくシンボル表記でダンプします。修飾子 v は次のオプションと共に使用できます。

-a (日付、ユーザー ID、グループ ID)

-f (クラス、データ、タイプ、マシン、バージョン、フラグ)

-h (タイプ、フラグ)

-L (値)

-o (タイプ、フラグ)

`-r` (名前、タイプ)

`-s` (可能であれば、セクションの内容を解釈する)

`-t` (タイプ、バインド)

修飾子 `-v` をオプション `-s` と共に使用すると、解釈可能なすべてのセクション (文字列テーブルやシンボルテーブルなど) が解釈されます。たとえば、`dump -sv -n .symtab` *filename...* を実行すると、`dump -tv` *filename...* と同じ書式で出力されます。一方、`dump -s -n .symtab` *filename...* を実行すると、raw データが 16 進数で出力されます。修飾子の付かない `dump -sv` *filename...* を実行すると、*file*内のすべてのセクションがダンプされます。このとき、解釈可能なセクションはすべて解釈され、解釈されないセクション (`.text` や `.data` など) は raw データとしてダンプされます。

dump ユーティリティは情報を意味のある書式で出力します。つまり、出力する情報により、文字、16 進数、8 進数、または 10 進数を適切に使い分けます。

属性

次の属性については、 `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWbtool

関連項目

`elfdump`(1), `nm`(1), `ar`(3HEAD), `a.out`(4), `attributes`(5)

dumpcs(1)

名前 | dumpcs – 現在のロケールのコードセット一覧表を表示

形式 | **dumpcs** [-0123vw]

機能説明 | dumpcs は、ユーザーの現在のロケールに対応するコードセットの印字可能文字を 16 進数のコード値とともに表示します。 ユーザーの表示端末はそのロケールの文字を表示できるものとします。

オプションなしで起動した場合は、現在のロケールの印字可能文字をすべて表示します。 1 つ以上の数字オプションを指定した場合には、指定した コードセット番号に対応する現在のロケールの EUC コードセット が、コードセット順に表示されます。 非印字文字は、その文字が属しているコードセットに 固有な表示幅分の個数の ASCII 空白文字で 置き換えられ、非印字文字であることが直前のアスタリスク (*) で示されます。

オプション

-0	ASCII または EUC 主コードセットの文字を表示します。
-1	EUC 補助コードセット 1 の文字を表示します (現在のロケールで使用されている場合)。
-2	EUC 補助コードセット 2 の文字を表示します (現在のロケールで使用されている場合)。
-3	EUC 補助コードセット 3 の文字を表示します (現在のロケールで使用されている場合)。
-v	「冗長」。 非印字文字をリストしてコード表を表示します。 このオプションがなく、非印字文字が 1 行以上連続した場合、最初の行の最初のカラムにアスタリスクがあるだけの 1 つの行で置き換えられます。
-w	コード値を、対応するワイド文字の値で置き換えます。

環境 | 環境変数 LC_CTYPE と LANG は、 dumpcs の文字分類を制御します。 dumpcs を起動すると、これらの環境変数は、 LC_CTYPE、 LANG の順にチェックされます。有効な値が検出されると、 あとの環境変数は無視されます。 たとえば、 LANG を新しく設定しても LC_CTYPE の現在有効な文字分類に上書きすることはありません。 有効な値がない場合、シェルの 文字分類はデフォルトで POSIX.1 の C ロケールに設定されます。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | localedef(1), attributes(5)

注意事項 | dumpcs は EUC ロケールでのみ使用できます。

名前 | echo – 引数の出力

形式 | **/usr/bin/echo** [*string*...]

機能説明 | echo ユーティリティは、 空白文字で区切られ、 復帰改行 (NEWLINE) で終わる引数を標準出力に出力します。 引数が指定されていない場合は、 NEWLINE 文字のみを書き出します。

echo は、コマンドファイル内で診断メッセージを生成したり、 既知のデータをパイプに送ったり、 環境変数の内容を表示したりする際に便利です。

C シェル 、 Korn シェル 、 Bourne シェル には、 echo 組み込みコマンドがあります。 デフォルトでは echo 組み込みコマンドは、 ユーザーがフルパス名なしで echo を呼び出した場合に 実行されます。 shell_builtins(1) を参照してください。 sh の echo 、 ksh の echo 、 /usr/bin/echo はバックスラッシュのついたエスケープ文字を認識し、 sh の echo を除いては、 警告文字として \ a を認識しません。 さらに、 ksh の echo には -n オプションは付きません。 sh の echo と /usr/bin/echo は、 SYSV3 環境変数が設定されている場合 (「環境」を参照) 、 -n オプションだけが付きます。 その場合、前述のバックスラッシュのついたエスケープ文字は 認識しません。 一方、 csh の echo と /usr/ucb/echo には -n オプションが付きますが、 バックスラッシュのついたエスケープ文字は認識しません。

オペランド | 次のオペランドを指定できます。

string 標準出力に書き出される文字列。 任意のオペランドが “ -n” である場合、 オプションではなく 文字列として扱います。 以下の文字の並びは、引数の中で特別に解釈されます。

\ a	警告文字
\ b	バックスペース
\ c	復帰改行なしで行を出力します。引数内で \c に続く文字はすべて無視されます。
\ f	用紙送り (form-feed)
\ n	復帰改行
\ r	キャリッジリターン
\ t	タブ
\v	垂直タブ
\\	バックスラッシュ
\ 0*n*	*n* は、 ASCII コードを 1 、 2 、 または 3 桁の 8 進数で表した 8 ビット文字です。

使用法 | 移植性の必要なアプリケーションには、 -n (第 1 の引数として) またはエスケープシーケンスを 使用しないでください。

printf(1) ユーティリティは、 以下のように echo ユーティリティの従来の動作を すべてエミュレートするために 使用できます。

- Solaris 2.6 オペーレーティング環境またはその互換バージョンの /usr/bin/echo は以下と同機能です。

```
printf "%b\n" "$*"
```

- /usr/ucb/echo は以下と同機能です。

```
if [ "X$1" = "X-n" ]
then
    shift
    printf "%s" "$*"
else
    printf "%s\n" "$*"
fi
```

新しいアプリケーションでは echo の代わりに printf を使用することをお薦めします。

使用例

例 **1** 現在のディレクトリがルートからどれくらい離れているかを調べる

echo を使用して、 現在のディレクトリが ルートディレクトリ (/) から見ていくつ目のサブディレクトリになるかを、 次のようにして判定することができます。

- 現在の作業中のディレクトリのフルパス名を表示する
- パスに組み込まれたスラッシュ文字を、 空白文字に変換するために tr を通して出力をパイプする
- ユーザーのパスの名前をカウントするために wc-w を通して出力をパイプする

```
example% /usr/bin/echo $PWD | tr '/' ' ' | wc -w
```

それぞれの機能については、 tr(1) と wc(1) を参照してください。

以下に、復帰改行 (NEWLINE) なしで文字列を表示するための別の方法を示します。

例 **2** /usr/bin/echo

```
example% /usr/bin/echo "$USER's current directory is $PWD\c"
```

例 **3** sh/ksh シェル

```
example$ echo "$USER's current directory is $PWD\c"
```

例 **4** csh シェル

```
example% echo -n "$USER's current directory is $PWD"
```

例 **5** /usr/ucb/echo

```
example% /usr/ucb/echo -n "$USER's current directory is $PWD"
```

環境

SYSV3　この環境変数は INTERACIVE UNIX システムと SCO UNIX のインストールスクリプトとの 互換性を提供するために使用します。互換性だけを目的とした環境変数で、 新しいスクリプトでは使用しないでください。

echo の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　入力ファイルはすべて、正常に出力された

>0　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

echo(1B), printf(1), shell_builtins(1), tr(1), wc(1), ascii(5), attributes(5), environ(5)

注意事項

エスケープ規則 \ 0*n* を使用して 8 ビット文字を表す際、 *n* の前に 必ずゼロ (0) を付けなければなりません。

たとえば、 echo 'WARNING:\ 07' と入力した場合、 WARNING: が出力され、 端末の " ベル " が鳴ります。 "07" の前に " \" を付けるときには、 これを保護する単一 (または二重) 引用符 (または 2 つのバックスラッシュ) を使用する必要があります。

\0 の後には、 8 進の出力文字を形成する最大 3 桁の文字が使用されます。 \0*n* の後に、 この 8 進表記に含まれない数値をさらに表示したい場合は、 *n* には全 3 桁を使用しなければなりません。 たとえば、 "ESC 7" と表示したい場合は、 \ 0 の後に、 2 桁の "33" だけでなく、 3 桁の "033" を使用しなければなりません。

2 桁	誤:	echo"0337	od -xc	
	結果:	df0a	(16 進)
	337	(ascii)	
3 桁	正:	echo "00337"	od -xc	

結果:	lb37 0a00	(16 進)
033 7	(ascii)	

各文字の 8 進表記については、 `ascii`(5) を参照してください。

名前 | ed, red – テキストエディタ

形式 | **/usr/bin/ed** [-s | -] [-p *string*] [-x] [-C] [*file*]

/usr/xpg4/bin/ed [-s | -] [-p *string*] [-x] [-C] [*file*]

/usr/bin/red [-s | -] [-p *string*] [-x] [-C] [*file*]

機能説明 | ed ユーティリティは標準のテキストエディタです。 file が指定されていると ed は e コマンド (下記参照) を指定されたファイルについてシミュレートします。 その結果、このファイルは ed のバッファに読み込まれ、編集できるようになります。

ed ユーティリティは、編集されるファイルのコピーに 対して操作を行います。 つまり、コピーに対して行われた変更は w (write) コマンドが実行されるまで、そのファイルに対して有効になりません。 編集されるテキストのコピーは バッファという一時ファイルに存在します。バッファは 1 つしかありません。

red ユーティリティは ed の制限付きバージョンです。 red は現在のディレクトリ内のファイルしか編集できず、 !*shell command* によるシェルコマンドの実行もできません。 これらの制限を無視しようとするとエラーメッセージ (*restricted shell*) が表示されます。

ed と red は、両方とも fspec(4) フォーマット機能をサポートしています。 デフォルトの端末モードは stty –tabs または stty tab3 で、タブ位置は 8 カラムごとに設定されます (stty(1) を参照) 。ただし file の先頭行にフォーマット指定が記述されていれば、デフォルトモード に優先してその指定が有効となります。 たとえば file の先頭行に次のように指定したとします。

```
<:t5,10,15 s72:>
```

この場合、タブ位置が 5 、 10 、 15 に、そして最大行長が 72 に設定されます。

ed コマンドは単純で、その構造は規則的です。最初に 0 から 2 個までのアドレス、次に 1 文字のコマンド、最後に (あれば) コマンドのパラメータが続きます。アドレスはバッファ内の 1 行または複数行を指定するものです。 アドレスが必要なコマンドはすべてデフォルトのアドレスを持っているので、 アドレスを省略することがよくあります。

一般に、 1 行に 1 つのコマンドだけ指定します。 コマンドの中には、テキストを入力するものもあります。 この時テキストは、バッファの中の適切な場所に格納されます。 ed がテキストを受け付けているときのことを「入力モード」であるといいます。 このモードではコマンドは認識されません。 すべての入力がそのまま受け付けられるだけです。 入力モードから抜けるには、行の先頭でピリオド (.) だけを入力して、キャリッジリターンを押します。

/usr/bin/ed | ed が引数のあるコマンドを実行する場合、 デフォルトのシェル /usr/bin/sh が使われます (sh(1) 参照) 。

/usr/xpg4/bin/ed | ed が引数のあるコマンドを実行する場合、/usr/xpg4/bin/sh が使われます (ksh(1) 参照) 。

正規表現

ed ユーティリティでは「正規表現」規則が使用できますが、 ある程度制限されています。 正規表現は、アドレスの中では行を指定するために、 また、いくつかのコマンド (たとえば、 s) の中では行のうちの置換される部分を指定するために用いられます。 ed におけるアドレス指定方法を理解するには、 常に「現在行」が存在することを認識する必要があります。 一般に、現在行はコマンドによって影響を受けた最後の行です。 現在行が受ける影響については、 各コマンドの説明の箇所で述べます。

国際化された標準の正規表現は、 システムに与えられたすべてのロケールで 使用されます。 詳細については regex(5) を参照してください。

ed コマンド

コマンドには 0 、 1 、または 2 個のアドレスを必要とします。 アドレスを必要としないコマンドにアドレスを指定するとエラーになります。 アドレスが必要なコマンドに必要な数のアドレスを指定しないと、 デフォルトのアドレスが採用されます。 必要以上にアドレスが指定されると、最後のアドレスの方から使用されます。

通常、アドレスはコンマ (,) によってお互いに区切られます。 セミコロン (;) によっても区切られます。 後者の場合、最初のアドレスが計算され、 現在行 (.) がその値に設定されます。 その後、 2 番目のアドレスが計算されます。 この機能は、順方向および逆方向検索の開始行を決定するのに使用できます (上記規則 5 および 6 を参照) 。

以下に示す ed コマンドのリストでは、コマンドの前の括弧はアドレスの一部ではなく、 デフォルトのアドレスを示します。

各アドレス部分の先頭には任意の 2 個の空白文字を付加できます。 またコマンド文字の先頭にも任意の数の空白文字を付加できます。 接尾文字 (l 、 n 、または p) を指定するのであれば、コマンドの直後に記述しなければなりません。

e 、 E 、 f 、 r 、 w の各コマンドには、省略可能な file パラメタがあります。これを指定する場合には、コマンド文字との間に最低 1 個の空白文字を置くことが必要です。

バッファ全体を書き換えた最後の w コマンド実行後にバッファの内容が変更されているとき、 e または q コマンドによりエディタバッファを破壊しようとすると、 ed は警告を発します。具体的には以下の文字列を標準出力に書き出します。

"?\ n"

なお H コマンドにより「ヘルプモード」が起動されていれば、 状況を説明するメッセージが続いて出力されます。 この警告出力後も、 ed はコマンドモードのままで、現在の行番号は変わりません。 ここで続けて e または q コマンドを再度入力すれば、そのコマンドが実行されます。

標準入力から次のコマンドを読み込もうとして ファイルの終わりを検出した場合、ed ユーティリティは q コマンドが入力された場合と同じように動作します。

一般に、 1 行に 2 つ以上のコマンドを指定するとエラーです。 しかし、すべてのコマンド (e 、 f 、 r 、 w を除く) は、 l 、 n 、または p コマンド (それぞれ、現在行をリストする、番号付けする、出力する) を接尾辞として付けることができます (l 、 n 、および p コマンドを参照) 。

(.)a

<*text*>

.

append コマンドは、 0 行以上のテキスト行を受け入れ、それをバッファ内 のアドレスが示す行の直後に追加します。現在行 (.) は、最後に挿入された行に設定されます。 挿入行がない場合は、アドレスが示す行に設定されます。 a コマンドではアドレス 0 は有効です。この場合、バッファの先頭にテキストを追加します。端末から入力できる最大文字数は 1 行あたり 256 文字です (復帰改行文字 (NEWLINE) も含む) 。

(.)c

<*text*>

.

change コマンドは、バッファからアドレスで指定された行を削除し、 0 行以上のテキスト行をバッファ内に受け入れ、 削除された行と置換します。現在行 (.) は最終入力行に、または入力行がない場合、 削除された最後の行の直後の行に設定されます。
バッファの最終行を削除した場合、 現在の行番号は新たに最終行となった行のアドレスに設定されます。 削除によりバッファ内に行が残っていなければ、 現在の行番号はゼロに設定されます。

C

基本的に、後述する x コマンドと同じです。ただし、 NULL キーが入力されないかぎり、 e および r コマンドによって読み込まれたすべてのテキストを 暗号化されているものと見なします。

(. , .)d

delete コマンドは、バッファからアドレスで指定された行を削除します。 削除された最後の行の直後の行が現在行に設定されます。 バッファの最終行を削除した場合、 新たに最終行となった行が現在行になります。 削除によりバッファ内に行が残っていなければ、 現在の行番号は ゼロとなります。

e *file*

edit コマンドは、バッファの全内容を削除して file の内容をバッファ内に読み込みます。 現在行 (.) は、バッファの最終行に設定されます。 file が指定されないと、 (もし、あれば) 現在記憶されているファイル名を使用します (f コマンド参照) 。 -s オプションが指定された場合を除き、読み込んだバイト数が以下の形式で 標準出力に書き出されます。

"%d\ n" 読み込んだバイト数

file は、後で使用する e 、 E 、 r 、および w コマンドのデフォルトのファイル名として記憶されます。 file の代わりに ! を指定すると、 ! 以降の文字列はシェル (sh(1)) コマンドと見な

され、その出力が読み込まれます。 このようなシェルコマンドは現在のファイル名として記憶されません。 後述の「診断」も参照してください。 e コマンドが正常に終了すると、すべてのマークは捨てられます。 バッファ全体が最後に書き換えた後でその内容が変更されているとき、 前述したように警告が発せられます。

E *file* Edit コマンドは、 基本的に、 e コマンドと同じです。ただし、最後の w コマンドを実行してからバッファの内容が変更されたか否かをチェックしません。

f *file* file が指定されると、 f コマンドは、現在記憶されているパス名を file に変更します。 次に、パス名を変更したか否かに関わらず、現在記憶されているパス名 (変更した場合は新しいパス名) を以下の形式で標準出力に書き出します。

```
"%s\ n" パス名
```

現在の行番号は変わりません。

(1 , $) g/*RE*/*command list* global コマンドは、まず *RE* で指定された正規表現と一致する各行をマークします。 そして、該当するすべての行に対して、 まず現在行 (.) をその行に設定してから、 *command list* で指定したコマンドリストを実行します。 g コマンドが終了したとき、現在の行番号の値は、コマンドリスト中の 最後のコマンドが指定した値となっています。 一致する行が見つからなかった場合には、 現在の行番号の値は変わりません。 単一コマンドまたはコマンドリストの最初のコマンドは global コマンドと同一行に現われなければなりません。 コマンドが複数行にまたがる場合、 最終行以外の各行は バックスラッシュ (\ ) で終わらなければなりません。 a 、 i、および c コマンドおよび関連する入力を複数行に指定できます。 入力モードを終了する . が command *list* の最後の行の場合、省略できます。空の command *list* は p コマンドと同じです。 g 、 G 、 v 、 V 、および ! コマンドは、 *command list* 内に書くことができません。 「注意事項」および「ファイル」 の前の最後のパラグラフも参照してください。 *RE* を区切る文字として、スラッシュの代わりに、 空白と復帰改行以外の任意の文字を使うことができます。 また区切り文字を *RE* 中で実際の文字として使いたければ、その前にバックスラッシュを 付加してください。

(1 , $) G/*RE*/ 対話型 Global コマンドでは、最初に、 *RE* で指定された正規表現と一致する各行をマークします。 そして、該当する各行について、その行が標準出力に書き出され、 現在行 (.) がその行に変更され、この時点で、 1 つのコマンド (a 、 c 、 i 、 g 、 G 、 v 、および V コマンドをのぞく) が入力でき、入力されたコマンドは実行されます。 コマンドが実行されると、次のマークされた行が出力され、 以下同じ処理を繰り返します。 改行は無効なコマンドとして扱います。&は、現在起動中の G 内で、 最後に実行された NULL でないコマンドを再実行します。 注意 : G コマンド 実行中のコマンド入力はバッファ内の どの行でもアドレス指定したり、

影響を与えたりできます。 現在の行番号の最終的な値は、最後に正常終了したコマンド が指定した値です (なお G で入力したコマンドが失敗した場合、または NULL コマンドが入力された場合は、 G 自身が「最後に正常終了したコマンド」となります)。 一致する行が見つからなければ、現在の行番号は変わりません。 G コマンドは SIGINT シグナルによって終了させることができます。また、 割り込みシグナル (ASCII DEL または BREAK) でも終了できます。 *RE* を区切る文字として、スラッシュの代わりに、 空白と復帰改行以外の任意の文字を使うことができます。 また区切り文字を *RE* 中で実際の文字として使いたければ、その前にバックスラッシュを 付加してください。

`h`

help コマンドは、最後の ? 診断の理由を説明する短いエラーメッセージを出力します。 現在の行番号は変わりません。

`H`

Help コマンドは、 以後発生するすべての ? 診断のエラーメッセージを出力するモードにします。 もしあれば前の ? も説明します。 H コマンドは、このモードを交互にオンおよびオフにします。 最初はオフです。 現在の行番号は変わりません。

`( . )i`

<*text*>

`.`

insert コマンドは、 0 またはそれ以上の行のテキストを受け入れ、 バッファ内のアドレス指定された行の前に挿入します。 現在行 (.) は最後に挿入された行に設定されます。 挿入行がない場合は、アドレス指定された行に設定されます。 i コマンドと a コマンドは、入力テキストの挿入位置だけ異なります。 このコマンドにはアドレス 0 は有効ではありません。 端末から入力できる最大文字数は 1 行あたり 256 文字です (復帰改行文字も含む) 。

`( . , .+1 )j`

join コマンドは、 適当な復帰改行文字を削除することによって連続する行を結合します。 1 つのアドレスしか指定されないと j コマンドは何もしません。 行の結合が行われると、現在の行番号は結合された側の行のアドレスに 設定されます。結合が行われなければ、現在の行番号は変わりません。

`( . )k`*x*

mark コマンドは、 *x* という名前でアドレス行をマークします。 *x* は ASCII の小文字 (a–z) でなければなりません。 ′*x* を指定すると、このマークされた行を指すようになります。 現在行 (.) は変更しません。

`( . , . )l`

list コマンドは、アドレスされた行をすべての情報が見える形で標準出力へ書き出します。 \\ 、 \ a 、 \ b 、 \ f 、 \ r 、 \ t 、 \ v はそれぞれ対応するエスケープシーケンスとして書き出されます。 テーブル中にある \ n は適用外です。テーブル中にない非印字文字に関しては、各バイトごと (最上位ビットから) に 3 桁の 8 進数で、前にバックスラッシュが付加された形式で出力されます。 長い行は折り返されます。折り返しの発生地点にはバックスラッシュと 復帰改行文字が表示されます。 折り返し地点

の長さは不定ですが、出力装置に適した値に なっています。各行の終わりは $ でマークされます。 l コマンドは、 e 、 E 、 f 、 q 、 Q 、 r 、 w 、 ! 以外のすべてのコマンドの後に付けられます。現在の行番号は、最後に出力された行のアドレスに設定されます。

(. , .)m*a*
: move コマンドは、アドレス行を *a* で示される行の後に移動します。アドレス 0 を *a* に書いても有効で、その場合アドレス行をファイルの先頭に移動します。 アドレス *a* が移動する行の範囲内にあるとエラーになります。 現在行 (.) は移動した最後の行に設定されます。

(. , .)n
: number コマンドは、行番号とタブの後に、アドレス行を出力します。 現在行 (.) は出力した最後の行に設定されます。 n コマンドは e 、 E 、 f 、 q 、 Q 、 r 、 w 、または ! 以外のすべてのコマンドの後に続きます。

(. , .)p
: print コマンドは、アドレス行を標準出力に出力します。 現在行 (.) は出力した最後の行に設定されます。 p コマンドは、 e 、 E 、 f 、 q 、 Q 、 r 、 w 、または ! 以外のすべてのコマンドの後に付けられます。 たとえば、 dp というコマンドは、現在行を削除して新しい現在行を出力します。

P
: コマンドは、 後続のすべてのコマンドにアスタリスク (*) (-p 指定時は 文字列) によるプロンプトをつけて入力を促します。 P コマンドは、このモードを交互にオンまたはオフします。 初期値は、 -p オプション指定時はオン、省略時はオフです。 現在行は変わりません。

q
: quit コマンドは、 ed を終了します。バッファ全体が書き換えられた後でバッファの内容が 変更されていると、警告が発せられます。 後述の「診断」を参照してください。

Q
: エディタは、最後の w コマンドの後、バッファが変更されたか否かをチェックしないで終了します。

($)r *file*
: read コマンドは、 file の内容をバッファ内に読み込みます。 file が指定されないと、(もしあれば) 現在記憶されているファイル名を使用します (e および f コマンド参照)。 file が ed 起動後参照された 最初のファイル名でないかぎり、現在記憶されているファイル名は 変更されません。 r コマンドにはアドレス 0 は有効です。この場合、バッファの先頭にファイルを読み込みます。読み込みに成功すると、 -s オプションが省略されていれば、 読み込まれた文字数が以下の形式で標準出力に書き出されます。

  ```
  %d\ n 読み込んだバイト数
  ```

  現在行 (.) は読み込まれた最後の行に設定されます。 file の代わりに ! を指定すると、 ! 以降の文字列はシェルコマンド (sh(1) 参照) と見なされ、その出力が読み込まれます。 たとえば、 $r

`!ls` は編集中のファイルの最後に現在のディレクトリを追加します。 このようなシェルコマンドは、現在のファイル名として記憶されません。

(. , .)s/*RE***/***replacement***/**
(. , .)s/*RE*/*replacement*/*count*, *count*=[1-512]
(. , .)s/*RE*/*replacement*/g
(. , .)s/*RE*/*replacement*/l
(. , .)s/*RE*/*replacement*/n
(. , .)s/*RE*/*replacement*/p

substitute コマンドは、各アドレス行について、 *RE* で示された正規表現を検索します。 これらの置換コマンドは、任意の数だけ指定できます。 一致が発生した各行に対して、 グローバル置換指示子 g がコマンドの後にあれば、すべての (重ならない) 一致した文字列を *replacement* に置換します。 グローバル指示子がなければ、 一致した文字列の最初のものだけを置換します。 数字 *count* がコマンドの後にあれば、各アドレス行内で一致した文字列のうち *count* 番目のものだけを置換します。 すべてのアドレス行について置換が失敗するとエラーになります。 正規表現 *RE* と *replacement* を区切るには、スラッシュ (/) の代わりに空白文字と復帰改行以外のすべての文字が使用できます。 現在行 (.) は置換が発生した最後の行に設定されます。 *RE* の区切り文字を *RE* 中で実際の文字として使いたければ、その前にバックスラッシュを付加してください。 「ファイル」 の前の最後のパラグラフも参照してください。 *replacement* 内のアンパサンド (&) は、現在行上で正規表現 *RE* と一致した文字列に置き換えられます。 この場合の & の特別な意味は、 \ を前につけることによって抑止できます。 さらに一般的な機能として、文字列 *n* (*n* は数字) は、 指定された正規表現 *RE* の \ （ と \ ） で囲まれた *n* 番目のサブ正規表現と一致するテキストに置換されます。 ネストされた括弧付きサブ正規表現が存在する場合、 *n* は左から数えた \ （ の発生回数によって決まります。 文字 % が *replacement* 内の唯一の文字であるとき、最後の置換コマンドで使用した *replacement* を現在の置換コマンドの *replacement* として使用します。 ただしそれ以前に置換コマンドがなかった場合、 このような % の使い方はエラーとなります。 % は、複数の文字の置換文字内にあるとき、または \ が前に付くときには、その特別な意味を失います。 *replacement* を先頭から終端まで走査する際にバックスラッシュ (\) が検出されると、後続の文字は特殊な意味を持っていたとしても その意味を失います。 なお & 、 \ 、 % および数字以外の文字については、どのような特殊な意味が与えられているか は不定です。 1 行を分割するには、復帰改行文字で置換します。 *replacement* 内の復帰改行文字は、 \ を前に付けてエスケープしなければなりません。 このような置換は、 g または v コマンドリストの一部としては実行できません。 現在の行番号は、置換が行われた最後の行のアドレスに設定されます。 置換がまったく行われないと、現在の行番号は変わりません。 行が分割されると、新たな現在の行番号を決定するため、 分割で発生した両方の行で置換が行われたものと見なされます。 置換文字列が置換対象となる文字列と同一の場合でも、 置換は発生したと見なされます。 置換コマンドは以下に示す指示子をサポートします。

count　各アドレス行で見つかった *RE* のうち、 *count* 番目のものだけを置換します。 *count* は 1 から 512 までの値でなければなりません。

g　最初のものだけでなく、すべての重なっていない *RE* を一括して置換します。 g と *count* の両方を指定した場合、結果は保証できません。

`l` 置換を行なった最後の行の内容を標準出力に書き出します。 出力形式は `l` コマンドのものと同じです。

`n` 置換を行なった最後の行の内容を標準出力に書き出します。 出力形式は `n` コマンドのものと同じです。

`p` 置換を行なった最後の行の内容を標準出力に書き出します。 出力形式は `p` コマンドのものと同じです。

`( . , . )t`*a*
`t` コマンドは、 `m` コマンドと同じように動作します。ただし、 アドレス行のコピーがアドレス `a` (0 でもよい) の後に置かれます。 現在行 (`.`) はコピーされた最後の行に設定されます。

`u`
undo コマンドは、 バッファの内容を変更した最後のコマンドの実行結果を無効にします。 無効にできるコマンドは、最後に実行した `a` 、 `c` 、 `d` 、 `g` 、 `i` 、 `j` 、 `m` 、 `r` 、 `s` 、 `t` 、 `u` 、 `v` 、 `G` 、または `V` コマンドです。 グローバルコマンドの `g` 、 `G` 、 `v` 、または `V` でバッファを変更していた場合、一括してその変更を無効にします。 グローバルコマンドで変更が行われていない (たとえば `g`/*RE* /`p`) 場合、 `u` コマンドは何も意味を持ちません。 現在の行番号は、無効にしたコマンドの開始直前に設定されていた値に戻ります。

`( 1 , $ )v`/*RE*/*command list*
`v` コマンドは、基本的に、グローバルコマンド `g` と同じです。ただし、 最初の段階でマークされる行は *RE* で示す正規表現に一致しない ものです。

`( 1 , $ )V`/*RE*/
`V` コマンドは、基本的に、対話型グローバルコマンド `G` と同じです。ただし、 最初にマークされる行は *RE* で示す正規表現に一致しない ものです。

`( 1 , $ )w` *file*
write コマンドはアドレス行を `file` に書き込みます。 `file` が存在しない場合は、ファイル生成マスクが他のモードで指定されていないかぎり、 モード `666` (すべてのユーザーが読み込み、書き込み可能) で作成します。 sh(1) 上での特殊コマンド `umask` の説明を参照してください。 `file` が `ed` 起動後参照された 最初のファイル名でないかぎり、現在記憶されているファイル名は 変更されません。 `file` が指定されないと、(もし、あれば) 現在記憶されているファイル名を使用します (`e` および `f` コマンド参照)。 現在行 (`.`) は変更されません。 コマンドが正常終了すると、 `-s` オプションが省略されていれば、書き込まれた文字数が以下の形式で 出力されます。

```
"%d\ n", 書き込んだバイト数
```

`file` の代わりに `!` を指定した場合、 `!` 以降のテキストは、アドレス行が標準入力であるシェル (sh(1) 参照) コマンドと見なされ、その出力が読み込まれます。 このようなシェルコマンドは、現在のパス名として記憶されません。 このような `!` を伴った `w` コマンドは、「バッファ全体を書き換えた最後の `w` コマンド」と見なされます。

(1 , $) W *file*
: W コマンドは基本的に上述の write コマンドと同じです。ただし、 アドレス行を file (存在する場合) の最後に追加します。 file が存在しない場合、上述の w コマンドで述べたようにファイルを作成します。

X
: e および r コマンドで編集するために 読み込まれたテキストが暗号化されているか否かを判定します。 キーとして空文字列を与えると暗号化を無効にします。これ以後の e 、 r 、および w コマンドは、このキーをテキストの暗号化または復合化に使用します 。 明示的に空文字列をキーとして指定した場合は 暗号化は無効になります。 ed のオプション -x も参照してください。

($)=
: アドレス行の行番号が、次に示す形式で標準出力に表示されます。

```
"%d\ n" 行番号
```

このコマンドによって、現在の行番号は変更されません。

!*shell command*
: ! 以降のテキストを UNIX システムシェル (sh(1) 参照) に送信し、コマンドとして解釈します。 コマンドテキストにエスケープされていない % 文字を指定すると、記憶されているファイル名に置換されます。 ! がシェルコマンドの最初の文字として現われる場合、 それは前のシェルコマンドのテキストで置換されます。 つまり、 !! は最後のシェルコマンドを繰り返します。 % または ! による置換が実行されると、変更された行の内容が command の実行前に標準出力に書き出されます。 -s オプションが省略されていれば、 ! コマンドは終了時に以下のメッセージを標準出力に書き出します。

```
"!\ n"
```

現在の行番号は変更されません。

(.+1) <new-line>
: 行にアドレスだけを指定すると、そのアドレス行を出力します。 復帰改行文字だけの場合、 .+1p と同じです。 つまり、バッファ内を進むのに使用します。 現在の行番号は、書き出した行のアドレスに設定されます。

割り込みシグナル (ASCII DEL または BREAK) が送信されると、 ed は "?\ n" を出力して、 ed のコマンドレベルに戻ります。 ed ユーティリティはすべてのシグナルに対して標準的な動作を行います。 ただし次の 2 つのシグナルは例外です。

SIGINT
: ed ユーティリティは現在の動作を中断し、文字列 "?\ n" を標準出力に書き出し、コマンドモードに戻ります。

SIGHUP
: バッファが空でなく、最後の書き込み処理以降に変更されている場合、 ed ユーティリティはファイル中にバッファのコピーを生成しようとします。 その対象ファイルとして、まず現在のディレクトリ中の ed.hup というファイルが選ばれます。それが失敗すると、環境変数 HOME が示すディレクトリ中の ed.hup というファイルが選ばれます。 いずれの場合も ed は、コマンドモードに戻らないで終了します。

いくつかのサイズ制限があります。 1 行は 512 文字以下、グローバル・コマンドリストは 256 文字以下、ファイルのパス名は 255 文字以下です (スラッシュを含む) 。 行数の制限はユーザーのメモリ容量によって異なります。 1 行には 1 ワード必要です。

ファイルを読むとき、 ed は ASCII と NULL 文字を破棄します。

ファイルが復帰改行文字で終了していないとき、 ed はそれを追加して、その旨を説明するメッセージを表示します。

正規表現 RE または置換文字列の終端区切り文字 (たとえば、/) が復帰改行文字の直前の文字のとき、区切り文字は省略できます。 いずれの場合もアドレスされた行が出力されます。 次の各組のコマンドは同じものと見なされます。

```
s/s1/s2         s/s1/s2/p

g/s1            g/s1/p

?s1             ?s1?
```

不正なコマンドが投入されると、 ed は以下の文字列を標準出力に書き出します。

```
"?\ n"
```

このとき H コマンドにより「ヘルプモード」が有効になっていれば、 状況を説明するメッセージが付加されます。 上記文字列出力後、 ed はコマンドモードを継続します。現在の行番号は変わりません。

オプション

`-C` 暗号化オプション。 基本的に、 -x オプションと同じです。ただし、 ed は、 C コマンドをシミュレートします。 C コマンドは、基本的に、 X コマンドと同じです。ただし、 読み込まれたすべてのテキストが暗号化されていると見なされます。

`-p` *string* ユーザーがプロンプト文字列を指定するのを許可します。 デフォルトではプロンプト文字列はありません。

`-s` | `-` e 、 r 、および w コマンドによる文字カウント、 e および q コマンドからの診断、および !*shell command* の後の ! プロンプトを出力しません。

`-x` 暗号化オプション。 ed は、 X コマンドをシミュレートして、ユーザーにキーの入力を要求してきます。 X コマンドは読み込まれたテキストが 暗号化されているか否かを判定するのに高度な推測を行います。 一時バッファファイルも、 -x オプションで入力したキーを変形したバージョンを用いて暗号化されます。 本マニュアルページの最後の節「注意事項」 も参照してください。

オペランド

以下のオペランドを指定できます。

file この file 引数を指定すると、 ed は、標準入力からコマンドを読み込む前に、パス名 file で示されるファイルに e コマンドを適用したかのように動作します。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の ed と red の動作については、largefile(5) を参照してください。

環境

ed の実行に影響を与える環境変数 HOME、 LC_CTYPE、 LC_COLLATE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	ファイルにもコマンドにもエラーはなく、正常終了した
>0	エラーが発生した

ファイル

$TMPDIR	この環境変数が NULL でない場合、 その値は一時作業ファイルのディレクトリ名として /var/tmp の代わりに使用されます。
/var/tmp	/var/tmp が存在する場合、一時作業ファイルのディレクトリ名として使用されます。
/tmp	環境変数 TMPDIR が存在しないか NULL で、さらに /var/tmp が存在しない場合、 /tmp が一時作業ファイルのディレクトリ名として使用されます。
ed.hup	端末がハングアップしたとき、作業ファイルがここに保存されます。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/red

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/ed

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

bfs(1), edit(1), ex(1), grep(1), ksh(1), sed(1), sh(1), stty(1), umask(1), vi(1), fspec(4), attributes(5), environ(5), largefile(5), regex(5), XPG4(5)

診断

?	コマンドエラー
?file	アクセス不可能なファイル (詳細については help および Help コマンドを使用してください)。

最後に w コマンドを使用して全バッファを書き出した後に バッファの内容が変更された場合、 e または q コマンドによってバッファの内容を破棄しようとすると、 ed はユーザーに対して 警告を発し、 ? を出力して編集を続行するか否かを聞いてきます。ここで再度 e または q コマンドを入力すると、これらのコマンドは実行されます。-s コマンド行オプションは、上述の機能を禁止します。

注意事項

– オプションは、サポートはされていますが、ドキュメント内では コマンド構文規格に準拠する -s オプションで置き換えました (intro(1) 参照) 。

g または v コマンドに対して ! コマンドは無効です。

! コマンドと e 、 r 、および w コマンドからのエスケープコマンド ! は、エディタが制限付きシェル (sh(1) 参照) から起動されているときには使用できません。

正規表現 RE 内の \ n シーケンスは復帰改行文字と一致しません。

エディタの入力がコマンドファイル (たとえば、 ed file < *ed_cmd_file*) からの場合、最初にエラーが発生した時点でエディタは終了します。

名前 | edit – テキストエディタ (ex の臨時ユーザー用の変形版)

形式 | **/usr/bin/edit** [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-t *tag*] [-v] [-V] [-x] [-w *n*] [-C] [+ | *command* | -c *command*] *filename...*

/usr/xpg4/bin/edit [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-t *tag*] [-v] [-V] [-x] [-w *n*] [-C] [+ | *command* | -c *command*] *filename...*

機能説明 | edit ユーティリティは、 コマンド指向エディタの使用を望む、 新規または臨時のユーザー対象に推奨されるもので、 テキストエディタ ex の変形です。 このエディタは、 次のオプションが自動的に設定され、 ex とまったく同じ様に動作します。

初心者	ON
レポート	ON
表示モード	ON
マジック	OFF

edit の使用を始めるにあたり、 役に立つ簡単な説明を次に示しますが、 CRT 端末を使用している場合には、 画面エディタ vi について学ぶ方をお勧めします。

既存のファイルの内容を編集する場合は、 シェルに対し、 コマンド edit *name* で始めます。 edit は、 そのユーザー編集用のファイルのコピーを作成して ユーザーが編集できるようにし、 そのファイル内の行数および文字数を表示します。 新規のファイルを作成する場合も、 コマンド edit にファイル名を付け、 edit *name* で始めます。 この場合には、 エディタは [New File] と表示して、 このファイルが新規のファイルであることを示します。

edit コマンドのプロンプトは、 コロン (:) であり、 エディタを起動した後には、 このコロンが表示されていなければなりません。 既存のファイルを編集する場合は、 edit のバッファ (編集中のファイルのコピーの名前) に何行か入っています。 編集開始時には、 edit はファイルの最終行を現在行とします。 edit のコマンドのほとんどは、 対象行をユーザーが指定しないかぎり、 現在行を対象とします。 したがって、 print(p と省略できます) を指定し、 キャリッジリターンを入力すると (これは、 edit のすべてのコマンドの後に必要です)、 現在行が表示されます。 現在行を delete (d) で削除すると、 通常、 ファイル中にある次の行が 新しい現在行として表示されます。 最終行を delete すると、 新しい最終行が現在行になります。

空のファイルに入力したい場合、 または、 新しい行を追加したい場合は、 append (a) コマンドを使用します。 このコマンドを実行 (append の後に、 キャリッジリターンを入力) した後は、 edit は、 ドット (.) だけからなる行を入力するまで、 端末から入力行を読み取ります。 入力行は、 現在行の後に追加されます。 insert(i) コマンドも append と同じ様な働きをしますが、 入力行は、 現在行の後ではなく、 前に挿入されます。

edit ユーティリティは、 バッファ内の行に行番号を付けます。 先頭行が、 行番号 1 になります。 コマンド 1 を実行すると、 edit はバッファの先頭行を表示します。 このときに、 コマンド d を実行すると、 edit は先頭行を削除し、 2 行目を 1 行目とし、 現在行 (新しい 1 行目) を表示して現在どこにいるかわかるようにします。 一般には、 現在行は常に、 その直前のコマンドの対象となっていた行です。

substitute (s) コマンド s/*old* /*new*/ を使用すると、 現在行の中のあるテキストを置き換えることができます。 この場合、 *old* には置き換えたい旧文字列を指定し、 *new* には *old* と置き換えたい新文字列を指定します。

filename (f) コマンドは、 編集中のバッファ内の行数を表示し、 このバッファに変更を加えた場合には [Modified] と表示します。 バッファを変更した後は、 write (w) コマンドを実行してファイルの内容をセーブすることができます。 quit (q) コマンドを実行すると、 エディタを終了することができます。 edit を起動したけれども、 そのバッファに変更を加えなかった場合は、 write コマンドを実行してファイルに書き込む必要はありません (ただし、実行しても問題はありません)。 バッファに変更を加えた後で、 ファイルに書き込まずに edit を quit コマンドで終了しようとすると、 メッセージ No write since last change (:quit! overrides) が表示され、 edit は次のコマンドの入力を待ちます。 そのバッファを書き込む必要がない場合は、 quit コマンドの後に感嘆符 を付けて (q!) 実行してください。 バッファは廃棄されて復元不能になり、 シェルに戻ります。

d および a コマンドが使え、 ファイル内の行を示す行番号の指定ができるようになると、 必要とするあらゆる変更を行うことができます。 ただし、 edit をもう少し頻繁に使用する場合には、 少なくとも、 他に 2 、 3 のコマンドを覚えておいたほうが便利です。

change(c) コマンドは、 現在行を入力行に変更します。 append の場合と同様、 ドット (.) だけからなる行を入力するまで、 複数の行を入力できます。 change では、 変更したい行の行番号を指定して、 複数行を変更することができます。 3,5c のように指定してください。 同じ方法を使って、複数行を表示することもできます。 1,23p と指定すると、 ファイルの先頭から 23 行目までを表示することができます。

undo (u) コマンドは、 直前に実行したコマンドにより変更されたバッファを、 元に戻します。 したがって、 substitute コマンドを実行し、 その結果が望んだものではなかった場合、 u コマンドを実行すると元の行の内容が復元されます。 undo コマンドを使って undo コマンドを取り消すこともできます。 edit は、 コマンドがバッファの複数行を対象にした場合、 警告メッセージを表示します。 write や quit などのコマンドは、 取り消しできないので注意してください。

バッファ内の次の行を見たい場合は、 キャリッジリターンを入力してください。 複数の行を見たい場合は、 キャリッジリターンの代わりに、 ^D (コントロールキーを押しながら、 d を押します) を入力してください。 これによって、 CRT に画面半分の行が表示されるか、 または、 ハードコピー端末に 12 行が出力されます。 z コマンドを実行すると、 前後のテキストを見ることができます。 現在行がテキスト表示の中心に表示され、 最後に表示される行が現在行になります。 '' を入力すると、 z コマンドを実行する前の現在行に戻ることができます。 z コマンドには、 他にもオプションがあります。 z- は、 元の現在行を最後の行とする 1 画面分のテキスト (または 24 行)

を表示します。 z+ は、 現在行の次の 1 画面分を表示します。 1 画面分より少ない行数で表示したい場合は、 z.11 と入力すると、 現在行の直前の 5 行と直後の 5 行が表示されます (z.*n* と入力すると、 *n* が奇数の場合は、 現在行が中心となり、 全部で *n* 行が表示されます。 *n* が偶数の場合は、 現在行を中心として、 *n*–1 行が表示されます)。 他のコマンドの後に行数を指定することができます。 たとえば、 コマンド d5 を使用すると、 現在行と、 現在行から数えて 5 行目まで の、合計 5 行を削除することができます。

ファイル内で何かを探す場合には、 行番号がわかっていれば、 行番号を使用することができます。 ただし、 行の挿入や削除を行なった場合には、 行番号は変わっているので、 これはあまりあてになりません。 ファイルの前方および後方に文字列を検索することができます。 /*text*/ という形式のコマンドを使用すると ファイルの最後の方に向かって *text* を検索でき、 ?*text*? を使用すると ファイルの先頭の方に向かって *text* を検索できます。 検索がファイルの終わりに到達しても *text* が見つからなかった場合は、 ラップして元の場所まで検索を続けます。 この検索の便利な機能に /^*text*/ という形式の検索があり、 これは、 行の先頭にある *text* を検索します。 同様に /*text*$/ は、 行の終わりにある *text* を検索します。 どちらのコマンドも、 最後に付ける / または ? を省略することができます。

現在行を表す記号名として、 ドット (.) があり、 これは行の範囲指定の際に最も便利です。 .,$p のように指定すると、 現在行から、 ファイルの最終行までが表示されます。 ファイルの最終行に移動するには、 最終行を表す記号名 $ を使用して参照することができます。 したがって、 コマンド $d は、 ファイルの最終行を削除し、 現在行は変わりません。 行参照では、 算術式も使用できます。 たとえば、 $–5 は、 最終行の前の行から数えて 5 番目の行を表し、 .+20 は、 現在行の次の行から数えて 20 番目の行を表します。

現在行を見つけるには、 '.=' と入力します。 テキストのある部分を、 ファイル内またはファイル間で、 移動またはコピーする場合に便利です。 コピーまたは移動したい行の、 最初と最後の行番号を調べます。 10 行目から 20 行目を移動する場合は、 10,20d a と入力すると、 これらの行はファイルから削除され、 a という名前のバッファに入ります。 edit には、 このようなバッファが、 a から z まで 26 個あります。 バッファ a の内容を現在行の後に入れるには、 put a と入力します。 これらの行を別のファイルに移動またはコピーする場合には、 行をコピーした後に edit (e) コマンドを実行します。 このとき、 edit chapter2 のように、 e コマンドの後に編集したい別のファイルの名前を指定します。 削除しないで行をコピーする場合には、 d の代わりに yank(y) を使用してください。 1 つのファイル内だけで移動またはコピーする場合には、 名前付きバッファを使用する必要はありません。 たとえば、 10 行目から 20 行目を ファイルの末尾に移動するには、 10,20m $ と入力します。

オプション

次のオプションは、 ex(1) 内で set コマンドを使用することによって、 オン/オフを切り替えることができます。

-| -s	ユーザーへのすべての対話型フィードバックを抑止します。 エディタスクリプト実行中に便利なオプションです。
-l	LISP プログラムの編集用にセットアップします。

`-L`	エディタまたはシステムのクラッシュが 原因で保存されたファイルの名前をすべて表示します。
`-R`	読み取り専用モード。 読み取り専用フラグが設定され、 ファイルの不慮の上書きを防ぎます。
`-r` *filename*	エディタもしくはシステムのクラッシュが発生した後で、 *filename* が示すファイルを編集します。つまりクラッシュ発生時にバッファ中にあったバージョンを復旧させます。
`-t` *tag*	*tag* が示すタグを含むファイルを編集します。そのタグが 定義されている地点が編集開始地点となります。
`-v`	`vi` を使った編集の画面表示を開始します。単に `vi` コマンドを入力しても同様に実行できます。
`-V`	冗長。 `ex` コマンドを標準入力で読み込んだ場合、 入力は標準エラーに表示されます。 シェルスクリプトの `ex` コマンド実行時に役に立ちます。
`-x`	暗号化オプション。 これを指定すると、 `edit` は、 `ex` の `X` コマンドをシミュレートし、 ユーザーにキーの入力を求めます。 このキーは、 `crypt` コマンドのアルゴリズムを使用したテキストの暗号化 および暗号解除を行うのに使用されます。 `X` コマンドは、 読み取られたテキストが暗号化されているかどうかを 経験的に類推して判断します。 一時バッファファイルも、 `-x` オプション用に入力された変形バージョンのキーを使用して、 暗号化されます。
`-w`*n*	デフォルト時のウィンドウサイズとして *n* を設定します。 低速の回線でエディタを使用している場合に便利です。
`-C`	暗号化オプション。上記 `-x` オプションと同様ですが、唯一の違いは `X` コマンドの代わりに `C` コマンドを実行する点です。この両コマンドの違いは、 `C` コマンドでは読み込まれたテキストは暗号化されているものと 無条件にみなされるという点です。
+*command* | `-c` *command*	`command` で示したエディタコマンドを冒頭に実行して編集処理を開始します。 通常は、検索または位置指定用のコマンドが用いられます。
filename	編集対象の 1 つまたは複数のファイル。

環境 次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/edit

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/edit

属性タイプ	属性値

使用条件	SUNWxcu4
CSI	対応済み

関連項目 ed(1), ex(1), vi(1), attributes(5), XPG4(5)

注意事項 暗号化オプションは、 セキュリティ管理ユーティリティパッケージで提供されていますが、 米国においてのみ使用可能です。

/usr/xpg4/bin/edit ユーティリティは /usr/bin/edit と同機能です。

名前 | egrep – 完全な正規表現を使用したファイル内のパターン検索

形式 | **/usr/bin/egrep** [-bchilnsv] [-e *pattern_list*] [-f *file*] [*strings*] [*file...*]

/usr/xpg4/bin/egrep [-bchilnsvx] [-e *pattern_list*] [-f *file*] [*strings*] [*file...*]

機能説明 | egrep (*expression grep*) ユーティリティは、ファイルの中の 文字パターンを検索し、そのパターンを含む行をすべて出力します。 egrep は、 パターンマッチング用に完全な正規表現 (英数字および特殊文字のフルセットを使用する文字列値による表現) を使用します。 また、 高速決定性アルゴリズムを使用しており、 これは指数的に増加する領域を必要とすることもあります。

ファイルが指定されない場合は、 egrep は、標準入力を入力とみなします。 検索された各行は、 通常、標準出力に出力されます。 複数の入力ファイルがある場合は、検索された各行の前にファイル名が出力されます。

/usr/bin/egrep | /usr/bin/egrep ユーティリティは、 regexp(5) のマニュアルページに述べるように完全な正規表現を受け付けますが、 \(と \) は除きます。 また、 以下の各項目が追加されます。

1. 後ろに + が付いている完全な正規表現。 これは、 その完全な正規表現が 1 回以上現れるものに一致します。
2. 後ろに ? が付いている完全な正規表現。 これは、 その完全な正規表現が 0 回 または 1 回現れるものに一致します。
3. | または復帰改行 (NEWLINE) で区切られた複数の完全な正規表現。 これは、 どちらかの完全な正規表現と一致する文字列に一致します。
4. グループ化のために、 丸かっこ () で囲まれた完全な正規表現。

完全な正規表現の中で $ 、 * 、 [、 ^ 、 | 、 (、) 、 \ の各文字を使用する際には注意が必要です。 これらの文字は、 シェルにとっても特別な意味があるからです。 完全な正規表現全体を単一引用符 '...' で囲むのがもっとも安全です。

演算子の優先順位は、 [ ] 、 * ? + 、 連結、 | 、 復帰改行の順です。

/usr/xpg4/bin/egrep | /usr/xpg4/bin/egrep ユーティリティは、 regex(5) のマニュアルページの「 EXTENDED REGULAR EXPRESSIONS 」の項で述べる 正規表現を使用します。

オプション | 以下のオプションは /usr/bin/egrep と /usr/xpg4/bin/egrep で使用できます。

-b	検索された各行の先頭にその行のブロック番号を付けます。 文脈によりブロック番号を見つけ出す場合に便利です (最初のブロックは 0 です)。
-c	一致したパターンを含む行の行数だけを出力します。
-e *pattern_list*	*pattern_list* (– で始まる完全な正規表現) を検索します。
-f *file*	file から完全な正規表現のリストを取り込みます。
-h	複数ファイルの検索時にファイル名を出力しません。

`-i` 比較時に大文字と小文字を区別しません。

`-l` 一致した行が 1 行でもあるファイルのファイル名だけを、 1 ファイルずつ復帰改行で区切って出力します。 パターンが 2 回以上見つかっても、 ファイル名の出力は 1 回だけです。

`-n` 各行の先頭にファイル内の行番号を付けます (最初の行は 1 です)。

`-s` エラーメッセージだけを表示します。 エラー状態を調べるのに便利です。

`-v` 一致するパターンを含む行以外のすべての行を出力します。

/usr/xpg4/bin/egrep

以下のオプションは、 `/usr/xpg4/bin/egrep` でのみ使用できます。

`-x` 固定文字列全体または正規表現と完全に一致する 入力行だけを、一致する行とみなします。

オペランド

以下のオペランドを指定できます。

file パターンを検索するファイルのパス名。 このオペランドを省略すると、標準入力が用いられます。

/usr/bin/egrep

pattern 入力の検索時に用いるパターンを指定します。

/usr/xpg4/bin/egrep

pattern 入力の検索時に用いる 1 つまたは複数のパターンを指定します。 このオペランドは、 `-e`*pattern_list* と指定されたものとして扱われます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `egrep` の動作については、`largefile`(5) を参照してください。

環境

`egrep` の実行に影響を与える環境変数 `LC_COLLATE`、`LC_CTYPE`、 `LC_MESSAGES`、`NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 一致するものが 1 つ以上見つかった

`1` 一致するものが 1 つも見つからなかった

`2` (一致するものが見つかった場合でも) 構文エラーが検出された、 またはアクセスできないファイルがあった

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/egrep

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/egrep

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目 `fgrep`(1), `grep`(1), `sed`(1), `sh`(1), `attributes`(5), `environ`(5), `largefile`(5), `regex`(5), `regexp`(5), `XPG4`(5)

注意事項 理想的には、 `grep` コマンドは 1 つだけにすべきですが、 広い範囲における消費領域と時間のかねあいに対応できるような 1 つのアルゴリズムはありません。

1 行は仮想記憶に使用できるサイズ に制限されています。

/usr/xpg4/bin/egrep `/usr/xpg4/bin/egrep` ユーティリティは `/usr/xpg4/bin/grep -E` と同一です (`grep`(1) を参照)。移植性の必要なアプリケーションは、 `/usr/xpg4/bin/grep -E` の方を使用してください。

名前 | eject – ドライブからの媒体 (CD-ROM やフロッピーなど) の取り出し

形式 | **eject** [-dfnpq] [*device* | *nickname*]

機能説明 | eject ユーティリティは、取り外し可能な媒体用のデバイスのうち、手動取り出しボタンが付いていないもの、あるいは、取り出しボタンが付いており、ボリューム管理 (vold(1M) のマニュアルページを参照) で管理されるものを対象に使用します。デバイスは名前またはニックネームで指定します。ボリューム管理が動作しており、デバイスを指定しない場合は、デフォルトのデバイスが使用されます。

このコマンドに応答するのは、プログラム制御下で eject コマンドをサポートしているデバイスだけです。eject コマンドの応答はボリューム管理が動作しているかどうかによって異なります。

ボリューム管理が動作している場合 | 手動でのみ取り出すことができる媒体で eject コマンドを実行すると、メディアの取り出し以外のすべての作業が行われます。たとえば、ファイルシステムがマウントされていれば、そのファイルシステムはマウント解除されます。この場合、eject は、手動でメディアを取り出す準備ができたことを知らせるメッセージを表示します。ウィンドウシステムが動作している場合、-p オプションを指定していなければ、このメッセージはポップアップウィンドウとして表示されます。ウィンドウシステムが動作していない場合、あるいは、-p オプションを指定した場合、このメッセージは標準エラー出力と、媒体が物理的に取り出されるシステムのシステムコンソールに表示されます。

ボリューム管理には、デバイスのパス名またはニックネームを指定しなかった場合に eject により使用されるデフォルトのデバイスの概念があります。どのデバイスがデフォルトで使用されるかを調べるには、-d オプションを使用します。

ボリューム管理が動作していない場合 | ボリューム管理が動作していないときにデバイスのパス名を指定した場合、eject はそのパス名に取り出しコマンドを送信します。パス名の代わりにニックネームを指定した場合、次のニックネームが eject により認識されます。

ニックネーム	パス
fd	/dev/rdiskette
fd0	/dev/rdiskette
fd1	/dev/rdiskette1
diskette	/dev/rdiskette
diskette0	/dev/rdiskette0
diskette1	/dev/rdiskette1
rdiskette	/dev/rdiskette
rdiskette0	/dev/rdiskette0
rdiskette1	/dev/rdiskette1

floppy	/dev/rdiskette
floppy0	/dev/rdiskette0
floppy1	/dev/rdiskette1

上記のリストは、-n オプションで再生成できます。

マウントされているファイルシステムが存在するデバイスから媒体を物理的に取り出してはなりません。eject はデバイスにマウントされているファイルシステムを自動的に検出し、媒体を取り出す前にファイルシステムをマウント解除しようとします (mount(1M) のマニュアルページを参照)。マウント解除が失敗した場合、eject は警告メッセージを出力して終了します。-f オプションを使用すると、デバイス上にファイルシステムがマウントされている場合でも、そのデバイスから媒体を強制的に取り出すことができます。-f オプションを使用できるのは、ボリューム管理が動作していない場合だけです。

eject は、デフォルトのデバイスとニックネームのリストを表示できます。

フロッピーディスクを挿入している場合、メディアを取り出す前に必ず volcheck(1) を実行し、ボリューム管理にフロッピーが入っていることを知らせなければなりません。

オプション

次のオプションを指定できます。

-d　デフォルトで取り出されるデバイス名を表示します。

-f　ビジー状態 (つまり、ファイルシステムがマウントされている状態) でも、強制的に媒体をデバイスから取り出します。ボリューム管理が動作していない場合にのみ有効です。

-n　ニックネームからデバイス名への変換テーブルを表示します。

-p　eject_popup プログラムを呼び出しません。

-q　メディアが存在するかどうかを照会します。

オペランド

次のオペランドを指定できます。

device　eject の実行対象となるデバイスを /dev ディレクトリ中のデバイス名で指定します。

nickname　eject の実行対象となるデバイスを、eject が認識できるニックネームで指定します。

使用例

例 1 ボリューム管理が動作している状態で CD-ROM を取り出す

ボリューム管理が動作しているときに CD-ROM ドライブから CD を取り出すには、次のように入力します。ただし、CD-ROM ドライブは 1 つだけと仮定します。

```
example> eject cdrom0
```

例 1 ボリューム管理が動作している状態で CD-ROM を取り出す　(続き)

例 2 ボリューム管理が動作していない状態で CD-ROM を取り出す

ボリューム管理が動作していないときに、パス名が /dev/dsk/c0t3d0s2 の CD-ROM ドライブから CD を取り出すには、次のように入力します。

```
example> eject /dev/dsk/c0t3d0s2
```

例 3 フロッピーディスクを取り出す

ボリューム管理が動作しているかどうかに関係なく、フロッピーディスクを取り出すには、次のように入力します。

```
example> eject floppy0
```

終了ステータス

次の終了ステータスが返されます。

- 0　操作が正常に終了した、あるいは、-q オプションを指定した場合には媒体がドライブに入っていた
- 1　操作が失敗した。あるいは、-q オプションを指定した場合には媒体がドライブに入っていなかった
- 2　無効なオプションが指定された
- 3　ioctl() 要求が失敗した
- 4　手動で媒体を取り出す準備ができた

ファイル

/dev/diskette0	デフォルトのフロッピーディスクファイル
/dev/sr0	デフォルトの CD-ROM ファイル (将来のリリースでは削除される予定)
/dev/dsk/c0t6d0s2	デフォルトの CD-ROM ファイル
/usr/lib/vold/eject_popup	手動で媒体を取り出す準備ができたことを知らせるポップアップ

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

volcancel(1), volcheck(1), volmissing(1), mount(1M), rmmount(1M), vold(1M), ioctl(2), rmmount.conf(4), vold.conf(4), attributes(5), volfs(7FS)

診断

未知のオプションを指定した場合、短いヘルプメッセージが出力されます。指定されたデバイスを開くことができなかった場合、あるいは、指定されたデバイスが eject をサポートしていない場合、診断メッセージが出力されます。

`Device Busy`	マウントされたファイルシステムが存在するデバイスから媒体を取り出そうとしました。マウントされたファイルシステムが存在するデバイスから強制的に媒体を取り出そうとすると、警告メッセージが出力されます。

使用上の留意点

ユーザーごとにデフォルトを変更する方法がありません。

ボリューム管理が動作していない場合、現在マウントされているボリュームを取り出すことができます (`mount`(1M) のマニュアルページを参照)。たとえば、`/dev/dsk/c0t3d0s2` にある CD-ROM ドライブが `/mnt` にマウントされている場合、(ボリューム管理が動作していなければ)、次のコマンドは動作します 。

```
example> eject /dev/dsk/c0t3d0s0
```

これは、スライス s0 と s2 の両方が CD-ROM ドライブ全体を参照しているためです。

名前 | elfdump – オブジェクトファイルの選択部分のダンプ

形式 | **elfdump** [-cdeihknprsvG] [-N *name*] [-w *file*] *filename...*

機能説明 | elfdump ユーティリティは、指定されたオブジェクトファイルの選択部分をシンボルでダンプします。オプションを指定することにより、オブジェクトファイルの特定部分を抽出できます。

elfdump ユーティリティは、dump(1) ユーティリティと機能的に似ています。elfdump のインタフェースの方が dump よりも新しく、ユーザーフレンドリです。ただし、シェルスクリプト内で使用するような場合は、dump ユーティリティの方が適しています。

elfdump は、ar(1) で作成したアーカイブファイルの検査にも使用できます。その場合、アーカイブ内の各オブジェクトは、指定されたオプションに従って処理されます。

表示される情報についての詳細は、『リンカーとライブラリ』を参照してください。

オプション | 次のオプションを指定できます。

-c　セクションヘッダー情報をダンプします。

-C　C++ シンボル名を復号化します。

-d　.dynamic セクションの内容をダンプします。

-e　ELF ヘッダーをダンプします。

-g　.group セクションの内容をダンプします。

-G　.got セクションの内容をダンプします。

-h　.hash セクションの内容をダンプします。

-i　.interp セクションの内容をダンプします。

-k　ELF チェックサムを計算します (gelf_checksum(3ELF) のマニュアルページを参照)。

-m　.SUNW_move セクションの内容をダンプします。

-n　.note セクションの内容をダンプします。

-N *name*　指定された *name* だけにオプションを適用します。たとえば、複数のシンボルテーブルが入っているファイルにおいて .dynsym テーブルだけを表示するには、次のコマンドを使用します。

```
example% elfdump -s -N .dynsym filename
```

-p　プログラムヘッダーをダンプします。

-r　再配置セクション (つまり、.rel[a]) の内容をダンプします。

`-s` シンボルテーブルセクション (つまり、`.dynsym` や `.symtab`) の内容をダンプします。アーカイブの場合は、アーカイブのシンボルテーブルをダンプします。各セクションは `-N` オプションで指定できます。アーカイブシンボルテーブルは、特殊なセクション名 `ARSYM` を使用した `-N ARSYM` オプションによって指定できます。

ver ヘッダーの下に、標準のシンボルテーブル情報に加えて、シンボルのバージョン定義索引も出力します。

`-v` バージョンセクション (つまり、`..SUNW_`*version*) の内容をダンプします。

`-w` *file* `-N` オプションで指定されたセクションの内容を指定されたファイル (*file*) に書き込みます。別の処理のために個々のセクションのデータを抽出するときに便利です。たとえば、ファイル内の `.text` セクションを抽出するには、次のように指定します。

```
example% elfdump -w text.out -N .text filename
```

`-y` `.SUNW_syminfo` セクションの内容をダンプします。

オペランド

次のオペランドを指定できます。

filename 指定されたオブジェクトファイルの名前

ファイル

`liblddbg.so` リンカーデバッグ用ライブラリ

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWbtool

関連項目

`dump`(1), `nm`(1), `pvs`(1), `elf`(3ELF), `attributes`(5)

『リンカーとライブラリ』

名前 | enable, disable – LP プリンタを使用可能または不可能に変更

形式 | **/usr/bin/enable** *printer*...

/usr/bin/disable [-c | -W] [-r [*reason*]] *printer*...

機能説明 | enable コマンドは、 プリンタを起動して、 lp コマンドで投入される要求を印刷できるようにします。 enable はプリンタサーバー上で動作します。

disable コマンドは、 プリンタを使用不可能にして、 lp コマンドによる要求を印刷できないようにします。 デフォルトでは、 *printer* 上で disable が起動されているとき、印刷を実行していた要求は、 使用可能になると *printer* または同じクラスに属する 他のプリンタ上ですべて再印刷されます。 disable はプリンタサーバー上で動作します。

なお、プリンタの状況を調べるには、 lpstat -p コマンドを実行してください。

enable と disable コマンドは、プリンタサーバーのスプール用システム上の 待ち行列に対してのみ有効です。 これらのコマンドをクライアントシステムから 実行しても、サーバー上には何も起こりません。

オプション | 以下のオプションを指定できます。

-c *printer* 上で現在印刷されている要求も キャンセルします。このオプションは -W と一緒に指定することはできません。 プリンタがリモートの場合には、 -c オプションは無視されますが、 それに関するメッセージは出力されません。

-W *printer* 上で現在印刷されている要求が 終了するのを待って、 *printer* を使用不可能にします。 このオプションは、 -c と一緒に指定することはできません。 プリンタがリモートの場合には、 -W オプションは無視されますが、 それに関するメッセージは出力されません。

-r [*reason*] プリンタを使用不可能にする理由を *reason* 引数で文字列として記述します。 複数のプリンタを指定した場合、そのすべてにこの理由が適用されます。 ここで指定した理由は、プリンタの状況を調べる lpstat -p コマンドの出力中に表示されます。 理由を示す文字列は、 空白文字を含んでいる場合には引用符で囲まなければなりません。 理由のデフォルト値は、既存の宛先に関しては unknown reason 、システムに追加されてまだ使用可能になったことのない宛先に関しては new destination となります。

オペランド | 以下のオペランドを指定できます。

printer 使用可能または不可能にするプリンタ名。 名前を使用して *printer* を指定します。 名前の命名規約については printers.conf(4) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 正常終了

0 以外　　エラーが発生した

ファイル

/var/spool/lp/*　　LP 印刷待ち行列

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpcu
CSI	対応済み

関連項目

lp(1), lpstat(1), printers.conf(4), attributes(5)

名前 | env – コマンド実行のための環境の設定

形式 | **/usr/bin/env** [-i | -] [*name=value...*] [*utility* [*arg* ..]]

/usr/xpg4/bin/env [-i | -] [*name=value...*] [*utility* [*arg* ..]]

機能説明 | env ユーティリティは現在の環境を読み込み、 引数の指定に従ってその値を変更し、 *utility* 引数で指定されたユーティリティを変更された状態の環境で実行します。

任意指定の引数も、 *utility* に渡されます。 *utility* が省略された場合、 変更後の環境の内容が標準出力に書き出されます。 この出力には、名前と値が *name=value* の形式で 1 行に 1 組ずつ書かれます。

/usr/bin | env が引数のあるコマンドを実行する場合、 デフォルトのシェル /usr/bin/sh が使われます (sh(1) 参照) 。

/usr/xpg4/bin | env が引数のあるコマンドを実行する場合、/usr/xpg4/bin/sh が使われます (ksh(1) 参照) 。

オプション | 以下のオプションを指定できます。

-i | - 現在のシェルから引き継いだ環境はすべて無視し、 引数で指定された環境だけを用いて *utility* を実行します。

オペランド | 以下のオペランドを指定できます。

name=value この形式でオペランドを指定すると、 *name* が示す環境変数に *value* が示す値を設定して実行環境を変更し、さらに *utility* を実行する前に、受け継いだ環境にその指定を割り当てます。

utility 実行するユーティリティの名前。 シェルの特殊な組み込みユーティリティを指定した場合の 結果は定義されていません。

arg 実行するユーティリティに引数として渡す文字列。

使用例 | **例 1** 新しい PATH の値でユーティリティを呼び出す

```
example% env   -i PATH=/mybin mygrep xyz myfile
```

ここでは環境変数 PATH を設定し、これを唯一の環境設定として mygrep という ユーティリティを実行しています。 つまり mygrep は、 PATH が示す /mybin 中に存在していなければなりません。

環境 | env の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | *utility* で指定したユーティリティが実行された場合、 そのユーティリティの終了ステータスが env の終了ステータスとなります。実行されなかった場合には、 env は以下の終了ステータスを返します。

0 正常終了

1-125 エラーが発生した

126 *utility* は見つかったが実行することができなかった

127 *utility* は見つからなかった

属性 次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/env

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/env

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目 ksh(1), sh(1), exec(2), profile(4), attributes(5), environ(5), XPG4(5)

名前 | eqn, neqn, checkeq – 数学的記述のタイプセット

形式 | **eqn** [-d *xy*] [-f *n*] [-p *n*] [-s *n*] [*file*...]

neqn [*file*...]

checkeq [*file*...]

機能説明 | eqn および neqn は、数式を記述するのに便利な言語プロセッサです。eqn は troff(1) 用のプリプロセッサで、 troff の出力を印刷できる装置用に提供されています。一方、neqn は nroff(1) 用のプリプロセッサで、端末での出力用に提供されています。通常この 2 つのコマンドは、以下の形式で指定します。

```
example% eqn file . . . | troff
example% neqn file . . . | nroff
```

ファイル名を表す *file*引数を省略すると、eqn または neqn は標準入力から読み込みます。数式の開始を示すには、行の先頭に .EQ を記述します。同様に数式の終了は、行の先頭に .EN を記述して表します。この 2 つの行は変換されないので、センタリングや番号付けなどを行うマクロパッケージ中に定義しておくこともできます。また、一対の文字を「区切り記号」として設定し、区切り記号に囲まれたテキストを eqn 入力として処理させることもできます。

区切り記号や .EQ/.EN が存在しない、または対で指定されていない場合、checkeq はメッセージを出力します。

オプション | 以下のオプションが指定できます。

-d*xy* コマンド行引数で指定される数式の区切り記号として、文字 *x* と *y* を指定します。ただしこの方法よりも、.EQ と .EN の間で delim *xy* を使って区切り記号を指定する方法がより一般的です。*x* と *y* には同じ文字を指定することも可能です。テキスト中に delim off と記述すると、区切り記号は有効でなくなります。区切り記号にも .EQ と .EN にも囲まれていないテキストは、すべてそのまま渡されます。

-f*n* ドキュメント全体を通じて使用するフォントとして *n* を指定します。このグローバルフォントの設定は、ドキュメントの本文中に gfont *n* 命令を指定して変更することもできます。*n* はフォント指定です。

-p*n* 下付きおよび上付きの添字のサイズを、直前の文字サイズより *n* ポイントだけ小さくします。 -p オプションを省略すると、添字のサイズは 3 ポイント小さくなります。

-s*n* ドキュメント全体を通じて使用する文字サイズとして *n* を指定します。このグローバルサイズの設定は、ドキュメントの本文中に gsize *n* 命令を指定して変更することもできます。*n* はポイントサイズです。

オペランド | 以下のオペランドが指定できます。

file eqn または neqn によって処理される nroff のファイルまたは troff のファイル。

EQN 言語

この説明を nroff を使って端末画面に表示した場合、端末画面の制限から neqn による出力箇所は正確には表示できません。出力の正確な表示を確認するために、このページを印刷してご覧ください。

eqn 中のトークンは、中括弧、二重引用符、チルド、山型記号、空白文字、タブ、または復帰改行文字で区切られます。中括弧 { } は、グループ分けに用いられます。一般的には、たとえば *x* のような 1 つの文字が記述できる箇所であれば、中括弧で囲んだ複雑な記述を代わりに指定できます。チルド (~) は出力中における 1 文字分の空白を、山型記号 (^) は半文字分の空白を表します。

下付きおよび上付きの添字:
これらは、キーワード sub と sup を使って生成できます。

`x sub i` という記述の出力結果は x_i になります。

`a sub i sup 2` の出力は $a_i^{\ 2}$ になります。

`e sup {x sup 2 + y sup 2}` の出力は $e^{x^2}+y^2$ になります。

分数:
分数は、キーワード over で指定します。

```
a over b
```

の出力結果は、

$$\frac{a}{b}$$

となります。

平方根の式:
平方根の式は、キーワード sqrt で指定します。

```
1 over sqrt {ax sup 2 +bx+c}
```

の出力結果は、

$$\frac{1}{\sqrt{ax^2+bx+c}}$$

となります。

制限値:
キーワード from と to は、種々の指定における最小値と最大値を表します。

```
lim from {n→ inf } sum from 0 to n x sub i
```

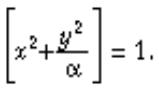
の出力結果は、

となります。

括弧:
大括弧、中括弧などを適切な高さで出力するには、左括弧に `left` を、右括弧には `right` をそれぞれ使用します。

```
left [ x sup 2 + y sup 2 over alpha right ] ~=~1
```

と記述すれば



という出力が得られます。なお、`right` 文節は省略することができます。キーワード `left` と `right` の直後に指定できる文字は、大括弧、中括弧、縦棒、上端と下端を表す `c` と `f`、何もない旨を示す `""` (対になるべき括弧のうち右括弧だけを使う場合に便利) です。

分数を縦に重ねる:
分数を縦に重ねるには、`pile`、`lpile`、`cpile`、または `rpile` を使用します。

```
pile {a above b above c}
```

という記述により

が出力されます。何重に積み重ねてもかまいません。文字を合わせる位置は、`lpile` は左詰め、`pile` と `cpile` はともにセンタリング (ただし縦方向の間隔が異なる)、そして `rpile` は右詰めとなります。

行列:
行列は `matrix`というキーワードで生成されます。

```
matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }
```

と記述すると、

x_i 1
y_2 2

が出力されます。カラムを右詰めにするには rcol を使用します。

文字の上下に付加する記号:
発音符記号のように文字の上下に付加する記号は、dot、dotdot、hat、tilde、bar、vec、dyad、underを使って指定できます。

```
x dot = f(t) bar
```
という記述の出力結果は、

$\dot{x} = \overline{f(t)}$

```
y dotdot bar ~=~ n under
```
の結果は、

$\ddot{\bar{y}} = \underline{n}$,

```
x vec ~=~ y dyad
```
の結果は、

$\vec{x} = \vec{\breve{y}}$.

文字のサイズとフォント:
文字のサイズやフォントの変更は、size *n* または size ±*n*、roman、italic、bold、font *n* で指定します。ドキュメント全体を通じてグローバルに使用する文字サイズとフォントは、gsize *n* と gfont *n* をドキュメント中に指定するか、またはコマンド行引数の -s*n* と -f*n* を使って変更できます。

表示引数の位置:
一連の表示引数の位置をそろえることもできます。先頭の数式において、そろえたい表示引数の直前に mark と記述します。さらに後続の数式において、それと合わせたい表示引数の直前に lineup と記述します。

短縮形:
入力の短縮形を定義したり既存のキーワードを再定義するには、define を使用します。次に例を示します。

`define` *thing* `%` *replacement* `%`
これにより *thing* というトークンが新たに定義され、その後このトークンが現れるたびに *replacement* に置き換えられます。なお `%` の位置には、任意の文字 (ただし *replacement* に含まれていないもの) を指定できます。

キーワードと短縮形:
`sum int inf` のようなキーワード、および `>=` → や `!=` のような短縮形も処理されます。

ギリシャ文字:
ギリシャ文字は `alpha` または `GAMMA` のように、大文字・小文字のうち希望する方のつづりで出力できます。

数学用語:
`sin`、`cos`、`log` のような数学用語は自動的にローマン字体で出力されます。

\(bu (•) のような 4 文字からなる `troff`(1) のエスケープコードは、どこでも記述できます。二重引用符に囲まれた文字列 `"..."` は、そのまま渡されます。これによりキーワードをテキストとして入力でき、また (他の方法が使えないとき) `troff` との通信用に使うことができます。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目

`nroff`(1), `tbl`(1), `troff`(1), `attributes`(5), `ms`(5)

使用上の留意点

数字や括弧をボールドで出力したい場合、`bold "12.3"` のように引用符で囲んでください。

名前 | ckrange, errange, helprange, valrange – 整数の入力要求とその検証

形式 | **ckrange** [-Q] [-W *width*] [-l *lower*] [-u *upper*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errange [-W *width*] [-e *error*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/helprange [-W *width*] [-h *help*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/valrange [-l *lower*] [-u *upper*] [-b *base*] *input*

機能説明 | ckrange ユーティリティは、ユーザーに指定範囲内の整数の入力を要求して、ユーザーの応答が有効かどうかを検証します。このユーティリティでは、ユーザーに指定範囲内の整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

また、このコマンドは、有効な入力範囲も定義します。下限または上限のいずれかが定義されていない場合、範囲は一方の限界だけに制限されます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckrange コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errrange (標準出力上にエラーメッセージを書式化して表示する)、helprange (標準出力上にヘルプメッセージを書式化して表示する)、および valrange (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

注：*input* 引数として負の値を指定すると、valrange 内の getopt で混乱が生じます。引数の前に - を付けると、getopt は処理を停止します。getopt のパラメータ処理については、getopt(1) および intro(1) のマニュアルページを参照してください。getopt は、位置指定パラメータを解析して、有効なオプションかどうかを検査するために使用されます。

オプション | 次のオプションを指定できます。

-b *base* | 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。基数を変換するには strtol(3C) を使用します。出力は常に、基数を 10 として実行されます。

-d *default*	*default* をデフォルト値として定義します。*default* は strtol(3C) を使用して指定した基数に変換できます。指定した基数に無効な文字があると、strtol はエラーを示すことなく変換を終了します。
-e *error*	*error* をエラーメッセージとして定義します。
-h *help*	*help* をヘルプメッセージとして定義します。
-k *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
-l *lower*	*lower* を範囲の下限として定義します。デフォルト値は、マシンの負の最大整数です。
-p *prompt*	*prompt* をプロンプトメッセージとして定義します。
-Q	有効な応答として終了 (quit) を使用できないようにします。
-s *signal*	終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。
-u *upper*	*upper* を範囲の上限として定義します。デフォルト値は、マシンの正の最大整数です。
-W *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定します。

input	上限、下限、および基数と照合される入力

使用例

例 1 デフォルトのプロンプト

ckrange のデフォルトのプロンプトは、次のとおりです (基数は 10)

```
example% ckrange Enter an integer between lower_bound and
upper_bound [lower_bound–upper_bound,?,q]:
```

例 2 デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/errange
ERROR: Please enter an integer between lower_bound \
     and upper_bound.
```

例 3 デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/helprange
Please enter an integer between lower_bound and upper_bound.
```

例 **3** デフォルトのヘルプメッセージ　*(続き)*

例 **4** 基数を 10 以外の数に変更した場合のメッセージ

基数を 10 以外の数に設定した場合、メッセージは "integer" から "base *base* integer" に変更されます。次に例を示します。

```
example% /usr/sadm/bin/helprange -b 36
```

例 **5** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **6** valrange モジュールの使用

`valrange` モジュールは、標準エラー出力に使用法に関するメッセージを作成します。正常終了したな場合は `0`、失敗した場合には `0` 以外の値を返します。

```
example% /usr/sadm/bin/valrange
usage: valrange [-l lower] [-u upper] [-b base] input
```

終了ステータス

次の終了ステータスが返されます。

`0`　正常終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`2`　使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`intro`(1), `face`(1), `getopt`(1), `strtol`(3C), `attributes`(5), `signal`(3HEAD)

名前 | ckdate, errdate, helpdate, valdate – 日付の入力要求とその検証

形式 | **ckdate** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errdate [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helpdate [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valdate [-f *format*] *input*

機能説明 | ckdate ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに日付の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザーの応答は、date コマンドで定義されている書式に一致したものでなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckdate コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errdate (エラーメッセージを書式化して表示する)、helpdate (ヘルプメッセージを書式化して表示する)、および valdate (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errdate および helpdate の各モジュールに format が定義されている場合、メッセージには、指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は、どのような書式も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。指定できる書式とその定義は、次のとおりです。

- %b = 省略された月の名前 (jan、feb、mar)
- %B = 完全な月の名前 %d = 日 (01 ～ 31)
- %D = %m/%d/%y の形式の日付 (デフォルトの書式)
- %e = 日 (1 ～ 31、1 桁の数字の前にはブランクが挿入されます)

`%h` = 省略形の月の名前。`%b`%と同じ

`%m` = 月 (01 〜 12)

`%y` = 西暦中の年 (たとえば、89)

`%Y` = CCYY の形式の年 (たとえば、1989)

`-h` *help* `help` をヘルプメッセージとして定義します。

`-k` *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

`-p` *prompt* *prompt* をプロンプトメッセージとして定義します。

`-Q` 有効な応答として終了 (quit) を使用できないように指定します。

`-s` *signal* 終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル `signal` を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、`width` の行長に書式化します。

オペランド

次のオペランドを指定できます。

input 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0` 正常な終了

`1` 入力で EOF が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`3` ユーザー終了 (quit)

`4` 不正な書式引数

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckdate` のデフォルトのプロンプトは、次のとおりです。

```
Enter the date [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter a date.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter a date. Format is <format>.
```

終了 (quit) オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valdate` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckgid, errgid, helpgid, valgid – グループ ID の入力要求とその検証

形式 | **ckgid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errgid [-W *width*] [-e *error*]

/usr/sadm/bin/helpgid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valgid *input*

機能説明 | ckgid は、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに既存のグループ ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckgid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errgid (エラーメッセージを書式化して表示する)、helpgid (ヘルプメッセージを書式化して表示する)、および valgid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような書式基準も満たす必要はありません。

-e *error* — *error* エラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するように指定します。

-m — ユーザーがヘルプを要求した場合、あるいはユーザーの入力が不正な場合は、グループ名の一覧を表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/group` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckgid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the name of an existing group [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

デフォルトのヘルプメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valgid` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckint, errint, helpint, valint – 整数値の入力要求とその検証

形式 | **ckint** [-Q] [-W *width*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errint [-W *width*] [-b *base*] [-e *error*]

/usr/sadm/bin/helpint [-W *width*] [-b *base*] [-h *help*]

/usr/sadm/bin/valint [-b *base*] *input*

機能説明 | ckint ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白（復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckint コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errint (エラーメッセージを書式化して表示する)、helpint (ヘルプメッセージを書式化して表示する)、および valint (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errint と helpint の各モジュールに *base* が定義されている場合、メッセージには入力する整数の基数が含まれます。

オプション | 次のオプションを指定できます。

-b *base* — 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。

-d *default* — *default* をフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

`-s` *signal*　終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-W` *width*　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　基数 *base* の基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　正常な終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckint` のデフォルトのプロンプトは、次のとおりです (基数は 10)。

```
Enter an integer [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
ERROR - Please enter an integer.
```

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
Please enter an integer.
```

基数を 10 以外の数に設定した場合は、上記のメッセージは "integer" から "base *base* integer" に変更されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valint` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前

ckitem, erritem, helpitem – メニューの作成、およびメニュー項目の入力要求とその検証

形式

ckitem [-Q] [-W *width*] [-uno] [-f *filename*] [-l *label*] [[-i *invis*] [,...]] [-m *max*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]] [*choice* [...]]

/usr/sadm/bin/erritem [-W *width*] [-e *error*] [*choice* [..]]

/usr/sadm/bin/helpitem [-W *width*] [-h *help*] [*choice* [..]]

機能説明

ckitem ユーティリティは、メニューを作成し、ユーザーにプロンプトを出力してメニュー項目から 1 つの項目を選択するように促します。さらに、ユーザーの応答入力を検証します。このコマンドのオプションでは、プロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) を定義します。

デフォルトでは、メニューは書式化されており、各項目の前に番号が付けられ、端末上に複数の列で出力されます。列の長さは、選択した最大長の項目によって決まります。項目はアルファベット順に表示されます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckitem コマンドには、2 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erritem (エラーメッセージを書式化して表示する) と helpitem (ヘルプメッセージを書式化して表示する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。これらのモジュールに *choice* が定義されている場合、メッセージには利用可能なメニューの選択項目が含まれます。

オプション

次のオプションを指定できます。

-d *default*
: *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error*
: *error* をエラーメッセージとして定義します。

-f *filename*
: *filename* を表示されるメニュー項目のリストを含むファイルとして定義します。(このファイルの書式は、token<tab>description です。ポンド記号 (#) で始まる行は注釈として指定され、無視されます。)

-h *help*
: *help* をヘルプメッセージとして定義します。

-i *invis*
: 非表示メニュー項目 (メニューに表示されない選択項目) を定義します。(たとえば、非表示項目として使用される "all" は、有効なオ

プションですが、メニューには表示されません。任意の数の非表示項目を定義できます。) 非表示項目があることを、プロンプトメッセージまたはヘルプメッセージのどちらかで、ユーザーに知らせるようにします。

`-k` *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

`-l` *label* メニューの上に表示するラベル、*label* を定義します。

`-m` *max* ユーザーが選択できるメニュー項目の最大数を定義します。デフォルト値は 1 です。

`-n` メニュー項目をアルファベット順で表示しないようにします。

`-o` 1 つのメニュートークンだけが返されるようにします。

`-p` *prompt* *prompt* をプロンプトメッセージとして定義します。

`-Q` 有効な応答として終了 (quit) を使用できないようにします。

`-s` *signal* 終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-u` メニュー項目に番号を付けずに表示するようにします。

`-W` *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

choice メニュー項目を定義します。項目は空白か復帰改行で区切ります。

終了ステータス

次の終了ステータスが返されます。

`0` 正常な終了

`1` 入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、`-f` オプションでファイルが開けない、あるいは使用法に誤りがあった

`3` ユーザー終了 (quit)

`4` 選択すべき項目がない

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

ユーザーは、メニュー項目に番号が付いている場合はその項目の番号を、あるいは、その項目を一意に識別するのに必要な長さの文字列を入力できます。長いメニューはページに分割され、各ページには 10 個の項目が表示されます。

メニュー項目が、`-f` オプションで指定したファイルとコマンド行の両方に定義されている場合、メニュー項目は通常、アルファベット順に表示されます。ただし、アルファベット順での表示を抑制する `-n` オプションが使用されている場合は、ファイルに定義された項目が最初に表示され、次にコマンド行に定義されたオプションが表示されます。

`ckitem` のデフォルトのプロンプトは次のとおりです。

```
Enter selection [?,??,q]:
```

1 つの疑問符はヘルプメッセージを表示してから、プロンプトを再表示します。2 つの疑問符は、ヘルプメッセージを表示してから、メニューラベル、メニュー、およびプロンプトを再表示します。

番号を入力した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Bad numeric choice specification
```

文字列を入力した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Entry does not match available menu selection.
Enter the number of the menu item you wish to select,
the token which is associated with the menu item,
or a partial string which uniquely identifies the token
for the menu item. Enter ?? to reprint the menu.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Enter the number of the menu item you wish to select,
the token which is associated with the menu item,
or a partial string which uniquely identifies the token for
the menu item. Enter ? to reprint the menu.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。

名前 | error – ソース行の右側へのコンパイラエラーメッセージの挿入

形式 | **error** [-n] [-q] [-s] [-v] [-t *suffixlist*] [-I *ignorefile*] [*filename*]

機能説明 | error は、コンパイラや言語プロセッサにより生成されたエラーメッセージを解析します。error は、従来の読みづらい省略形だらけのエラーメッセージを解析し、エラーメッセージとソースコードを同時に表示できるようにします。

error は、指定されたファイル (*filename*) または標準入力からエラーメッセージを読み取り、次の処理を行います。

- 各エラーメッセージを生成した言語プロセッサを判別します。
- エラー行を含むファイルの名前と行番号を判別します。
- ソースファイル内でエラー行の直前にエラーメッセージを挿入します。

言語プロセッサや内容を特定できなかったエラーメッセージは、ファイルに挿入されず標準出力に送られます。error がソースファイルを処理するのは、すべての入力を読み込んだ後だけです。

error は、その標準入力がエラーメッセージの送信元とパイプを介して接続されている状態で実行するように設計されています。言語プロセッサの中には、エラーメッセージを独自の標準エラーファイルに格納するものも、標準出力に送信するものがあります。いずれの場合でも、エラーメッセージの送り元はパイプを介して error に接続されていなければなりません。たとえば、csh 構文を使用していると仮定すると、次のコマンドは、lint の実行時に make(1S) によって実行されたプログラムが生成したすべてのエラーメッセージを解析します。

```
example% make -s lint | & error -q -v
```

error は、as(1)、cpp(1)、ld(1)、cc(1B)、make(1S) などのコンパイラが生成したエラーメッセージを識別できます。Pascal 以外のすべての言語において、エラーメッセージは 1 行に制限されています。エラーメッセージの中には、複数のファイル内の複数の行を示すものもあります。この場合、error はエラーメッセージを複製し、該当するすべての場所にエラーメッセージを挿入します。

オプション | 次のオプションを指定できます。

-n　ファイルを処理しません。すべてのエラーメッセージは標準出力に送られます。

-q　error はファイルを処理するかどうかを尋ねます。継続するには、質問に対して y または n で答えます。-q オプションを指定しない場合、すべての参照されるファイル (破棄されるエラーメッセージを参照するファイルを除く) が処理されます。

-s　エラーの分類に従った統計を出力します。

-v　すべてのファイルを処理した後で、処理したすべてのファイルを vi で開き、最初に処理したファイル内の最初のエラーの位置を示します。vi(1) が見つからない場合、ex(1) または ed(1) を標準の場所から実行しようとします。

`-t` *suffixlist*　後続の引数を接尾辞リストとして扱います。接尾辞リストに現れない接尾辞を持つファイルは処理されません。接尾辞リストはドットで区切られたリストで、`*` ワイルドカードも使用できます。たとえば、次の接尾辞リストを使用すると、`error` は `.c`、`.y`、`.f*`、`.h` で終了するファイルを処理します。

```
.c.y.f*.h
```

`error` は割り込みと終了を捕捉し、適切に終了します。

使用例

例 **1** `error` コマンドの例

次の C シェル（`/usr/bin/csh`）の例では、`error` は FORTRAN コンパイラから入力を受け取ります。

```
example% f77 -c any.f |& error options
```

K シェル（`/usr/bin/ksh`）を使用した場合の例は、次のとおりです。

```
example% f77 -c any.f 2>&1 | error options
```

使用法

`error` は、次の 6 つの分類基準に従ってエラーメッセージを処理します。

同期　言語プロセッサの中には、現在処理中のファイルを示す簡単なエラーメッセージを生成するものがあります。言語プロセッサが各エラーメッセージ中にファイル名を含めない場合、`error` はこのような簡単なエラーメッセージを使用してファイル名を判別します。このような同期メッセージは、`error` により完全に取り除かれます。

破棄　`lint` からのエラーメッセージの中で、2 つの `lint` ライブラリ、`/usr/lib/lint/llib-lc` と `/usr/lib/lint/llib-port` のいずれかに関連するメッセージは破棄され、間違ってこれらのライブラリが処理されないようにします。このようなメッセージも `error` により完全に取り除かれます。

無視　`lint` からのエラーメッセージの中で、あまり大切でない診断メッセージを生成することがわかっている関数を示すメッセージは無視することができます。無視されたエラーメッセージはソースファイルには挿入されず、標準出力に書き込まれます。無視する関数名は、ユーザーのホームディレクトリにある `.errorrc` ファイル、または `-I` オプションで指定したファイルから取られます。このファイルが存在しない場合、エラーメッセージは無視されません。このファイルが存在するときは、無視する関数名はそのファイル内で 1 行に 1 つずつ記述されていなければなりません。

ファイル固有でない	判別できないエラーメッセージは 1 つのグループにまとめられ、ファイルが処理される前に標準出力に書き込まれます。このようなエラーメッセージは、ソースファイルには挿入されません。
ファイルに固有	特定のファイルは示されているが特定の行が示されていないエラーメッセージは、そのファイルが処理されるときに標準出力に書き込まれます。
本当のエラー	判別可能なエラーメッセージは、そのエラーメッセージが示すファイルに挿入される候補となります。

ソースファイルに挿入されるのは、本当のエラーメッセージだけです。他のエラーメッセージは、`error` により完全に取り除かれるか、標準出力に書き込まれます。`error` はソースファイル内で、エラーメッセージの行番号が示す行の前にエラーメッセージを挿入します。各エラーメッセージはその言語におけるコメントとなります。内部的には、エラーの先頭には文字列 `###` が付けられ、エラーの終わりには文字列 `%%%` が付けられます。これにより、エディタで作業するとき、エラーを簡単に検出し削除できます。さらに、各エラーメッセージには、そのエラーメッセージが示す行の行番号が入っています。記述規則に則っとったソースプログラムであれば、エラーメッセージを削除しなくてもそのまま再コンパイルでき、エラーメッセージ自体が新たなエラーを引き起こすこともありません。C や Pascal のように記述規則を持たない言語で書かれた、記述の整っていないソースプログラムでは、コメントを別のコメント内に挿入することは可能ですが、新たにコンパイルするときに問題が発生する原因となります。これを回避するには、コメントの終わりと同じ行に言語文がないように、ソースプログラムの記述方法を変更します。

ファイル

`~/.errorrc`	`lint` のエラーメッセージで無視する関数名
`/dev/tty`	ユーザーの端末

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWbtool

関連項目

`as`(1), `cc`(1B), `cpp`(1), `csh`(1), `ed`(1), `ex`(1), `make`(1S), `ld`(1), `vi`(1), `attributes`(5)

使用上の留意点

tty デバイスは、ユーザー入力用に直接開いてください。

リンク付きのソースファイルは、リンクを 1 つだけ持つファイルのコピーを新たに作成します。

言語プロセッサのエラーメッセージの形式を変更すると、`error` がエラーメッセージを判別できなくなります。

error は純粋に機械的な処理を行うため、構文として重要でない 1 つのエラーによって発生した大量のエラーをフィルタすることはできません。このようなエラーは手動で破棄してください。

Pascal のエラーメッセージは実際にエラーが発生した行の後に置かれますが、error はメッセージを前に置きます。'|' マークによるエラー箇所の指定も error のメッセージではずれてしまいます。

error は、ある程度高速な CRT で動作するように設計されています。低速の端末では使いづらく、ハードコピー端末で使用するようには設計されていません。

名前 | ckpath, errpath, helppath, valpath – パス名の入力要求とその検証

形式 | **ckpath** [-Q] [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errpath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-e *error*]

/usr/sadm/bin/helppath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-h *help*]

/usr/sadm/bin/valpath [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] *input*

機能説明 | ckpath ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーにパス名の入力を促すプロンプトメッセージ、ヘルプメッセージおよびエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

パス名は、最初のオプショングループで指定した基準に従わなければなりません。基準が定義されていない場合、パス名はまだ存在していない通常のファイルに対するものでなければなりません。-a (絶対) と -l (相対) のいずれも指定されていない場合は、どちらかが有効であるとみなされます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckpath コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errpath (標準出力上にエラーメッセージを書式化して表示する)、helppath (標準出力上にヘルプメッセージを書式化して表示する)、および valpath (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-a — パス名は絶対パスでなければなりません。

-b — パス名はブロック型特殊ファイルでなければなりません。

-c — パス名は文字型特殊ファイルでなければなりません。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

`-e` *error*	*error* をエラーメッセージとして定義します。
`-f`	パス名は通常ファイルでなければなりません。
`-h` *help*	*help* をヘルプメッセージとして定義します。
`-k` *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
`-l`	パス名は相対パスでなければなりません。
`-n`	パス名が存在していてはなりません (新規のものでなければなりません)。
`-o`	パス名が存在していなければなりません (既存のものでなければなりません)。
`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (`quit`) を使用できないようにします。
`-r`	パス名は読み取り可能でなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-t`	パス名は作成可能 (処理可能) でなければなりません。パス名が、まだ存在していない場合には作成されます。
`-w`	パス名は書き込み可能でなければなりません。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。
`-x`	パス名は実行可能でなければなりません。
`-y`	パス名はディレクトリでなければなりません。
`-z`	パス名には、`0` バイトを超えるサイズのファイルがなければなりません。

オペランド

次のオペランドを指定できます。

input	検証オプションと照合される入力

使用例

`ckpath` のデフォルトメッセージのテキストは、使用されている基準オプションによって異なります。

例 1 デフォルトのプロンプト

`ckpath` (`-a` オプションを使用) のデフォルトプロンプトの例は、次のとおりです。

```
example% ckpath -a
Enter an absolute pathname [?,q]
```

例 1 デフォルトのプロンプト　(続き)

例 2 デフォルトのエラーメッセージ

デフォルトのエラーメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/errpath -a
ERROR: A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 3 デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/helppath -a
A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 4 終了オプション

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード 3 と共に q が返されます。終了を入力すると、後に復帰改行が続きます。

例 5 valpath モジュールの使用

valpath モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valpath
usage: valpath [-[a|l][b|c|f|y][n|[o|z]]rtwx] input
    .
    .
    .
```

終了ステータス

次の終了ステータスが返されます。

0	正常終了
1	入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった
2	同時に指定できないオプション
3	ユーザー終了 (quit)
4	同時に指定できないオプション

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 face(1), signal(3HEAD), attributes(5)

名前 | ckstr, errstr, helpstr, valstr – 文字列の入力要求とその検証

形式 | **ckstr** [-Q] [-W *width*] [[-r *regexp*] [...]] [-l *length*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [- s *signal*]]

/usr/sadm/bin/errstr [-W *width*] [-e *error*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/helpstr [-W *width*] [-h *help*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/valstr [-l *length*] [[-r *regexp*] [...]] *input*

機能説明 | ckstr ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、文字列の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

このコマンドから返される応答は、定義済みの正規表現に一致していなければならず、指定された長さ以内でなければなりません。正規表現が指定されていない場合、有効な入力は、内部に空白を含まず、また先行や後続する空白がない、指定された長さ以内の文字列でなければなりません。長さが指定されていない場合、長さは検査されません。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckstr コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errstr (標準出力上にエラーメッセージを書式化して表示する)、helpstr (標準出力上にヘルプメッセージを書式化して表示する)、および valstr (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-l *length* — 入力の最大長を指定します。

`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (quit) を使用できないようにします。
`-r` *regexp*	入力を検証するときの基準となる正規表現、 *regexp* を指定します。空白を含めることができます。複数の式が定義されている場合、応答はその内のいずれかに一致しなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを *width* の行長に書式化します。

オペランド　次のオペランドを指定できます。

input	書式の長さや正規表現の基準に対して検証される入力

使用例

例 **1** デフォルトのプロンプト

`ckstr` のデフォルトのプロンプトは、次のとおりです。

```
example% ckstrEnter an appropriate value [?,q]:
```

例 **2** デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、適用される妥当性検査のタイプによって異なります。ユーザーには、長さまたはパターン照合のいずれかが失敗したことが通知されます。デフォルトのエラーメッセージは、次のとおりです。

```
example% /usr/sadm/bin/errstr
ERROR: Please enter a string which contains no embedded,
leading or trailing spaces or tabs.
```

例 **3** デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージも、適用される妥当性検査のタイプによって異なります。正規表現が定義されている場合、メッセージは次のようになります。

```
example% /usr/sadm/bin/helpstr -r regexp
Please enter a string which matches the following pattern:
regexp
```

他のメッセージは、文字列の長さの要件と定義を指定します。

例 **4** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **5** valstr モジュールの使用

valstr モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valstr
usage: valstr [-l length] [[-r regexp] [ . . . ]] input
```

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは、使用法に誤りがあった

2 無効な正規表現

3 ユーザー終了 (quit)

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

face(1), signal(3HEAD), attributes(5)

名前 | cktime, errtime, helptime, valtime – 時刻の入力要求とその検証

形式 | **cktime** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errtime [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helptime [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valtime [-f *format*] *input*

機能説明 | cktime ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに時刻の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザー応答は、時刻について定義されている書式に一致しなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

cktime コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errtime (エラーメッセージを書式化して表示する) と helptime (ヘルプメッセージを書式化して表示する) と、valtime (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errtime および helptime の各モジュールに format が定義されている場合、メッセージには指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。次の書式と定義を指定できます。

```
%H  =  hour (00 - 23)
%I  =  hour (00 - 12)
%M  =  minute (00 - 59)
%p  =  ante meridian or post meridian
%r  =  time as %I:%M:%S %p
%R  =  time as %H:%M (the default format)
%S  =  seconds (00 - 59)
%T  =  time as %H:%M:%S
```

-h *help* *help* をヘルプメッセージとして定義します。

-k *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* *prompt* をプロンプトメッセージとして定義します。

-Q 有効な応答として終了 (quit)を使用できないようにします。

-s *signal* 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

-W *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった

3 ユーザー終了 (quit)

4 無効な書式引数

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

attributes(5)

注意事項

cktime のデフォルトのプロンプトは、次のとおりです。

```
Enter a time of day [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter the time of day.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the time of day.  Format is <format>.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード 3 と共に q が返されます。valtime モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前 | ckuid, erruid, helpuid, valuid – ユーザー ID の入力要求とその検証

形式 | **ckuid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erruid [-W *width*] [-e *error*]

/usr/sadm/bin/helpuid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valuid *input*

機能説明 | ckuid ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに既存のユーザー ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckuid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erruid (エラーメッセージを書式化して表示する) と helpuid (ヘルプメッセージを書式化して表示する) と、valuid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-m — ヘルプが要求された場合、またはユーザーがエラーを犯した場合は、すべてのログインのリストを表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/passwd` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、または使用法に誤りがあった

`2`　　使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、 `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckuid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the login name of an existing user [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR:  Please enter one of the following login names:  <List>
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのヘルプメッセージは、次のとおりです。

```
Please enter one of the following login names:  <List>
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。 `valuid` モジュールは出力を何も生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckyorn, erryorn, helpyorn, valyorn – yes/no の入力要求とその検証

形式 | **ckyorn** [-Q] [-W *width*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erryorn [-W *width*] [-e *error*]

/usr/sadm/bin/helpyorn [-W *width*] [-h *help*]

/usr/sadm/bin/valyorn *input*

機能説明 | ckyorn は、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、「はい (yes)」または「いいえ (no)」の応答を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckyorn コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erryorn (エラーメッセージを書式化して表示する) と helpyorn (ヘルプメッセージを書式化して表示する) と、valyorn (応答を検証する) です。これらのモジュールは、FACE オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* デフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

-W *width* — プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input y、yes、または n、no (大文字小文字の任意の組み合わせ) に対して検証される入力

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった

2 使用法に誤りがあった

3 ユーザー終了 (quit)

属性

次の属性については、 attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

attributes(5)

注意事項

ckyorn のデフォルトのプロンプトは、次のとおりです。

```
Yes or No [y,n,?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter yes or no.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
To respond in the affirmative, enter y, yes, Y, or YES.
To respond in the negative, enter n, no, N, or NO.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード 3 と共に q が返されます。valyorn モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前 | exec, eval, source – 他のコマンドを実行するためのシェル組み込み関数

sh | **exec** [*argument*...]

eval [*argument*...]

csh | **exec** *command*

eval *argument*...

source [-h] *name*

ksh | ***exec** [*arg*...]

***eval** [*arg*...]

sh | exec コマンドは このシェルの代わりに、 *argument* で指定されたコマンドを、 新規プロセスは生成せずに 実行します。 入出力引数が指定可能で、 それら以外の引数を指定しない場合には、 シェルの入出力を変更します。

eval の組み込みの *argument* をシェルへの入力として読み取り、 生成されるコマンドを実行します。

csh | exec は現在のシェルの代わりに *command* を実行します。シェルは終了します。

eval は引数をシェルへの入力として読み取り、 生成されるコマンドを実行します。通常この指定は、コマンドまたは 変数置換の結果として生成されたコマンドを 実行するために使用します。

source は *name* からコマンドを読み取ります。 source コマンドは入れ子にできますが、 あまり深く入れ子にするとシェルのファイル記述子が 不足する可能性があります。 ソースファイル中のエラーは、それがいかなるレベルであろうと、 入れ子にされたすべての source コマンドを終了させます。

-h　*name* が示す、履歴のリスト上のファイル からコマンドを持ってきますが、実行はしません。

ksh | exec 組み込み関数を使用して *arg* を指定すると、 このシェルの代わりに 引数で指定されたコマンドを、 新規プロセスは生成せずに 実行します。 入出力引数が指定可能で、 現在のプロセスに影響を及ぼす場合があります。 引数を指定しない場合は、ファイル記述子が 入出力リダイレクトリストの指定どおりに 変更されることになります。 この場合、 この機能によりオープンされた 2 より大きい番号のファイル記述子は、 別のプログラムを起動するとクローズされます。

eval に続く引数をシェルへの入力として読み取り、 生成されるコマンドを実行します。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている ksh(1) コマンドは、以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。

3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、チルド置換は = 符号の後に実行され、ワード分割とファイル名生成は実行されません。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), sh(1), attributes(5)

名前 | ex – テキストエディタ

形式 | **/usr/bin/ex** [-| -s] [-l] [-L] [-R] [-r [*file*]] [-t *tag*] [-v] [-V] [-x] [-w *n*] [-C] [+ | *command* | -c *command*] *file...*

/usr/xpg4/bin/ex [-| -s] [-l] [-L] [-R] [-r [*file*]] [-t *tag*] [-v] [-V] [-x] [-w *n*] [-C] [+ | *command* | -c *command*] *file...*

機能説明 | ex ユーティリティはエディタ群 ex と vi のもとになるエディタで、ed のスーパーセットです。ed と比較すると、画面上での編集用に優れた拡張機能が備わっています。画面中心のテキスト編集は vi が目的とする機能です。

CRT 端末を使っているユーザーは、画面上でテキストを編集することが多く、その場合には ex の画面編集機能である vi(1) を使ってください。

ed を使った経験のあるユーザーは、ex では ed のすべてのコマンドがサポートされているばかりでなく、CRT 端末上で便利なさまざまな機能が備わっていることに気づくはずです。インテリジェント端末や高速端末では vi でも十分に便利な機能を果たせます。一般的に言えば、ex エディタは ed と比べて端末の機能を引き出すことができます。端末機能データベース (terminfo(4) を参照) を参照し、さらに環境変数 TERM が示す端末タイプを参照して、端末を効率よく使う方法を決定します。ex エディタでは、その画面上テキスト編集機能である visual コマンド (vi と略すことが可能) が持つ文字や行の挿入/削除といった機能を利用することができます。この処理モードは、vi コマンドを使用する際の中心的なモードです。

ex ユーティリティには、ファイル中のテキストを簡単に画面表示するための便利な機能が用意されています。たとえば z コマンドにより、テキストのウィンドウへ容易にアクセスできます。また ^D (CTRL-D) はウィンドウ半分だけテキストをスクロールするもので、改行キーを押す方法よりも簡単にファイルの内容を見ることができます。もちろん画面主導の visual モードも使用でき、これにより編集中のテキストにアクセスできます。

入力操作を誤ってしまった場合、ex ユーティリティには便利な補助機能が用意されています。undo (u) コマンドは、誤って実行してしまった変更処理 (1 つ) を取り消して元の状態に戻します。ex はユーザーに対して様々なフィードバックを行います。変更された行の内容を出力したり、またあるコマンドが多くの行の内容を変更した場合にその旨をユーザーに知らせます。これにより、予期していなかった行にまでコマンドが影響を与えてしまった場合、ユーザーはただちにそれを発見できます。

また本エディタは通常、現在編集中のもの以外のファイルへの上書きを抑止します。したがって、誤って別のファイルに書き込んでしまうというミスを防ぐことができます。システムやエディタがクラッシュしたり、ユーザーが誤って電話を切ってしまった場合でも、エディタの recover コマンド (または -r *file* オプション) を使えば作業を再開できます。このため、作業が中断した地点よりも数行ほど戻るだけで済みます。

さらに、複数のファイルを同時に扱うための機能も ex ユーティリティには備わっています。コマンド行上でファイルの 並びを指定し、next (n) コマンドを実行すれば順番にファイルが処理されます。next コマンドに対して、処理の対象とする一群のファイルを指定するために、ファイル名のリストやシェルが扱うようなパターンを与

えることができます。エディタにおけるファイル名は、完全シェル・メタシンタクスで表されるのが 通常です。メタキャラクタ % を使ってファイル名を定義することもでき、現ファイルの名前に置き換えられます。

エディタには、ASCII の小文字 (a–z) からなる名前を持つバッファが備わっています。ユーザーはテキストをバッファに書き込み、後でファイル中の他の位置へそのテキストを挿入することができます。edit (e) コマンドを使って別の新たなファイルの編集を開始しても、バッファの内容は以前のまま残されます。

ex には、最後に実行した置換 (substitute) コマンドを繰り返すための & コマンドが用意されています。また確認付き置換コマンドもあります。特定の範囲の中で置換を行うよう指示すると、エディタは 個々の該当箇所について実際に置換してよいか否かを問い合わせる、いわゆる対話型置換を実行します。

検索/置換処理において、大文字と小文字は同一とみなすよう指示することも できます。また語のマッチング用に正規表現を指定することも可能です。この便利な機能を使うと、たとえば検索対象として "edit" と 指定したとき、"editor" という語も検索できます。

ex が提供するオプションは、ユーザーが自分の希望に合わせて設定できます。便利なオプションの例として autoindent があります。これは自動インデントを行うもので、各行の先頭に空白を置いて 自動的にテキストをインデントします。表示されたテキストに対して、^D により後方へのタブを行なったり、空白文字やタブキーを入力したりして行の位置を変更できます。

その他の便利な機能としては、連結した行の間に自動的に空白を挿入する join (j) コマンド、複数の行を一度にシフトする < および > コマンド、さらに sort などのコマンドを通じてバッファの一部を フィルタする機能などがあります。

オプション

以下のオプションを指定できます。

- | -s	ユーザーへのすべての対話型フィードバックを抑止します。エディタスクリプト実行中に便利なオプションです。
-l	LISP を編集するための設定を行います。
-L	エディタもしくはシステムのクラッシュ発生によって保存された全ファイルの名前を表示します。
-R	読み取り専用モード。readonly フラグがセットされ、ファイルの上書きは不可能となります。
-r *file*	エディタもしくはシステムのクラッシュが発生した後で、*file* が示すファイルを編集します。つまりクラッシュ発生時にバッファ中にあったバージョンを復元します。
-t *tag*	*tag* が示すタグを含むファイルを編集します。そのタグが定義されている地点が編集開始地点となります。
-v	vi を使った編集の画面表示を開始します。単に vi コマンドを入力しても同様に実行できます。

`-V` 冗長。`ex` コマンドを標準入力で読み込んだ場合、入力は標準エラーに表示されます。シェルスクリプトの `ex` コマンド実行時に役に立ちます。

`-x` 暗号化オプション。`X` コマンドと同様にキーの入力をユーザーに要求します。このキーにより、暗号化と復号化が `crypt` コマンドのアルゴリズムを使って実行されます。`X` コマンドは、その高度な推定能力を使って、読み込まれたテキストが暗号化 されているか否かを判定します。一時バッファのファイルも、この `-x` オプション用にユーザーが入力したキーから生成した別バージョンのキーを使って暗号化されます。

`-w`*n* ウィンドウサイズのデフォルト値を *n* に設定します。低い回線速度でエディタを使用する場合に便利なオプションです。

`-C` 暗号化オプション。上記 `-x` オプションと同様ですが、唯一の違いは `X` コマンドの代わりに `C` コマンドを実行する点です。この両コマンドの違いは、`C` コマンドでは読み込まれたテキストは暗号化されているものと無条件にみなされるという点です。

`+`*command* | `-c` *command* `command` で示したエディタコマンドを冒頭に実行して編集処理を開始します。通常は、検索または位置指定用のコマンドが用いられます。

/usr/xpg4/bin/ex

`-t` *tag* と `-c` *command* の 2 つのオプションがともに指定された場合は、`-t` *tag* の方が先に処理されます。つまりタグを含んだファイルが `-t` により選択され、その後でコマンドが実行されます。

オペランド

以下のオペランドを指定できます。

file 編集するファイルのパス名。

ex の状態

コマンド これが通常もしくは初期の状態です。“`:`” がプロンプトとして表示され、入力が可能になっています。行削除文字の入力によりコマンドの一部を取り消すことができます。

挿入 `a`、`i`、または `c` の入力によりこの状態になり、任意のテキストを入力できます。この状態を終了させるには、ピリオド (.) だけからなる行を入力するか、あるいは割り込みを発生させます。後者の場合には異常終了となります。

ビジュアル `vi` と入力するとこの状態になり、`Q` または ^\ (CTRL-\) と入力すると終了します。

ex のコマンド名と簡略形

abbrev	ab	map		set	se
append	a	mark	ma	shell	sh
args	ar	move	m	source	so
change	c	next	n	substitute	s

copy	co	number	nu	unabbrev	unab
delete	d	preserve	pre	undo	u
edit	e	print	p	unmap	unm
file	f	put	pu	version	ve
global	g	quit	q	visual	vi
insert	i	read	r	write	w
join	j	recover	rec	xit	x
list	l	rewind	rew	yank	ya

ex コマンド引数

以下に示すすべての ex コマンドにおいて、数値を表す count と範囲を表す range の両方が指定され、ともに有効な場合、対象となる行の数には、範囲ではなく数値の方が用いられます。コマンドの開始行は、範囲が表す先頭の行となります。

省略	ab[brev] word rhs
追加	[line] a[ppend][!]
引数	ar[gs]
変更	[range] c[hange][!] [count]
ディレクトリ変更	chd[ir][!] [directory]; cd[!] [directory]
コピー	[range] co[py] line [flags]; [range] t line [flags]
削除	[range] d[elete] [buffer] [count] [flags]
編集	e[dit][!] [+line][file]; ex[!] [+line] [file]
ファイル	f[ile] [file]
グローバル	[range] g[lobal] /pattern/ [commands]; [range] v /pattern/ [commands]
挿入	[line] i[nsert][!]
連結	[range] j[oin][!] [count] [flags]
制御文字の表示	[range] l[ist] [count] [flags]
マップ	map[!] [x rhs]
マーク	[line] ma[rk] x; [line] k x
移動	[range] m[ove] line
次のファイル	n[ext][!] [file ...]
行番号	[range] nu[mber] [count] [flags]; [range] # [count] [flags]
オープン	[line] o[pen] /pattern/ [flags]

保存	pre[serve]
プリント	[range] p[rint] [count] [flags]
バッファからの取り出し	[line] pu[t] [buffer]
終了	q[uit][!]
読み込み	[line] r[ead][!] [file]
復元	rec[over] file
リワインド	rew[ind][!] Set se[t] [option[=[value]]...] [nooption...] [option?...] [all]
シェル	sh[ell]
ソース	so[urce] file
置換	[range] s[ubstitute] [/pattern/repl/[options] [count] [flags]]
中断	su[spend][!]; st[op][!]
タグ	ta[g][!] tagstring
省略の解除	una[bbrev] word
取り消し	u[ndo]
マップの解除	unm[ap][!] x
ビジュアルモードに変更	[line] vi[sual] [type] [count] [flags]
書き込み	[range} w[rite][!] [>>] [file]; [range} w[rite] [!] [file]; [range} wq[!] [>>] [file]
書き込んで終了	[range] x[it][!] [file]
バッファへの取り込み	[range] ya[nk] [buffer] [count]
ウィンドウ調整	[line] z [type] [count] [flags]
シェルエスケープ	! command [range]! command
左にシフト	[range] < [count] [flags]
右にシフト	[range] > [count] [flags]
再置換	[range] & [options] [count] [flags]; [range] s[ubstitute] [options] [count] [flags]; [range] ~ [options] [count] [flags]
スクロール	EOF
行番号表示	[line] = [flags]

実行　@ buffer; * buffer

ex コマンド

強制暗号化	`C`	必要に応じた暗号化	`X`
再置換	`&`	次行表示	CR
右にシフト	`>`	左にシフト	`<`
スクロール	`^D`	ウィンドウ	`z`
シェルにエスケープ	`!`		

ex 行指定

n	*n* 番目の行	*/pat*	*pat* に一致する次の行
`.`	現在行	`?`*pat*	*pat* に一致する前の行
`$`	最終行	*x-n*	*x* 番目の *n* 行前
`+`	次の行	*x,y*	*x* 番目の行から *y* 番目の行まで
`-`	前の行	*'x*	文字 *x* でマークされた行
+n	*n* 行後	*''*	直前にいた行
`%`	1,$		

初期化オプション

`EXINIT`	`set` による設定を格納する環境変数
`$HOME/.exrc`	エディタ初期化ファイル
`./.exrc`	エディタ初期化ファイル
`set` *x*	*x* オプションを有効とする
`set no`*x*	*x* オプションを無効とする
`set` *x*`=val`	*x* オプションの値を `val` とする
`set`	変更されたオプションを表示
`set all`	全オプションを表示
`set` *x*`?`	*x* オプションの値を表示

特に便利なオプションとその簡略形

autoindent	`ai`	インデントを自動挿入
autowrite	`aw`	変更を加える前にファイルの内容を書き出し
directory		一時作業ファイル用ディレクトリのパス名

exrc	ex	vi/ex が現在のディレクトリ中の .exrc を読み込めるようにする。本オプションはEXINITシェル変数もしくは$HOMEディレクトリ中の.exrcファイルに設定される。
ignorecase	ic	検索時に大文字/小文字の違いを無視
list		タブに対しては ^I を、行末には$ を出力する
magic		パターン中の . [* を特別な意味を持つと解釈
modelines		先頭と最後の5 行ずつは、その形式が ex:command: または vi:command: であれば、vi/ex コマンドであると解釈して実行
number	nu	行番号を付加
paragraphs	para	パラグラフを開始するマクロ名
redraw		高度機能の端末のシミュレート
report		最後に実行したコマンドが report 変数の値を超える数の行を更新した場合にその旨を通知
scroll		コマンドモード行
sections	sect	セクションを開始するマクロ名
shiftwidth	sw	<、>、または ^D に対するシフト数
showmatch	sm	入力された ） と }に対応する （ と{ を示す。
showmode	smd	vi における挿入モードを表示
slowopen	slow	挿入時の更新の中止
term		vi に対して使用中の端末のタイプを指定 デフォルトは環境変数 TERM が示す値
window		ビジュアルモード行
wrapmargin	wm	行の自動分割
wrapscan	ws	検索の際バッファの終端 (または先頭) で止まらない。

パターン情報の検索

^	行の先頭
$	行の終端
.	任意の文字
\<	語の先頭

\>	語の終端
[str]	*str* に含まれている任意の文字
[^str]	*str* に含まれていない任意の文字
[*x*–*y*]	*x* と *y* の間の任意の文字
*	任意の数の直前の文字

環境

ex の実行に影響を与える環境変数 HOME 、 PATH 、SHELL 、 TERM 、 LC_COLLATE 、 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

COLUMNS	システムが選択した値の代わりに用いる、画面の水平方向のサイズ値を 指定します。
EXINIT	エディタの開始時に、最初のファイルの読み込み前に実行する ex コマンドを定義します。複数のコマンドを記述する場合には、縦線 (|) で区切ってください。
LINES	システムが選択した値の代わりに用いる、画面の垂直方向のサイズ値を指定します。画面全体の行数、およびビジュアルモードでの 垂直画面サイズとして用いられます。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した

ファイル

/var/tmp/Ex*nnnnn*	エディタの一時ファイル
/var/tmp/Rx*nnnnn*	名前付きバッファ用一時ファイル
/usr/lib/expreserve	保存 (preserve) コマンド
/usr/lib/exrecover	復旧 (recover) コマンド
/usr/lib/exstrings	エラーメッセージ
/usr/share/lib/terminfo/*	端末の機能ファイル
/var/preserve/login	保持ディレクトリ (login は当該ユーザーのログイン ID)
$HOME/.exrc	エディタ初期化用ファイル
./.exrc	エディタ初期化用ファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/ex

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/ex

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

ed(1), edit(1), grep(1), sed(1), sort(1), vi(1), curses(3CURSES), term(4), terminfo(4), attributes(5), environ(5), standards(5)

『*OpenWindows* ユーザーズガイド *(*上級編*)*』

作者

vi と ex ユーティリティは、米国カリフォルニア州バークレー市にあるカリフォルニア大学のコンピュータサイエンス 学部、電子技術・コンピュータサイエンス学科が開発したソフトウェアにもとづいています。

注意事項

オプションの中には、サポートされ続けてはいるものの、マニュアル上ではコマンド構文標準 (intro(1) を参照) に準拠している他のオプションに置き換わっているものがあります。たとえば – オプションは -s に変わっています。また引数指定可能な -r オプションは、-L に変わっています。さらに +command は -c command に変わっています。

ファイルが読み込まれたときに、メッセージ file too large to recover with -r option が表示されることがあります。これは、このファイルの編集および保存はできるが、万一編集内容が失われた場合に、-r オプションで回復できないことを意味します。

z コマンドが出力する行数は、物理的ではなく論理的な値です。したがって長い行が存在すると、出力の量は 1 画面分を超えてしまうことがあります。

コマンド行で -s オプションが指定されていると、ファイル入出力エラーが発生しても名前は表示されません。

編集環境は構成用オプションにデフォルトで設定されています。編集作業を開始するとき、ex は EXINIT 環境変数を読み込もうとします。変数が定義されていればエディタは EXINIT の値を使い、定義されていなければ $HOME/.exrc 中に設定された値を使います。$HOME/.exrc がなければ、デフォルト値を使います。

$HOME 以外の現ディレクトリにある .exrc のコピーを使う場合は、EXINIT または $HOME/.exrc の中の *exrc* オプションを設定してください。*exrc* を EXINIT または $HOME/.exrc 中で設定すれば、EXINIT で設定されているオプションをローカルな .exrc で無効にすることができます。

大文字/小文字の区別をしないで単一の検索を行う簡単な方法はありません。

名前付きバッファ中にテキストがあり、エディタ終了以前にそのテキストが使用されなくとも、エディタは警告を発しません。

入力ファイル中の NULL 文字は捨てられます。結果として生成されるファイル中には現れません。

標準の Solaris バージョンの `ex` は、いずれは POSIX.2 に準拠したバージョンに置き換えられます (`standards`(5) を参照)。アドレス指定や機能で `ex` ファミリを使用するスクリプトは、これらのユーティリティの `/usr/xpg4/bin` バージョンを使ってください。

名前 | exec, eval, source – 他のコマンドを実行するためのシェル組み込み関数

sh | **exec** [*argument*...]

eval [*argument*...]

csh | **exec** *command*

eval *argument*...

source [-h] *name*

ksh | ***exec** [*arg*...]

***eval** [*arg*...]

sh | exec コマンドは このシェルの代わりに、 *argument* で指定されたコマンドを、 新規プロセスは生成せずに 実行します。 入出力引数が指定可能で、 それら以外の引数を指定しない場合には、 シェルの入出力を変更します。

eval の組み込みの *argument* をシェルへの入力として読み取り、 生成されるコマンドを実行します。

csh | exec は現在のシェルの代わりに *command* を実行します。シェルは終了します。

eval は引数をシェルへの入力として読み取り、 生成されるコマンドを実行します。通常この指定は、コマンドまたは 変数置換の結果として生成されたコマンドを 実行するために使用します。

source は *name* からコマンドを読み取ります。 source コマンドは入れ子にできますが、 あまり深く入れ子にするとシェルのファイル記述子が 不足する可能性があります。 ソースファイル中のエラーは、それがいかなるレベルであろうと、 入れ子にされたすべての source コマンドを終了させます。

-h　*name* が示す、履歴のリスト上のファイル からコマンドを持ってきますが、実行はしません。

ksh | exec 組み込み関数を使用して *arg* を指定すると、 このシェルの代わりに 引数で指定されたコマンドを、 新規プロセスは生成せずに 実行します。 入出力引数が指定可能で、 現在のプロセスに影響を及ぼす場合があります。 引数を指定しない場合は、ファイル記述子が 入出力リダイレクトリストの指定どおりに 変更されることになります。 この場合、 この機能によりオープンされた 2 より大きい番号のファイル記述子は、 別のプログラムを起動するとクローズされます。

eval に続く引数をシェルへの入力として読み取り、 生成されるコマンドを実行します。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている ksh(1) コマンドは、以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。

3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、チルド置換は = 符号の後に実行され、ワード分割とファイル名生成は実行されません。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), sh(1), attributes(5)

名前 | exit, return, goto – シェルの連続した処理を分岐して実行するためのシェル組み込み関数

sh | **exit** [*n*]

return [*n*]

csh | **exit** [(*expr*)]

goto *label*

ksh | ***exit** [*n*]

***return** [*n*]

sh | `exit` はシェルまたはシェルスクリプトを *n* で指定した終了状態で終了させます。 *n* を省略すると、 最後に実行されたコマンドの終了状態 がシェルの終了状態になります。 ファイルの終わりを検出した場合もシェルが終了します。

`return` は関数を、 *n* が示す戻り値で終了させます。 *n* を省略すると、戻り値は最後に実行された コマンドの終了状態になります。

csh | `exit` はシェルまたはシェルスクリプトを終了させ、 状態変数の値または式 *expr* で指定された値が返されます。

`goto` 組み込み関数は *label* をコマンド中で検索の引数として指定します。 シェルは可能なかぎり入力をさかのぼり、 *label*: という形式の行を探します。 *label*: の前には空白文字またはタブ文字がある可能性もあります。 指定された行の次から実行が再開します。 `while` または `for` 組み込みコマンドと、対応する `end` との間にあるラベルへジャンプするとエラーになります。

ksh | `exit` はシェルまたはシェルスクリプトを *n* で指定した終了状態で終了させます。 具体的には、指定した値の最下位 8 ビットが終了状態の値となります。 *n* を省略すると、 最後に実行されたコマンドの終了状態 がシェルの終了状態になります。 トラップ実行中に `exit` が発生した場合、ここで言う最後に実行されたコマンドとは、トラップ呼び出し直前に実行されたコマンドを指します。 なお、 `ignoreeof` オプション (後述の `set` を参照) が有効になっているシェルを除き、 ファイルの終わりを検出した場合もシェルが終了します。

`return` はシェル関数またはドット (.) スクリプトを、 *n* で指定された戻り値で 呼び出し側スクリプトに戻します。 *n* で指定した値の最下位 8 ビットが戻り値となります。 *n* を省略すると、 戻り値は最後に実行された コマンドの戻り値になります。 関数やドット (.) スクリプト実行中以外で `return` を起動すると、 結果は `exit` と同一になります。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている `ksh`(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。

4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

break(1), csh(1), ksh(1), sh(1), attributes(5)

名前 | expand, unexpand – タブ文字を空白文字に展開する、またはその反対

形式 | **expand** [-t *tablist*] [*file*...]

expand [–*tabstop*] [–*tab1, tab2,. . ., tabn*] [*file*...]

unexpand [-a] [-t *tablist*] [*file*...]

機能説明 | expand は 1 つ以上の *file* (または標準入力) のタブ文字を空白文字に展開して標準出力へコピーします。 バックスペース文字は出力中に保存され、タブのカラム幅の計算の際にカラム幅を 1 減算します。 expand はタブ文字を含む文字ファイルの前処理 (ソートをする前や、 特定のカラムを探す前など) を行う際に役に立ちます。

unexpand は 1 つ以上の *file* (または標準入力)を、 タブ文字を復活させて標準出力へコピーします。 オプションの指定がないときは、 行頭の空白文字とタブ文字だけがタブ列に置き換えられます。 -a オプションを指定するとこの指定は無効となります(「オプション」の項参照)。

オプション | expand のオプション

-t*tablist* タブの位置を指定します。引数 *tablist* は、 1 つまたは複数の 10 進整数からなります。複数個指定する場合には、昇順に並べて 空白文字またはコンマで区切らなければなりません。 1 つの整数だけを指定すると、そのカラム数ごとにタブが設定されます。 デフォルトでは 8 カラムおきです。 複数の整数を指定すると、それらのカラム位置にタブが設定されます。 指定する各カラム位置 (*N*) は、ゼロより大きい整数でなければなりません。 またカラム位置は昇順に指定する必要があります。 行を出力する際、カラム位置 *N* にタブを進めるということは、次の文字が *N*+1 カラムに出力されることになります。 複数のタブ位置が指定され、その最後のタブ位置を超えた地点で タブ文字の出力を処理する必要が生じた場合、 expand はそのタブを 1 つの空白文字に置き換えて出力します。

–*tabstop* 1 つの数を指定し、その個数分の空白文字ごとに、 タブを設定します。 省略時の値は 8 です。

–*tab1, tab2,. . ., tabn* 引数で指定された位置にタブ文字を設定します。

unexpand のオプション

-a 置き換えていく際に 2 つ以上空白文字が連続していたらタブ文字を挿入します。 より小さな出力ファイルを生成します。

-t*tablist* タブの位置を指定します。引数 *tablist* は、 1 つまたは複数の 10 進整数からなります。複数個指定する場合には、昇順に並べて 空白文字またはコンマで区切らなければなりません。 1 つの整数だけを指定すると、そのカラム数ごとにタブが設定されます。 デフォルトでは 8 カラムおきです。 複数の整数を指定すると、それらのカラム位置にタブが設定されます。 指定する各カラム位置 (*N*) は、ゼロより大きい整数でなければなりません。 またカラム位置は昇順に指定する必要があります。 行を出力する際、カラム位置 *N* にタブを進めるということは、次の文字が *N*+1 カラムに出力され

ることになります。 -t オプションを省略すると、デフォルトとして -t 8 を指定したことと同等となります。ただし、後述するように -a との関連については異なります。 複数のタブ位置が指定された場合、その最後のタブ位置を超えた地点では 空白文字からタブ文字への文字変換は発生しません。 -t オプションを指定すると、 -a オプションは意味を持たなくなり、タブ変換は 先行する空白文字だけに制限されることはありません。

オペランド

以下のオペランドを指定できます。

file 入力に用いるテキストファイルのパス名。

環境

expand と unexpand の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 正常終了

>0 エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

tabs(1), attributes(5), environ(5)

名前 | set, unset, setenv, unsetenv, export – 現在のシェルおよびそこから起動されたプロセスでの環境変数の特性を決定するシェル組み込み関数

sh | **set** [--aefhkntuvx [*argument*]]...

unset [*name*...]

export [*name*...]

csh | **set** [*var* [= *value*]]

set *var* [*n*] = *word*

unset *pattern*

setenv [*VAR* [*word*]]

unsetenv *variable*

ksh | **set** [±aefhkmnopstuvx] [±o *option*]... [±A *name*] [*arg*...]

unset [-f] *name*...

****export** [*name* [=*value*]]...

sh | set 組み込みコマンドには次のようなオプションがあります。

`--` どのフラグも変更しません。 $1 に - を設定する際に便利です。

`-a` 修正または作成された変数にエクスポートのマークを付けます。

`-e` コマンドが 0 以外の終了状態で終了した場合、直ちに終了します。

`-f` ファイル名を生成しないようにします。

`-h` 関数の定義時に、関数コマンドを検索しその位置を記憶します (通常、関数コマンドは関数実行時に検索されます)。

`-k` コマンド名の前にあるキーワード引数だけでなく、 すべてのキーワード引数がコマンド用の環境に置かれます。

`-n` コマンドを読み取るが、実行しません。

`-t` 1 つのコマンドを読み取り、実行したあと、終了します。

`-u` 未設定の変数を置換時にエラーとして扱います。

`-v` シェル入力行の読み取り時に、その内容を表示します。

`-x` コマンドの実行時に、コマンドと引数の内容を表示します。

- の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェルの起動時にも使用できます。 現在セットされているフラグは、 $- で見つけられます。 *argument* は定位置パラメタで、 $1、$2、... へ順に割り当てられます。 *argument* が指定されない場合、 すべての名前の値が出力されます。 unset は *name* ごとに、対応する変数または関数値を削除します。 変数 PATH 、PS1、 PS2、 MAILCHECK および IF は設定解除できません。

export 組み込みコマンドでは、 指定された *name* に対し、ひきつづき実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。 引数を省略すると、 現在のシェル実行中にエクスポートのマークが付けられた 変数名を一覧表示します。 関数名はエクスポートされません。

csh 引数を指定しないと set はすべてのシェル変数の値を表示します。 複数ワードからなる値は括弧でくくられて表示されます。 引数 *var* だけを指定すると、 set は空 (NULL) の値を *var* が示す変数に割り当てます。 引数を *var* = *value* の形式で指定すると、 set は、変数 *var* に値 *value* を割り当てます。 *value* は次のいずれかです。

word 単一ワード (もしくは引用符付きの文字列)

(*wordlist*) 空白で区切られた、括弧付きワードの並び

値は、割り当てられる前に、コマンドおよびファイル名展開されます。 set *var*[*n*]=*word* 形式は、複数ワードからなる値の *n* 番目のワードを *word* に置き換えます。

unset は *pattern* が示すファイル名置換パターンに一致する名の変数を削除します。 'unset *' と指定すると、すべての変数が削除されます。 ただしこれは、 csh の動作に悪影響をおよぼします。

引数を指定しないと setenv はすべての環境変数を表示します。引数 *VAR* を指定すると、 setenv は環境変数 *VAR* に空の値 (NULL) を設定します (慣習上、環境変数名は大文字で指定されるのが通常)。 *VAR* と *word* の両引数を指定すると、 setenv は、*VAR* に単一ワードまたは引用符付き文字列である値 *word* を設定します。 環境変数 PATH は、コロンで区切られた複数の *word* 引数を指定できます (後述の「使用例」を参照) 。 最もよく使用される環境変数 USER 、 TERM 、 PATH は、自動的に csh 変数 user、term、path から (へ) インポート (エクスポート) されます。 これらの変数を変更する場合には setenv を使用してください。 さらにシェルは、 csh 変数 cwd が変更されるたびに、その値を 環境変数 PWD へ設定します。

環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、LC_COLLATE、LC_NUMERIC、LC_MONETARY は、C シェル内で変更されると新しい値が即座に有効になります。 これらの環境変数の詳細については environ(5) を参照してください。

unsetenv は環境から *variable* が示す変数を削除します。 unset のようなパターンマッチングは行いません。

ksh set コマンドのフラグの意味は以下のとおりです。

-A 配列の代入。 *name* で示される変数の設定を解除し、 *arg* リストから順々に値を割り当てます。 +A は、 最初の変数設定を解除しません。

-a 定義される後続の変数すべてを自動的にエクスポートします。

-e コマンドの終了状態が 0 でない場合、 ERR トラップ (設定されていれば) を実行し、 終了します。 このモードは、 プロファイル読み取り時は無効です。

-f ファイル名の生成を無効にします。

`-h` 各コマンドは、 最初に検出された時点で、 検索済み別名になります。

`-k` コマンド名に先行するものだけでなく、 すべての変数代入引数をコマンドの環境に格納します。

`-m` バックグラウンドジョブを個別のプロセスグループで実行し、 完了時にメッセージを 1 行表示します。 バックグラウンドジョブの終了状態は 完了メッセージで報告されます。 ジョブ制御を備えたシステムでは、 このフラグは 対話型シェルに対して自動的に有効になります。

`-n` コマンドを読み取り、 構文エラーがないかチェックします。 実行は行いません。 対話型シェルに対しては無視されます。

`-o` このフラグの後に指定する引数は、 以下のオプション名のいずれかです。

`allexport`	`-a` と同じです。
`errexit`	`-e` と同じです。
`bgnice`	バックグラウンドジョブをすべて低い優先度で実行します。 これはデフォルトモードです。
`emacs`	コマンド入力用に、 `emacs` 形式のインラインエディタを起動します。
`gmacs`	コマンド入力用に、 `gmacs` 形式のインラインエディタを起動します。
`ignoreeof`	ファイルの終わりを検出してもシェルは終了しません。 終了させるには `exit` コマンドを使用する必要があります。
`keyword`	`-k` と同じです。
`markdirs`	ファイル名生成によって生成されるディレクトリ名には、 すべて最後に `/` を付加します。
`monitor`	`-m` と同じです。
`noclobber`	`>` によるリダイレクトが存在するファイルを切り捨てないようにします。 このオプションが有効なとき、 ファイルを切り捨てるには `>|` 指定が必要です。
`noexec`	`-n` と同じです。
`noglob`	`-f` と同じです。
`nolog`	履歴ファイルに関数定義を保存しません。
`nounset`	`-u` と同じです。
`privileged`	`-p` と同じです。
`verbose`	`-v` と同じです。
`trackall`	`-h` と同じです。

vi	vi 形式のインラインエディタの挿入モードになります。 033 というエスケープ文字を押すと、 挿入モードから制御モードに変わります。 Return で行を送信します。
viraw	各文字を vi モードで入力されたときと同様に処理します。
xtrace	-x と同じです。

オプション名を指定しない場合、 -o は現在のオプション設定を表示します。

-p	$HOME/.profile ファイルを処理しないようにし、 ENV ファイルの代わりに /etc/suid_profile ファイルを使用します。 このモードは、 実効ユーザー ID が実ユーザー ID と等しくないとき、また実効グループ ID が実グループ ID と等しくないときには 必ず有効になります。 このモードを無効にすると、 実効ユーザー ID が実ユーザー ID に、 実効グループ ID が実グループ ID にそれぞれ設定されます。
-s	定位置パラメタを辞書編集方式の順にソートします。
-t	コマンド 1 つを読み取って実行し、 終了します。
-u	置換を行う際に、設定されていないパラメタを エラーとして扱います。
-v	シェルへの入力行を読み取り時に表示します。
-x	コマンドとその引数を実行時に表示します。
–	-x フラグと -v フラグを無効にし、 フラグに対する引数の検査を停止します。
– –	どのフラグも変更しません。 このフラグは、 $1を – で始まる値に設定する際に便利です。 このフラグの後に引数がない場合、 定位置パラメタが設定解除されます。

– の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェル起動時にも使用できます。 現在設定されているフラグは、 $– で見ることができます。 -A を指定しないかぎり、 残りの引数は定位置パラメタとなり、 $1 $2 ... に順番に割り当てられます。 引数を 1 つも指定しない場合には、 すべての変数の名前と値を標準出力上に表示します。

name が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、 MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

unset を使用すると *name* が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、

MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

export 組み込みコマンドでは、 指定された *name* に対し、 後で実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

1 つまたは 2 つの (*) アスタリスクが先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

csh 次の例では、 /bin、/usr/bin、/usr/sbin、/usr/ucb/bin ディレクトリにあるファイルをその順番で検索するために、 環境変数 PATH を設定しています。

```
setenv PATH "/bin:/usr/bin:/usr/sbin:usr/ucb/bin"
```

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), read(1), sh(1), typeset(1), attributes(5), environ(5)

名前 | expr – 引数を式として評価する

形式 | **/usr/bin/expr** *argument…*

/usr/xpg4/bin/expr *argument…*

機能説明 | expr ユーティリティは式を評価して、その結果を標準出力に書き出します。文字 0 はゼロ値として書き出され、何もない場合は NULL 文字列として書き出されます。

オペランド | *argument* オペランドは式として評価されます。式の項は空白文字で区切る必要があります。シェルに対する特殊文字は、エスケープする必要があります (sh(1) 参照)。空白文字またはその他の特殊文字を含む文字列は、引用符で囲む必要があります。式の長さは LINE_MAX (2048 文字) に制限されます。

演算子とキーワードを以下に示します。優先度の低いものから順に並べてあり、等しい優先度の演算子は、{ } 記号内にグループ分けされています。演算子はすべてその左に関わります。

expr \| *expr*
1 番目の *expr* が NULL または 0 のどちらでもない場合、1 番目の *expr* の評価を返します。それ以外の場合、2 番目の expr が NULL でなければ、2 番目の *expr* の評価を返します。そうでなければ、0 を返します。

expr \& *expr*
expr が両方とも NULL または 0 以外の場合、最初の expr を返します。それ以外の場合は、0 を返します。

expr { =, \>, \>=, \<, \<=, != } expr
両引数とも整数の場合、整数比較の結果を返します。それ以外の場合、ロケール固有の連立シーケンスを使用して、文字列比較の結果を返します。各比較の結果は、指定された関係が TRUE の場合は 1、FALSE の場合は 0 になります。

expr { +, – } expr
整数値引数の加減。

expr { *, /, % } expr
整数値引数の乗除または剰余。

expr : expr
マッチング演算子 : (コロン) は、最初の引数と 2 番目の引数を比較します。2 番目の引数は国際化された基本正規表現である必要があります。詳しくは regex(5) マニュアルページ、または 「注意事項」を参照してください。通常、/usr/bin/expr のマッチング演算子は一致したバイト数を返し、/usr/xpg4/bin/expr のマッチング演算子は一致した文字数を返します (失敗時は 0)。2 番目の引数に少なくとも 1 つの、国際化された基本正規表現のサブエクスプレッション [\ (. . . \ )] が含まれる場合、マッチング演算子は \1 に対応する文字列を返します。

integer
数字だけからなる引数。先頭に負記号を付けることもできます。

string
integer 引数、または演算子符号の 1 つとして識別することができない文字列の引数です。

互換演算子 **(x86** のみ**)**

次の演算子は INTERACTIVE UNIX システムとの互換性のためだけに用意されるもので、INTERACTIVE UNIX システムではないスクリプトでは使用できません。

`index` *string character–list*
character-list にあるバイトのいずれかが、*string* のバイトに一致する最初の位置を報告します。

`length` *string*
string の長さ (バイト数) を返します。

`substr` *string integer–1 integer–2*
integer–1 の位置で始まり、長さが *integer–2* バイトである *string* の部分文字列を抽出します。*integer–1* が *string* のバイト数よりも大きい値である場合、`expr` は NULL 文字列を返します。*string* にあるバイト数よりも多く抽出しようとした場合、`expr` は *string* から残りのすべてのバイトを返します。*integer-1* または *integer-2* のどちらかが負の値である場合、結果は予測できません。

使用例

例 **1** 整数をシェル変数に追加する

```
example$ a=`expr  $a  +  1`
```

例 **2** パス名の要素を返す

`basename`(1) をエミュレートします。`basename`(1) は、`$a` というパス名の最後の要素を返します。`$a` が `/usr/abc/file` か、ただの `file` のいずれかと等価の場合に、この例は、`file` を返します。引数としての単独の `/` に注意してください。`expr` は `/` を除数演算子とみなします (後述の 「注意事項」 を参照)。

```
example$ expr $a : '.*/\(.*\)' \| $a
```

例 **3** `//` 文字を使用して式を単純にする

例 2. を改善したものです。`//` 文字を追加してあるので、除算演算子についてのあいまいさはなくなり、式全体が単純になっています。

```
example$ expr //$a : '.*/\(.*\)'
```

/usr/bin/expr

例 **4** 変数にバイト数を返す

`$VAR` 中のバイト数を返します。

```
example$ expr "$VAR" : '.*'
```

/usr/xpg4/bin/expr

例 **5** 変数に文字数を返す

`$VAR` 中の文字数を返します。

```
example$ expr "$VAR" : '.*'
```

環境

expr の実行に影響を与える環境変数 LC_COLLATE、LC_CTYPE、LC_MESSAGES、NLSPATH の詳細については、environ(5) を参照してください。

終了ステータス

式を評価した結果として、expr は以下の終了ステータスを返します。

0	式は NULL でも 0 でもなかった
1	式が NULL または 0 だった
2	不正な式だった
>2	エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

basename(1)、ed(1)、sh(1)、Intro(3)、attributes(5)、environ(5)、regex(5)、XPG4(5)

診断

syntax error	演算子とオペランドのエラー
non-numeric argument	数値ではない文字列に対して演算を行う

注意事項

シェルによる引数の処理後、expr は、値による場合を除き、演算子とオペランドの違いを区別できなくなります。$a が = の場合、

```
example$ expr $a = '='
```

というコマンドは、

```
example$ expr = = =
```

と等価になります。これは、引数が expr に渡される (さらに、すべて引数が = 演算子と見なされる) ためです。次のコマンドは正しく動作します。

```
example$ expr X$a = X=
```

正規表現

旧バージョンのあるバージョンとは違って、expr はシステムに提供されたすべてのロケールに対して、国際化された基本正規表現を使用します。国際化された正規表現に関しては regex(5) のマニュアルページを参照してください。

名前 | exstr – ソースファイルからの文字列の抽出

形式 | **exstr** *filename...*

exstr -e *filename...*

exstr -r [-d] *filename...*

機能説明 | exstr ユーティリティは、 C 言語のソースファイルから文字列を抽出し、 それをメッセージ検索関数の呼び出し (gettxt(3C) を参照) に置き換えます。 このユーティリティは、 printf コマンドや printf ルーチンへの引数として指定された文字列だけではなく、 二重引用符で囲まれている文字列もすべて抽出します。第 1 の形式では、exstr はソースファイルからすべての文字列を取り出して、標準出力上に書き出します。 各文字列の先頭には、ソースファイル名とコロン (:) が付加されます。

最初のステップとして、まず exstr -e を使って一群の文字列を入力ファイルから取り出し、 別のファイルに保存します。 次にそれらのうちどの文字列が、 あとでメッセージ検索関数で検索できるように変換可能かを 見つけ出します。 次に、変換可能でない行を削除し、変換可能な行に関しては メッセージファイル名とメッセージ番号を第 4 項目 (*msgfile*) と第 5 項目 (*msgnum*) として追加します。 指定するメッセージファイルは、 mkmsgs(1) で作成したもので、 /usr/lib/locale/locale/LC_MESSAGES ディレクトリ中に存在している必要があります。 このパス中の locale は、テキスト文字列が書かれている言語に対応しています。 詳しくは setlocale(3C) を参照してください。 指定するメッセージ番号は、メッセージファイル中の文字列の シーケンス番号に一致している必要があります。

次に、このようにして編集したファイルを入力に指定して exstr -r を実行します。これにより元の C 言語のソースファイルの新バージョンとして、 文字列がメッセージ検索関数 gettxt() の呼び出しに置き換えられているファイルが生成されます。 *msgfile* と *msgnum* の 2 つのフィールドは、gettxt() へ渡す第 1 引数を構築するために用いられます。gettxt() への第 2 引数は、実行時にメッセージ検索が失敗したときに印刷されます。 この引数は、-d オプションが指定された場合を除き、 NULL 文字列です。

このユーティリティは、 すべての場合に文字列を変換できるわけではありません。 たとえば、静的初期化文字列を関数呼び出しに 置き換えることはできません。 また文字列が、変換できないエスケープシーケンスの形式を とっている場合もあります。 既存のコードを破壊しないようにするため、exstr -e を使って生成したファイルの中身をよく検査して、 関数呼び出しに置換できない文字列を 含んでいる行を必ず削除してください。 また場合によっては、文字列が抽出されて メッセージ検索関数の呼び出しに置き換えられるように、 コードを修正する必要があります。

オプション | 以下のオプションを指定できます。

-e　　指定した C 言語のソースファイルから、文字列リストと 位置情報を取り出します。 リストは以下に示す形式で標準出力上に 書き出されます。

file:line:position:msgfile:msgnum:string

file　　C 言語のソースファイル

line ファイル中の行番号

position その行における文字位置

msgfile NULL 文字列

msgnum NULL 文字列

string 抽出されたテキスト文字列

この出力先としてファイルを指定することも可能です。 ファイルに書き出せば、あとでその内容を編集して *msgfile* と *msgnum* 用に使いたい値を追加することができます。

msgfile *string* を置き換えるテキスト文字列を含んでいるファイル。mkmsgs(1) ユーティリティを使って、この名前を持つファイルを作成して 適切なディレクトリに置いておく必要があります。

msgnum *msgfile* 中のシーケンス番号

次の手順として、 exstr -r を使って *file* の *string* を置き換えます。

-r C 言語のソースファイル中の文字列を、 メッセージ検索関数 gettxt() の呼び出しに置き換えます。

-d このオプションは、 -r オプションとともに使用します。 実行時に gettxt() を呼び出してメッセージの検索が成功しなかった場合、 抽出された文字列が印刷されます。 国際的環境で動作してメッセージを 複数の言語で印刷するような アプリケーションプログラム上では、 exstr が提供する機能を使います。 exstr は、テキスト文字列を、メッセージデータベース中の 文字列を指す関数呼び出しに置き換えます。 どのデータベースを用いるかは、環境変数 LC_MESSAGES の実行時の値により決まります (environ(5) を参照)。

使用例

例 1 exstr の使用例

example.c というファイル中に、次のような 2 つの文字列が含まれていると想定します。

```
main()

{

        printf("This is an example\n");

        printf("Hello world!\n");

}
```

このとき、以下のように example.c を引数として指定して exstr を実行すると、これらの文字列が取り出されて 標準出力上に書き出されます。

```
example% exstr example.c
```

これにより次の出力が得られます。

例 1 exstr の使用例　(続き)

```
example.c:This is an example\n
example.c:Hello world!\n
```

次に -e オプションと出力ファイル名を指定した例を示します。

```
example% exstr -e example.c > example.stringsout
```

この場合には、 example.stringsout というファイルに以下の出力が得られます。

```
example.c:3:8:::This is an example\n
example.c:4:8:::Hello world!\n
```

次にこの example.stringsout ファイルを編集して、 これらの文字列が検索関数呼び出しに置き換えられる前に、*msgfile* と *msgnum* の両フィールド用に使用したい値を追加します。 たとえばメッセージファイルの名前が UX で、ファイル中の文字列のシーケンス番号が 1 と 2 であったとすると、 example.stringsout ファイルを編集して次のような内容に変更します。

```
example.c:3:8:UX:1:This is an example\n
example.c:4:8:UX:2:Hello world!\n
```

このように準備を整えたら、 -r オプション付きで exstr ユーティリティを実行して、 ソースファイル中の文字列をメッセージ検索関数 gettxt() の呼び出しに置き換えることができます。 次のコマンド例を見てください。

```
example% exstr -r example.c <example.stringsout >intlexample.c
```

これを実行すると以下のような出力が得られます。

```
extern char *gettxt();

main()

{

    printf(gettxt("UX:1", ""));

    printf(gettxt("UX:2", ""));

}
```

また -d オプションを指定することもできます。

```
example% exstr -rd example.c <example.stringsout >intlexample.c
```

この場合には、抽出された文字列が gettxt() 関数への第 2 引数として用いられます。

```
extern char *gettxt();

main()

{

        printf(gettxt("UX:1", "This is an example\n"));

        printf(gettxt("UX:2", "Hello world!\n"));

}
```

例 1 exstr の使用例　（続き）

ファイル　/usr/lib/locale/locale/LC_MESSAGES/mkmsgs(1) が生成したファイル

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWtoo

関連項目　gettxt(1), mkmsgs(1), printf(1), srchtxt(1), gettxt(3C), printf(3C), setlocale(3C), attributes(5), environ(5)

診断　exstr が発行するエラーメッセージは、 メッセージテキストを読めば意味が分かるようになっています。 エラーメッセージはコマンド行で見つかったエラー、また 入力ファイル中で見つかった形式エラーを 表します。

face(1)

名前 | face – FACE (Framed Access Command Environment Interface) 用の実行可能ファイル

形式 | **face** [-i *init_file*] [-c *command_file*] [-a *alias_file*] [*filename...*]

機能説明 | FACE (Framed Access Command Environment Interface) は、ユーザーが FACE ユーザーとして正しく設定されていれば、ファイルとファイルフォルダをメニューとフォームの形式で画面に示します。

filename は、メニューの場合は Menu.*xxx*、フォームの場合は Form.*xxx*、テキストファイルの場合は Text.*xxx* の命名規約に従わなければなりません (各 *xxx* は UNIX システムにおけるファイルの命名規約に準拠する文字列)。FMLI (Form and Menu Language Interpreter) 記述子 lifetime は face に対する引数により開かれたすべてのフレームにおいて、無視されます。デフォルトでは、このようなフレームの lifetime は immortal です。コマンド行に *filename* を指定しない場合、FACE メニュー は LOGINWIN 環境変数で指定されたオブジェクトに従って開かれます。これらの環境変数は、ユーザーの .environ ファイルに設定されています。

オプション | 次のオプションを指定できます。

-a *alias_file*	別名ファイル
-c *command_file*	コマンドファイル
-i *init_file*	初期ファイル

オペランド | 次のオペランドを指定できます。

filename	最初に開くオブジェクトを示すファイルの完全パス名

終了ステータス | ユーザーが FACE ユーザーとして正しく設定されていない場合、face コマンドは 0 以外の終了ステータスを返します。

ファイル | $HOME/pref/.environ

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfac

関連項目 | env(1), attributes(5)

名前 | factor – 素因数の分解

形式 | **factor** [*integer*]

機能説明 | factor は、10^{14} 以下の正の整数の、すべての素因数を標準出力に書き込みます。素因数は適切な回数だけ書き込まれます。

factor を引数なしで実行した場合、factor は整数が入力されるのを待ちます。整数が入力されると、factor はその整数を素因数に分解し、適切な回数だけ出力して、再び整数が入力されるのを待ちます。0 または数値以外の文字を入力すると、factor は終了します。

引数 (整数) を指定して factor を実行すると、factor はその整数を出力し、素因数に分解して、すべての素因数を上記のと同様に出力し、終了します。引数が 0 または数値以外の文字の場合、factor は 0 を出力して終了します。

整数を素因数分解する最大時間は sqrt(n) (n は入力する整数) に比例します。*n* が素数または素数の二乗である場合、factor は、素因数の計算にこの最大時間を必要とします。

オペランド | 次のオペランドを指定できます。

integer 10^{14} 以下の正の整数

終了ステータス | 次の終了ステータスが返されます。

0 正常終了

1 エラーが発生した

診断 | 範囲外の数値や意味のない文字の入力に対しては、factor は「Ouch!」というエラーメッセージを出力します。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | attributes(5)

名前 | history, fc – コマンドの履歴リストの処理

形式 | **/usr/bin/fc** [*first* [*last*]]

/usr/bin/fc -l [-nr] [*first* [*last*]]

/usr/bin/fc -s [*old* = *new*] [*first*]

csh | **history** [-hr] [*n*]

ksh | **fc** -e - [*old* = *new*] [*command*]

fc [-e *ename*] [-nlr] [*first* [*last*]]

/usr/bin/fc | fc ユーティリティは、以前に対話型 sh に入力されたコマンドの一覧表示、または 編集と再実行を行います。

コマンドの履歴リストは、番号によってコマンドを参照します。 リスト中の最初の番号は任意に選択できます。 コマンド番号とコマンドとの関係は変わることはありません。 ただしユーザーがログインしたときに、どのプロセスも リストをアクセスしていない場合を除きます。 この場合には、システムはコマンド番号をリセットするので、 保存されている最も古いコマンドに新たな番号 (通常は 1) が与えられます。コマンド番号が、HISTSIZE の値と 128 の大きい方の値に到達したとき、シェルは番号を循環させるので、次のコマンドが再び最小値 (通常は 1) から始まります。このように番号が循環しても、fc は時間の順序は把握しています。 たとえば 4 つのコマンドにそれぞれ 32 766 、32 767、 1 (循環後)、 2 という番号が与えられたとき、 32 767 は 1 よりも大きい値であっても 1 以前に実行されたものであると認識しています。

コマンドが編集されたとき (-l オプション省略時) 、結果として生成された行が履歴リストの最後尾に記録され、sh によって再実行されます。編集作業を起動した fc コマンド自身は履歴リストには記録されません。 エディタがゼロ以外の終了ステータスを返した場合、 履歴リストへの記録や再実行は抑止されます。fc コマンドに指定したコマンド行変数割り当てや リダイレクト演算子は、fc コマンド自身および生成されるコマンドの両方に対して有効です。次の例を見てください。

fc -s -- -1 2>/dev/nullこの例は以前のコマンドを再度呼び出すものですが、 fc および以前のコマンドの両方に対して標準エラー出力を抑止します。

csh | 履歴のリストを表示します。 *n* を指定すると、 *n* 個の最新の履歴を表示します。

-r　出力を、古い履歴からではなく 最近の履歴から順に並べ変えます。

-h　先頭に番号を付加せずに履歴の リストを出力します。 -h オプションを使用することにより、 csh 組み込みコマンドの source(1) の入力に適したファイルを出力することができます。

履歴置換: 履歴置換を使用すれば、 以前入力したコマンド行のワードを、 これから入力するコマンド行で使用できます。 これにより、 綴りの訂正、 複雑なコマンドや引数の繰り返し入力が簡単になります。 コマンド行は履歴リストに保存されます。 履歴リストのサイズは history 変数によって変更できます。 history シェル変数は履歴ファイルに保存されるコマンド行の 最大数に設定されます。 たとえば

```
set history = 200
```

は、履歴リストに最新の 200 コマンド行のトラックを保管します。 設定を行わない場合、 C シェルは最新のコマンドだけを保存します。

履歴置換の指定は ! で始まり (histchars 変数により他の文字に変更可能)、 コマンド行のどこに現われてもかまいません。 ただし履歴置換のネストはできません。 ! を \ でエスケープすれば、 その特別な意味を抑制できます。

履歴置換を含んだ入力行は、 展開された後で、 他の置換が起こる前または コマンドが実行される前に、 端末上に表示されます。

イベント指示子

イベント指示子は、 履歴リスト内の コマンド行エントリを参照するものです。

!	次の文字が空白文字、 タブ、 復帰改行、 =、 または (でなければ、 履歴置換を開始します。
!!	直前のコマンドを指します。 他の文字を加えずにこれだけを入力すると、 直前のコマンドを繰り返します。
!*n*	*n* 番のコマンド行を指します。
!-*n*	入力中のコマンドから *n* 個前のコマンドを指します。
!*str*	*str* で始まる最新のコマンドを指します。
!?*str*?	*str* を含む最新のコマンドを指します。
!?*str*? *additional*	*str* を含む最新のコマンドを指し、 その参照コマンドに *additional* を追加します。
!{command} *additional*	command で始まる最新のコマンドを指し、 その参照コマンドに *additional* を追加します。
^*previous_word*^*replacement*^	文字列 *previous_word* を文字列 *replacement* に置き換えて、 直前のコマンド行を繰り返します。 これは履歴置換と同じ動作です。 !:s/*previous_word*/*replacement*/ .以前の特定のコマンドを再実行し、さらに同じような置換を行う場合、たとえば 6 番目のコマンドを再実行するには、次のようにします。 !:6s/*previous_word*/*replacement*/ .

ワード指示子:

A ':' (コロン) はイベント指示子とワード指示子とを区切ります。 ワード指示子が ^、 $、 *、 -、 または % で始まるときは省略できます。 直前のコマンドからワードを選択する場合、 イベント指示子の 2 番目の ! は省略できます。 たとえば !!:1 と !:1 は、 両方とも直前のコマンドの最初のワードを指します。 また、 !!$ と !$ は、 両方とも直前のコマンドの最後のワードを指します。 ワード指示子には以下のものがあります。

#	今までに入力したすべてのコマンド行
0	最初に入力したワード (コマンド)
n	*n* 番目の引数
^	最初の引数。 1 と同じ
$	最後の引数
%	最新の ?*s* による検索で一致したワード
x-*y*	ワードの範囲。 -*y* は 0-*y* の省略形
*	すべての引数。 イベント中に 1 ワードしか存在しないときは NULL
*x**	*x*-$ の省略形
x-	*x** と同じだが、 ワード $ を省略

修飾子:

オプションのワード指示子の後に、 : で始まる 1 つ以上の修飾子を追加できます。

h	パス名の後部分の構成要素を除去して、 前部分を残します。
r	'.*xxx*' の形の接尾辞を除去して、 ベース名を残します。
e	接尾辞以外はすべて除去して、 拡張部分を残します。
s/*oldchars*/*replacements*/	*replacements* を *oldchars* に置換します。 *oldchars* は組み込まれた空白文字を含む文字列ですが、 イベント指示子、 ^*oldchars*^*replacements*^ にある *previous_word* は空白文字を含みません。
t	パス名の前部分の構成要素を除去して、 後部分を残します。
&	前の置換を繰り返します。
g	各ワード内の最初の一致が発生した箇所を、 上記のオプションに接頭辞を付けて変更します (たとえば g&) 。
p	新しいコマンドを表示するだけで、 実行はしません。

q	置換されたワードをクォートして、 それ以上の置換をエスケープします。
x	q と同じですが、 空白文字、 タブ、 復帰改行文字 (NEWLINE) ごとにワードに分割します。

g を先頭に付加しないと、*oldchars* に一致する最初の文字列だけが変更されます。 一致する文字列がなければ、 エラーとなります。

置換部分の左側は正規表現ではなく文字列です。 / の箇所には、 区切り文字としてどのような文字でも使用できます。 区切り文字用の文字はバックスラッシュで囲まれます。 右側の & 文字は、 左側のテキストで置換されます。 & はバックスラッシュでクォートすることができます。 *oldchars* が NULL のとき、 直前の文字列における *oldchars* 、または !?*s*. における文脈検索文字列 *s* を使用します。 同様に *replacements* の直後に復帰改行がある場合、 文脈検索の最右にある ? は省略できます。

イベントが指定されないと、 履歴リファレンスは前のコマンドか、 (もしあれば) そのコマンド行上での 前の履歴リファレンスを参照します。

ksh fc -e- [*old=new*] [*command*] の形式で fc を指定すると、 *old=new* の置換を行なった後で *command* が再実行されます。*command* 引数を省略すると、最後に行なったコマンドが実行されます。

fc [-e *ename*] [-nlr] [*first* [*last*]] の形式で fc を指定すると、端末から最近入力された HISTSIZE 個のコマンドの中から、*first* から *last* までの範囲のコマンドを選択します。*first* と *last* の両引数は、 数値または文字列で指定できます。 文字列の場合、その文字列で始まる最新のコマンドを見つけます。 負の数値は、 現在のコマンド番号からのオフセットとなります。-l オプションを指定すると、標準出力上にコマンドを一覧表示します。 -l を指定しないと、これらのキーボードコマンドの入ったファイル上で -e *ename* で示すエディタプログラムを起動します。 *ename* が省略されていると、 変数 FCEDIT (デフォルトは /bin/ed) の値をエディタとして使用します。 編集が完了すると、 編集されたコマンドを実行します。*last* を省略すると、*first* と同一値に設定されます。*first* を省略すると、デフォルトは、編集については直前のコマンドに、 一覧表示については –16 になります。-r オプションはコマンドの順序を逆にします。-n オプションは一覧表示時にコマンド番号の出力を抑止します (コマンド行編集の詳細については ksh(1) を参照) 。

HISTFILE シェル起動時にこの変数が設定されていると、その値はコマンド履歴 を格納するために使用されるファイルのパス名になります。

HISTSIZE シェル起動時にこの変数が設定されていると、このシェルで使用可能な入力済みコマンドの数が、この値以上になります。デフォルト値は 128 です。

コマンド再入力:

端末装置から最近入力された HISTSIZE が示す個数 (デフォルトは 128 個) のコマンドのテキストは、履歴ファイルに保存されています。 $HOME/.sh_history というファイルは、HISTFILE 変数が設定されていない場合、または変数が示すファイルが書き込み不可能な場合に使用されます。シェルは、同じ名前の HISTFILE を使用する

対話型シェルすべてのコマンド履歴を使用できます。fc という特殊コマンドは、 このファイルの一部をリスト表示または編集するときに使用します。 編集またはリスト表示すべきファイルの部分は、 番号か、 またはコマンドの最初の文字 を 指定することによって選択できます。 単一のコマンドを指定することも、コマンドの範囲を指定することも可能です。fc の引数としてエディタプログラムが指定されていないと、FCEDIT という変数の値が使用されます。FCEDIT が未定義の場合は、/bin/ed が使われます。編集されたコマンドは、 エディタを終了した時点で 表示および再実行されます。エディタ名に – を指定すると、 編集段階が省かれ、コマンドが再実行されます。 この場合、 *old=new* という形式の代入パラメタを使用すれば、 実行前にコマンドを変更できます。 たとえば、 r が 'fc -e -' の別名として定義されているとき 'r bad=good c' と入力すると、c という文字で始まるコマンドのうち最新のものが、その記述中の最初の bad という 文字列を good に置き換えられて再実行されます。

複合コマンドの中に fc 組み込みコマンドを指定すると、 すべてのコマンドが履歴ファイルから 削除されます。

オプション

以下のオプションを指定できます。

`-e` *editor*
: *editor* が示すエディタを使ってコマンドを編集します。文字列 *editor* はユーティリティ名で、 PATH 変数の値に従って検索されます。-e を省略すると、 FCEDIT 変数の値がデフォルトとして用いられます。FCEDIT の値が NULL または未設定のときは、エディタとして ed が使用されます。

`-l`
: (文字のエル) エディタを呼び出して編集する代わりに、コマンドを一覧表示します。 *first* と *last* の両オペランドで指定した範囲のコマンドを、 -r オプションがあればそれに従って、順番にコマンド番号付きで表示します。

`-n`
: -l オプションによる一覧表示において、コマンド番号を出力しません。

`-r`
: コマンドの一覧表示 (-l 指定時) または編集 (-l および -s 省略時) において、順序を逆にします。

`-s`
: エディタを呼び出さずにコマンドを再実行します。

オペランド

以下のオペランドを指定できます。

first

last
: 表示または編集するコマンドを選択します。いくつまでさかのぼってコマンドをアクセスできるかは、HISTSIZE 変数の値により決まります。*first* と *last* の値は、それぞれ以下のいずれかです。

  [+]*number*
  : コマンド番号を表す正の整数。過去に実行した各コマンドの番号は、 -l オプションを使えば確認できます。

  –number
  : いくつ前のコマンドかを示す負の整数。 たとえば直前に実行したコマンドなら –1 となります。

string 指定した文字列で始まっていたコマンドのうち、 最後に実行したコマンド。*old=new* オペランドが `-s` オプションなしで指定された場合、文字列形式の *first* オペランドを使用するなら、その文字列中に等記号を含めることはできません。

「形式」の項で示した形式で、`-s` を指定する場合、

- *first* を省略すると、直前のコマンドが使用されます。

「形式」の項で示した形式で、`-s` を指定しない場合、

- *last* を省略すると、デフォルト値は `-l` 指定時は直前のコマンドとなり、`-l` 省略時は *first* の値となります。
- *first* と *last* の両方を省略すると、`-l` 指定時は直前の 16 個のコマンドの表示、`-l` 省略時は直前の 1 つのコマンドの編集となります。
- *first* と *last* の両方を指定すると、 *first* から *last* までのすべてのコマンドが表示 (`-l` 指定時) または編集 (`-l` 省略時) されます。複数のコマンドを一度に、 各々を新たな行で開始してエディタに渡せば、複数コマンドの編集が可能です。*first* が示すコマンドが *last* が示すものより新しい場合、`-r` 指定時と同じように逆の順序で表示または編集されます。次の例を見てください。1 行目の 2 つのコマンドは、2 行目のそれぞれ真下にあるコマンドと同じ意味を持ちます。

```
fc -r 10 20     fc    30 40
fc   20 10      fc -r 40 30
```

- 一連のコマンドを範囲指定する場合、 履歴リストに存在していない値を *first* や *last* に指定してもエラーとはなりません。 `fc` は、存在している最も古いまたは新しい番号を、その代わりに使用します。 たとえば、履歴リスト中に 10 個のコマンドが記録されていて、そのコマンド番号が 1 から 10 となっているとします。このとき、以下のコマンドはいずれも 10 個のコマンドすべてを表示または編集することになります。

```
fc -l
fc 1 99
```

old=new 再実行対象のコマンド中に最初に現れた文字列 *old* を、他の文字列 *new* に置き換えます。

出力 `-l` オプションを使ってコマンドを表示する場合、 その出力形式は次のとおりです。

`"%d\t%s\n", <`*line number*`>, <`*command*`>`

`-l` と `-n` の両オプションを指定すると、各コマンドの出力形式は 次のようになります。

"\t%s\n", <*command*>

command が複数の行で構成されている場合、 2 行目以降は以下のように表示されます。

"\t%s\n", <*continued-command*>

使用例

例 **1** history と fc の使用例

```
            csh                                    ksh

% history                              $ fc -l
  1   cd /etc                            1   cd /etc
  2   vi passwd                          2   vi passwd
  3   date                               3   date
  4   cd                                 4   cd
  5   du .                               5   du .
  6   ls -t                              6   ls -t
  7   history                            7   fc -l

% !d                                   $ fc -e - d
  du .                                   du .
  262   ./SCCS                           262   ./SCCS
  336   .                                336   .

% !da                                  $ fc -e - da
  Thu Jul 21 17:29:56 PDT 1994           Thu Jul 21 17:29:56 PDT 1994

%                                      $ alias \!='fc -e -'

% !!                                   $ !
  date                                   alias ='fc -e -'
  Thu Jul 21 17:29:56 PDT 1994
```

環境

fc の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

FCEDIT　この変数は、シェルによって展開されると、 e *editor* オプションの *editor* のデフォルト値を定義します。 FCEDIT が NULL または未設定の場合、エディタとして ed が使用されます。

HISTFILE　コマンドの履歴ファイル名を表すパス名を定義します。 HISTFILE 変数が設定されていないと、シェルはユーザーのホームディレクトリ内で .sh_history というファイルをアクセスまたは生成しようとします。この履歴ファイルへの読み込みおよび書き込みアクセスができない、または生成できない場合、シェルは何らかのメカニズムを使って 履歴が正しくとれるようにします。 (この項で述べる履歴ファイルの参照とは、 このメカニズムが使用される場合があることを前提としています。) 履歴ファイルの初期化時にのみ fc をアクセスするようにすることもできます。初期化は、 fc または sh がユーザーからのコマンドに従って、最

初にこのファイル、または ENV 変数が指定するファイル、または /etc/profile のようなシステム起動 ファイルからのエントリ検索もしくはエントリの追加を試みたときに 発生します。 なお履歴ファイル用の初期化処理は、 システム起動ファイルの内容に依存させることもできます。 つまり、ユーザーが設定した HISTFILE や HISTSIZE 値を効果的に上書きするようなコマンドを、このファイルに記述することが可能です。たとえば set -o nolog オプションが設定されていなければ、関数定義コマンドが 履歴ファイルに記録されます。 ENV ファイルの前に呼び出されるシステムスタートアップファイル中に、 システム管理者が関数定義を記述しておけば、 ユーザーが履歴ファイルの属性を変更するような動作が可能になる前に、 履歴ファイルが初期化されます。 シェルが呼び出されると、最初に HISTFILE 変数が参照されます。 他のシェルが呼び出されるまでは、HISTFILE に対する変更は反映されません。

HISTSIZE　以前のコマンドを最大いくつまでさかのぼってアクセスできるかを 10 進数を使って定義します。この変数が設定されていないと、128 以上の不定の値がデフォルトとして用いられます。 シェルが呼び出されると、最初に HISTSIZE 変数が参照されます。 他のシェルが呼び出されるまでは、HISTSIZE に対する変更は反映されません。

終了ステータス

以下の終了ステータスが返されます。

0　一覧表示の正常終了

>0　エラーが発生した。

上記以外の場合、 fc により実行されたコマンドの終了ステータスがそのまま返されます。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ed(1), ksh(1), set(1), set(1F), sh(1), source(1), attributes(5), environ(5)

名前 | fdformat – フロッピーディスクまたは PCMCIA メモリーカードのフォーマット

形式 | **fdformat** [-dDeEfHlLmMUqvx] [-b *label*] [-B *filename*] [-t *dostype*] [*devname*]

機能説明 | fdformat ユーティリティは、フロッピーディスクと PCMCIA メモリーカードをフォーマットします (fdformat の大部分の機能は、rmformat(1) ユーティリティにも提供されています)。新しいフロッピーディスクや PCMCIA メモリーカードは、使用する前にフォーマットする必要があります。

fdformat は、媒体をフォーマットして検証し、不良セクターが存在するかどうかを示します。フォーマットを行うと、フロッピーディスクまたは PCMCIA メモリーカード上にあった既存データは消去されます。デバイス名 (*devname*) を指定しない場合、fdformat はデフォルトでフロッピーディスクを使用します。

デフォルトでは、fdformat はドライブが持つ能力を使用して、フロッピーディスクをフォーマットします。3.5 インチ高密度ドライブは 1.44M バイトのフォーマット能力を持ちます。5.25 インチ高密度ドライブは 1.2M バイトのフォーマット能力を持ちます。どちらの場合も fdformat に密度オプションを指定する必要はありません。ただし、ドライブのデフォルト密度よりも低い密度のフロッピーディスクを使用する場合は、密度オプションを指定しなければなりません。超高密度 (ED) ドライブで高密度フロッピーディスク (1.44M バイト) をフォーマットするには、-H オプションを使用します。高密度 (HD) ドライブまたは超高密度 (ED) ドライブで倍密度 (低密度) フロッピーディスク (720K バイト) をフォーマットするには、-D、-l、または -L オプションを使用します。中密度フロッピーディスク (1.2M バイト) をフォーマットするには、-M オプション (-m オプションと同じ) と -t nec を組み合わせて使用します。

拡張密度は、両面拡張密度または超高密度 (DS/ED) フロッピーディスクで使用します。中密度および高密度は、両面高密度 (DS/HD) フロッピーディスクで使用します。倍密度 (低密度) は、両面倍密度 (DS/DD) フロッピーディスクで使用します。通常、ある密度のフロッピーディスクをより高いまたはより低い密度のフロッピーディスクに変更することは不可能です。フロッピーディスクの密度と異なる密度でフォーマットした場合、データの完全性は保証できません。

512K バイトから 64M バイトまでの密度の PCMCIA メモリーカードをフォーマットできます。

-x オプションを指定しない限り、fdformat は、すべてのトラック上のセクターごとに新しい識別フィールドとデータフィールドを書き込みます。-v オプションを指定した場合、fdformat はフロッピーディスクの各セクターを検証します。

フォーマットと検証の後、fdformat はオペレーティングシステムのラベルをブロック 0 に書き込みます。-t dos オプション (-d オプションと同じ) を使用すると、fdformat はフォーマット後に、 MS-DOS ファイルシステムをフロッピーディスクまたは PCMCIA メモリーカードに書き込みます。-t nec オプションと -M オプション (-m オプションと同じ) を組み合わせて使用すると、fdformat は NEC-DOS ファイルシステムをフロッピーディスクに書き込みます。上記以外の場合、fdformat は SunOS ラベルをブロック 0 に書き込みます。

オプション | 次のオプションを指定できます。

-b *label*	媒体にボリュームラベル (*label*) を書き込みます。SunOS のボリュームラベルは 8 文字までの英数字です。MS-DOS のボリュームラベルは 11 文字までの大文字の英数字です。
-B *filename*	特別なブートローダーを MS-DOS フロッピーディスク上の *filename* にインストールします。このオプションを指定するときには、必ず -d (または -t dos) オプションも指定する必要があります。
-D	720K バイト (3.5 インチ) または 360K バイト (5.25 インチ) 倍密度フロッピーディスクをフォーマットします (-l または -L オプションと同じ)。倍密度タイプのドライブでは、このオプションがデフォルトの動作です。高密度または拡張密度タイプのドライブでは、このオプションを指定してください。
-e	フォーマット後、フロッピーディスクを取り出します。この機能は、一部のシステムでは利用できません。
-E	2.88M バイト (3.5 インチ) 拡張密度フロッピーディスクをフォーマットします。拡張密度タイプのドライブでは、このオプションがデフォルトの動作です。
-f	強制的にフォーマットします。つまり、ユーザーによる確認なしでフォーマットを開始します。
-H	1.44M バイト (3.5 インチ) または 1.2M バイト (5.25 インチ) 高密度フロッピーディスクをフォーマットします。高密度タイプのドライブでは、このオプションがデフォルトの動作です。拡張密度タイプのドライブでは、このオプションを指定してください。
-M	高密度フロッピーディスクを 1.2M バイト (3.5 インチ) 中密度でフォーマットします。このオプションを指定するときには、必ず -t nec オプションも指定する必要があります。-M と -m は同じです。 この機能は、一部のシステムでは利用できません。
-q	(Quiet) 状態メッセージを出力しません。
-t dos	フォーマット後、MS-DOS ファイルシステムとブートセクターをフロッピーディスクにインストールします。このオプションは、-d オプションまたは MS-DOS の FORMAT コマンドと同じです。
-t nec	フォーマット後、NEC-DOS ファイルシステムとブートセクターをフロッピーディスクにインストールします。このオプションを指定するときには、必ず -M オプションも指定する必要があります。この機能は、一部のシステムでは利用できません。
-U	すべてのファイルシステム上で umount を実行し、その後でフォーマットします。mount(1M) のマニュアルページを参照してください。

`-v` フォーマット後、フロッピーディスクの各ブロックを検証します。

`-x` フォーマットを実行せずに、SunOS ラベルまたは MS-DOS ファイルシステムだけを書き込みます。

オペランド　次のオペランドを指定できます。

devname 第 1 ドライブを使用する場合は、`rdiskette0` (ボリューム管理なしのシステム) または `floppy0` (ボリューム管理ありのシステム) を *devname* に指定します。第 2 ドライブを使用する場合は、`rdiskette1` (ボリューム管理なしのシステム) または `floppy1` (ボリューム管理ありのシステム) を *devname* に指定します。*devname* を指定しない場合、第 1 ドライブが (存在すれば) 使用されます。

PCMCIA メモリーカードの場合、`/dev/rdsk/c`*N*`t`*N*`d`*N*`s`*N* または `/dev/dsk/c`*N*`t`*N*`d`*N*`s`*N* に存在する PCMCIA メモリーカード用のデバイス名を *devname* に指定します。*devname* を指定しない場合、デフォルトのフロッピーディスクドライブが (存在すれば) 使用されます。*N* は 10 進数の番号で、次のように指定します。

`c`*N* コントローラ *N*

`t`*N* テクノロジタイプ *N*:

0x1	`ROM`
0x2	`OTPROM`
0x3	`EPROM`
0x4	`EEPROM`
0x5	`FLASH`
0x6	`SRAM`
0x7	`DRAM`

`d`*N* テクノロジタイプ内のテクノロジ領域 *N*

`s`*N* スライス *N*

次のオプションは、以前のバージョンの `fdformat` との互換性のために残されています。できるだけ使用しないようにしてください。

`-d` MS-DOS フロッピーディスクまたは PCMCIA メモリーカードをフォーマットします (`-t dos` と同じ)。`MS-DOS` の `FORMAT` コマンドと同じです。

`-l` 720K バイト (3.5 インチ) または 360K バイト (5.25 インチ) 倍密度フロッピーディスクをフォーマットします (`-D` または `-L` と同じ)。倍密度タイプのドライブでは、このオプションがデフォルト

の動作です。高密度または拡張密度タイプのドライブでは、このオプションを指定してください。

`-L` 720K バイト (3.5 インチ) または 360K バイト (5.25 インチ) 倍密度フロッピーディスクをフォーマットします (`-l` または `-D` と同じ)。倍密度タイプのドライブでは、このオプションがデフォルトの動作です。

`-m` 高密度フロッピーディスクを 1.2M バイト (3.5 インチ) 中密度でフォーマットします。このオプションを指定するときには、必ず `-t nec` オプションも指定する必要があります。`-m` と `-M` は同じです。この機能は、一部のシステムでは利用できません。

ファイル

`/vol/dev/diskette0`	フロッピーディスクドライブ `0` 内にある媒体に、ブロック型デバイスとしてアクセスするためのディレクトリ。
`/vol/dev/rdiskette0`	フロッピーディスクドライブ `0` 内にある媒体に、文字型デバイスとしてアクセスするためのディレクトリ。
`/vol/dev/aliases/floppy0`	フロッピーディスクドライブ `0` 内にある媒体用の文字型デバイスへのシンボリックリンク。
`/dev/rdiskette`	一次フロッピーディスクドライブ (通常はドライブ `0`) 内にある媒体に、文字型デバイスとしてアクセスするためのディレクトリ。
`/vol/dev/dsk/c`*N*`t`*N*`d`*N*`s`*N*	PCMCIA メモリーカードにブロック型デバイスとしてアクセスするためのディレクトリ。*N* の値については、「オペランド」の項を参照してください。
`/vol/dev/rdsk/c`*N*`t`*N*`d`*N*`s`*N*	PCMCIA メモリーカードに文字型デバイスとしてアクセスするためのディレクトリ。*N* の値については、「オペランド」の項を参照してください。
`/vol/dev/aliases/pcmem`*S*	ソケット *S* 内にある PCMCIA メモリーカード用の文字型デバイスへのシンボリックリンク (*S* は PCMCIA のソケット番号を示す)。
`/dev/rdsk/c`*N*`t`*N*`d`*N*`s`*N*	PCMCIA メモリーカードに、文字型デバイスとしてアクセスするためのディレクトリ。*N* の値については、「オペランド」の項を参照してください。
`/dev/dsk/c`*N*`t`*N*`d`*N*`s`*N*	PCMCIA メモリーカードに、ブロック型デバイスとしてアクセスするためのディレクトリ。*N* の値については、「オペランド」の項を参照してください。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | cpio(1), eject(1), rmformat(1), tar(1), volcancel(1), volcheck(1), volmissing(1), volrmmount(1), mount(1M), newfs(1M), prtvtoc(1M), vold(1M), rmmount.conf(4), vold.conf(4), attributes(5), pcfs(7FS), volfs(7FS)

IA のみ | fd(7D)

注意事項 | SPARC ベースのシステム上で fdformat と newfs(1M) コマンドを使用して作成された UFS ファイルシステムを含むフロッピーディスクまたは PCMCIA メモリーカードは、IA ベースのシステム上で作成された UFS ファイルシステムを含むフロッピーディスクまたは PCMCIA メモリーカードと同じではありません。これらのプラットフォーム間で UFS のフロッピーディスクまたは PCMCIA メモリーカードを交換しないでください。これらのプラットフォーム間でフロッピーディスクまたはメモリーカード上のファイルを転送するには、 cpio(1) または tar(1) コマンドを使用してください。

-t dos (または -d) オプションを使用して MS-DOS 用にフォーマットされたフロッピーディスクまたは PCMCIA メモリーカードには、必要なシステムファイルがインストールされません (つまり、ブートできません)。このようなフロッピーディスクまたは PCMCIA メモリーカードを使用して PC 上でブートしようとすると、次のようなメッセージが表示されます。

```
Non-System disk or disk error.
Replace and strike any key when ready
```

使用上の留意点 | 現在のところ、フロッピーディスクまたは PCMCIA メモリーカード上の不良セクターの場所を検出する機能は、サポートされていません。このため、fdformat を実行してエラー (不良セクター) が検出された場合、そのフロッピーディスクまたは PCMCIA メモリーカードを使用できません。

名前 | jobs, fg, bg, stop, notify – プロセスの実行の制御

sh

jobs [-p | -l] [% *job_id*...]

jobs -x *command* [*arguments*]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

csh

jobs [-l]

fg [% *job_id*]

bg [% *job_id*...]

notify [% *job_id*...]

stop % *job_id*...

stop *pid*...

ksh

jobs [-lnp] [% *job_id*...]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

sh

ジョブ制御が有効なとき、Bourne シェルに組み込まれた jobs は、停止中またはバックグラウンドで実行中の すべてのジョブを表示します。 %*job_id* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが表示されます。次のオプションを使って、ジョブに関する表示を変更できます。

-l　ジョブのプロセスグループ ID および作業ディレクトリを表示します。

-p　ジョブのプロセスグループ ID のみを表示します。

-x　*command* または *argument* 中に見つかった *job_id* を、対応するプロセスグループ ID に置き換え、*command* に *argument* を渡して実行します。

シェルを jsh として呼び出すと、sh の説明で述べたすべての機能に加えて、ジョブ制御が可能になります。通常、ジョブ制御は対話型シェルに対してだけ可能です。通常、非対話型シェルは、ジョブ制御の機能 を使用しません。

ジョブ制御が可能なとき、ユーザーが端末から入力したコマンドまたはパイプラインは、すべて *job_id* と呼ばれます。どのジョブも、必ずフォアグラウンド、バックグラウンド、または停止のいずれかの状態にあります。 これらの用語の定義を次に示します。

1. フォアグラウンド状態にあるジョブは、 制御している端末への読み取りおよび 書き込みアクセス権を持っています。
2. バックグラウンド状態にあるジョブは、 制御している端末への読み取りアクセスを拒否されていますが、 条件付き書き込みアクセス権は持っています (`stty`(1) を参照)。
3. 停止ジョブは保留状態に置かれたジョブであり、通常は `SIGTSTP` シグナルにより、この状態になります (`signal`(3HEAD) を参照)。

シェルが起動するすべてのジョブには、 ジョブ番号 (*job-id number*) と呼ばれる正の整数が割り当てられます。シェルはこの番号を把握し、 特定のジョブを示す識別子として使用します。また、シェルは現在 (*current*) および前回 (*previous*) のジョブも把握しています。現在のジョブとは、最後に起動または再起動されたジョブです。 前回のジョブとは、その直前のジョブです。

ジョブ識別子の正しい構文は次のような形式です。

`%`*job_id*

job_id は、次のいずれかの形式で指定できます。

`%` または `+`	現在のジョブ
–	前回のジョブ
`?`<*string*>	*string* を含むコマンド行 (一意に表す) に対応したジョブ
n	ジョブ番号が *n* のジョブ
pref	コマンド名の先頭が *pref* のコマンド。 たとえば `ls -l` *name* がバックグラウンドで実行中だった場合、`% ls` と指定すればこのコマンドを示すことができます。*pref* は、引用符で囲まない限り、空白文字を含めることができません。

ジョブ制御が有効なとき、`fg` は中断しているジョブの実行を フォアグラウンドで再開します。 またバックグラウンドで実行中のジョブを フォアグラウンドに移動します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

ジョブ制御が有効なとき、`bg` は中断されているジョブの実行を バックグラウンドで再開します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

`stop` は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (`ps`(1) を参照) 。

csh C シェルに組み込まれた `jobs` は、引数なしでジョブ制御下で活動中のジョブを一覧表示します。

`-l`	通常の情報の他に、プロセス ID を表示します。

シェルは、番号の付いた *job_id* を各コマンドシーケンスと対応付けて、バックグラウンドで動作中のコマンド、または TSTP シグナル (通常は Control-Z) によって停止したコマンドの動作を追跡します。コマンドまたはコマンドシーケンス (セミコロンで区切られたリスト) をメタキャラクタ & を使用してバックグラウンドで起動した場合、 シェルは角括弧で囲まれたジョブ番号と 関連するプロセス番号のリストを表示します。以下に例を示します。

[1] 1234現在のジョブリストを見るには、組み込みコマンド jobs を使用します。最後に停止したジョブ (停止したジョブがない場合は、 最後にバックグラウンドに投入されたジョブ) を「現在のジョブ」といい、 '+' で示します。前のジョブは '-' で示します。 現在のジョブが終了したりフォアグラウンドに移された場合、 前のジョブが新しく現在のジョブになります。

ジョブの操作方法については、組み込みコマンド bg、 fg、 kill、 stop、 % の説明を参照してください。

ジョブの参照は '%' で始まります。 パーセント記号だけの指定は、現在のジョブを示します。

% %+ %%	現在のジョブ
%-	前のジョブ
%*j*	'kill -9 %*j*' のようにジョブ *j* を参照します。 *j* はジョブ番号、またはジョブを起動した コマンド行を一意に表す文字列です。 たとえば 'fg %vi' は、停止した vi ジョブをフォアグラウンドに移します。
%?*string*	*string* を含むコマンド行 (一意に表す) に対応したジョブを指定します。

バックグラウンドで動作中のジョブは、端末からの読み取り時に停止します。 バックグラウンドジョブは、通常出力を生成しますが、'stty tostop' コマンドを使用して抑止することも可能です。

fg は現在のジョブまたは指定された *job_id* をフォアグラウンドへ移します。

bg はバックグラウンドで、現在のジョブ または指定されたジョブを実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

notify は現在のジョブまたは指定されたジョブの状態が変わったとき、 その旨非同期にユーザーに知らせます。

ksh jobs は、現在のシェル環境で開始されたジョブの状況を表示します。 jobs がジョブの終了を報告したとき、シェルはそのジョブのプロセス ID を、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除します。

特定のジョブの報告だけが必要なら、*job_id* を使ってジョブを指定します。*job_id* を 1 つも指定しないと、全ジョブに関する情報が出力されます。

以下のオプションは、 `jobs` の出力を変更または拡張するために使用します。

`-l` (文字のエル) 個々のジョブに関して詳細な情報を出力します。具体的には、 ジョブ番号、現在のジョブ、プロセスグループ ID 、状態、ジョブを生成したコマンドを出力します。

`-n` 前回通知を受けた後に停止または終了した ジョブだけを表示します。

`-p` 選択されたジョブのプロセスグループリーダのプロセスグループ ID だけを出力します。

デフォルトでは、`jobs` は、停止しているすべてのジョブの状態、実行中のバックグラウンドジョブの 状態、そして状態が変わったのにシェルによりまだ報告されていない すべてのジョブの状態を表示します。

`set` コマンドの `monitor` オプションを有効にすると、対話型シェルが `job` を各パイプラインと関連付けます。 このオプションは、`jobs` コマンドが表示する現在のジョブのテーブルを維持し、これらのジョブに整数番号を割り当てます。ジョブを `&` で非同期に起動すると、シェルは、`[1] 1234` という形式の行を表示します。 非同期に起動されたジョブはジョブ番号 1 であり、プロセス ID が 1234 であるプロセスが 1 つ (トップレベル) あることを示します。実行中のジョブがあるが、 別に実行したいジョブがある場合、`^Z` (Control-Z) キーを押せば、 現在のジョブに `STOP` シグナルが送信されます。 そうするとシェルは通常、 ジョブが「停止」されたことを示し (後述の「出力」の項を参照)、 プロンプトを表示します。これで、このジョブの状態を `bg` コマンドでバックグラウンドで処理するか、または他のコマンドを実行してから、 `fg` というコマンドで ジョブをフォアグラウンドに移すことができます。`^Z` は直ちに有効になります。つまり `^Z` は、 保留中の出力や読み取られていない入力が 直ちに中止されるという点で、割り込みに似ています。

シェル内のジョブを参照する方法はいくつかあります。そのジョブのいずれかのプロセスの ID を使っても、また以下のいずれかを使っても参照できます。

`%`*number* *number* が示す番号のジョブ

`%`*string* コマンド行が *string* で始まっていたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%?`*string* コマンド行が *string* を含んでいたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%%` 現在のジョブ

`%+` `%%` と同じ

`%-` 直前のジョブ

シェルは、 プロセスの状態が変更すると、 直ちにそれを検出します。 ジョブがブロックされてそれ以上進めない状態になると、 シェルはその旨をユーザーに通知します。 ただし、ユーザーの作業の妨げにならないよう、 この通知はプロンプトを発行

する直前にだけ行います。 モニタモードが有効なとき、 完了した各バックグラウンドジョブは、 CHLD に設定されているトラップを起こします。 ジョブの実行中または停止中にシェルを終了しようとすると、 「停止中 (実行中) のジョブがある (‘You have stopped (running) jobs.’) 」旨の警告を受けます。 jobs コマンドを使用すれば、 どのジョブが該当するのかを確認できます。これを実行するか、 または直ちにシェルを再終了しようとすると、 シェルは 2 度目の警告は出さず、 停止中のジョブは終了します。

fg は、バックグラウンドジョブを、現在の環境からフォアグラウンドへ移します。 fg を使ってジョブをフォアグラウンドへ移した場合、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除されます。 fg コマンドが使えるのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をフォアグラウンドで実行します。*job_id* が指定されないと、 現在のジョブをフォアグラウンドで実行します。

bg は、現在の環境で中断されたジョブを、バックグラウンドジョブとして 実行することにより再開します。 *job_id* が示すジョブがすでにバックグラウンドジョブを実行している場合、bg は何も行わず正常に終了します。bg を使ってジョブをバックグラウンドへ移した場合、あたかも非同期リストから起動されたかのように、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID 」の 1 つとなります。 bg コマンドが使えるのは、 ジョブ制御をサポートするシステム上だけです。*job_id* が省略された場合は、 現在のジョブをバックグラウンドで実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

出力

-p オプションを指定すると、各プロセス ID に対して次に示す 1 行の情報が出力されます。

"%d\n", *"process ID"*

-p を省略すると、 -l オプションも省略されていれば、以下の形式の一連の行が出力されます。

"[%d] %c %s %s\n", *job-number*, *current*, *state*, *command*

各フィールドの意味を以下に説明します。

current

文字 + は、 fg および bg コマンド用のデフォルトとして使用するジョブを表します。 このデフォルトジョブは、 *job_id* %+ または %% を使って指定することもできます。文字 – は、現在のデフォルトジョブが終了してしまった場合に デフォルトとなるジョブを表します。このジョブは、 *job_id* %– を使って指定することもできます。 その他のジョブに関しては、このフィールドは空白文字として出力されます。 + や – を使って表せるジョブの数は、どちらも最大 1 つです。停止中の

ジョブがあれば、現在のジョブも停止ジョブとなります。 停止中のジョブが 2 つ以上あれば、以前のジョブも停止ジョブとなります。

job-number

`wait` 、 `fg` 、 `bg` 、 `kill` の各ユーティリティ用にプロセスグループを識別するのに使用する番号。 これらのユーティリティを使うと、ジョブはジョブ番号の後に `%` を付加することにより識別できます。

state

以下の文字列 (POSIX ロケールにある) のいずれかです。

`Running`	ジョブはシグナルによって中断されておらず、 終了もしていないことを表す。
`Done`	ジョブは終了して、ゼロの終了ステータスを返したことを表す。
`Done(`*code*`)`	ジョブは正常に終了し、指定されたゼロ以外の終了ステータス (*code* が示す 10 進数) を返したことを表す。
`Stopped`	
`Stopped (SIGTSTP)`	`SIGTSTP` シグナルがジョブを停止したことを表す。
`Stopped (SIGSTOP)`	`SIGSTOP` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTIN)`	`SIGTTIN` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTOU)`	`SIGTTOU` シグナルがジョブを停止したことを表す。

利用者側で、文字列 `Stopped` の代わりに `Suspended` を使うよう定義することができます。 ジョブをシグナルが終了した場合、 `state` の形式は不定ですが、ここに示した他の `state` 形式とは明確に区別できるものです。その出力上で、 ジョブを終了させたシグナルの名前または説明が与えられます。

command

シェルに与えられた関連コマンド。

`-l` オプションを指定すると、プロセスグループ ID を示すフィールドが `state` フィールドの前に挿入されます。さらに、プロセスグループ内の より多くのプロセスが別の行に出力されることがあります。 その内容は、プロセス ID と `command` フィールドだけです。

環境

jobs 、 fg 、 bg の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

jobs 、 fg 、 bg は、以下の終了ステータスを返します。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), kill(1), ksh(1), ps(1), sh(1), stop(1), shell_builtins(1), stty(1), wait(1), signal(3HEAD), attributes(5), environ(5)

名前 | fgrep – ファイル内の固定文字列のみを検索

形式 | **/usr/bin/fgrep** [-bchilnsvx] [-e *pattern_list*] [-f *pattern –file*] [*pattern*] [*file...*]

/usr/xpg4/bin/fgrep [-bchilnsvx] [-e *pattern_list*] [-f *pattern –file*] [*pattern*] [*file...*]

機能説明 | fgrep (高速 grep) ユーティリティは、ファイルで文字列を検索し、その文字列を含むすべての行を出力します。 fgrep は文字列を検索するため、 正規表現と一致するパターンを検索する grep(1) および egrep(1) とは異なります。 fgrep は高速で単純なアルゴリズムを使用します。

$、 *、 [、 ^、 |、 (、)、 および \ の文字は、 fgrep により文字どおりに解釈されます。すなわち、 fgrep は、 egrep の場合と異なり、正規表現をすべて認識できるわけではありません。 これらの文字はシェルに対して特別の意味を持つので、 '...' のように全 *string* を単一引用符で囲むことをお勧めします。

ファイルを何も指定しないと、 fgrep は標準入力を想定します。通常、発見された各行は標準出力にコピーされます。複数の入力ファイルがある場合、表示された各行の前にファイル名が出力されます。

オプション | 以下のオプションを指定できます。

-b	各行の先頭に、その行が発見されたブロック番号を出力します。これは、ブロック番号を文脈別に見つける際に役立ちます。最初のブロックは 0 です。
-c	パターンを含む行の数だけを出力します。
-e *pattern_list*	*pattern_list* にある *string* を検索します。これは、*string* が - ではじまる場合に有効です。
-f *pattern–file*	*pattern–file* からパターンのリストを取り出します。
-h	複数のファイルを検索するときにファイルを出力しません。
-i	比較中に大文字と小文字の区別を無視します。
-l	一致する行のあるファイルの名前を 1 行ずつ復帰改行文字で区切って出力します。パターンが 2 度以上見つかるときは、ファイルの名前を繰り返しません。
-n	各行の先頭に、その行のファイル中の行番号を出力します。最初の行は 1 です。
-s	エラーメッセージを表示するだけです。エラーステータスをチェックするのに有用です。
-v	パターンを含む行を除いたすべての行を出力します。
-x	完全に一致する行だけを出力します。

オペランド | 以下のオペランド を指定できます。

file パターンを検索するファイルのパス名を指定します。このオペランドを省略すると、標準入力が用いられます。

/usr/bin/fgrep

pattern 入力の検索時に用いるパターンを指定します。

/usr/xpg4/bin/fgrep

pattern 入力の検索時に用いる 1 つまたは複数のパターンを指定します。オペランドは -e*pattern_list* が指定されたものとして扱われます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `fgrep` の動作については、`largefile`(5) を参照してください。

環境

`fgrep` の実行に影響を与える環境変数 `LC_COLLATE` 、 `LC_CTYPE`、`LC_MESSAGES`、 `NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 一致するものが 1 つ以上見つかった

`1` 一致するものが 1 つも見つからなかった

`2` 構文エラーが検出された、 またはアクセスできないファイルがあった (一致するものが見つかった場合でも)

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/fgrep

属性タイプ	属性値
使用条件	SUNWcsu

/usr/xpg4/bin/fgrep

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

`ed`(1), `egrep`(1), `grep`(1), `sed`(1), `sh`(1), `attributes`(5), `environ`(5), `largefile`(5), `XPG4`(5)

注意事項

理想的には `grep` コマンドを 1 つだけにすべきなのですが、 空間と時間を調節できるだけの広範なアルゴリズムはありません。

1 行は仮想記憶に使用できるサイズに 制限されています。

/usr/xpg4/bin/fgrep

`/usr/xpg4/bin/fgrep` ユーティリティは `/usr/xpg4/bin/grep -F` と同じです (`grep`(1) 参照) 。 移植性が必要なアプリケーションでは `/usr/xpg4/bin/grep -F` を使用してください。

名前 | file – ファイルタイプの判別

形式 | **file** [-h] [-m *mfile*] [-f *ffile*] *file...*

file [-h] [-m *mfile*] -f *ffile*

file -c [-m *mfile*]

機能説明 | file ユーティリティは、*file* で指定した各ファイル、およびオプションとして指定した *ffile* 中に記述された個々のファイルを分類します。 指定したファイルが通常ファイル以外である場合、そのタイプを判別します。判別されるタイプには、ディレクトリ、FIFO、ブロック特殊ファイル、文字特殊ファイルなどがあります。ファイル長がゼロの通常ファイルは、空ファイルと判別します。

file がテキストファイルである場合、file は最初の 512 バイトを検査し、そのプログラミング言語を判定します。*file* が実行可能ファイル a.out である場合、file はバージョンスタンプを出力します。ただし、これは、バージョンスタンプが 0 より大きい場合に限られます。*file* がシンボリックリンクである場合、デフォルトでは、file はそのリンクをたどって参照先のファイルをテストします。

/usr/lib/locale/*locale*/LC_MESSAGES/magic ファイル (日本語ロケールの場合は /usr/lib/locale/ja/LC_MESSAGES/magic) が存在する場合、file はデフォルトで、このマジックファイルを使用してマジック番号を持つファイルを識別します。/usr/lib/locale/*locale*/LC_MESSAGES/magic が存在しない場合、file は /etc/magic ファイルを使用します。マジック番号は、ファイルタイプを表す数値定数または文字列定数です。 /etc/magic の形式については magic(4) を参照してください。

file が存在しない、読み取りできない、または状態が判別できない場合でも、終了ステータスに影響を与えるようなエラーとは見なされません。この場合、ファイルの処理は行われたがタイプは判断できなかったということが示されます。

オプション | 以下のオプションを指定できます。

-c	マジックファイルにフォーマットエラーがないかどうかをチェックします。 効率上の問題から、この妥当性検査は通常実行しません。
-h	シンボリックリンクが参照するファイルをテストしません。
-f *ffile*	*ffile* は、検査すべきファイルの一覧が記述されているファイルの名前を表します。
-m *mfile*	*mfile* を /etc/magic に代わる代替マジックファイルとして使用します。

オペランド | 以下のオペランドを指定できます。

file	テストするファイルのパス名。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の file の動作については、largefile(5) を参照してください。

使用例

例 1 実行可能なバイナリファイル

引数として指定したファイルがバイナリの実行可能ファイルであるかどうかを検査する例です。

```
file "$1" | grep -Fq executable &&          printf "%s is executable.\n" "$1"
```

環境

`file` の実行に影響を与える環境変数 `LC_CTYPE` 、 `LC_MESSAGES` 、 `NLSPATH` の詳細については、 `environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 正常終了

`>0` エラーが発生した

ファイル

`/etc/magic` `file` のマジック番号ファイル

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

`ls`(1), `magic`(4), `attributes`(5), `environ`(5), `largefile`(5)

診断

`-h` オプションが指定され、*file* がシンボリックリンクである場合、`file` は次のエラーメッセージを出力します。

`symbolic link to` *file*

名前 | file – ファイルの内容を検査してファイルタイプを識別

形式 | **/usr/ucb/file** [-f *ffile*] [-cL] [-m *mfile*] *filename...*

機能説明 | file コマンドは、*filename* 引数で指定された個々のファイルを検査し、その内容を確かめます。内容が ASCII テキストの場合、file は先頭の 512 バイトをチェックしてその言語を推定します。

file コマンドは /etc/magic ファイルを使用して、ある種のマジック番号すなわちタイプを示す数値定数または文字列定数を含むファイルを識別します。

オプション

-c	マジック番号ファイルを検査し、フォーマットエラーの有無を調べます。 処理効率が良くないので、この検査を実施することはほとんどありません。 本オプション指定時には、ファイルタイプのチェックは行いません。
-f *ffile*	タイプを識別するファイル名のリストを *ffile* で示すファイルから得ます。
-L	ファイルがシンボリックリンクの場合、 リンク自体ではなくリンクが参照しているファイルを検査します。
-m *mfile*	代替マジック番号ファイル名として、 *mfile* が示すファイルを使用します。

使用例 | **例 1** あるユーザーのディレクトリ中の全ファイルに対して file コマンドを実行する

あるユーザーのディレクトリ中の全ファイルに対して file コマンドを実行した例を以下に示します。

```
example% pwd
/usr/blort/misc

example% /usr/ucb/file  *

code:           mc68020 demand paged executable
code.c:         c program text
counts:         ascii text
doc:            roff,nroff, or eqn input text
empty.file:     empty
libz:           archive random library
memos:          directory
project:        symboliclink to /usr/project
script:         executable shell script
titles:         ascii text
s5.stuff:       cpio archive
example%
```

環境 | file の実行時、 LC_CTYPE、 LANG、 LC_default の 3 つの環境変数が文字分類を制御します。 file の開始時、 LC_CTYPE の次に LANG、その次に LC_default、の順序で変数の値がチェックされます。正当な値を持つ変数が見つかれば、 その値が用

いられ残りの変数の値は無視されます。したがって、たとえば LANG に新たな値を設定しても、現在 LC_CTYPE が示す文字分類規則が有効であれば、規則は変更されません。いずれの変数の値も正当でないと、シェル文字分類規則のデフォルトとして POSIX.1 C ロケールが使用されます。

ファイル

/etc/magic

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWscpu

関連項目

magic(4), attributes(5)

使用上の留意点

file はしばしば認識を誤ることがあります。特にコマンドファイルを C プログラムファイルと認識してしまうことがよくあります。

また file は Pascal や LISP を認識することができません。

名前 | mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式 | **mailp** [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明 | `mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション | 次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D` 出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。

`-F` メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。

`-h` バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。

`-l` 横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。

`-P` *printer* `-d` オプションを指定した場合と同じです。

`-s` *subject* *subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename 読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`1` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

名前 | filesync – 通常ファイル、ディレクトリファイル、特殊ファイルの同期

形式 | **filesync** [-aehmnqvy] [-o src | dst] [-f src | dst | old | new] [-r *directory...*]

filesync [-aehmnqvy] -s *source-dir* -d *dest-dir filename...*

機能説明 | filesync ユーティリティは、複数のコンピュータシステム (通常は、サーバーとポータブルコンピュータ) 間でファイルを同期させます。filesync は、通常ファイル、ディレクトリファイル、特殊ファイルの同期を取ります。本来、モバイルシステムでの使用を目的としていますが、filesync は据え置き型のシステムにおけるバックアップやファイルの複製などにも便利です。

システム間でファイルを同期させた場合、それぞれのシステム上の対応するファイルが同じものになります。一方または両方のシステムでファイルを変更すると、対応するファイルは同じではなくなります (つまり、同期していません)。もう一度ファイルを同じにするには、ファイル間の違いを調整しなければなりません。filesync がどのようにファイルを調整して同期させるかについての詳細は、「ファイルの調整と同期」を参照してください。

filesync コマンドには 2 つの形式があります。最初の形式の filesync はファイル引数なしで呼び出します。この形式の filesync は $HOME/.packingrules ファイルに指定されているファイルおよびシステム間の違いを調整します。$HOME/.packingrules は、filesync および cachefspack 用のパッキング規則リストで、同期させるファイルのリストが入っています。packingrules(4) と cachefspack(1M) のマニュアルページを参照してください。

2 番目の形式の filesync は、ソースシステム上のディレクトリから宛先システム上のディレクトリに特定のファイルをコピーします。さらに、この形式の filesync は、引数 (*filename*) として指定した (1 つまたは複数の) ファイルを $HOME/.packingrules に追加します。ソースシステムと宛先システム上のディレクトリを指定する方法については、-s オプションと -d オプションを参照してください。ファイル (*filename*) 引数を指定する方法については、「オペランド」の項を参照してください。

filesync コマンドは複数累積できます。つまり、filesync コマンドに指定したファイルは既存のパッキング規則ファイルに追加されます。「複数の filesync コマンド」を参照してください。

ファイルの調停と同期 | filesync はコンピュータシステム間でファイルを同期させるために、次の 2 つの処理を行います。

1. filesync は、両方のシステム上にあるパッキング規則ファイルに指定されているディレクトリとファイルを調べ、同じかどうかを判定します。異なるファイルには調整が必要です。

   filesync はまた、監視されるすべてのファイルについて、$HOME/.filesync-base ファイル内のベースライン要約を保守します。このファイルには、最後の調整時におけるファイルの名前、タイプ、およびサイズのリストが入っています。

2. ベースラインファイル内の情報と指定されたオプションに基づいて (「filesync の衝突の解決」を参照)、filesync はどちらのコピーが正しいものかを判別し、正しくない方のシステムに対して必要な変更を加えます。この変更が加えられると、2 つのコピーは再び同じになります (つまり、同期しています)。

   ソースファイルが変更され、宛先ファイルが変更されていない場合、ソースファイルの変更は宛先システムに伝播されます。宛先ファイルが変更され、対応するソースファイルが変更されていない場合、宛先ファイルの変更はソースシステムに伝播されます。両方のシステム上でファイルが変更されたが、それでもまだファイルが同じでない場合、手動で衝突を解決するようにユーザーに促す警告メッセージが出力されます。「filesync の衝突の解決」を参照してください。

filesync の衝突の解決

両方のシステム上のファイルが変更された場合、filesync は、どちらのバージョンを選択するべきかを決定しようとします。どちらのバージョンを選択すればいいのかを自動的に決定できない場合、filesync は警告メッセージを出力し、2 つの互換性のないファイルを調整しないままにします。

この場合、ユーザーが手動で違いを解決しなければなりません。あるいは、どちらのバージョンを選択するかを filesync に指示しなければなりません。衝突を解決する方法を filesync に指示するには、-o オプションと -f オプションを使用します (「オプション」の項を参照)。

あるいは、衝突するファイルごとに、ユーザーが 2 つのバージョンを調べ、どちらを変更せずにおくかを決定し、手動で 2 つのバージョンを同期させます (つまり、所有権や保護モードを、コピー・削除・変更して修正します)。その後、filesync を再実行すると、衝突がまだ残っているかどうかがわかります。

パッキング規則ファイル

パッキング規則ファイル $HOME/.packingrules には、同期すべきファイルのリストが入っています。このファイルの構文については、packingrules(4) のマニュアルページを参照してください。

$HOME/.packingrules ファイルは、ユーザーが filesync を filename 引数付きで呼び出すと自動的に作成されます。filesync のオプションを使用することにより、ユーザーは $HOME/.packingrules ファイル内のパッキング規則を増やすことができます。

パッキング規則ファイルは手動でも作成および編集できます。任意のエディタで $HOME/.packingrules を編集すると、恒久的な変更を $HOME/.packingrules に加えたり、コマンド行からは利用できないより強力なオプション (IGNORE コマンドなど) を使用できます。複雑なワイルドカード式を入力する場合も、$HOME/.packingrules ファイルを編集する方がはるかに簡単です。

ベースラインファイル

$HOME/.filesync-base は 、filesync ベースライン要約ファイルです。filesync は $HOME/.filesync-base 内の情報を使用して、調整処理と同期処理の間にファイル間の差異を識別します。ユーザーはベースラインファイルを作成および編集する必要はありません。ベースラインファイルは filesync により自動的に作成されます。このファイルには、保守されるすべてのファイルについて、最後の調整時における状態が記録されます。

複数の filesync コマンド

しばらくすると、同期させたいファイルのセットが変わってしまうことがあります。たとえば、ノートブックには現在進行中のプロジェクトに関するファイルだけを保存したいと考えることはよくあることです。これまで係ってきたすべてのプロジェクトに関するファイルを同期させようとすると、ノートブックのハードディスクは古いファイルでいっぱいになってしまいます。filesync コマンドを実行するごとに、不必要なファイルの更新に時間を費やしてしまいます。

ユーザーがノートブック上のファイルを削除すると、filesync はサーバー上の対応するファイルも削除しようとします。これは好ましいことではありません。 この場合、 filesync に指示を与え、一部のファイルを同期させないようにする方法が必要になります。そのためには、次の2つの方法を利用します。

1. $HOME/.packingrules を編集します。削除したいファイルの規則を削除します。
2. $HOME/.packingrules を削除します。その後に filesync コマンドを使用して、同期させたいファイルを指定します。

どちらの方法も有効です。使いやすい方を選んでください。変更したいファイルの規則が少ない場合、$HOME/.packingrules を編集する方が簡単です。変更したいファイルの規則が多い場合は、$HOME/.packingrules を削除し、最初から作り直す方が簡単です。

filesync によるファイルセットの同期を停止すれば、サーバー上のファイルを気にすることなく、ノートブック上のファイルを削除できます。

モバイルマシン

filesync を使用してモバイルマシンとサーバー間でファイルを同期させる場合、パッキング規則ファイルとベースラインファイルは、サーバー上ではなく、モバイルマシン上に格納します。ノートブックにログインするときに、HOME 環境変数がノートブック上のディレクトリを正しく指定していない場合、FILESYNC 環境変数を使用すれば、パッキング規則ファイルとベースラインファイルの代替位置を指定できます。

各モバイルマシンは独自のパッキング規則ファイルとベースラインファイルを持っていなければなりません。サーバーにベースラインファイルが格納されており、複数のモバイルマシンがサーバーのベースラインに基づいて調整を試みると、ファイルの同期は正しくなくなります。この場合、モバイルマシンはファイルの状態を正確に記述していないベースラインファイルを使用することになり、結果として間違った調整が行われます。

単一のベースラインファイルを複数のマシン間で共有する際の危険性を回避するために、filesync はデフォルトの規則を新しいパッキング規則ファイルごとに追加します。このデフォルト規則により、パッキング規則ファイルとベースラインファイルがコピーされるのを防ぐことができます。

オプション

次のオプションを指定できます。

-a
ACL (Access Control List, アクセス制御リスト) を検査し、新しいファイルと変更されたファイルのすべてに対して ACL の同期を強制します。特定のファイルで ACL を設定できない場合、filesync はそのファイルに対する ACL 同期を停止します。

ファイルシステムの中には、ACL をサポートしていないものがあります。ACL をサポートしているファイルシステムとサポートしていないファイルシステム間で ACL を同期させることはできません。このようなファイルシステム間で ACL を同期させようとすると、多数のエラーメッセージが生成されます。

-d *dest-dir*
filename をコピーする宛先システム上のディレクトリを指定します。-s *source-dir* オプションと *filename* オペランドと共に使用します。-s オプションおよび「オペランド」の項を参照してください。

-e
すべての違いにフラグを立てます。filesync を root ユーザーとして実行していない限り、保護モードや所有権に関するすべての衝突を完全に解決することは不可能です。通常、ファイルの所有権または保護モードを変更できない場合、filesync は所有権と保護モードの衝突を無視します。ただし、-e (すべての同期) フラグを指定した場合、filesync はこのような違いにもフラグを立てます。

-f src | dst | old | new
-f オプションは、衝突する変更をどのように解決するかを filesync に指示します。両方のシステム上でファイルが変更されているときに、-f オプションを指定すると、filesync は指定されたシステム上の変更をそのまま (有効) にし、指定されなかったシステム上の変更を破棄します。

-f src を指定すると、ソースシステム上のファイルを有効にします。-f dst を指定すると、宛先システム上のファイルを有効にします。-f old を指定すると、古いバージョンのファイルを有効にします。-f new を指定すると、新しいバージョンのファイルを有効にします。

-f オプションと -o オプションを組み合わせるときは、それぞれのsrc と dst が競合しないように指定しなければなりません。-f オプションと -o オプションで競合が発生した場合、-f オプションが無視されます。-o オプションを参照してください。

-h
エラー発生時に停止します。通常、ファイルのコピー時に読み取りエラーまたは書き込みエラーが発生した場合、filesync はエラーメッセージを出力し、別のファイルを調整しようとします。-h オプションを指定した場合、filesync はこのようなエラーが発生するとすぐに停止し、別のファイルの調整は行いません。

-m
ファイルの両方のコピーにおいて変更時間が同じであることを保証します。デフォルトでは、新たにコピーされるファイルの変更時間は調整時の時間に設定されます。ファイルの変更の順番は変更時間の昇順で決定されます。つまり、ファイルが伝播される順番は、ソースファイルの変更時間の相対的な順番と同じです。一般

的に、2 つのシステム間には時間のずれがあり、これらのシステム間で変更時間を転送すると、おかしな結果になることがある、という点に注意してください。

filesync でディレクトリ内のいくつかのファイル (すべてではない) を更新すると、make プログラムが混乱してしまうことがあります。たとえば、filesync が .c ファイルの同期は取るが、.o ファイルを無視する場合、変更された .c ファイルが (変更される前の .c ファイルから生成された) .o ファイルよりも前の変更時間を示すことがあります。

-n
実際の変更は行いません。-n オプションを指定した場合、filesync は、ファイルにどのような変更が加えられており、どのような調整が必要であるかを判別し、その情報を標準出力に表示します。ファイルの変更は行われません (パッキング規則ファイルも含む)。

-n オプションと -o オプションの両方を指定すると、filesync は、-o オプションで指定されたシステムを解析し、そのシステムで加えられた変更を報告します。マシンが接続されていない (つまり、サーバーにアクセスできない) ときにローカルマシン上で加えられた変更を知りたい場合には、-n オプションと -o オプションを組み合わせて使用すると便利です。-o オプションを参照してください。

-o src | dst
-o オプションを指定すると、ソースシステム (src) または宛先システム (dst) のどちらか一方向だけの調整を行います。

-o src を指定すると、filesync はソースシステムから宛先システムへの変更だけを伝播します。宛先システム上で加えられた変更は無視されます。ソースディレクトリまたは宛先ディレクトリにアクセスできない場合、filesync は中断します。

-o dst を指定すると、filesync は宛先システムからソースシステムへの変更だけを伝播します。ソースシステム上で加えられた変更は無視されます。ソースディレクトリまたは宛先ディレクトリにアクセスできない場合、filesync は中断します。

-n オプションと -o オプションの両方を指定すると、filesync は、-o オプションで指定されたシステムを解析し、そのシステム上で加えられた変更を報告します。マシンが接続されていない (つまり、サーバーにアクセスできない) ときにこのローカルマシン上で加えられた変更を知りたい場合には、-n オプションと -o オプションを組み合わせて使用すると便利です。-n オプションを参照してください。

-f オプションと -o オプションを組み合わせるときは、それぞれのsrc と dst が競合しないように指定しなければなりません。-f オプションと -o オプションで競合が発生した場合、-f オプションが無視されます。-f オプションを参照してください。

-q
実行した各再調整アクションを記述する標準のfilesync メッセージを抑制します。

標準の `filesync` メッセージは、UNIX のシェルコマンドの形式 (たとえば、`mv` 、`ln` 、`cp` 、`rm` 、`chmod` 、`chown` 、`chgrp` 、`setfacl` など) で各調整アクションを記述します。

`-r` *directory*
directory だけに調整を制限します。複数のディレクトリを指定するときは、複数の `-r` オプションを指定します。

`-s` *source-dir*
コピーする *filename* が存在しているソースシステム上のディレクトリを指定します。`-d` *dest-dir* オプションと *filename* オペランドと共に使用します。`-d` オプションおよび「オペランド」の項を参照してください。

`-v`
ファイルが比較されるたびに、追加の情報を標準出力に表示します。

`-y`
安全検査の確認応答を省略します。モバイルマシンはドメイン間を移動することがあり、`filesync` が操作するファイルの多くは NFS 経由でアクセスされることが予想されます。したがって、間違ったファイルシステムまたはサーバー上において、ローカルの変更を調整する危険性があります。間違ったファイルシステムまたはサーバー上で調整が行われること、多数のファイルが間違って変更または削除されます。そのような事態を回避するために、`filesync` は調整する前にいくつかの安全検査を実行します。多数のファイルを削除したい場合、あるいは高いレベルのディレクトリの `i` ノード番号を変更した場合、`filesync` は調整する前にユーザーにそのことをプロンプトで確認します。このような変更が行われることをすでに知っており、確認を省略したい場合は、`-y` (yes) オプションを使用して、そのような確認に対して自動的に `-y` (yes) で応答します。

オペランド　次のオペランドを指定できます。

filename
指定したソースディレクトリ (*source-dir*) 内にある、同期すべき通常ファイル、ディレクトリ、シンボリックリンク、または特殊ファイルの名前。複数のファイルを指定するには、各 *filename* をスペースで区切って指定します。*filename* オペランドは `-s` オプションと `-d` オプションと共に使用します。「オプション」の項を参照してください。

filename が通常ファイルの場合、指定した宛先ディレクトリ (*dest-dir*) 内に、その通常ファイルが (同じファイル名で) 複製されます。

filename がディレクトリの場合、指定した宛先ディレクトリ (*dest-dir*) 内に、そのディレクトリおよびそのディレクトリの下のサブディレクトリとファイルが再帰的に複製されます。

filename がシンボリックリンクの場合、指定した宛先ディレクトリ (*dest-dir*) 内に、そのシンボリックリンクのコピーが複製されます。

filename が特殊ファイルの場合、指定した宛先ディレクトリ (*dest-dir*) 内に、同じメジャーデバイス番号とマイナーデバイス番号を持つ特殊ファイルが複製されます。filesync を使用して特殊ファイルを作成できるのはスーパーユーザーだけです。

宛先ディレクトリ (*dest-dir*) に作成されたファイルは、ソースディレクトリ内のファイルと同じアクセス許可 (所有者、グループ、他のユーザーについてのアクセス権) を持ちます。

filename 内にエスケープされたシェルのワイルドカード文字が含まれている場合、そのワイルドカード文字は $HOME/.packingrules に格納され、filesync が実行されるたびに評価されます。

たとえば、次のコマンドは、現在 $RHOME 内にある 2 つの特殊ファイルが $HOME に複製されたことを確認します。$HOME:

```
filesync -s $RHOME  -d $HOME a.c  b.c
```

次の例は、$RHOME 内にある *.c ファイルが (後で作成されなくても) すべて $HOME に複製されたことを確認します。

```
filesync -s $RHOME -d $HOME '*.c'
```

任意の宛先ファイルがすでに存在している場合、filesync は、これらのファイルが同じであることを確認し、同じでない場合は警告を発行します。

ファイルがコピーされた後は、ソースと宛先に区別はありません。 (-o オプションと -f オプションで指定した場合を除く)。

環境

FILESYNC	filesync のパッキング規則ファイルとベースラインファイルのデフォルトの位置を指定します。この環境変数のデフォルト値は $HOME です。パッキング規則ファイルとベースラインファイルの名前には接尾辞 .packingrules と .filesync-base が付加されます。
LC_MESSAGES	診断メッセージと情報メッセージが表示される方法を決定します。C ロケールでは、メッセージはプログラム自身と同じデフォルトの形式 (ほとんどの場合は英語) で表示されます。

終了ステータス

一般に、すべてのファイルがすでに最新のものである場合、あるいは、すべてのファイルが正常に調整されている場合、filesync は 0 の終了ステータスで終了します。ただし、-n オプションが指定された場合、あるいは、エラーが発生した場合は、終了ステータスは次の論理和になります。

0	衝突がなかった。すべてのファイルが最新ものになっている。
1	解決可能な衝突がいくつかある。

2 手動で解決しなければならない衝突がいくつかある。

4 指定したファイルがいくつか存在しない。

8 いくつかのファイルでなアクセス権が不足している。

16 パッキング規則ファイルまたはベースラインファイルにアクセスするときにエラーが発生した。

32 無効な引数

64 指定した src または dst ディレクトリのどちらか (あるいは、その両方) にアクセスできなかった。

128 その他の障害が発生した。

ファイル

$HOME/.packingrules	同期すべきファイルのリスト
$HOME/.filesync-base	ベースライン要約ファイル

属性 次の属性については、 attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 cachefspack(1M), packingrules(4), attributes(5)

名前 | mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式 | **mailp** [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明 | `mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション | 次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D` 出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。

`-F` メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。

`-h` バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。

`-l` 横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。

`-P` *printer* `-d` オプションを指定した場合と同じです。

`-s` *subject* *subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename 読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`1` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

名前 | find – ファイルの検索

形式 | **/usr/bin/find** *path... expression*

/usr/xpg4/bin/find *path... expression*

機能説明 | `find` ユーティリティは、*path* で指定した各パス名に含まれているディレクトリ階層を再帰的に下降し、以下に示す一次子に書き込まれたブール型の式 *expression* と一致するファイルを検索します。

`find` は、ファイル階層をどこまでも深く下降することができます。*path* オペランドのアプリケーションによる指定値の長さが `PATH_MAX` を超えない限り、パスの長さが原因でエラーになることはありません。

オペランド | 以下のオペランドを指定できます。

path ディレクトリ階層構造における、下降を開始する地点のパス名を指定します。

expression 第一引数は – で始まるか、あるいは引数自身が ! または (で、後続の引数は以下に説明する一次子と演算子で構成される式として解釈されます。以下の説明で、*n* が一次引数として使われている場合、*n* は 10 進整数とみなされます。この整数には、次に示すように先頭に正 (+) や負 (–) の符号を付加することも可能です。

+n *n* より大きい

n *n* に等しい

-n *n* より小さい

式 | 有効な式は次のとおりです。

`-atime` *n* ファイルが *n* 日前にアクセスされている場合、真になります。*path* に含まれているディレクトリのアクセス時間は、 `find` 自体によって変更されます。

`-cpio` *device* 常に真です。*device* 上に、現在のファイルを `cpio` フォーマット (5120 バイトレコード) で書き込みます。

`-ctime` *n* ファイルのステータスが *n* 日前に変更されている場合、真になります。

`-depth` 常に真です。ディレクトリ階層を下降して、あるディレクトリにあるすべてのエントリを、そのディレクトリ自体よりも先に検査します。これは、`find` と `cpio`(1) を組み合わせて書き込み権のないディレクトリ内のファイルを転送したい場合に役立ちます。

`-exec` *command* 実行した *command* が終了ステータスとして 0 の値を返す場合、真です。*command* の終わりには、コマンドの終了を示すセミコロン (`;`) を付ける必要があります。`-exec` の最後の引数が { } であり、セミコロン (`;`) ではなくプラス (+) を指定した場合、コマンドが複数回呼び出され、{ } がパス名のグループに置き換えられます。

`-follow` 常に真です。シンボリックリンクをたどり、アクセスしたディレクトリを記憶します。これは、無限ループを検出するためです。無限ループは、シンボリックリンクが親ディレクトリを指している場合などに発生します。この式を `-type l` 式と組み合わせて使用することはできません。

`-fstype` *type* ファイルが属するファイルシステムの形式が *type* の場合、真です。

`-group` *gname* ファイルが *gname* というグループに属している場合、真です。*gname* が数値で、`/etc/group` ファイルや NIS テーブル、NIS+ テーブルに存在しない場合、この数値はグループ ID と見なされます。

`-inum` *n* ファイルが *n* という i ノード番号を持つ場合、真です。

`-links` *n* ファイルにリンクが *n* 個ある場合、真です。

`-local` ファイルシステムが `/etc/dfs/fstypes` で定義されたリモートファイルシステムでない場合に、真です。`/etc/dfs/fstypes` ファイルが存在しない場合、デフォルトのリモートファイルシステムとして `nfs` が使用されます。このオプションは、ローカルでないディレクトリの階層を下降して検索します。ディレクトリ階層を下降しないでローカルファイルを検索する例については、「使用例」の項を参照してください。

`-ls` 常に真です。現在のパス名と、このパスの統計情報を出力します。統計情報には以下の項目が含まれています。

- i ノード番号
- KB (1024 バイト) 単位のサイズ
- 保護モード
- ハードリンクの数
- ユーザー
- グループ
- バイト単位のサイズ
- 更新時刻 ファイルが特殊ファイルの場合、サイズフィールドにはメジャーデバイス番号とマイナーデバイス番号が入ります。

ファイルがシンボリックリンクの場合、'→' の後に、リンクされたファイルのパス名が表示されます。書式は、`ls -gilds` の書式と同じです (`ls`(1B) 参照)。この書式は、`ls` で設定されているのではなく、`find` 内部で設定されています。

`-mount` 常に真です。検索の範囲を、指定されたディレクトリを持つファイルシステムに限定します。他のファイルシステムへのマウントポイントは表示しません。

`-mtime` *n* ファイルのデータが *n* 日前に変更されている場合、真です。

`-name` *pattern* *pattern* が現在のファイル名と一致する場合、真です。通常のシェルファイル名生成文字 (`sh`(1) 参照) を使用できます。*pattern*

内のエスケープ文字としては、バックスラッシュ (\ ) を使用します。find をシェルから起動する場合は、*pattern* をエスケープするか、*pattern* 全体を引用符で囲む必要があります。

/usr/bin/find では、ピリオド文字 (.) を *pattern* の先頭に明示的に指定しない場合、ピリオド文字 (.) で始まる現在のファイル名は *pattern* に一致しません。/usr/xpg4/bin/find では、この区別は行われません。つまり、ワイルドカードファイル名生成文字はピリオド文字 (.) で始まるファイル名に一致します。

-ncpio *device*
常に真です。現在のファイルを *device* に cpio -c フォーマット (5120 バイトレコード) で書き込みます。

-newer *file*
現在のファイルが、*file* 引数で示されているファイルの更新時刻以降に変更されている場合、真です。

-nogroup
ファイルが /etc/group ファイルに記載されているグループに属していないか、NIS テーブル、NIS+ テーブルに記載されていない場合、真です。

-nouser
ファイルが /etc/passwd ファイルに記載されているユーザーのものでないか、NIS テーブル、NIS+ テーブルに記載されていない場合、真です。

-ok *command*
-exec と同様の機能ですが、生成されるコマンド行の先頭には疑問符が出力されます。また、ユーザーがプロンプトに答えて y を入力した場合以外は、コマンド行を実行しません。

-perm [-]*mode*
mode 引数は、ファイルモードビットを表します。書式は、chmod(1) のシンボリックモードオペランド (symbolic_mode_list) と同じです。すべてのファイルモードビットがオフになっているテンプレートを想定して、次の演算子記号（*op*）があればそれを解釈します。

+ テンプレート上で、モードビットをオンにする

– ビットをオフにする

= プロセスのファイルモード生成マスクの内容に関係なく、モードビットをオンにする このうち – 記号は、*mode* 引数の最初の文字として記述することはできません。これは、ハイフンではじまるオプションとの混同を避けるためです。初期値としてすべてのモードビットがオフになっているので、最初の文字として – を指定する必要のあるシンボリックモードはありません。

ハイフンが省略されると、一次子は、ファイルのアクセス権ビットの値が結果のテンプレートの値と等しければ、真と判定します。

mode 引数がハイフンではじまる場合は、結果のテンプレートのビットがすべてファイルのアクセス権ビットで設定されていれば、真と判定します。

`-perm` [`-`]*onum*　ファイルのアクセス権フラグが *onum* に指定した 8 進数字に一致すれば、真になります (`chmod`(1) を参照)。*onum* の先頭がマイナス記号 (–) の場合、ファイルのアクセス権フラグと 比較されるのは、*onum* 中で設定されているビットだけになります。比較の結果、一致すれば、真と判定されます。

`-print`　常に真です。現在のパス名を出力します。

`-prune`　常に真です。一致する ディレクトリ構造内で、*pattern* より下位のディレクトリやファイルは検査しません (「使用例」を参照)。`-depth` が指定されている場合は、`-prune` の指定は無効になります。

`-size` *n*[`c`]　ファイルが *n* ブロック長 (1 ブロックは 512 バイト)である場合、真です。*n* の後に `c` を指定すると、サイズはバイト単位で示されます。

`-type` *c*　ファイルの形式が *c* である場合、真です。ここで *c* は、`b` (ブロック型特殊ファイル)、`c` (文字型特殊ファイル)、`d` (ディレクトリ)、`D` (door)、`f` (プレーンファイル)、`l` (シンボリックリンク)、`p` (FIFO (名前付きパイプ))、`s` (ソケット) のいずれかです。

`-user` *uname*　ファイルの所有者が *uname* に指定したユーザーである場合、真です。*uname* が数値で、`/etc/passwd` ファイルに記載されているログイン名でないか、NIS テーブルまたは NIS+ テーブルに記載されていない場合、この数値はユーザー ID と見なされます。

`-xdev`　`-mount` 一次子と同じ。

`-xattr`　ファイルが拡張属性を持っている場合、真です。

複合式

一次子は、以下の演算子 (優先度の高い順) を使用して組み合わせることができます。

1) (*expression*)　括弧で囲まれた式が真である場合、真 (括弧は、シェルの特殊文字とみなされるため、エスケープする必要がある)

2) `!`*expression*　一次子の否定 (`!` は単項 *not*) 演算子

3) *expression* [`-a`] *expression*　一次子の連結 (*and* 演算子は 2 つの一次子を並置することにより示される)

4) *expression* `-o`*expression*　一次子の代替 (`-o` は *or* 演算子)

`find` を `cpio` と組み合わせて使用する場合、`cpio` に `-L` オプションが指定されているときは、`find` に `-follow` 式を使用する必要があります。`cpio` に `-L` オプションが指定されていない場合は、`find` に `-follow` 式を使用してはなりません。この条件に従わないと、予期しない結果になります。

expression を省略すると、式として -print が使用されます。*expression* を指定し、その式の中に -exec、 -ok、 -print のいずれも含まれていなければ、その式は次に示すものに置き換えられます。

(*given_expression*) -print

ここで、*given_expression* は、 ユーザーが指定した任意の *expression* を指します。-user、-group、-newer の各一次子がそれぞれの引数を評価するのは 1 回だけです。-exec または -ok オプションで指定したコマンドを呼び出しても、同ファイル中の後続の一次子は影響を受けません。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の find の動作については、largefile(5) を参照してください。

使用例

例 1 ディレクトリ階層を出力する

次に示す 2 つのコマンドは、同じ意味を持ちます。

```
example% find .
example% find . -print
```

どちらも、現在のディレクトリ以下の階層構造をすべて出力します。

例 2 ファイルを削除する

次の例は、ホームディレクトリにある a.out または *.o という名前のファイルのうち、1 週間アクセスされなかったものをすべて削除します。

```
example% find $HOME \( -name a.out -o -name ´*.ó \) \  -atime +7 -exec rm {} \;
```

例 3 すべてのファイル名 (SCCS ディレクトリを除く) を出力する

次の例は、現在のディレクトリとそれ以下のディレクトリ内にあるファイル名すべてを再帰的に出力します。ただし、SCCS ディレクトリはスキップします。

```
example% find . -name SCCS -prune -o -print
```

例 4 すべてのファイル名と SCCS ディレクトリを出力する

次の例は、現在のディレクトリとそれ以下のディレクトリ内にあるファイル名すべてを再帰的に出力します。SCCS ディレクトリの内容は出力しませんが、SCCS ディレクトリ名は出力します。

```
example% find . -print -name SCCS -prune
```

例 5 新しいファイルを検査する

次のコマンドは、-nt を指定した test(1) と基本的に同じです。

例 **5** 新しいファイルを検査する　(続き)

```
example$ if [ -n "$(find
file1 -prune -newer file2)" ]; then

printf %s\\n "file1 is newer than file2"
```

例 **6** 24 時間モードを使用してファイルを選択する

-atime、-ctime、-mtime で指定する *n* の最小単位は 24 時間です。 たとえば、ある日の 23:59 にアクセスされたファイルがあり、その 2 分後、つまり翌日の 00:01 に次のコマンドを実行したとします。

```
example% find . -atime -1 print
```

この場合、ファイルは選択の対象となります。日付は変わっていても 24 時間は経過していないためです。午前 0 時が間にはさまっていても、24 時間を単位とする計算には影響はありません。

例 **7** 指定したファイルアクセス権に一致したファイルを出力する

次の例は、ユーザーによる読み取り・書き込み・実行、グループによる読み取り・実行、その他のユーザーによる読み取り・実行が許可されており、その他のアクセス権は許可されていないファイル名すべてを再帰的に出力します。

```
example% find . -perm u=rwx,g=rx,o=rx
```

次のように指定することもできます。

```
example% find . -perm a=rwx,g-w,o-w
```

例 **8** その他のユーザーによる書き込みが許可されているファイルを出力する

次の例は、その他のユーザーによる書き込みが 許可されているファイル名すべてを再帰的に出力します (読み取り、書き込みが許可されているかどうかは関係ありません)。

```
example% find . -perm -o+w
```

例 **9** ローカルファイル (ローカルでないディレクトリ階層は下降しない) を出力する

```
example% find . ! -local -prune -o -print
```

例 **10** 拡張属性を持つ名前空間内のファイルを出力する

```
example% find . -xattr
```

環境

find の実行に影響を与える環境変数 LC_COLLATE、LC_CTYPE、LC_MESSAGES、NLSPATH の詳細については、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	オペランドで指定されたパスはすべて正常に検査された
>0	エラーが発生した

ファイル

/etc/passwd	パスワードファイル
/etc/group	グループファイル
/etc/dfs/fstypes	分散ファイルシステムパッケージを登録したファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み
インタフェースの安定性	安定

関連項目

chmod(1), cpio(1), ls(1B), sh(1), test(1), stat(2), umask(2), attributes(5), environ(5), largefile(5)

警告

次のオプションは旧式で、将来のリリースではサポートされなくなります。

-cpio *device*	常に真です。*device* 上に、現在のファイルを cpio フォーマット (5120 バイトレコード) で書き込みます。
-ncpio *device*	常に真です。*device* 上に、現在のファイルを cpio -c フォーマット (5120 バイトレコード) で書き込みます。

注意事項

find を使用して、一定時間内に変更されたファイルを判別する場合は、-print 引数の前に -time 引数を指定します。そうしないと、すべてのファイル名が出力されます。

Solaris のルートファイルシステム下にあるファイルは、実際には、mntfs や namefs などの仮想ファイルシステムのマウントポイントである場合もあります。ufs ファイルシステムと比較するときに、-mount または -xdev を find 式に指定した場合、このようなファイルは選択されません。

名前 | finger – ローカルユーザーとリモートユーザーに関する情報の表示

形式 | **finger** [-bfhilmpqsw] [*username...*]

finger [-l] [*username@hostname* 1 [*@hostname* 2 .. .*@hostname n...*]]

finger [-l] [*@hostname* 1 [*@hostname* 2 .. .*@hostname n...*]]

機能説明 | 引数を省略した場合、 finger は複数行のフォーマットでログインしている全ユーザーに関して以下の情報を表示します。

- ユーザー名
- 正式な氏名
- 端末名 (書き込み権が拒否されていると先頭に * が付く)
- アイドル時間
- ログイン時刻
- リモートでログインしていたらホスト名

アイドル時間には 3 通りの表記方法があります。 数値 1 つの場合は「分」を表し、2 つの数値が : (コロン) をはさんでいる場合は「時」と「分」を表し、2 つの数値が d をはさんでいる場合は「日」と「時」を表します。

username 引数を 1 つ以上指定すると、それらのユーザーに関するより詳細な情報が、当該ユーザーがログインされているいないに関わらず出力されます。 *username* はローカルユーザー名である必要がありますが、 そのユーザーの姓でも、名でも、アカウント名でもかまいません。 引数の指定により複数行にわたって出力される詳細情報には、 以下の内容が含まれます。

- 当該ユーザー名と正式名
- 当該ユーザーのホームディレクトリとログインシェル
- 現在ログインされていればそのログイン時刻、されていなければ 最後にログインした時刻。さらにログインを行なった端末またはホスト
- 当該ユーザーが最後にメールを受信した時刻、および最後にメールを読んだ時刻
- もしあれば、当該ユーザーのホームディレクトリ内の .project ファイルの最初の行
- もしあれば、当該ユーザーのホームディレクトリ内の .plan の内容

/etc/passwd のコメント (GECOS) フィールドにコンマが組み込まれている場合、finger はそのコンマの後に続く情報を表示しないので、注意してください。

*username@hostname*1[*@hostname*2 . . .*@hostname n*] または *@hostname*1[*@hostname 2 . . .@hostname n*] の形式で引数を指定すると、要求はまず *hostname n* に送られ、それぞれ *hostname n*–1 を通して hostname1 に送られます。本プログラムは finger user information protocol (finger ユーザー情報プロトコル RFC 1288 を参照) を使って、 指定されたユーザー (指定されていれば) またはログインしているユーザーに関する情報を、指定されたリモートホストに照会します。 表示された情報はサーバーによって異なります。

RFC 1288 で規定されるように finger は印刷可能である 7 ビットの ASCII データだけを送ります。この動作は、 システム管理者が /etc/default/finger に PASS オプションを使用することによって修正できます。 PASS=low を指定すると、 10 進数 32 ASCII よりも小さい文字をすべて送ります。 PASS=high を指定すると、10 進数 126 ASCII よりも大きい文字をすべて送ります。PASS=low,high または PASS=high,low を指定すると、 32 ASCII よりも小さい文字と 126 ASCII よりも大きい文字をすべて送ります。

オプション

username@hostname 形式が -l オプションだけで指定できるほかは、 以下のオプションを使用できます。

-b　長形式の出力において、 ユーザーのホームディレクトリとシェルの出力を省略します。

-f　長形式以外の出力において、通常出力されるヘッダーを省略します。

-h　長形式の出力において、 .project ファイルの出力を省略します。

-i　アイドル形式の出力を生成します。短形式との違いは、ログイン名、端末、 ログイン時刻、およびアイドル時間のみが出力されることです。

-l　長形式の出力を生成します。

-m　引数をユーザー名 (姓や名ではない) と見なしてマッチングを行います。

-p　長形式の出力において、 .plan ファイルの出力を省略します。

-q　クィック形式の出力を生成します。短形式との違いは、ログイン名、端末、 およびログイン時刻のみが出力されることです。

-s　短形式の出力を生成します。

-w　短形式の出力において、正式な氏名の出力を省略します。

ファイル

$HOME/. plan	ユーザーの計画
$HOME/. project	ユーザーのプロジェクト
/etc/default/finger	finger オプションのファイル
/etc/passwd	パスワードのファイル
/var/adm/lastlog	最後にログインした時刻
/var/adm/utmp	アカウンティング

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

passwd(1), who(1), whois(1), passwd(4), attributes(5)

Zimmerman, D., *The Finger User Information Protocol*, RFC 1288, Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), Rutgers University, December 1991.

注意事項

本コマンドをリモート形式で実行する場合、使用可能なオプションは `finger user information protocol` (finger ユーザー情報プロトコル) により制限されます。

名前 | fmli – FMLI の呼び出し

形式 | **fmli** [-a *alias_file*] [-c *command_file*] [-i *initialization_file*] *filename...*

機能説明 | fmli コマンドは、FMLI (Form and Menu Language Interpreter) を呼び出し、*filename* 引数で指定されたフレームを開きます。*filename* 引数は初期フレーム定義ファイルのパス名です。filename は、メニューの場合は Menu.*xxx*、フォームの場合は Form.*xxx*、テキストフレームの場合は Text.*xxx* の命名規約に従わなければなりません。各 *xxx* は UNIX システムのファイルの命名規約に準拠する文字列です。FMLI 記述子 lifetime は、fmli に対する引数によって開かれたすべてのフレームについて無視されます。デフォルトでは、このようなフレームの lifetime は immortal です。

オプション | 次のオプションを指定できます。

-a *alias_file*
: -a を指定した場合、*alias_file* は、 alias =*pathname* 形式の行を含むファイルの名前です。このオプションを指定すると、$alias を定義ファイルで使用することにより、パス名が長いオブジェクトやデバイスへの参照を簡略化したり、検索パス (UNIX シェルにおける $PATH に似ている) を定義できます。

-c *command_file*
: -c を指定した場合、*command_file* は、デフォルトの FMLI コマンドを無効にし、新しいアプリケーション固有のコマンドを定義できるファイルの名前です。*command_file* の内容は FMLI コマンドのメニューに反映されます。

-i *initialization_file*
: -i を指定した場合、*initialization_file* は、アプリケーション全体に適用される次のような特性を定義できるファイルの名前です。

- 製品情報を表示する一時的な導入フレーム
- 表示文字と、その位置、その他の要素
- 画面のすべての要素の色属性

使用例 | 例 **1** fmli コマンドの例

fmli を呼び出すには、次のように入力します。

```
example% fmli Menu.start
```

Menu.start は、上述したメニュー定義ファイルの命名規約に従って命名された *filename* です。

fmli を呼び出し、initialization_file を指定するには、次のように入力します。

```
example% fmli -i init.myapp Menu.start
```

init.myapp は initialization_file の例です。

変数 | LOADPFK
: この環境変数を未設定のままにすると、AT&T 5620 や 630 などの端末では、FMLIはファンクションキーを使用するための代替文字シーケンスをダウンロードし、それらを端末のプログラム可能な

ファンクションキーに割り当て、ユーザーがすでにファンクションキーに設定しておいた設定を無効にします。この環境変数を LOADPFK=NO に設定すると、ダウンロードは行われません。

COLUMNS　TERM 環境変数に設定された、端末セット用に定義されている論理画面の幅を変更します。たとえば、132 カラムモードの端末では、次のように FMLI を呼び出すと、より広い画面幅を使用できます。

```
COLUMNS=132 fmli frame-file
```

LINES　TERM 環境変数に設定された、端末セット用に定義されている論理画面の長さ (高さ) を変更します。

ファイル

/usr/bin/fmli

属性

次の属性については、 attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

vsig(1F), attributes(5)

診断

fmli コマンドに *filename* を指定しなかった場合、fmli は次のメッセージを返します。

```
Initial object must be specified.
```

filename が存在しない、あるいは *filename* が読み取れない場合、fmli はエラーメッセージを返して終了します。上記の例のコマンド行では、fmli は次のメッセージを返して終了します。

```
Can't open object "Menu.start"
```

filename が存在しても、*filename* が 3 つの正しいオブジェクト名 (Menu.、 Form.、 Text.) のいずれからも始まらない場合、あるいは、*filename* が正しい名前であっても、そのファイルに適切なデータが入っていない場合、fmli は、ファンクションキー用の画面ラベルを表示した後、次のメッセージを出力して終了します。

```
I do not recognize that kind of object
```

名前 | fmt – 簡単なテキストフォーマッタ

形式 | **fmt** [-cs] [-w *width* | -width] [*inputfile*...]

機能説明 | fmt は簡単なテキストフォーマッタで、行を詰めたり結合することによって、-w *width* オプションで指定した文字数までの出力行を作成します。デフォルトの *width* は 72 カラムです。fmt は引数に指定した *inputfile* を結合します。*inputfile* を指定しなかった場合、fmt は標準入力からのテキストを書式化します。

空白行は、単語間の空白としてそのまま出力されます。fmt は、nroff(1) との互換性のため . (ドット) で始まる行を詰めたり結合することはありません。また、メールヘッダーと判別された (つまり、最初の行が "From" で始まる) 連続する空白以外の行も詰めたり結合することはありません。

インデントはそのまま出力されます。インデントが異なる入力行は結合されません (-c オプションを使用しない場合)。

また、fmt は、vi(1) のインラインテキストフィルタとしても使用できます。

```
!}fmt
```

上記の vi コマンドを入力すると、カーソルの位置から段落の最後までのテキストが書式化されます。

オプション | 次のオプションを指定できます。

-c	クラウンマージンモード。1 つの段落内の最初の 2 行のインデントを保持し、後続行の左マージンを 2 番目の行に揃えます。タグ付き段落に便利です。
-s	行の分割のみ。短い行を連結してより長い行にはしません。これにより、コード例などの書式化されたテキストが不当に連結されるのを防ぐことができます。
-w *width* | -*width*	出力行を *width* カラムの幅にします。

オペランド | *inputfile* 入力ファイル

環境 | fmt の実行に影響を与える環境変数 LC_CTYPE については、environ(5) のマニュアルページを参照してください。

属性 | 次の属性については、 attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | nroff(1), vi(1), attributes(5), environ(5)

注意事項 | *-width* オプションは BSD 互換性のために提供されていますが、将来のリリースでは廃止される可能性があります。

名前 | fmtmsg – stderr またはシステムコンソールへのメッセージ表示

形式 | **fmtmsg** [-c *class*] [-u *subclass*] [-l *label*] [-s *severity*] [-t *tag*] [-a *action*] *text*

機能説明 | fmtmsg は、メッセージの分類構成要素に基づいて、書式付きメッセージを stderr またはコンソールに書き出します。

書式付きメッセージは、 標準構成要素で構成されます (「環境」の項の環境変数 MSGVERB を参照)。このうち分類構成要素とサブクラス構成要素の 2 つは、標準メッセージの一部としては表示されません。 これらはメッセージの発行元を定義し、書式付きメッセージの表示先を指定するために使われます。

オプション | -c *class* メッセージ発行の原因となった状態の種類を表します。 *class* としては以下のキーワードが指定できます。

hard	状態の発生元はハードウェア
soft	状態の発生元はソフトウェア
firm	状態の発生元はファームウェア

-u *subclass* メッセージをさらに定義しメッセージの表示先を示すための、一連のキーワードを *subclass* に指定します。 キーワード間はコンマで区切ります。以下のキーワードが指定可能です。

appl	状態の発生はアプリケーション中に起因している。このキーワードは util および opsys と同時には指定できない。
util	状態の発生はユーティリティ中に起因している。このキーワードは appl および opsys と同時には指定できない。
opsys	状態の発生はカーネル中に起因している。このキーワードは appl および util と同時には指定できない。
recov	アプリケーションをその状態から回復させる。このキーワードは nrecov と同時には指定できない。
nrecov	アプリケーションをその状態から回復させない。このキーワードは recov と同時には指定できない。
print	標準エラーストリーム stderr にメッセージを書き出す。
console	システムコンソールにメッセージを書き出す。print と console を両方指定することも可能。

-l *label* メッセージの発行元を指定します。

`-s` *severity*
: エラーの重大度を指定します。標準レベルの重大度として *severity* に指定できるキーワードとその意味を以下に説明します。

  `halt`
  : アプリケーションが重大なエラーを検出し、処理を中断している。

  `error`
  : アプリケーションがエラーを検出した。

  `warn`
  : アプリケーションが、通常は起こらない状態を検出した。エラーの可能性がある。

  `info`
  : アプリケーションは、エラーではない単なる情報を通知している。

`-t` *tag*
: メッセージの識別子を含んだ文字列を指定します。

`-a` *action*
: エラー回復処理の第 1 ステップを述べた文字列を指定します。*action* 引数は、それ全体が 1 つの文字列であると正しく解釈されるように 記述しなければなりません。 `fmtmsg` は、この *action* で指定した文字列の前に `TO FIX:` を付加します。

text
: 状態を表すテキスト文字列を指定します。*text* 引数は、それ全体が 1 つの文字列であると正しく解釈されるように 記述しなければなりません。

使用例

例 1 標準メッセージ形式

次に示す `fmtmsg` は、標準メッセージ形式で完全なメッセージを生成し、それを標準エラーストリームに書き出す例です。

```
example% fmtmsg -c soft -u recov,print,appl -l UX:cat \
     -s error -t UX:cat:001 -a "refer to manual" "invalid syntax"
```

これを実行すると次の出力が得られます。

```
UX:cat: ERROR: invalid syntax
TO FIX: refer to manual   UX:cat:138
```

例 2 MSGVERB を使用する

環境変数 `MSGVERB` が次のように設定されているとします。

```
MSGVERB=severity:text:action
```

このとき上記例 1 の `fmtmsg` コマンドを実行すると、以下の出力が得られます。

```
ERROR: invalid syntax
TO FIX: refer to manual
```

例 3 SEV_LEVEL を使用する

環境変数 `SEV_LEVEL` が次のように設定されているとします。

```
SEV_LEVEL=note,5,NOTE
```

例 3 SEV_LEVEL を使用する　(続き)

このとき次の fmtmsg コマンドを実行します。

```
example% fmtmsg -c soft -u print -l UX:cat -s note -a "refer
to manual" "invalid syntax"
```

以下の出力が得られます。

```
NOTE: invalid syntax
TO FIX: refer to manual
```

また、stderr にメッセージが出力されます。

環境

fmtmsg の動作は、 2 つの環境変数 MSGVERB と SEV_LEVEL により制御されています。MSGVERB はシステム管理者によって、システム用の /etc/profile 中に設定されています。ユーザーは、このシステムに設定された値以外の MSGVERB 値を使用することができます。その方法としては、ユーザーの .profile ファイル中にある MSGVERB をリセットする、現在のシェルセッション内で値を変更する、の 2 通りがあります。SEV_LEVEL はシェルスクリプト中で使用できます。

メッセージを stderr に出力するとき、どの構成要素を選択すべきかを MSGVERB が fmtmsg に伝えます。MSGVERB の値は一連のキーワードで、キーワード間はコロンで区切ります。MSGVERB は次に示す形式で設定できます。

```
MSGVERB=[keyword[:keyword[:...]]]
export MSGVERB
```

keywords としては、 label 、 severity 、 text 、 action 、 tag が指定できます。MSGVERB が構成要素用のキーワードを含んでいて、その構成要素の値が NULL 文字列ではない場合、 fmtmsg はメッセージを stderr に書き出す際にその構成要素をメッセージ中に挿入します。MSGVERB が構成要素用のキーワードを含んでいなければ、 その構成要素はメッセージ表示には含まれません。上記のキーワードは任意の順序で指定できます。MSGVERB が定義されていない、値が NULL 文字列である、値の形式が正しくない、上記のもの以外のキーワードを含んでいる、といった場合には、fmtmsg はすべての構成要素を選択します。

MSGVERB は、どの構成要素を選択するかを表しますが、 これは表示用のメッセージに関してだけです。コンソールへ出力されるメッセージには、常にすべての構成要素が含まれます。

SEV_LEVEL は重大度レベルを定義し、fmtmsg で使用できるように印刷文字列をそれらのレベルに対応させます。なお以下に示す標準重大度レベルは、変更することはできません。これ以外のレベルは、定義したり、再定義したり、削除したりすることが可能です。

0　(対応するものなし)

1　HALT

2 ERROR

3 WARNING

4 INFO

SEV_LEVEL は次の形式で設定できます。

description は 3 つのフィールドで構成され、フィールド間は コンマで区切ります。

SEV_LEVEL= [*description*[:*description*[:...]]]
export SEV_LEVEL

description=*severity_keyword*,*level*,*printstring*

最初のフィールド *severity_keyword* は、 fmtmsg の -s *severity* オプションで指定できるキーワードの いずれかと同じ文字列です。

次のフィールド *level* は、評価の結果が正の整数となる文字列です。ただしその整数値は、標準重大度レベルとして予約されている 0 、 1 、 2 、 3 、または 4 であってはなりません。キーワード *severity_keyword* が指定されていると、*level* の値は重大度の値として fmtmsg(3C) に渡されます。

3 番目のフィールド *printstring* は、前述の重大度値 *level* が用いられる際に fmtmsg が標準メッセージ形式で使用する文字列です。

SEV_LEVEL が定義されていない場合、またはその値が NULL 文字列の場合には、デフォルトの重大度レベル値だけが使用可能です。コロンで区切られた一連の *description* の中に、その内容がコンマで区切られた 3 つのフィールドで構成されていないものがあるとき、または第 2 フィールドの値が正の整数に評価されないものがあるとき、その *description* は無視されます。

終了ステータス

以下の終了ステータスが返されます。

0 指定された機能はすべて正常に実行された

1 コマンド中に、構文の誤り、不正なオプション、 またはオプションに対する不正な引数が検出された

2 機能は部分的に正常終了したが、メッセージは stderr に出力されなかった

4 機能は部分的に正常終了したが、メッセージは システムコンソールに出力されなかった

32 要求された機能は 1 つも正常に終了しなかった

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 addseverity(3C), fmtmsg(3C), attributes(5)

名前 | fnattr – FNS 名前付きオブジェクトに関連する属性の更新と検査

形式 | **fnattr** [-AL] *composite_name* [[-O | -U] *identifier*...]

fnattr [-L] *composite_name* [{-a [-s] [-O | -U] *identifier* [*value*...]} | {-d [[-O | -U] *identifier* [*value*...]]} | {-m [-O | -U] *identifier old_value new_value*}...]

機能説明 | fnattr コマンドは、FNS 名前付きオブジェクトに関連する属性を更新および検査します。このコマンドには 4 つの使用法があります。属性または値の追加、属性または値の削除、属性値の変更、および属性の内容リストの表示です。

オプション | 属性とその値の追加、変更、および削除用のオプションは、同じコマンド行内で結合できます。変更は指定された順番で実行されます。

コマンド行に複数の変更を指定したときに、そのうちのいくつかの変更が失敗した場合、以降の変更は中断されます。失敗の前に実行された変更は有効です。失敗した変更は fnattr の出力として表示されます。

-a　*composite_name* で指定された名前付きオブジェクトに関連する属性を追加するか、その属性に値を追加します。*identifier* は操作する属性の識別子で、その形式は FN_ID_STRING です (-O オプションまたは U オプションを指定しない場合)。*value* . . は、追加する属性値を表します。*value* を格納するために使用される属性構文は fn_attr_syntax_ascii です。

-A　権限を持つ情報源に問い合わせて、属性情報を取得します。

-d　*composite_name* で指定された名前付きオブジェクトに関連する属性を削除します。*identifier* を指定しない場合、指定した名前付きオブジェクトに関連するすべての属性が削除されます。値 (*value* . . .) を付けずに*identifier* を指定した場合、*identifier* で指定した属性全体が削除されます。個々の属性値 (*value* . . .) を指定した場合、指定した属性値だけが属性から削除されます。属性の最後の値を削除すると、属性を削除したことになります。*identifier* の形式は FN_ID_STRING です (-O オプションまたは -U オプションを指定しない場合)。

-L　*composite_name* が XFN リンクにバインドされている場合、そのリンクが指すオブジェクトに関連する属性を操作します。-L を使用しない場合、XFN リンクに関連する属性が操作されます。

-m　*composite_name* で指定された名前付きオブジェクトに関連する、*identifier* で指定された属性の値を変更します。指定した属性の *old_value* が *new_value* に変更されます。*composite_name* に関連する別の属性や値には影響しません。*identifier* の形式は FN_ID_STRING です (-O オプションまたは -U オプションを指定しない場合)。

-O　*identifier* の形式は FN_ID_ISO_OID_STRING (ASN.1 のドットで区切られた整数リスト文字列) です。

`-s` 優先モードで追加します。*identifier* と同じ識別子を持つ属性がすでに存在する場合、その属性の値をすべて削除し、*value* に変更します。このオプションを省略した場合、指定した属性の値は、既存の値と *value* を結合したものになります。

`-U` *identifier* の形式は `FN_ID_DCE_UUID` (文字列形式の DCE UUID) です。

オペランド

次のオペランドを指定できます。

composite_name FNS 名前付きオブジェクト

追加

`-a` オプションは属性と値を追加します。次のコマンドは、`user/jane` の属性 `shoesize` の値を値 `7.5` に変更します。

```
eg% fnattr user/jane -as shoesize 7.5
```

次のコマンドは、`user/jane` の属性 `project` に値 `Chameleo` を追加します。

```
eg% fnattr user/jsmith -a project Chameleo
```

削除

`-d` オプションは属性と値を削除します。次のコマンドは、`user/jane` に関連するすべての属性を削除します。

```
eg% fnattr user/jane -d
```

次のコマンドは、`user/jane` に関連する属性 `shoesize` を削除します。

```
eg% fnattr user/jane -d shoesize
```

次のコマンドは、`user/jane` に関連する属性 `projects` から属性値 `old_project` を削除します。

```
eg% fnattr user/jane -d projects old_project
```

変更

`-m` オプションは属性値を変更します。次のコマンドは、`user/jsmith` に関連する属性 `projects` の値 `Chameleo` を `Dungeon` に変更します。

```
eg% fnattr user/jsmith -m projects Chameleo Dungeon
```

次のコマンドは、変更が失敗する例です。このコマンドを実行するユーザーは `user/jane` の属性を更新する権限を持っていませんが、新しい属性を追加することは許可されています。次のコマンドを実行すると属性 `hatsize` は追加されますが、`-d shoesize` が失敗してコマンドが中断されるため、属性 `shoesize` は削除されず、属性 `dresssize` は変更されません。

```
eg% fnattr user/jane -a hatsize medium -d shoesize -m dresssize 5 6
```

リスト

属性とその値のリストを表示するには、オプションを指定しません。次のコマンドは、user/jane に関連するすべての属性のリストを表示します。

```
eg% fnattr user/jane
```

次のコマンドは、user/jane の属性 project の値のリストを表示します。

```
eg% fnattr user/jane project
```

次のコマンドは、user/jane の属性 project と shoesize の値のリストを一覧表示します。

```
eg% fnattr user/jane project shoesize
```

終了ステータス

次の終了ステータスが返されます

0　　操作が正常終了した

1　　操作が失敗した

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfns

関連項目

fnlookup(1), attributes(5), fns(5)

注意事項

組み込み属性 (ユーザーの onc_unix_passwd など) は fnattr コマンドでは更新できません。組み込み属性の内容は、基盤となっているネームサービス (NIS+ や NIS など) の更新により影響を受けます。

名前 | fnbind – FNS 名への参照のバインド

形式 | **fnbind** [-s] [-v] [-L] *name new_name*

fnbind -r [-s] [-v] *new_name* [-O | -U] *ref_type* { [-O | -U] *addr_type* [-c | -x] *addr_contents* }...

機能説明 | fnbind ユーティリティは、*name* で指定された参照を名前 *new_name* にバインドします。2 番目の (-r オプションを使用する) 形式の fnbind は、コマンド行に指定された引数を使用して構築した参照を名前 *new_name* にバインドします。

オプション | 次のオプションを指定できます。

-s　すでにバインドされている場合でも、*new_name* に参照をバインドします。このオプションを省略した場合、*new_name* がすでにバインドされていれば、fnbind は失敗します。

-v　*new_name* にバインドされている参照を表示します。

-L　*name* を使用して XFN リンクを作成し、このリンクを *new_name* にバインドします。

-r　*ref_type* を参照タイプとし、1 つまたは複数の *addr_type* と *addr_contents* のペアをアドレスの参照リストとして参照を作成し、その参照を *new_name* にバインドします。-O オプションまたは -U オプションを指定しない限り、FN_ID_STRING が *ref_type* と *addr_type* の識別子の形式として使用されます。-c オプションまたは -x オプションを指定しない限り、*addr_contents* は XDR 暗号化文字列として格納されます。

-c　指定された形式で *addr_contents* を格納します。XDR 暗号化を使用してはいけません。

-x　*addr_contents* は 16 進数文字列を指定します。この文字列を 16 進数表記に変換し、格納します。つまり、XDR 暗号化は使用してはいけません。

-O　識別子の形式は FN_ID_ISO_OID_STRING (ASN.1 のドットで区切られた整数リスト文字列) です。

-U　識別子の形式は FN_ID_DCE_UUID (文字列形式の DCE UUID) です。

使用例 | 例 **1** プリンタへのサービスのバインド

次のコマンドは、名前 thisorgunit/service/pr を thisorgunit/service/printer で指定された参照にバインドします。thisorgunit/service/pr にバインドされていた参照は変更されます。

```
example% fnbind -s thisorgunit/service/printer thisorgunit/service/pr
```

例 **2** XFN リンクへのバインド

次のコマンドは、名前 thisorgunit/service/pr を、名前 thisorgunit/service/printer を使用して構築された XNF リンクにバインドします。

例 2 XFN リンクへのバインド　(続き)

```
example% fnbind -L thisorgunit/service/printer thisorgunit/service/pr
```

例 3 アドレスタイプへのバインド

次のコマンドは、名前 thisorgunit/service/calendar を、参照タイプが SUNW_cal、アドレスタイプが SUNW_cal_deskset_onc、アドレス内容が staff@exodus の参照にバインドします。

```
example% fnbind -r thisorgunit/service/calendar SUNW_cal \
SUNW_cal_deskset_onc staff@exodus
```

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfns

関連項目

fnlookup(1), fnrename(1), fnunbind(1), FN_identifier_t(3XFN), xdr(3NSL), attributes (5), fns(5), xfn_links(3XFN)

名前 | fnlist – FNS コンテキストにバインドされている名前と参照の表示

形式 | **fnlist** [-Alv] [*composite_name*]

機能説明 | fnlist は、*composite_name* のコンテキストにバインドされている名前と参照を表示します。

composite_name を指定しない場合、デフォルトの初期コンテキストが表示されます。

オプション | 次のオプションを指定できます。

-A　権限を持つ情報源に問い合わせて、情報を取得します。

-l　*composite_name* のコンテキストにバインドされている名前と共に、その参照を表示します。このオプションを指定しない場合、名前だけが表示されます。

-v　参照を詳細に表示します。onc_fn_* 参照の場合、このオプションは、*composite_name* のコンテキストにバインドされているすべての名前の参照を格納している NIS+ テーブルの名前を導き出すときに便利です。

オペランド | 次のオペランドを指定できます。

composite_name　FNS 名前付きオブジェクト。*composite_name* は、UNIX のファイル名と同様に、作成される下位コンテキストに依存します。次に、有効な *composite_name* オペランドを使用したコマンドの例を示します。

```
eg% fnlist thisorgunit
eg% fnlist thisorgunit/service
eg% fnlist thisorgunit/service/printer
```

FNS が展開された場合、*composite_name* は展開されているサイトに固有です。

使用例 | **例 1** fnlist コマンドの例

次の例では、コマンドにオペランドが指定されていないため、初期コンテキストの参照タイプとアドレスタイプのリストが表示されます。

```
eg% fnlist -l
```

次の例では、ユーザーコンテキストが指定されているため (つまり、*composite_name* = user/) 、まず、FNS が fncreate(1M) 経由で、NIS、NIS+、または files のいずれかのネームサービスを使用して展開されていなければなりません。FNS が展開されていない場合、ユーザーコンテキストが存在しないため、このコマンドはエラーメッセージ "Name not found" を表示して失敗します。

次のコマンドは、user/ のコンテキストにバインドされている名前を表示します。

```
eg% fnlist user/
```

例 1 fnlist コマンドの例　(続き)

次のコマンドは、user/ のコンテキストにバインドされている名前と参照を表示します。

```
eg% fnlist -l user/
```

終了ステータス

次の終了ステータスが返されます。

0　　操作が正常終了した

1　　操作が失敗した

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfns

関連項目

fnbind(1), fnlookup(1), fnunbind(1), fncreate(1M), fndestroy(1M), attributes(5), fns(5), fns_references(5)

fnlookup(1)

名前 | fnlookup – NFS 名にバインドされている参照の表示

形式 | **fnlookup** [-ALv] *composite_name*

機能説明 | fnlookup は、*composite_name* のバインドを表示します。

オプション | 次のオプションを指定できます。

-A　権限を持つ情報源に問い合わせて、情報を取得します。

-L　*composite_name* が XFN リンクにバインドされている場合、そのリンクがバインドされている参照を表示します。-L オプションを指定しない場合、fnlookup は XFN リンクを表示します。

-v　参照を詳細に表示します。onc_fn_* 参照の場合、このオプションは、*composite_name* の参照と (もしあれば) その参照の文字列表現を格納している NIS+ テーブルの名前を導き出すときに便利です。

オペランド | 次のオペランドを指定できます。

composite_name　FNS 名前付きオブジェクト

使用例 | 例 **1** fnlookup コマンドの例

次のコマンドは、名前 user/jsmith/service/calendar (ユーザー jsmith のカレンダーを参照する) がバインドされている参照を表示します。

```
eg% fnlookup user/jsmith/service/calendar
```

次のコマンドは、名前 user/jsmith/service (ユーザー jsmith のサービスコンテキストを参照する) がバインドされている参照を表示します。

```
eg% fnlookup user/jsmith/service
```

このリンクが XFN リンクにバインドされている場合、次のコマンドは、このリンクがバインドされている参照を表示します。

```
eg% fnlookup -L user/jsmith/service
```

終了ステータス | 次の終了ステータスが返されます。

0　操作が正常終了した

1　操作が失敗した

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfns

関連項目 | fnbind(1), fnlist(1), fnunbind(1), fncreate(1M), fndestroy(1M), xfn_links(3XFN), attributes(5), fns(5), fns_references(5)

名前 | fnrename – FNS 名のバインド名の変更

形式 | **fnrename** [-s] [-v] *context_name old_atomic_name new_atomic_name*

機能説明 | fnrename は、*context_name* のコンテキストにおいて、*old_atomic_name* のバインド名を *new_atomic_name* に変更します。*old_atomic_name* と *new_atomic_name* は両方とも原子名でなければならす、*context_name* で指定されたコンテキストにおいて解決されなければなりません。

オプション | 次のオプションを指定できます。

-s　*new_atomic_name* にすでにバインドされている参照を変更します。このオプションを省略した場合、 *new_atomic_name* がすでにバインドされていればfnrename は失敗します。

-v　変更されるバインド名を表示します。

使用例 | 例 **1** fnrename コマンドの例

次のコマンドは、user/jsmith/service/ で指定されたコンテキストにおいて、clendar にバインドされた参照に calendar をバインドし、clendar をバインド解除します。

```
eg% fnrename user/jsmith/service/ clendar calendar
```

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfns

関連項目 | fnbind(1), fnlist(1), fnunbind(1), fncreate(1M), fndestroy(1M), xfn_links(3XFN), attributes(5), fns(5), fns_references(5)

名前 | fnsearch – 指定した属性による FNS オブジェクトの検索

形式 | **fnsearch** [-AlLv] [-n *max*] [-s *scope*] *composite_name* [-a *ident*]... [-O | -U]*filter_expr* [*filter_arg*]...

機能説明 | fnsearch コマンドは、指定したフィルタ式 (*filter_expr*) を満たす属性を持つ *composite_name* またはその下位のコンテキストにバインドされているオブジェクトの名前を (オプションで指定されていなければ属性と参照も) 表示します。フィルタ式はオブジェクトの属性と参照の識別子と値を含む論理式で、検索時にテストされます。

FNS の概要については、 fns(5) のマニュアルページを参照してください。

オプション | 次のオプションを指定できます。

-a *ident* — フィルタ式を満たす各オブジェクトの属性を表示します。-a オプションを指定しない場合、すべての属性が表示されます。空の *ident* (シェルから見ると " ") を指定すると、属性は一切表示されません。-a オプションは複数指定できます。*ident* の完全な構文については、「選択した属性の表示」を参照してください。

-A — 権限を持つ情報源に問い合わせて、情報を取得します。

-l — フィルタ式を満たす各オブジェクトの参照を表示します。

-L — 検索時に、XFN リンクをたどります。

-n *max* — 表示するオブジェクトの最大数を 指定した数 (正の整数) に制限します。デフォルトは無制限です。

-s *scope* — 検索の適用範囲を設定します。*scope* は次のいずれかです。

- object — オブジェクト *composite_name* だけを検索します。
- context — *composite_name* に直接バインドされているオブジェクトを検索します。
- subtree — *composite_name* またはその下位コンテキストにバインドされているオブジェクトを検索します。
- constrained_subtree — subtree と似ていますが、コンテキストの実装で定義されている下位コンテキストだけを検索します。

scope には o や cont などの省略形を使用できます。このオプションを指定しない場合、デフォルトの動作は -s context です。

`-v` フィルタ式を満たす各オブジェクトの参照を詳細に表示します。このオプションは -l よりも優先されます。

オペランド

次のオペランドを指定できます。

composite_name FNS 名前付きオブジェクト

簡単なフィルタ式

最も簡単な形式のフィルタ式は、属性の存在をテストするものです。このフィルタ式には属性の名前を指定するだけです。次のコマンドは、for_sale という属性を持つオブジェクトを検索します。

```
% fnsearch composite_name for_sale
```

もう 1 つの簡単なフィルタ式は、属性の値をテストするものです。次のコマンドは、age が 17 未満 (つまり、17 歳未満) のオブジェクトを検索します。

```
% fnsearch composite_name "age < 17"
```

文字列値を指定するときは、文字列を単一引用符で囲みます。次のコマンドは、color が red (つまり、色が赤) のオブジェクトをすべて検索します。

```
% fnsearch composite_name "color == 'red'"
```

この例の二重引用符 (") はフィルタ式の一部ではないことに注意してください。この二重引用符は、シェルが、空白文字や単一引用符を式の一部として解釈しないようにします。

論理演算子

論理演算子 and、or、および not を使用すると、簡潔なフィルタ式を指定できます。次に例を示します。

```
% fnsearch composite_name "age >= 35 and us_citizen"
```

括弧を使用すると、複数のフィルタ式をグループにまとめることができます。

```
% fnsearch composite_name "not (make == 'olds' and year == 1973)"
```

演算子の優先順位は昇順です。

```
or
と
not
関係演算子 （以下の「関係演算子」を参照）
```

論理演算子 and と or は左側への結合が優先されます。

関係演算子

次に、属性と指定された値を比較するための関係演算子を示します。

`==` 属性値の少なくとも 1 つが指定された値と同じ場合、真になります。

`!=` いずれの属性値も指定された値と同じでない場合、真になります。

`<` 属性値の少なくとも 1 つが指定された値よりも小さい場合、真になります。

`<=` 属性値の少なくとも 1 つが指定された値以下である場合、真になります。

`>` 属性値の少なくとも 1 つが指定された値よりも大きい場合、真になります。

`>=` 属性値の少なくとも 1 つが指定された値以上である場合、真になります。

`~=` コンテキスト固有の近似一致基準に従って比較を行い、属性値の少なくとも 1 つが指定された値に一致する場合、真になります。

比較と順番は、テストされる属性の構文または規則に固有です。

選択した属性の表示

デフォルトでは、`fnsearch` コマンドは、検索基準を満たす各オブジェクトの名前とすべての属性を表示します。表示される属性のリストを制限するには、`-a` コマンド行オプションを使用します。次の例では、オブジェクト `small` の属性 `color` と `shape` だけが表示されます。

```
% fnsearch composite_name -a color -a shape "size == 'small'"
```

属性識別子の形式は、デフォルトで `FN_ID_STRING` (ASCII 文字列) です。OSI OID または DCE UUID の属性識別子を指定するには、それぞれ、属性名の前に `-O` または `-U` の接頭辞を付けます。

`-O` 識別子の形式は `FN_ID_ISO_OID_STRING` (ASN.1 のドットで区切られた整数リスト文字列) です。

`-U` 識別子の形式は `FN_ID_DCE_UUID` (文字列形式の DCE UUID) です。

次に例を示します。

```
% fnsearch composite_name -a -O 2.5.4.0 "shoe_size < 9"
```

もう一つ例を示します。

```
% fnsearch composite_name -a -U 0006a446-5e97-105f-9828-8190285baa77 \
"bowling_avg > 200"
```

フィルタ引数

フィルタ式の一部は、置換トークン (パーセント記号 (`%`) に単一の文字が続いたもの) で置き換えることができます。置換トークンの値は、`printf`(1) の場合と同様に、フィルタ式に続くフィルタ引数で指定します。次に、利用できる置換トークンを示します。

`%a` 属性

`%s` 文字列

`%i` 識別子

`%v` 属性値 (現時点で使用できる構文は `fn_attr_syntax_ascii` だけです。)

たとえば、次のコマンドを見てください。

```
% fnsearch composite_name "color == 'red'"
```

上記のコマンドは、次のように記述することができます。

```
% fnsearch composite_name "%a == 'red'" color
```

あるいは、次のようにも記述できます。

```
% fnsearch composite_name "%a == %s" color red
```

置換トークンは、フィルタ引数の値が実行時に計算されるようなシェルスクリプトを書くときに使用すると便利です。

デフォルトでは、属性 (上記属性 color など) の識別子の形式は FN_ID_STRING (ASCII 文字列) です。置換トークンでは OSI OID と DCE UUID も使用できます。前述のコマンド行オプションと同様に、それぞれ、フィルタ引数の前に -O または -U の接頭辞を付けます。

-O　識別子の形式は FN_ID_ISO_OID_STRING (ASN.1 のドットで区切られた整数リスト文字列) です。

-U　識別子の形式は FN_ID_DCE_UUID (文字列形式の DCE UUID) です。

次に例を示します。

```
% fnsearch composite_name "%a -O 2.5.4.0
```

もう一つの例を示します。

```
% fnsearch composite_name "%a" =='red'" \
 -U 0006a446-5e97-105f-9828-8190285baa77
```

ワイルドカード付き文字列

ワイルドカード付き文字列は、ワイルドカード指示子と文字列の並びから構成されます。ワイルドカード指示子はアスタリスク (*) で示し、任意の文字が 0 回または複数回発生することを意味します。

ワイルドカード付き文字列は、部分文字列の照合を指定するときに使用します。次に、ワイルドカード付き文字列とその意味を説明します。

*	任意の文字列。
'tom'	文字列 tom。
'harv'*	harv で始まる任意の文字列。
*'ing'	ing で終わる任意の文字列。
'a'*'b'	a で始まり、b で終わる任意の文字列。
'jo'*'ph'*'ne'*'er'	jo で始まり、その後に ph という部分文字列を含み、さらにその後に ne という部分文字列を含み、er で終わる任意の文字列。
%s*	フィルタ引数として指定された文字列で始まる任意の文字列。
'bix'*%s	bix で始まり、フィルタ引数として指定された文字列で終わる任意の文字列。

拡張演算

拡張演算子は TRUE または FALSE を返す述語 (機能) で、フィルタ式内で他の演算子と自由に組み合わせることができます。

拡張演算を指定するには、単一引用符で囲んだ文字列 (演算名) と、その後に括弧で囲んだ引数を指定します。次に、現在定義されている 3 つの拡張演算を示します。

'name'(ワイルドカード付き文字列)	オブジェクトの名前が指定されたワイルドカード付き文字列に一致する場合、TRUE です。
'reftype'(識別子)	オブジェクトの参照タイプが指定された識別子と等しい場合、TRUE です。
'addrtype'(識別子)	オブジェクトの参照内の任意のアドレスタイプが指定された識別子と等しい場合、TRUE です。

次のコマンドは、名前が bill で始まり、80 を超える属性 IQ を持つオブジェクトを検索します。

```
% fnsearch composite_name "'name'('bill'*) and IQ > 80"
```

フィルタ式の文法

次に、フィルタ式の完全な文法を示します。これは、XFN 仕様で定義された文法に基づいています (FN_search_filter_t(3XFN) を参照)。

この文法では、文字数リテラルは二重引用符で囲みます。引用符自身はフィルタ式の一部ではありません。中括弧はグループ化に使用します。角括弧は任意の要素を示します。引用符で囲まれていないアスタリスク (*) は、直前の要素が 0 回または複数回発生することを示します。プラス記号 (+) は、直前の要素が 1 回または複数回発生することを示します。

FilterExpr : : = [*Expr*]

Expr : : =

Expr "or" *Expr* | *Expr* "and" *Expr* | "not" *Expr*
| "(" *Expr* ")"
| *Attribute* [*RelOp Value*]
| *Ext*

RelOp : : = "==" | "!=" | "<" | "<=" | ">" | ">=" | "~="

Attribute : : =

*Char** | "%a"

Value : : =

Integer | *WildcardedString* | "%v"

WildcardedString : : =

"*" | *String* | {*String* "*"}+ [*String*]
| {"*" *String*}+ ["*"]
(that is, an alternating sequence of *String* and "*")

String : : =

"'" *Char** "'"
| "%s"

Ext : : =

"'name'(" *WildcardedString* ")"
| "'reftype'(" *Identifier* ")"
| "'addrtype'(" *Identifier* ")"

Identifier : : =

"'" *Char** "'"
| "%i"

Char : : =

an element of the Portable Character Set (ASCII)
| a character in the repertoire
of a string representation

終了ステータス

次の終了ステータスが返されます。

`0` 操作が正常に終了した

`1` 操作が失敗した

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfns

関連項目

`printf`(1), `FN_search_control_t`(3XFN), `FN_search_filter_t`(3XFN), `fn_attr_ext_search`(3XFN), `fn_attr_search`(3XFN), `attributes`(5), `fns`(5)

注意事項

空のフィルタ式は、`TRUE` に評価されます (すべてのオブジェクトを満たします)。

フィルタ式の任意の部分式にある識別子がオブジェクトの属性として存在しない場合、その識別子を含む最も内側にある論理式が `FALSE` に評価されます。

名前 | fnunbind – FNS 名からの参照のバインド解除

形式 | **fnunbind** *composite_name*

機能説明 | fnunbind は、*composite_name* の参照をバインド解除します。

たとえば、次のコマンドは名前 user/jsmith/fs/ にバインドされている参照をバインド解除します。

```
eg% fnunbind user/jsmith/fs/
```

fnunbind はコンテキスト名をバインド解除できません。コンテキストをバインド解除するには、まず、そのコンテキストをコマンド fndestroy で破壊しておかなければなりません。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWfns

関連項目 | fnbind(1), fnlist(1), fnlookup(1), fnrename(1), fncreate(1M), fndestroy(1M), attributes(5), fns(5)

名前 | fold – 行の折り返し用フィルタ

形式 | **fold** [-bs] [-w *width* | -width] [*file...*]

機能説明 | fold ユーティリティは、入力ファイルから読み込んだ各行を、 行の最大長が *width* で示すカラム数 (-b オプション指定時はバイト数) になるように 分割して複数行に表示するフィルタです。 つまり各出力行 (以降、セグメントと呼ぶ) の長さが 指定されたカラム数またはバイト数を超えないように、 復帰改行文字を挿入していきます。 分割は文字の途中で発生することはありません。 入力中の 1 つの文字が占有するカラム数より小さい値を *width* に指定した場合、その結果は予測できません。

-b オプションが省略された場合、入力中に 改行、バックスペース、タブのいずれかの文字を検出すると、 以下に述べるような特殊な処理が行われます。

バックスペース (BACKSPACE)	現在の行の幅の値が 1 だけマイナスされます。 ただし負の数になることはありません。 fold は、バックスペース文字の直前や直後に復帰改行を挿入することはありません。
改行 (CARRIAGE-RETURN)	現在の行の幅の値が 0 に設定されます。 fold は、改行文字の直前や直後に復帰改行を挿入することはありません。
タブ (TAB)	カラム位置ポインタを次のタブ位置へ進めます。 タブ位置は、カラム 1 から 8 カラムごと (1, 9, 17 ...) に 設定されています。

オプション | 以下のオプションを指定できます。

-b	*width* の値はカラム数ではなくバイト数で示されていることを指定します。
-s	入力行に空白文字が含まれている場合、 出力セグメントの長さが *width* カラムまたはバイトを超えない範囲で、最後の空白文字の直後で 行を分割します。 この条件に合う空白文字が存在しなければ、 -s オプションはそのセグメントに関しては無意味となります。
-w *width* | -width	出力セグメントの最大長をカラム数 (-b 指定時はバイト数) で指定します。 *width* の値が正の整数でないと、エラーが発生します。 デフォルト値は 80 です。

オペランド | 以下のオペランドを指定できます。

file	行の折り返しを行うテキストファイルのパス名。 このオペランドを省略すると、標準入力とみなされます。

使用例 | 例 **1** 長い行を含んでいるファイルをプリンタに送る

長い行を含んでいるファイルをプリンタに送る場合、 そのプリンタに lp(1) が割り当てた行の幅を知っていれば、以下のようなコマンドにより 出力行を折り返せます。

```
example% fold -w 132 bigfile | lp
```

環境

fold の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　指定されたファイルはすべて正常に処理された

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

cut(1), pr(1), attributes(5), environ(5)

注意事項

fold と cut(1) は、どちらも長い行を持つファイルから新たなテキストファイルを 生成する目的で使われます。 fold は行の内容を連続させるべき場合に使用するのに対して、 cut は行 (またはレコード) の数を一定にしておく場合に使用します。

fold は、最大印刷幅 (通常は 80 または 132 カラム) を超える行は切り捨ててしまうような プリンタにテキストファイルを出力させるためによく使われます。

下線が入力ファイルに含まれている場合には、 fold は正しく動作しないことがあります。

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式 | **mailp** [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明 | `mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション | 次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D` 出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。

`-F` メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。

`-h` バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。

`-l` 横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。

`-P` *printer* `-d` オプションを指定した場合と同じです。

`-s` *subject* *subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename 読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`1` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

ftp(1)

名前 | ftp – ファイル転送プログラム

形式 | **ftp** [-dginptv] [-T *timeout*] [*hostname* [*port*]]

機能説明 | ftp コマンドは、インターネット標準ファイル転送プロトコル (FTP) のユーザーインタフェースとして機能します。このコマンドを使用すると、リモートネットワークサイトとファイルをやりとりすることができます。

ftp の通信相手であるホストおよびポート (任意指定) は、コマンド行に指定することができます。クライアントホスト名をコマンド行に指定した場合 ftp は、ただちにそのホストの FTP サーバーとの接続を確立しようとします。ホスト名がコマンド行で指定されなかった場合は、コマンドインタプリタに制御が移り、ユーザーからの指示を待ちます。指示を待っている間、ftp> というプロンプトが表示されます。

オプション | 以下に示すオプションは、コマンド行中に指定することも、コマンドインタプリタに対して指定することもできます。

-d — デバッグモードを有効にします。

-g — ファイル名の “グロビング” (展開) 機能を無効にします。

-i — 複数ファイルの転送中に、対話形式のプロンプトを出力しないようにします。

-n — 初期接続時に “自動ログイン” を行いません。自動ログインが有効になっている場合、ftp はユーザーのホームディレクトリ内の .netrc ファイルを検査し、リモートマシンにログインするアカウントが記述されているかどうかを確認し、アカウントが見つからない場合は、ログイン名の入力を要求します (デフォルトはローカルマシン上のログイン名)。さらに、もし必要であれば、パスワードとログインするアカウントの入力も要求します。

-p — データ転送を受動モードで行います。このオプションは、ファイアウォールを介して接続されているリモートホストと通信する場合に便利です。

-t — パケットトレース機能を有効にします (この機能はまだ実装されていません)。

-T *timeout* — 大域接続タイマーを有効にします (秒単位、10 進数で指定)。タイマーには、サーバーにデータが送信されるとリセットされ、クライアントシステムにユーザーの入力を求めるプロンプトが表示されている間は無効になるという制御接続用のタイマーと、着信方向または発信方向のデータ接続を監視するためのタイマーの 2 種類があります。

-v — データ転送統計情報のレポートとともに、リモートサーバーからのすべての応答を表示します。ftp が対話形式で動作している場合は、この機能はデフォルトで有効になります。

以下に示すコマンドは、コマンドインタプリタに対して指定することができます。

`!` [*command*]
command で指定したコマンドを、ローカルマシン上で シェルコマンドとして実行します。*command* を省略した場合、対話型シェルが呼び出されます。

`$` *macro-name* [*args*]
macro-name で指定したマクロを実行します。このマクロは `macdef` コマンドで定義したものです。引数 *args* は、そのままマクロに渡されます。

`account` [*passwd*]
リモートシステムに正常にログイン完了した後、資源のアクセスに必要な補助パスワードを指定します。*password* を省略した場合、ユーザーはアカウントのパスワードを入力するように要求されます。このとき、入力したパスワードは画面上には表示されません。

`append` *local-file* [*remote-file*]
[*remote-file*] *local-file* で指定したローカルファイルを、リモートマシン上の *remote-file* で指定したファイルの末尾に追加します。 *remote-file* を省略した場合、ローカルファイル名が `ntrans` または `nmap` の設定に従って変更され、リモートファイル名として使用されます。ファイル転送時、“表現タイプ”、“ファイル構造”、“転送モード” には、現在の設定値が使用されます。

`ascii`
“表現タイプ” を、デフォルトの “ネットワーク ASCII” に設定します。

`bell`
各ファイル転送コマンドが終了するたびに、ベルを鳴らします。

`binary`
“表現タイプ” を、“イメージ” に設定します。

`bye`
リモートサーバーとの FTP セッションを終了し、`ftp` コマンドの実行を終了します。なお `EOF` (ファイルの終わり) に到達した場合もセッションを終了し、コマンドの実行を終了します。

`case`
`mget` コマンドを実行している間、リモートコンピュータ上にあるファイル名の大文字と小文字を対応づける機能を有効にします (デフォルトでは無効になっています)。この機能を有効にすると、英大文字の名前を持つリモートコンピュータ上のファイルは、英小文字の名前でローカルディレクトリに書き込まれます。

`cd` *remote-directory*
リモートマシン上の作業ディレクトリを、*remote-directory* で指定したディレクトリに変更します。

`cdup`
リモートマシン上の作業ディレクトリを、現在の作業ディレクトリの親ディレクトリに変更します。

`close`
リモートサーバーとの FTP セッションを終了し、コマンドインタプリタに戻ります。定義されたマクロがあれば削除されます。

`cr`
“ネットワーク ASCII” タイプのファイル検索中に復帰改行 (RETURN) を除去する処理のオン・オフの切り替えを行います。“ネットワーク ASCII” タイプのファイル転送時には、 復帰改行/改行 (RETURN/`LINEFEED` シーケンス) によってレコードが認識されます。 `cr` がオン (デフォルト) のとき、復帰改行はこのシーケンスから取り除かれます。 これにより、改行のみでレコードを区切っている UNIX システムの仕様に準拠できます。UNIX 以外のシステムのリモートホスト上のレコードには、改行のみを区切り記号として使用している場合があります。“ネットワーク ASCII” タイプの転送を行ったとき、これらの改行文字は `cr` がオフの場合にのみレコード区切り記号とは 区別して扱われます。

`delete` *remote-file*
リモートマシン上の *remote-file* で示したファイルを削除します。

`debug`
デバッグモードのオン・オフの切り替えを行います。 デバッグモードがオンのとき、`ftp` はリモートマシンに送られたコマンドを、先頭に `-->` を付加して印刷します。

`dir` [*remote-directory*] [*local-file*]
remote-directory で示すリモートディレクトリの内容を出力します。出力先は *local-file* で示したローカルファイルです。 *remote-directory* を省略した場合は、 現在の作業ディレクトリの内容を出力します。 ローカルファイルを省略した場合、または *local-file* として `-` を指定した場合は、出力先は端末となります。

`disconnect`
`close` オプションと同一機能です。

`form` [*format-name*]
“表現タイプ” のうちのキャリッジ制御 フォーマットサブタイプを、*format-name* で示したフォーマットに設定します。なお *format-name* として指定可能なものは `non-print` だけです。これはデフォルトの “ノンプリント” サブタイプに対応します。

`get` *remote-file* [*local-file*]
remote-file で示したリモートファイルを取り出し、 *local-file* で示した名前でローカルマシン上に格納します。 *local-file* を省略した場合、リモートマシン上のファイル名が `case`、`ntrans`、`nmap` 設定に従って変更され、ローカルファイル名として使用されます。ファイル転送時、“表現タイプ”、“ファイル 構造”、“転送モード” に関しては、現在の設定値が用いられます。

`glob`
`mdelete`、 `mget`、 `mput` 時のファイル名の展開、あるいは “グロビング” 機能のオン・オフの切り替えを行います。グロビング機能をオフにすると、ファイル名は展開されずそのまま使用されます。

`mput` 時のグロビングは、`sh`(1) での処理と同一です。 `mdelete` と `mget` に関しては、各リモートファイル名は リモートマシン上で個々に展開され、リストはマージされません。

ディレクトリ名の展開は、通常ファイル名の展開とはかなり異なります。すなわちグロビングの結果は、リモートオペレーティングシステムと FTP サーバーにより決まります。その値は `mls` *remote-files* – コマンドの実行によりあらかじめ得ることができます。

`mget` と `mput` は、ファイルのディレクトリサブツリーを すべて転送するためのものではありません。そのような転送を行いたいときは、`tar`(1) アーカイブをサブツリーに転送してください (その際には `binary` コマンドの実行で設定される “イメージ” という “表現タイプ”を使用してください)。

`hash`
個々のデータブロックを転送するたびにハッシュサイン (#) を印刷する機能のオン・オフの切り替えを行います。各データブロックのサイズは 8192 バイトです。

`help` [*command*]
command で示したコマンドの意味を説明するメッセージを出力します。*command* を省略した場合は、定義されているコマンドの一覧表が出力されます。

`lcd` [*directory*]
ローカルマシン上の作業ディレクトリを、*directory* で示したディレクトリに変更します。*directory* を省略した場合は、ユーザーのホームディレクトリが指定されたものとみなされます。

`ls`[*remote-directory* | `-al`] [*local-file*]
remote-directory で示すリモートファイル上のディレクトリの内容を、要約した形式で出力します。出力先は *local-file* が示すローカルファイルです。*remote-directory* を省略した場合は、現在の作業ディレクトリの内容が出力されます。

`-a` オプションは全てのエントリを表示します。通常は表示されないドット (.) で始まるファイルも含みます。`-l` オプションはモード、リンク数、所有者、グループ、バイト数、最後の修正日時をファイルごとに表示します。特殊ファイルの場合、サイズフィールドにはサイズではなくメジャーやマイナのデバイス数が含まれます。ファイルがシンボリックリンクの場合、ファイル名は “→” に続き、次にリンク先のファイルのパス名が表示されます。ローカルファイルを省略した場合、または *local-file* として – を指定した場合は、出力先は端末となります。

`macdef` *macro-name*
マクロを定義します。後続の行で指定された内容が、*macro-name* という名前のマクロとして格納されます。マクロ定義の終わりは空の (NULL) 行 (ファイル中では連続した復帰改行 (NEWLINE) 文字、端末入力では連続した復帰改行 (RETURN) 文字) によって表されます。定義可能なマクロ数は最大 16 個、その合計文字数は最大 4096 文字です。いったん定義されたマクロは、`close` コマンドが実行されるまで定義されたままとなります。

マクロプロセッサは、 `$` と `\` を特殊文字として解釈します。 `$1`、 `$2`、 `$3` ... のように `$` の後に 1 個 (または複数個) の数値を指定すると、それは、マクロ呼び出しコマンド行内の対応する引数に置き換えられます。`$` の後に文字 `i` を指定すると、そのマクロはループされる旨がマクロプロセッサに通知されます。つまり最初のパスでは `$i` はマクロ呼び出しコマンド行内の 第 1 引数で置き換えられ、2 回目の実

行時には 第 2 引数で置き換えられ、という方法で繰り返されます。\ の後に何らかの文字を指定すると、\ はその文字で置き換えられます。これにより、$ を特殊文字ではなく通常の文字として使用することができます。

`mdelete` *remote-files*
リモートマシン上にある、*remote-files* で示すファイルを削除します。

`mdir` *remote-files local-file*
`dir` と同機能ですが、複数のリモートファイル名を指定できる点が異なります。対話形式のプロンプトが有効な場合、最後の引数が `mdir` の出力先としてのターゲットのローカルファイルであることを確認するためのプロンプトを出力します。

`mget` *remote-files*
リモートマシン上にある、*remote-files* で示すファイル名を展開し、それにより生成された名前を持つ各ファイルに対して `get` を実行します。ファイル名の展開の詳細については、`glob` の説明を参照してください。生成されたファイル名は、`case`、`ntrans`、`nmap` の設定値に従って処理されます。ファイルはローカル作業ディレクトリに転送されます。ローカル作業ディレクトリは `lcd` *directory* により変更できます。 新たなローカルディレクトリを作成するには、`!` `mkdir` *directory* を使用してください。

`mkdir` *directory-name*
directory-name で示したディレクトリをリモートマシン上に生成します。

`mls` *remote-files local-file*
`ls`(1) と同機能ですが、複数のリモートファイル名を指定できる点が異なります。対話形式のプロンプトが有効な場合、最後の引数が `mls` の出力先としてのターゲットのローカルファイルであることを確認するためのプロンプトを出力します。

`mode` [*mode-name*]
“転送モード” を、 *mode-name* に設定します。なお *mode-name* として指定可能なものは `stream` だけです。これはデフォルトの “ストリーム” モードに対応します。現在のシステムでサポートしているモードは `stream` だけなので、これを指定しなければなりません。

`mput` *local-files*
local-files で示したローカルファイル名のリスト中のワイルドカードを展開し、それにより生成された名前を持つ各ファイルに対して `put` を実行します。ファイル名展開の詳細については `glob` の説明を参照してください。生成されたファイル名は、 `ntrans` および `nmap` の設定値に従って処理されます。

`nmap` [*inpattern outpattern*]
ファイル名マッピングメカニズムを 設定または解除します。引数を省略した場合は、解除となります。引数を指定した場合は、リモートファイル名は 「対象のリモートファイル名指定のない `mput` および `put` コマンド」の実行時にマップされます。 同様にローカルファイル名は 「対象のローカルファイル名指定のない `mget` および `get` コマンド」の実行時にマップされます。

このコマンドは、異なったファイル名規約を持つ、UNIX 以外のシステムのリモートホストに接続する場合に便利です。マッピング処理は、*inpattern* と *outpattern* で設定されたパターンに従って行われます。*inpattern* は入力ファイル名 (すでに `ntrans` および `case` の設定により処理されている場合もある) 用のテンプレート

です。 *inpattern* 中に $1、$2 ... $9 シーケンスを含めることにより、変数を使用したテンプレート処理も可能です。 $ を特殊文字ではなく通常の文字として扱う必要があれば、 \ を使用してください。その他の文字はすべて入力されたとおりに処理され、nmap の *inpattern* 変数の値を決定するために用いられます。

たとえば *inpattern* として $1.$2 を指定し、リモートファイル名が mydata.data のとき、 $1 の値は mydata に、 $2 の値は data になります。

outpattern は、マッピングの結果生成されるファイル名を決定します。$1、$2 ... $9 というシーケンスを指定すれば、*inpattern* のテンプレートから得られる値により 置き換えられます。シーケンス $0 は、元のファイル名に置き換えられます。さらに、[*seq1* , *seq2*] というシーケンスは、 *seq1* が NULL でなければ *seq1* に、NULL であれば *seq2* に置き換えられます。

たとえば nmap $1.$2.$3 [$1,$2].[$2,file] というコマンドを実行すると、入力ファイル名が myfile.data または myfile.data.old のときは出力ファイル名が myfile.data になり、 myfile のときは myfile.file になり、 .myfile のときは myfile.myfile になります。また、たとえば nmap $1 | sed "s/ *$//" > $1 のように、 *outpattern* 中に空白文字が含まれていても構いません。 $、[、]、, の 4 つの各文字を、特殊文字ではなく通常の文字として扱う必要があれば、 \ を使用してください。

ntrans [*inchars* [*outchars*]]
ファイル名の文字変換メカニズムを 設定または解除します。引数を省略した場合は、解除となります。引数を指定した場合は、リモートファイル名を構成する文字は 「対象のリモートファイル名指定のない mput および put コマンド」の実行時に変換されます。同様にローカルファイル名の文字は 「対象のローカルファイル名指定のない mget および get コマンド」の実行時に変換されます。

このコマンドは、異なったファイル名規約を持つ、UNIX 以外のシステムのリモートホストに接続する場合に便利です。

inchars で指定した文字がファイル名に含まれていると、その文字は *outchars* 中の対応する文字に変換されます。 *inchars* 内におけるその文字の位置が *outchars* の長さを超えている場合、その文字はファイル名から削除されます。

ntrans コマンドを ftp で使用すると 16 文字のみを変換します。 全アルファベットを転換する場合には、 case (上記参照) を使います。

open *host* [*port*]
host が示すホストの FTP サーバーとの接続を確立します。 オプションであるポート番号を指定すると、 ftp はそのポートにおいて FTP サーバーと接続しようと試みます。また 自動ログインオプションが有効 (デフォルト値) のとき、 ftp はユーザーを自動的に FTP サーバーにログインしようと試みます。

passive
受動モードのオン・オフを切り替えます。受動モードがオンの場合、ftp クライアントは、データ接続用のポートをオープンし、そのポートのアドレスを返すことを要求する PASV コマンドを FTP サーバーに送信します。リモートサーバーはそのポートで待機し、クライアントはそのポートに接続します。受動モードがオフの場合、ftp クライアントは、リモートサーバーが接続するべきアドレスを指定する

PORT コマンドを送信します。受動モードは、ftp クライアントへの接続が何らかの形で制御されている場合 (例: ファイアウォール配下にある場合など) に便利です。IPv6 対応の FTP サーバーに接続する場合は、PASV の代わりに EPSV、PORT の代わりに EPRTが使用される場合があります。

prompt
対話形式のプロンプトのオン・オフの切り替えを行います。オンに設定すれば、複数個のファイル転送を行う場合、読み書きの対象とするファイルを 個別に選択することが可能となります。 デフォルトはオンです。 オフのときは、 mget や mput コマンドはすべてのファイルを転送し、 mdelete コマンドはすべてのファイルを削除します。

proxy *ftp-command*
二次制御接続上で FTP コマンドを実行します。 すなわちこのコマンドは、2 つのリモート FTP サーバーに対して同時に接続を確立し、それらのサーバー間での転送を可能とします。最初に発行する proxy コマンドは、二次制御接続を確立するための open コマンドでなければなりません。二次接続上で実行可能なその他のコマンドを知るには、 proxy ? コマンドを使用してください。

以下に述べるコマンドは、 proxy を伴って入力された場合、 次のように、 通常とは異なった動作をします。 open は自動ログイン処理中には新たなマクロの定義は行いません。close は既存のマクロの削除は行いません。get および mget は一次制御接続上のホストから 二次制御接続上のホストへファイルを転送します。put、mput、および append は、 二次制御接続上のホストから一次制御接続上のホストへファイルを転送します。

第三者によるファイル転送は、二次制御接続上のサーバーが PASV コマンドをサポートしているか否かにより異なります。

put *local-file*[*remote-file*]
local-file で示すローカルファイルを、 *remote-file* という名でリモートマシン上に格納します。 *remote-file* を省略した場合は、 ローカルファイル名が ntrans または nmap 設定に従って変更され、 リモートファイル名として使用されます。 ファイル転送時、“表現タイプ”、“ファイル 構造”、“転送モード” に関しては、現在の設定値が用いられます。

pwd
現在の作業ディレクトリの名前を リモートマシン上に出力します。

quit
bye と同一機能です。

quote *arg1 arg2* . . .
指定された引数をそのままリモート FTP サーバーに送信します。 FTP からは応答コードが 1 つ返されます。 (有効な引数の一覧を表示するには、 remotehelp コマンドを使用してください。)

quote コマンドの使用は、FTP プロトコルを熟知している、 経験豊かなユーザーだけに限るべきです。

recv *remote-file*[*local-file*]
get と同一機能です。

`reget` *remote-file* [*local-file*]
`reget` コマンドの動作は `get` コマンドと似ています。ただし、ローカルファイル (*local-file*) が存在し、かつ、そのサイズがリモートファイル (*remote-file*) より小さい場合は、*local-file* を転送中の *remote-file* の部分コピーであるとみなして、障害が発生したと考えられる時点から転送を継続するという点が異なります。このコマンドは、接続が途切れる可能性の高いネットワーク経由での大きなファイル転送に便利です。

`remotehelp` [*command-name*]
リモート FTP サーバーのヘルプ機能を呼び出します。 *command-name* が指定されていれば、それもサーバーに渡されます。

`rename` *from to*
リモートマシン上に存在する *from* で示したファイルの名前を *to* という名前に変更します。

`reset`
応答用のキューをクリアします。このコマンドは、 コマンドと応答との順序を、リモート FTP サーバーに合わせ直します (再同期化)。リモートサーバーが FTP のプロトコルに違反する動作を行った場合、 この再同期化処理が必要となることがあります。

`restart` [*marker*]
指定したマーカー (*marker*) の値で、直後にある `get` または `put` を再起動します。UNIX システムでは、*marker* は通常ファイルへのバイトオフセットです。直後のコマンドが `mget` の場合、`restart` は、1 つめの `get` に対して適用されます。*marker* に `0` を指定した場合は、再起動マーカーはクリアされます。引数を指定しなかった場合は、現在の再起動状態が表示されます。

`rmdir` *directory-name*
リモートマシン上の *directory-name* で示すディレクトリを削除します。

`runique`
ローカルシステムにファイルを格納する際に一意なファイル名を与える処理のオン・オフの切り替えを行います。この機能がオンのとき、 `get` または `mget` コマンドの対象となるローカルファイルの名前と 同じファイル名がすでに存在していれば、 ファイル名の終わりに `.1` を付加して一意な名前を新たに生成します。 さらにその名前と同じファイル名もすでに存在していれば、 `.2` を使用します。このようにして数値を増加させていき、 `.99` に到達しても一意名を生成できない場合、ファイル転送は中止され、エラーメッセージが表示されます。一意なファイル名が生成できたら、 その名前が報告されます。この `runique` 機能は、シェルコマンドで生成する ローカルファイルの名前には影響を与えません。 デフォルトはオフです。

`send` *local-file* [*remote-file*]
`put` と同一機能です。

`sendport`
`PORT` コマンドの使用のオン・オフの切り替えを行います。デフォルトでは、 `ftp` は各データ転送処理に対して接続を確立する際に `PORT` コマンドの使用を試みます。 複数のファイル転送を実施する場合に、 `PORT` コマンドを用いると時間の遅れを防ぐことができます。 `PORT` コマンド実行が失敗すると、 `ftp` はデフォルトの

データポートを使用します。 PORT コマンドの使用がオフの (使用しない) 状態では、 データ転送処理に対しての PORT コマンドの使用の試みは抑止されます。このオフ指定は、 PORT コマンドを無視するのにもかかわらず受け付けた旨を (誤って) 表してしまうような FTP システムに接続されている場合に便利です。

`site` *arg1* [*arg2*] ...
指定した引数をそのまま SITE コマンドとして、リモートの FTP サーバーに送信します。

`status`
ftp の現在の状態を表示します。

`struct` [*struct-name*]
ファイル構造を *struct-name* で示す値に設定します。なお *struct-name* として指定可能なものは file だけです。これはデフォルトの "ファイル" 構造にあたります。現在のシステムでサポートされている構造は file だけで、これ以外は指定できません。

`sunique`
リモートシステムにファイルを格納する際に 一意なファイル名を与える処理のオン・オフの切り替えを行います。 この機能を使うには、リモート FTP サーバーが STOU コマンドをサポートしていなければなりません。 リモートサーバーは、生成した一意名を報告します。 デフォルトはオフです。

`tcpwindow` [*size*]
データ接続に使用する TCP ウィンドウのサイズ (*size*) を設定します。0 を指定すると、データ接続時の TCP ウィンドウサイズは明示的には設定されません。引数を指定しなかった場合は、現在の設定が表示されます。

`tenex`
"表現タイプ" を、 TENEX マシンと通信するために必要な値に設定します。

`trace`
パケットトレース機能のオン・オフの切り替えを行います (未実装)。

`type` [*type-name*]
"表現タイプ" を、 *type-name* に設定します。 *type-name* として指定可能なものは次のとおりです。 "ネットワーク ASCII" の場合は ascii、"イメージ" の場合は binary または image、バイト長が 8 の "ローカルバイトサイズ" の場合は tenex (TENEX マシンとの通信に使用)。 *type-name* を省略した場合は、現在の表現タイプが表示されます。 デフォルト時のタイプは "ネットワーク ASCII" です。

`user` *user-name* [*password*] [*account*]
リモート FTP サーバーに対して、 ユーザー自身の名前などの属性を通知します。サーバーがパスワードを必要としているのに *password* を省略した場合は、 ftp はローカルエコーをオフにしてから パスワードの入力を要求します。 同様にサーバーがアカウントを必要としているのに *account* 指定を省略した場合は、 ftp はアカウントの入力を要求します。 アカウントが指定された場合、 (リモートサーバーがログイン時に アカウントを必要としなければ) ログイン処理完了後にアカウントコマンドが リモートサーバーに渡されます。この処理は、 ftp が "自動ログイン使用不可" 状態で呼ばれた場合を除き、 FTP サーバーへ最初に接続が行われた時点で 自動的に実行されます。

verbose
冗長表示モードのオン・オフの切り替えを行います。 このモードがオンのとき、FTP サーバーからの応答はすべて表示されます。 さらにファイル転送が終了するたびに、その転送処理の効率に関する統計データも 報告されます。デフォルトは、ftp のコマンドが端末から送られてくる場合はオン、それ以外の場合はオフです。

? [*command*]
help と同一機能です。

空白文字を含んだコマンド引数は、 引用符 (") で囲むことができます。

省略できないコマンド引数を省略すると、 ftp はその引数の入力を要求するプロンプトを出力します。

ファイル転送の中止

ファイル転送処理を中止したい場合、端末の割り込みキーを使用します。 その転送処理が送信の場合は、ただちに中止されます。 受信の場合には、リモートサーバーに対して FTP プロトコルの ABOR コマンドを送信し、以後受信するデータを 破棄することにより中止されます。この処理に要する時間は、リモートサーバーが ABOR 処理をサポートしているかどうかで違ってきます。リモートサーバーが ABOR コマンドをサポートしていないと、 要求されたファイルをリモートサーバーが完全に送信し終わるまで ftp> というプロンプトは現れません。

ftp がローカル処理を終えてリモートサーバーからの 応答を待っている状態の場合には、 端末から割り込みキーを入力しても無視されます。 このモードでは遅延時間が長くなることがあります。 その場合の理由としては、上記の ABOR 処理によるもの、または リモートサーバー側の誤った動作 (たとえば、 ftp プロトコル違反) によるものが考えられます。 後者が原因の場合、ローカル ftp プログラムを手動で終了させることが必要となります。

ファイル命名規約

ftp コマンドの引数として指定されたローカルファイル名は、 以下の規則に従って処理されます。

1) ファイル名として – を指定すると、読み込みの場合は標準入力、 書き出しの場合は標準出力が使用されます。

2) ファイル名の最初の文字が | の場合、引数の残りの部分は シェルコマンドであると解釈されます。 ftp はその引数を指定して popen(3C) を使ってシェルを呼び出し、 そのシェルの標準入力からデータを読み込みます (または標準出力にデータを書き出します)。 シェルコマンドが空白文字を含んでいる場合には、 引用符で囲まなければなりません (たとえば、 "| ls -lt")。 この機能の便利な使い方の典型的な例は "dir | more" です。

3) 上記の 2 種類に該当しないローカルファイル名は、グロビング (展開) メカニズムが有効な場合、sh(1) が使用する規則に従って展開されます (詳細は glob の項を参照)。 put のようにローカルファイル名を 1 つだけ必要とする ftp コマンドの場合には、グロビング処理で 最初に生成されるファイル名だけが用いられます。

4) mget または get コマンドに対してローカルファイル名を省略すると、リモートファイル名が case、ntrans、nmap の設定値に従って変更され、

ローカルファイル名として使用されます。 runique がオンの場合は、その名前がさらに変更されることがあります。

5) mput または put コマンドに対してリモートファイル名を省略すると、ローカルファイル名が ntrans および nmap の設定値に従って変更され、リモートファイル名として使用されます。 sunique がオンの場合は、その名前をリモートサーバーがさらに変更することがあります。

ファイル転送用パラメータ

FTP 仕様では、ファイル転送に影響を与えうるパラメータをいくつか指定できます。

“表現タイプ” として指定できる値は “ネットワーク ASCII”、“EBCDIC”、“イメージ”、バイト長 指定つきの “ローカルバイトサイズ”(おもに PDP10 および PDP20 シリーズ用) です。 “ネットワーク ASCII” および “EBCDIC” の場合には、さらにサブタイプ指定を伴います。 このサブタイプは、垂直方向のフォーマット制御 (復帰改行 (NEWLINE) 文字やフォームフィードなど) がそのまま渡される (“ノンプリント”) か、TELNET フォーマット中に存在する (“TELNET フォーマット制御”) か、あるいは ASA (FORTRAN) (“キャリッジ制御 (ASA)”) フォーマット中に存在するかを指定します。 ftp がサポートする表現タイプは、“ネットワーク ASCII”(サブタイプは “ノンプリント” のみ)、“イメージ”、およびバイト長が 8 の “ローカルバイトサイズ”(TENEX マシンとの通信に使用) のみです。

“ファイル構造” に指定可能な値は “file”(レコード構造なし) 、 “record”、および “page” です。 ftp はデフォルトである “file” のみをサポートします。

“転送モード” に指定できる値は “stream”、“block”、 および “compressed” です。 ftp はデフォルトである “stream” のみをサポートします。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の ftp の動作については、largefile(5) を参照してください。

ftp コマンドは IPv6 に対応しています。詳細は ip6(7P) のマニュアルページを参照してください。

ファイル

~/.netrc

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	enabled

関連項目

ls(1), rcp(1), sh(1), tar(1), in.ftpd(1M), popen(3C), ftpusers(4), netrc(4), attributes(5), largefile(5), ip6(7P)

『*RFC 2428, FTP Extensions for IPv6 and NATs*』、Allman, M., Ostermann, S., および Metz, C. 共著、The Internet Society 発行、1998 年 9 月

『*RFC 959, File Transfer Protocol (FTP)*』Postel, Jon, Joyce Reynolds 著、Network Information Center 発行、1985 年 10 月

『*RFC 1639, FTP Operation Over Big Address Records (FOOBAR)*』、Piscitello, D. 著、Network Working Group 発行、1994 年 6 月

注意事項

アカウントが `/etc/ftpusers` ファイルにリストされていると、リモート FTP サーバーに明示的に拒否されることが原因で、ログインに失敗する場合があります。`in.ftpd`(1M) および`ftpusers`(4) のマニュアルページを参照してください。

多くのコマンドの実行が正常に行われるか否かは、リモートサーバーの動きによって決まります。

以前は、表現タイプが “ネットワーク ASCII” の場合の 4.2 BSD コードハンドリング転送においてキャリッジリターンの扱いに誤りがありましたが、この誤りは修正されました。ただしその修正の影響で、表現タイプが “ネットワーク ASCII ”のとき、 4.2 BSD とのバイナリファイル転送が 正しく行われないかも知れません。この問題を回避するには、表現タイプとして “イメージ” を使用してください。

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル **(sh)** の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | gencat – 書式付きメッセージカタログの生成

形式 | **gencat** *catfile msgfile...*

機能説明 | gencat コマンドは、*msgfile* というメッセージテキストのソースファイルを *catfile* という書式付きメッセージデータベースにマージします。 *catfile* データベースがまだ存在していない場合は、データベースが作成されます。 *catfile* が存在する場合、そのメッセージは新しい *catfile* に挿入されます。 セット番号とメッセージ番号が衝突する場合、 *msgfile* に定義された新しいメッセージテキストが *catfile* に現在入っている古いメッセージテキストと置き換わります。 gencat に対するメッセージテキストのソースファイル(またはファイルのセット)入力には、セット番号とメッセージ番号の両方、またはメッセージ番号だけを含めることができます。 後者の場合、 NL_SETD (nl_types(3HEAD) 参照) というセットが想定されます。

メッセージテキストのソースファイルフォーマット | メッセージテキストのソースファイルのフォーマットは次のとおりに定義します。 メッセージテキストのソース行のフィールドは、 ASCII 空白文字または ASCII タブ文字 1 つで区切ることに注意してください。 他の ASCII 空白文字または ASCII タブは、後続のフィールドの一部と見なされます。

$set *n comment*	ここで、 *n* は次の $set 、 $delset 、 またはファイルの終わりまでの後続メッセージのセット識別子を指定します。 *n* は (1–{NL_SETMAX}) の範囲内の数にする必要があります。 1 個のソースファイル内のセット識別子は連続する必要はありません。 セット識別子の後の文字列はコメントと見なされます。 メッセージテキストのソースファイルに $set 宣言を指定しないと、メッセージはすべてデフォルトのメッセージセット、 NL_SETD に入ります。
$delset *n comment*	既存のメッセージカタログから *n* というメッセージセットを削除します。 セット番号の後の文字列はコメントと見なされます。 (注意 : *n* は、有効なセットでない場合、無視されます)。
$*comment*	ドル記号の $ で始まり、その後に ASCII 空白文字または ASCII タブ文字が続く行は、コメントと見なされます。
m message-text	*m* はメッセージ識別子であり、 (1-{NL_MSGMAX}) の範囲内の数になります。 *message-text* は直前の $set 宣言で指定されたセット識別子とメッセージ識別子の *m* とともにメッセージカタログに格納されます。 *message-text* が空で、 ASCII 空白文字または ASCII タブのフィールドセパレータが存在する場合は、メッセージカタログに空の文字列を格納します。 メッセージソース行にメッセージ番号があるが、フィールドセパレータも *message-text* もない場合、カタログからその番号 (もしあれば)の既存のメッセージを削除します。

メッセージ識別子は連続する必要はありません。 *message-text* の長さは、 (0–{NL_TEXTMAX}) の範囲内にする必要があります。

`$quote`*c* この行は、オプションの引用符文字である *c* を指定します。 この文字を使用して *message-text* を囲めば、メッセージソース行において後方の空白または NULL (空)メッセージが見えるようにすることができます。 デフォルト時、または空の `$quote` 宣言を指定するとき、 *message-text* の引用は認識されません。

メッセージテキストのソースファイル内の空の行は無視されます。 テキスト文字列には、下記の表に定義された特殊文字およびエスケープシーケンスを入れることができます。

説明	シンボル	シーケンス
改行	NL(LF)	\n
水平タブ	HT	\t
垂直タブ	VT	\v
バックスペース	BS	\b
キャリッジリターン	CR	\r
フォームフィード	FF	\f
バックスラッシュ	\	\\
ビットパターン	ddd	\ddd

ddd というエスケープシーケンスは、バックスラッシュと、それに続く 1 つ、 2 つ、または 3 つの 8 進数からなります。 これらの数字は、希望の文字の値を指定するのに使用します。 バックスラッシュの後の文字が、指定された文字のいずれでもない場合、バックスラッシュは無視されます。

また、バックスラッシュの後に ASCII 改行文字を指定すれば、後続の行に文字列を継続できます。 したがって、下記の 2 行は単一のメッセージ文字列を記述します。

```
1 This line continues \
to the next line
```

これは、次の行と等価です。

```
    1 This line continues to the next line
```

オペランド　以下にオペランドを示します。

catfile フォーマットされたメッセージカタログのパス名。 – を指定すると、 標準出力とみなされる。

msgfile メッセージテキストのソースファイルのパス名。 *msgfile* の例として – を指定すると、 標準入力とみなされる。 メッセージテキストソースファイルの書式は 前述の「メッセージテキストのソースファイルフォーマット」 で定義されている。

環境

gencat の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 入力ファイルはすべて、正常に出力された

>0 エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWloc
CSI	対応済み

関連項目

mkmsgs(1), catgets(3C), catopen(3C), gettxt(3C), attributes(5), environ(5), nl_types(3HEAD)

名前 | getopt – コマンドオプションの解析

形式 | **set** - - ` getopt *optstring* $ * `

機能説明 | getopts コマンドが、 getopt に取って代わります。 詳細は、 以下の「注意事項」の項を参照してください。

getopt は、シェルプロシージャによる解析を容易にするために コマンド行のオプションを分解し、 各オプションが正当であるかどうかの確認に使用します。 *optstring* は、 認識されるオプションの文字列です。 getopt(3C) を参照してください。 オプション文字のあとにコロン (:) が付いている場合は、 そのオプションに引数があるとみなされます。 コロンと引数の間は、 空白で区切られていても、区切られていなくてもかまいません。 特殊なオプション - - は、 オプションの終わりを表すのに使用されます。 このオプションを明示的に使用すると getopt はこれを認識し、 省略された場合には getopt がこれを生成します。 いずれの場合でも、 getopt はオプションの終わりにこの特殊オプションを付けます。 シェルの位置パラメタ ($1 $2 . . .) が再設定されて、 各オプションの前に - が付き、 各オプションはそのオプションの位置パラメタに入ります。 各オプション引数も、 同様に その引数の位置パラメタに入るように解析されます。

使用例 | 例 **1** コマンド引数の処理

次の部分コードに、 オプション -a または -b 、および引数の必要なオプション -o を使用するコマンドについて、 引数の処理例を示します。

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
           echo $USAGE
           exit 2
fi
for i in $*
do
           case $i in
           -a | -b)     FLAG=$i; shift;;
           -o)           OARG=$2; shift 2;;
           --)           shift; break;;
           esac
done
```

このコードは、 次のすべてのコマンドを同一内容として受け付けます。

```
cmd -aoarg filename1 filename2
cmd -a -o arg filename1 filename2
cmd -oarg -a filename1 filename2
cmd -a -oarg -- filename1 filename2
```

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値

使用条件	SUNWcsu
CSI	対応済み

関連項目 intro(1), getopts(1), getoptcvt(1), sh(1), shell_builtins(1), getopt(3C), attributes(5)

診断 getopt は、 *optstring* に含まれていないオプション文字を検出すると、 標準エラー出力に エラーメッセージを表示します。

注意事項 getopt は、次のメジャーリリースではサポートされません。 このリリースにおいては、 変換ツール getoptcvt が提供されています。 詳細については、 getopts(1) および getoptcvt(1) のマニュアルページを参照してください。

オプションを再走査する際は、 optind に 1 を再設定してください。

getopt は、オプションに続くオプション引数を、 空白および引用符で区切って指定できるという コマンド構文標準規則 8 の一部をサポートしません (intro(1) 参照)。 たとえば、 次のコマンド行は正しく処理されません。

```
cmd -aa    -b -o "xxx z yy" filename
```

この不完全性を正すためには、 getopt コマンドの代わりに getopts コマンドを使用してください。

オプション引数を持つあるオプションに、 *optstring* で示されているオプションの 1 つと同じ値が続いている場合 (前述の「使用例」の項を参照。 ただし、コマンド行 cmd -o -a *filename*を使用する場合)、 getopt は、常に -a を -o のオプション引数として扱うので、 -a はオプションとして認識されません。 この場合、 この例の for ループは、*filename* 引数を通り越してシフトします。

名前 | getoptcvt – コマンドオプションを解析するために getopts に変換

形式 | **/usr/lib/getoptcvt** [-b] *filename*

/usr/lib/getoptcvt

機能説明 | /usr/lib/getoptcvt は *filename* 内のシェルスクリプトを読み取り、 getopt の代わりに getopts を使用できるように それを変換し、 その結果を標準出力に出力します。

getopts は、 Bourne シェル組み込みコマンドの 1 つであり、 定位置パラメタの解析および オプションの妥当性チェックに使用されます。 sh(1) を参照してください。 このコマンドは、 コマンド構文標準規格 (intro(1) のルール 3 〜 10 を参照) のうち、該当するルールすべてをサポートします。 このコマンドは、getopt コマンドの代わりに使用してください (以下の「注意事項」の項を参照) 。シェルの getopts 組み込みコマンドの構文を 次に示します。

getopts *optstring name* [*argument* . . .]

optstring には、getopts を使用するコマンドが認識できる オプション文字を入れなければなりません。 オプション文字の後にコロン (:) が付いている場合は、 そのオプションには 1 つまたは複数の引数があるとみなされます。 オプションと引数の間は、空白によって区切られていなければなりません。

getopts は、呼び出されるたびに、 次のオプションをシェル変数 *name* に入れ、 処理すべき次の引数へのインデックスをシェル変数 OPTIND に入れます。 シェルまたはシェル手続きが呼び出されるたびに、 OPTIND の値は 1 に初期設定されます。

オプションにオプション引数が必要な場合、getopts はその引数をシェル変数 OPTARG に入れます。

不正なオプションが検出されると、? が *name* に入ります。

オプションの終わりが検出されると、getopts はゼロ以外の終了状態で終了します。特殊オプション -- を使用して、オプションの終わりを表すことができます。

デフォルトでは、 getopts は定位置パラメタを解析します。 getopts コマンド行に追加の引数 (*argument* . . .) が指定されると、 getopts は定位置パラメタの代わりにその引数を解析します。

新しいコマンドはすべて、 intro(1) で説明している コマンド構文標準規格に 従うようにするために、 定位置パラメタの解析、 および オプションが そのコマンドの有効なオプションであるかどうかの チェックに、 getopts または getopt を使用しなければなりません (以下の「注意事項」の項を 参照してください) 。

オプション | 次のオプションを指定できます。

-b　変換されたスクリプトを、 UNIX システムの旧リリースへ 移植可能にします。/usr/lib/getoptcvt は、 *filename* 内のシェルスクリプトを変更し、変更後のシェルスクリプトを実行すると、 実行時に getopts と getopt のどちらを呼び出すか判別できるようにします。

使用例

例 1 引数の使用例

次に示すシェルプログラムの一部分では、オプション -a または -b 、およびオプション引数の必要なオプション -o を使用するコマンドについて、引数の使用例を示します。

```
while getopts abo: c
do
      case $c in
      a | b)      FLAG=$c;;
      o)          OARG=$OPTARG;;
      \?)         echo $USAGE
                  exit 2;;
      esac
done
shift `expr $OPTIND - 1`
```

例 2 同一内容のコード式

このコードは、 次のコマンドすべてを同一内容として受け付けます。

```
cmd -a -b -o "xxx z yy" filename
cmd -a -b -o "xxx z yy" -filename
cmd -ab -o xxx,z,yy filename
cmd -ab -o "xxx z yy" filename
cmd -o xxx,z,yy b a filename
```

環境

getopts の実行に影響する環境変数 LC_CTYPE、LC_MESSAGES、および NLSPATH については、environ(5) のマニュアルページを参照してください。

OPTIND　この変数は、次に処理すべき引数の索引として getoptcvt によって使用されます。

OPTARG　この変数は、オプションが引数を使用している場合に、引数を格納するために getoptcvt によって使用されます。

終了ステータス

次の終了ステータスが返されます。

0　*optstring* によって指定または指定解除されたオプションが見つかった

>0　オプションの終わりに到達したか、エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

intro(1), getopts(1), sh(1), shell_builtins(1), getopt(3C), attributes(5)

診断

getopts は、 *optstring* に含まれていないオプション文字を検出すると、 標準エラー出力にエラーメッセージを表示します。

注意事項

現在の実装においては、 次のようにコマンド構文規格（ intro(1) を参照 ）に従わない使用方法が認められていますが、 このような使用方法は、 将来、システムのリリースではサポートされないので、 使用しないでください。 前述の 「使用例」 と同様に、 -a および -b がオプションであり、 -o はオプション引数を持つオプションとします。 次の例では、 オプション引数を持つオプションは 別のオプションと一緒にまとめることはできない、 というルール 5 に違反しています。

```
example% cmd -aboxxx filename
```

次の例では、 オプション引数を持つオプションの後には、 空白がなければならない、 というルール 6 に違反しています。

```
example% cmd -ab oxxx filename
```

シェル変数 OPTIND の値を変更したり、 異なる引数の集まりを解析したりした場合の結果は不定です。

名前 | getopts – ユーティリティオプションの解析

形式 | **/usr/bin/getopts** *optstring name* [*arg*...]

sh | **getopts** *optstring name* [*argument*...]

ksh | **getopts** *optstring name* [*arg*...]

/usr/bin/getopts

getopts ユーティリティを使えば、引数リストからオプションやオプション引数を検索できます。

getopts は、呼び出されるたびに、*name* オペランドで指定されているシェル変数に次のオプションを入れ、シェル変数 OPTIND に処理すべき次の引数へのインデックスを入れます。シェルが呼び出されるたびに、OPTIND は 1 に初期設定されます。

オプション引数を必要とするオプションの場合、getopts はオプション引数をシェル変数 OPTARG に入れます。オプションがない場合または見つかったオプションにオプション引数がない場合、OPTARG は設定されません。

オプション文字の位置に、*optstring* オペランドに含まれていないオプション文字があると、*name* で指定されているシェル変数が疑問符 (?) 文字に設定されます。この場合、*optstring* の最初の文字がコロン (:) なら、シェル変数 OPTARG は見つかったオプション文字に設定されますが、標準エラー出力には何も出力されません。それ以外の場合、シェル変数 OPTARG は設定されず、標準エラー出力には診断メッセージが出力されます。この状況は、呼び出しアプリケーションに引数を渡す方法においてエラーがあったと判断され、getopts 処理のエラーとはなりません。

オプション引数がない場合は、以下のように処理されます。

- *optstring* の最初の文字がコロン (:) なら、*name* で指定されているシェル変数はコロン文字に設定され、シェル変数 OPTARG は見つかったオプション文字に設定されます。
- それ以外の場合、*name* で指定されているシェル変数は疑問符 (?) 文字に設定され、シェル変数 OPTARG は設定されず、標準エラー出力には診断メッセージが出力されます。この状況は、呼び出しアプリケーションに引数を渡す方法においてエラーがあったと判断され、getopts 処理のエラーとはなりません。この判断を示した診断メッセージが出力されますが、終了ステータスは 0 になります。

オプションの最後にくると、getopts はゼロより大きい戻り値で終了します。シェル変数 OPTIND は最初に現われた非オプション引数のインデックスに設定されます。このとき、最初に現われた – – 引数は、他の非オプション引数が先に現われない場合にオプションとしてみなされます。非オプション引数が 1 つもない場合は、$# + 1 に設定され、*name* 変数は疑問符 (?) 文字になります。オプションの最後を判断するのは、特殊オプション – – がある場合、– 以外で始まる引数が見つかった場合、またはエラーが発生した場合です。

シェル変数 OPTIND と OPTARG は getopts の呼び出し元のローカル変数であり、デフォルトではエクスポートされません。

name オペランドで指定されているシェル変数、OPTIND、および OPTARG は、現在のシェル実行環境に影響を及ぼします。

アプリケーションが OPTIND を 1 に設定すると、新しいパラメータセットが使用できるようになります。現在の位置パラメータまたは新しい *arg* 値です。すべての呼び出しで同じパラメータ (位置パラメータまたは *arg* オペランド) を使おうとしない場合、または OPTIND を1 以外の値に変更した場合、1 つのシェル実行環境で getopts を複数回呼び出すと、結果は保証されません。

sh getopts は、Bourne シェル組み込みコマンドの1 つであり、定位置パラメタの解析およびオプションの妥当性チェックに使用されます。sh(1) を参照してください。このコマンドは、コマンド構文標準規格 (intro(1) のルール 3 〜 10を参照) のうち、該当するルールすべてをサポートします。このコマンドは、getopt コマンドの代わりに使用してください。

optstring には、getopts を使用するコマンドが認識できるオプション文字を入れなければなりません。オプション文字の後にコロンが付いている場合は、そのオプションには 1 つまたは複数の引数があるとみなされます。オプションと引数の間は、空白によって区切られていなければなりません。

getopts は、呼び出されるたびに、次のオプションをシェル変数 *name* に入れ、処理すべき次の引数へのインデックスをシェル変数 OPTIND に入れます。シェルまたはシェル手続きが呼び出されるたびに、OPTIND の値は 1 に初期設定されます。

オプションにオプション引数が必要な場合、getopts はその引数をシェル変数 OPTARG に入れます。

不正なオプションが検出されると、? が *name* に入ります。

オプションの終わりが検出されると、getopts はゼロ以外の終了状態で終了します。特殊オプション – を使用して、オプションの終わりを表すことができます。

デフォルトでは、getopts は定位置パラメタを解析します。getopts コマンド行に追加の引数 (*argument* . . .) が指定されると、getopts は定位置パラメタの代わりにその引数を解析します。

/usr/lib/getoptcvt は、*filename* 内のシェルスクリプトを読み取り、getopt の代わりに getopts を使用できるようにそれを変換し、その結果を標準出力に出力します。

新しいコマンドはすべて、intro(1) で説明しているコマンド構文標準規格に従うようにするために、定位置パラメタの解析、およびオプションがそのコマンドの有効なオプションであるかどうかのチェックに、getopts または getopt を使用しなければなりません。

getopts は、*optstring* に含まれていないオプション文字を検出すると、標準エラー出力にエラーメッセージを表示します。

現在の実装においては、次のようにコマンド構文標準規格 (intro(1) を参照) に従わない使用方法が認められていますが、このような使用方法は、将来、システムのリリースではサポートされないので、使用しないでください。以下のセクションの例と同様に、a および b がオプションであり、o はオプション引数を持つオプションとします。

次の例では、オプション引数を持つオプションは別のオプションと一緒にまとめることはできない、というルール 5 に違反しています。

```
example% cmd – aboxxx filename
```

次の例では、オプション引数を持つオプションの後には、空白がなければならない、というルール 6 に違反しています。

```
example% cmd – ab oxxx filename
```

シェル変数 OPTIND の値を変更したり、異なる引数の集まりを解析したりした場合の結果は不定です。

ksh *arg* が正当なオプションを示しているかどうかをチェックします。*arg* を省略すると、定位置パラメタが使用されます。オプション引数は + または – で始まります。+ または – 以外の文字で始まっているオプション、また – 引数があると、オプションの終わりとみなされます。*optstring* には、getopts が認識する文字を記述します。文字の後に : が続く場合、そのオプションには引数があるとみなされます。オプションと引数とは空白文字で区切ることができます。

+ で始まる場合、getopts は + を起動するたびに*name* 内で見つかった次のオプション文字を設定します。次の *arg* のインデックスは OPTIND に格納されます。オプション引数がある場合は OPTARG に格納されます。

optstring 内で先頭に : がある場合は、getopts は無効なオプション文字を OPTARG に格納し、*name* を ? (未定義のオプションが指定された場合) または : (必要なオプション引数が省略されている場合) に設定します。getopts はエラーメッセージを表示します。オプションがなくなると、終了ステータスは 0 以外になります。

シェルの getopts 組み込みコマンドに関する詳細は、このマニュアルページ内で前述した Bourne シェル (sh) の説明を参照してください。

オペランド 次のオペランドを使用できます。

optstring getopts を呼び出すユーティリティによって識別されるオプション文字が入った文字列。文字の後ろにコロンが付いている場合、そのオプションには引数があることを意味します。引数は別個に指定する必要があります。アプリケーションは、オプション文字とそのオプション引数を別々に指定しなければなりませんが、そのように指定されているかどうかに関わらず、 getopts は、引数を取るオプションに続く文字をその引数と解釈します。オプション引数として NULL を渡すには getopts起動時に明示的に引数とする必要があります。getopt(3C) を参照してください。アプリケーションでは、疑問符 (?) とコロン (:) をオプション文字としては使用できません。英数字以外のオプション文字を使用した場合の結果は保証できません。オプション引数とオプション文字とを別々に指定しないと、 OPTARG の値からオプション文字と – が取り除かれます。オプション文字がわからない

とき、またはオプション引数がないとき、*optstring* の最初の文字によって、 getopts のふるまいが決まります。

name getopts によって、見つかったオプション文字に設定されるシェル変数の名前。

デフォルトでは、getopts は、呼び出し側のシェルプロシージャに引き渡す位置パラメータの構文を解析します。*arg s* が指定されていれば、位置パラメータの代わりに解析されます。が指定されていれば、位置パラメータの代わりに解析されます。

使用法

getopts は現在のシェル実行環境に影響を及ぼすので、通常、シェルに組み込まれています。以下のように、サブシェルまたは別のユーティリティ実行環境から呼び出されると、getopts は呼び出し側の環境のシェル変数には影響しません。

```
(getopts abc value "$@")
 nohup getopts ...
 find . – exec getopts ... \;
```

なお、位置パラメータが変更されても、シェル関数は呼び出し側シェルと OPTIND を共有します。引数の構文を解析するために getopts を使用したい関数は、処理の最初で OPTIND の値を保存し、戻る前に復元するのが一般的です。ただし、呼び出し側シェル用に関数が OPTIND を変更する場合もあります。

使用例

例 1 引数の解析と表示

以下に示すスクリプトは、引数を解析して表示する例です。

```
aflag=
bflag=
while getopts ab: name
do
     case $name in
     a)      aflag=1;;
     b)      bflag=1
             bval="$OPTARG";;
     ?)     printf "Usage: %s: [-a] [-b value] args\n"  $0
            exit 2;;
     esac
done
if [ ! -z "$aflag" ]; then
   printf "Option -a specified\n"
fi
if [ ! -z "$bflag" ]; then
     printf 'Option -b "%s" specified\n' "$bval"
fi
shift $(($OPTIND - 1))
printf "Remaining arguments are: %s\n" "$*"
```

例 2 オプション付きコマンド用の引数の処理

以下のシェルプログラムの一部は、コマンドに対する引数を処理する方法を示す例です。この例では、オプション引数を伴わない -a または -b と、オプション引数を伴う -o を使用します。

例 2 オプション付きコマンド用の引数の処理　(続き)

```
while getopts abo: c
do
      case $c in
     a | b)   FLAG=$c;;
     o)       OARG=$OPTARG;;
     \?)      echo $USAGE
        exit 2;;
     esac
done
shift `expr $OPTIND - 1`
```

例 3 等価コード表現

以下のコードは、すべて同じ処理を実行します。

```
cmd -a -b -o "xxx z yy" filename
cmd -a -b -o "xxx z yy" -- filename
cmd -ab -o xxx,z,yy filename
cmd -ab -o "xxx z yy" filename
cmd -o xxx,z,yy -b -a filename
```

環境

getopts の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

OPTIND　この変数は、次に処理する引数へのインデックスとして getopts が使用します。

OPTARG　この変数は、オプションに引数が使用された場合、その引数を保存するために getopts が使用します。

終了ステータス

以下の終了ステータスが返されます。

0　*optstring* で指定した、または指定しなかったオプションが見つかった。

>0　オプションの並びの終わりを検出した、またはエラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

intro(1), getoptcvt(1), ksh(1), sh(1), getopt(3C), attributes(5), environ(5)

診断

エラーが検出されたとき、*optstring* オペランドの最初の文字がコロン (:) 以外なら、以下の情報が入った診断メッセージが不定フォーマットで標準エラー出力に出力されます。

- 呼び出し側のプログラム名がメッセージ内に示されます。呼び出し側のプログラム名は、`getopts` が呼び出された時点で、シェル特殊パラメータ `0` の値を持ちます。

  basename `"$0"`
  と等価の名前が使われます。

- *optstring* にないオプションが見つかった場合、エラーと認識されて、無効なオプション文字がメッセージ内に示されます。
- オプション引数を必要とするオプションが見つかったが、オプション引数がない場合、エラーと認識されて、無効なオプション文字がメッセージ内に示されます。

名前 | gettxt – テキスト文字列のメッセージデータベースからの検索

形式 | **gettxt** *msgfile* : *msgnum* [*dflt_msg*]

機能説明 | gettxt は、 /usr/lib/locale/locale/LC_MESSAGES というディレクトリ内のメッセージファイルからテキスト文字列を検索します。 locale というディレクトリ名は、テキスト文字列を書き込む際の言語に該当します。 setlocale(3C) を参照してください。

msgfile — *msgnum* を検索する先の /usr/lib/locale/locale/LC_MESSAGES ディレクトリ内のファイル名。 *msgfile* の名前は、最大 14 文字の長さにできますが、 \0 (NULL) や、 / (スラッシュ) または : (コロン) などの ASCII コードを入れることはできません。

msgnum — *msgfile* から検索すべき文字列のシーケンス番号。 *msgfile* 内の文字列は、 *1* から *n* まで連続して番号を付けます。 ここで、 *n* はファイル内の文字列の数です。

dflt_msg — gettxt が *msgnum* の *msgfile* からの検索に失敗した場合に表示すべきデフォルト文字列。 グラフィック以外の文字はアルファベットのエスケープシーケンスとして表記する必要があります。

検索すべきテキスト文字列は、 *msgfile* ファイル内にあります。 このファイルは mkmsgs(1) ユーティリティで作成し、 /usr/lib/locale/locale/LC_MESSAGES ディレクトリの下にインストールします。 LC_MESSAGES という環境変数を設定すると、対応したディレクトリを検索します。 LC_MESSAGES を設定しないと、 LANG 環境変数が使用されます。 LANG を設定しないと、文字列の入ったファイルは、 /usr/lib/locale/C/LC_MESSAGES ディレクトリの下になります。

gettxt が、要求された言語でのメッセージの検索に失敗すると、 /usr/lib/locale/C/LC_MESSAGES/*msgfile* において同一のメッセージを検索します。 これも失敗し、かつ、 *dflt_msg* が存在し NULL 以外である場合、 *dflt_msg* の値を表示します。 *dflt_msg* が存在しないか NULL である場合、 Message not found! ! という文字列を表示します。

使用例 | LANG 環境変数または LC_MESSAGES 環境変数をそのデフォルト値以外に設定していない場合、たとえば、

```
example% gettxt UX:10 "hello world\n"
```

では、 /usr/lib/locale/C/UX/*msgfile* から 10 番目のメッセージを検索します。検索が失敗すると、 "hello world" というメッセージが表示され、その後に復帰改行が続きます。

環境 | gettxt の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGESについての詳細は、 environ(5) を参照してください。

LC_CTYPE gettxt の文字の処理方法を決定します。 LC_CTYPE に有効な値が設定されていると、 gettxt は、その ロケール にあった文字を含むテキストやファイル名を表示および処理できます。 gettxt は拡張 Unix コード(EUC)も表示および処理できます。 この場合、文字は 1 バイト幅、 2 バイト幅、 3 バイト幅のいずれも使用できます。 また、 gettxt は 1 、 2 、 またはそれ以上のカラム幅の EUC 文字も処理することができます。 C ロケール においては、 ISO 8859-1 の文字だけが有効です。

LC_MESSAGES 診断メッセージや情報メッセージの表示方法を決定します。 また、メッセージの言語とスタイル、 そして肯定応答および否定応答の正しい形も決定します。 C ロケール においては、 メッセージはプログラム自身が使用しているデフォルトの形で表示されます(通常、米語)。

ファイル

/usr/lib/locale/C/LC_MESSAGES/*
  mkmsgs(1) が作成するデフォルトのメッセージファイル

/usr/lib/locale/locale/LC_MESSAGES/*
  mkmsgs(1) が作成するさまざまな言語用のメッセージファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWloc
CSI	対応済み

関連項目

exstr(1), mkmsgs(1), srchtxt(1), gettxt(3C), setlocale(3C), attributes(5), environ(5)

名前 | glob – 単語リストを展開するためのシェル組み込み関数

csh | **glob** *wordlist*

csh | glob は *wordlist* に指定される単語リストを使用してファイル名を展開します。 エスケープ \ を認識しない点を除いて、 echo と同様です。 出力のワードは NULL 文字によって区切られます。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | csh(1), echo(1) attributes(5)

名前 | exit, return, goto – シェルの連続した処理を分岐して実行するためのシェル組み込み関数

sh | **exit** [*n*]

return [*n*]

csh | **exit** [(*expr*)]

goto *label*

ksh | ***exit** [*n*]

***return** [*n*]

sh | `exit` はシェルまたはシェルスクリプトを *n* で指定した終了状態で終了させます。 *n* を省略すると、 最後に実行されたコマンドの終了状態 がシェルの終了状態になります。 ファイルの終わりを検出した場合もシェルが終了します。

`return` は関数を、 *n* が示す戻り値で終了させます。 *n* を省略すると、戻り値は最後に実行された コマンドの終了状態になります。

csh | `exit` はシェルまたはシェルスクリプトを終了させ、 状態変数の値または式 *expr* で指定された値が返されます。

`goto` 組み込み関数は *label* をコマンド中で検索の引数として指定します。 シェルは可能なかぎり入力をさかのぼり、 *label*: という形式の行を探します。 *label*: の前には空白文字またはタブ文字がある可能性もあります。 指定された行の次から実行が再開します。 `while` または `for` 組み込みコマンドと、対応する `end` との間にあるラベルへジャンプするとエラーになります。

ksh | `exit` はシェルまたはシェルスクリプトを *n* で指定した終了状態で終了させます。 具体的には、指定した値の最下位 8 ビットが終了状態の値となります。 *n* を省略すると、 最後に実行されたコマンドの終了状態 がシェルの終了状態になります。 トラップ実行中に `exit` が発生した場合、ここで言う最後に実行されたコマンドとは、トラップ呼び出し直前に実行されたコマンドを指します。 なお、 `ignoreeof` オプション (後述の `set` を参照) が有効になっているシェルを除き、 ファイルの終わりを検出した場合もシェルが終了します。

`return` はシェル関数またはドット (.) スクリプトを、 *n* で指定された戻り値で 呼び出し側スクリプトに戻します。 *n* で指定した値の最下位 8 ビットが戻り値となります。 *n* を省略すると、 戻り値は最後に実行された コマンドの戻り値になります。 関数やドット (.) スクリプト実行中以外で `return` を起動すると、 結果は `exit` と同一になります。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている `ksh`(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。

4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

break(1), csh(1), ksh(1), sh(1), attributes(5)

名前 | grep – ファイルにおけるパターンの検索

形式 | **/usr/bin/grep** [-bchilnsvw] *limited-regular-expression* [*filename...*]

/usr/xpg4/bin/grep [-E | -F] [-c | -l | -q] [-bhinsvwx]
-e *pattern_list...* [-f *pattern_file...*] [*file...*]

/usr/xpg4/bin/grep [-E | -F] [-c | -l | -q] [-bhinsvwx]
[-e *pattern_list...*] -f *pattern_file...* [*file...*]

/usr/xpg4/bin/grep [-E | -F] [-c | -l | -q] [-bhinsvwx] *pattern*
[*file...*]

機能説明 | grep ユーティリティは、テキストファイルにおいてパターンを検索し、そのパターンを含むすべての行を出力します。grep は、単純な非決定性アルゴリズムを使用します。

pattern_list での $、*、[、^、|、(、)、\ などの文字の使い方に注意してください。これらの文字はシェルに対して特別な意味を持ちます。 '...' のように *pattern_list* 全体を単一引用符で囲むことをお勧めします。

ファイルを指定しないと、grep は標準入力を入力とします。通常、発見された各行は標準出力に出力されます。入力ファイルが複数ある場合、発見された各行の前にファイル名を出力します。

/usr/bin/grep | /usr/bin/grep ユーティリティは、 regexp(5) のマニュアルページに説明されている正規表現に似た限定正規表現 を使用して、パターンマッチングを行います。

/usr/xpg4/bin/grep | -E と -F の両オプションは、/usr/xpg4/bin/grep の *pattern_list* の解釈の仕方に影響を与えます。-E を指定すると、/usr/xpg4/bin/grep は *pattern_list* を完全な正規表現と解釈します (詳細は -E の説明を参照) 。-F を指定すると、grep は *pattern_list* を固定文字列と解釈します。両方のオプションを省略すると、grep は *pattern_list* を基本正規表現と解釈します (詳細は regex(5) のマニュアルページを参照)。

オプション | 以下のオプションは、/usr/bin/grep と /usr/xpg4/bin/grep で指定できます。

-b　各行の先頭に、その行が発見されたブロックの番号を出力します。これは、ブロック番号を文脈別に見つける際に役立ちます (最初のブロックは 0 です)。

-c　パターンを含む行の数だけを出力します。

-h　複数のファイルを検索するときにファイル名を出力しません。

-i　比較中に大文字と小文字を区別しません。

-l　一致した行が 1 行でもあるファイルのファイル名だけを、1 ファイルずつ復帰改行で区切って出力します。パターンが 2 回以上見つかっても、ファイル名の出力は 1 回だけです。

-n　各行の先頭に、ファイルにおけるその行の行番号を出力します (最初の行は 1 です) 。

`-s` 存在しないファイルや読めないファイルに対してのエラーメッセージを抑制します。

`-v` パターンを含む行を除いたすべての行を出力します。

`-w` 表現が \< と \> で囲まれている場合のように、表現を単語として検索します。

/usr/xpg4/bin/grep 以下のオプションは `/usr/xpg4/bin/grep` のみで指定できます。

`-e`*pattern_list* 入力の検索時に用いるパターンを 1 つ以上指定します。複数のパターンを *pattern_list* に指定する場合、各々を復帰改行文字 (NEWLINE) で区切る必要があります。空のパターンは、復帰改行文字を 2 つ連続して指定することにより表せます。 `-E` または `-F` オプションを指定した場合を除き、各パターンは基本正規表現として 扱われます。複数個の `-e` および `-f` オプションを指定することが可能です。行のマッチングにおいて、 指定されたパターンはすべて使用されますが、 評価の順序は未定です。

`-E` 完全な正規表現を使ってマッチングを行います。 指定した各パターンは完全な正規表現として扱われます。 ある完全な正規表現の全体が入力行と一致したとき、 その行は一致したことになります。 空の完全な正規表現は、すべての行と一致します。 各パターンは、 `regex`(5) のマニュアルページに記述されているように、完全な正規表現 として解釈されます。ただし \(や \) は例外で、以下の指定も含まれます。

1. 完全な正規表現の後に + が付加されているとき。これはその表現の 1 回以上の出現に一致します。
2. 完全な正規表現の後に ? が付加されているとき。これはその表現の 0 または 1 回の出現に一致します。
3. 複数の完全な正規表現が | または復帰改行文字で区切られているとき。 これは、いずれかの表現と一致する文字列に一致します。
4. 完全な正規表現が括弧 () で囲まれているとき。これはグループ化を表します。 これらの演算子を優先順位の高い順に並べると、まず `[ ]`、次に `* ? +`、次が連結、その次が | と復帰改行となります。

`-f` *pattern_file* *pattern_file* が示すパス名で指定したファイルから、 1 つまたは複数のパターンを読み込みます。 *pattern_file* 中のパターンの終了は、復帰改行文字により表されます。 空のパターンは、*pattern_file* 中に空の行を記述することにより表せます。 `-E` または `-F` オプションを指定した場合を除き、各パターンは基本正規表現として 扱われます。

`-F` 固定文字列を使ってマッチングを行います。 指定された各パターンを、正規表現ではなく文字列と見なします。 いずれかのパターンが連続したバイトとして入力行の中に現れた場合、 その行は一

致します。空の文字列はすべての入力行と一致します。 詳細については fgrep(1) を参照してください。

`-q` 行が一致したかどうかに関わらず、 標準出力には何も書き出しません。入力行に一致した行があった場合、 0 の終了ステータスで処理を終了します。

`-x` 固定文字列全体または正規表現と完全に一致する入力行だけを、一致する行とみなします。

オペランド

以下のオペランドを指定できます。

file パターンを検索するファイルのパス名。このオペランドを省略すると、標準入力が用いられます。

/usr/bin/grep

pattern 入力の検索時に用いるパターンを指定します。

/usr/xpg4/bin/grep

pattern 入力の検索時に用いる 1 つまたは複数のパターンを指定します。このオペランドは、`-e` *pattern_list* が指定されたものとして扱われます。

使用法

`-e` *pattern_list* オプションの指定は、 *pattern_list* オペランドと同じ結果をもたらしますが、 *pattern_list* がハイフン区切り文字で始まる場合には便利な方法です。 また、複数のパターンを別個の引数として用いたい場合にも便利な方法です。

`-e` と `-f` の両オプションは、それぞれ複数個指定できます。 入力テキスト行のマッチングにおいて、 `grep` は指定されたすべてのパターンを使用します。 なお評価の順序は指定されないので注意してください。 空の文字列をパターンとして使用したいときは、 すべての行と一致するそのパターンを最初に記述し、 それ以外のパターンをうまく無視させることができます。

`-q` オプションは、一群のファイル中に特定のパターンまたは文字列が存在しているかどうかを簡単に見分ける手段として使えます。複数のファイルを検索する場合、最初に一致するものを見つけたところで 処理を終了するので、性能が向上します。また、引数として複数のファイルを指定する場合でも、それほど注意は 必要ありません。なぜなら、前の方の `file` オペランドで指定したファイルでアクセスエラーや読み取りエラーが 発生しても、そのあとで一致するものが見つかれば、`grep` は 0 の終了ステータスで終了するためです。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `grep` の動作については、`largefile`(5) を参照してください。

使用例

例 1 ある文字列をすべて検索する

以下の例は、 `text.mm` というファイルを検索し、 `Posix` という文字列 (大文字と小文字は区別しない) が現れている箇所すべてを行番号とともに出力します。

```
example% /usr/bin/grep -i -n posix text.mm
```

例 2 空の行を検索する

次の例は、標準入力中の空の行を見つけるものです。

```
example% /usr/bin/grep ^$
```

これは次のように指定することもできます。

```
example% /usr/bin/grep -v .
```

例 3 ある文字列を含む行を検索する

以下の例はすべて、abc と def という 2 つの文字列のどちらかまたは両方を含んでいる行をすべて出力します。

```
example% /usr/xpg4/bin/grep 'abc
def'
example% /usr/xpg4/bin/grep -e 'abc
def'
example% /usr/xpg4/bin/grep -e 'abc' -e 'def'
example% /usr/xpg4/bin/grep -E 'abc|def'
example% /usr/xpg4/bin/grep -E -e 'abc|def'
example% /usr/xpg4/bin/grep -E -e 'abc' -e 'def'
example% /usr/xpg4/bin/grep -E 'abc
def'
example% /usr/xpg4/bin/grep -E -e 'abc
def'
example% /usr/xpg4/bin/grep -F -e 'abc' -e 'def'
example% /usr/xpg4/bin/grep -F 'abc
def'
example% /usr/xpg4/bin/grep -F -e 'abc
def'
```

例 4 ある文字列と完全に一致する行を検索する

以下の 2 つの例はともに、abc または def のどちらかの文字列と完全に一致している行をすべて出力します。

```
example% /usr/xpg4/bin/grep -E '^abc$ ^def$'
example% /usr/xpg4/bin/grep -F -x 'abc def'
```

環境

grep の実行に影響を与える環境変数 LANG、LC_ALL、LC_COLLATE、 LC_CTYPE、LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	一致するものが 1 つ以上見つかった
1	一致するものが 1 つも見つからなかった
2	(一致するものが見つかった場合でも) 構文エラーが検出された、またはアクセスできないファイルがあった

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/grep

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/grep

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み
インタフェースの安定性	標準

関連項目 `egrep`(1), `fgrep`(1), `sed`(1), `sh`(1), `attributes`(5), `environ`(5), `largefile`(5), `regex`(5), `regexp`(5), `XPG4`(5)

/usr/bin/grep

1 行は、 仮想記憶に使用できるサイズに制限されています。 入力行中に空文字がある場合、 `grep` は最初の空文字までマッチングを行います。最初の空文字までのマッチングで一致した場合は、行全体が出力されます。

/usr/xpg4/bin/grep

入力ファイルが `LINE_MAX` バイトよりも長い行を含んでいる場合、 またはバイナリデータを含んでいる場合の結果は特定されません。 `LINE_MAX` は `/usr/include/limits.h` で定義されます。

名前 | groups – ユーザーのグループメンバーシップの出力

形式 | **groups** [*user*...]

機能説明 | groups コマンドは、このコマンドを実行したユーザーの、または指定したユーザーの所属グループを標準出力に書き込みます。各ユーザーは /etc/passwd 内に指定されているグループに属し、さらに /etc/group 内に指定されている他のグループにも属している可能性があります。/etc/passwd では、グループの ID (gid) は数値で指定されています。groups コマンドは gid をグループ名に変換して出力します。

使用例 | 出力の形式は次のとおりです。

```
example% groups tester01 tester02
tester01 : staff
tester02 : staff
example%
```

ファイル | /etc/passwd

/etc/group

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | group(4), passwd(4), attributes(5)

名前 | hash, rehash, unhash, hashstat – ディレクトリの内容の内部ハッシュテーブルの評価

形式 | **/usr/bin/hash** [*utility*]

/usr/bin/hash [-r]

sh | **hash** [-r] [*name*...]

csh | **rehash**

unhash

hashstat

ksh | **hash** [*name*...]

/usr/bin/hash | /usr/bin/hash ユーティリティは、見つかったユーティリティの位置を現在のシェル環境が どのように記憶するか、その記憶方法を変更します。 具体的には、指定された引数に従って、新たなユーティリティを ユーティリティ位置リストに追加したり、リストの内容を消去したりします。引数を指定しないと、リストの内容が報告されます。

シェルの組み込みユーティリティとして提供されているものについては、 hash は報告対象としません。

sh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。-r オプションを指定すると、シェルは記憶したすべての位置を破棄します。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。 この出力表示において、 *Hits* の列はシェルプロセスによってコマンドが呼び出された回数を表します。*Cost* は、検索パスのコマンドを見つけるのに必要な作業量です。 コマンドが検索パスの「相対」ディレクトリにある場合、 そのディレクトリの変更後にそのコマンドが格納された位置が再計算されます。 この再計算が行われる予定のコマンドに対しては、*Hits* 情報の隣にアスタリスク (*) が示されます。 *Cost* の値は、再計算が行われるたびに増加します。

csh | rehash は新しく追加されたコマンドに合わせて、 path 環境の変数に記憶しているディレクトリの内容の内部ハッシュテーブルを再計算します。

unhash は内部ハッシュテーブルを使用不能にします。

hashstat は内部ハッシュテーブルがコマンドの検索 (実行を伴わない) にどの程度有効に働いて いるかを示す統計情報を出力します。 ハッシュ関数がヒットの可能性を示しているパス の各構成要素と、' / ' で始まらない各構成要素について、実行しようとします。

ksh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。引数が与えられない場合、hash は記憶されたコマンドに関する情報を表示します。

オペランド | 以下のオペランドを指定できます。

utility 位置のリスト内で検索する、あるいはリストに 追加するユーティリティ名。

出力　引数を 1 つも指定しないと、hash の標準出力が使用されます。その形式は決まっていませんが、 現在のシェル環境用の位置リストにある各ユーティリティの パス名は含まれています。このリストは、以前に実行された hash 呼び出し中に指定されていたユーティリティをすべて含んでおり、さらに通常のコマンド検索プロセスで呼び出され見つけられた ユーティリティを含んでいることもあります。

環境　hash の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、 NLSPATH についての詳細は、environ(5) を参照してください。

PATH　*utility* の位置を指定します。

終了ステータス　以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目　csh(1), ksh(1), sh(1), attributes(5), environ(5)

名前 | hash, rehash, unhash, hashstat – ディレクトリの内容の内部ハッシュテーブルの評価

形式 | **/usr/bin/hash** [*utility*]

/usr/bin/hash [-r]

sh | **hash** [-r] [*name*...]

csh | **rehash**

unhash

hashstat

ksh | **hash** [*name*...]

/usr/bin/hash | /usr/bin/hash ユーティリティは、見つかったユーティリティの位置を現在のシェル環境が どのように記憶するか、その記憶方法を変更します。 具体的には、指定された引数に従って、新たなユーティリティを ユーティリティ位置リストに追加したり、リストの内容を消去したりします。引数を指定しないと、リストの内容が報告されます。

シェルの組み込みユーティリティとして提供されているものについては、 hash は報告対象としません。

sh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。-r オプションを指定すると、シェルは記憶したすべての位置を破棄します。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。 この出力表示において、 *Hits* の列はシェルプロセスによってコマンドが呼び出された回数を表します。*Cost* は、検索パスのコマンドを見つけるのに必要な作業量です。 コマンドが検索パスの「相対」ディレクトリにある場合、 そのディレクトリの変更後にそのコマンドが格納された位置が再計算されます。 この再計算が行われる予定のコマンドに対しては、*Hits* 情報の隣にアスタリスク (*) が示されます。 *Cost* の値は、再計算が行われるたびに増加します。

csh | rehash は新しく追加されたコマンドに合わせて、 path 環境の変数に記憶しているディレクトリの内容の内部ハッシュテーブルを再計算します。

unhash は内部ハッシュテーブルを使用不能にします。

hashstat は内部ハッシュテーブルがコマンドの検索 (実行を伴わない) にどの程度有効に働いて いるかを示す統計情報を出力します。 ハッシュ関数がヒットの可能性を示しているパス の各構成要素と、' / ' で始まらない各構成要素について、実行しようとします。

ksh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。引数が与えられない場合、hash は記憶されたコマンドに関する情報を表示します。

オペランド | 以下のオペランドを指定できます。

utility 位置のリスト内で検索する、あるいはリストに 追加するユーティリティ名。

出力　引数を 1 つも指定しないと、hash の標準出力が使用されます。その形式は決まっていませんが、 現在のシェル環境用の位置リストにある各ユーティリティの パス名は含まれています。このリストは、以前に実行された hash 呼び出し中に指定されていたユーティリティをすべて含んでおり、さらに通常のコマンド検索プロセスで呼び出され見つけられた ユーティリティを含んでいることもあります。

環境　hash の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、 NLSPATH についての詳細は、environ(5) を参照してください。

PATH　*utility* の位置を指定します。

終了ステータス　以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目　csh(1), ksh(1), sh(1), attributes(5), environ(5)

名前 | head – ファイルの最初の数行の表示

形式 | **head** [-*number* | -n *number*] [*filename...*]

機能説明 | head ユーティリティは、*filename* で指定された各ファイルについて、先頭の何行か (行数は *number* で指定) を標準出力に出力します。*filename* を指定しない場合、head は、標準入力の最初の数行を出力します。 *number* のデフォルト値は 10 行です。

複数のファイルが指定された場合、各ファイルの内容の先頭に次のような行が表示されます。

==> *filename* <==

したがって、複数の小さいファイルの内容を、各ファイルを識別できるように表示する際、一般的な方法として、次のようにこのコマンドを使用することができます。

```
example% head -9999 filename1 filename2 . . .
```

オプション | 以下のオプションを指定できます。

-n *number* — 各入力ファイルの先頭の何行を標準出力に書き出すか、 その行数を *number* で指定します。 *number* の値は正の整数でなければなりません。

- *number* — *number* は正の整数で、意味は上記 -n *number* 指定と同じです。

オプションを省略すると、 head は -n 10 というオプションが指定されたものとみなします。

オペランド | 以下のオペランドを指定できます。

filename — 入力ファイルのパス名。このオペランドを 1 つも指定しないと、標準入力が用いられます。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の head の動作については、largefile(5) を参照してください。

使用例 | **例 1** ディレクトリ中のすべてのファイルの先頭 10 行を出力する

ディレクトリ中のすべてのファイル (ピリオドで始まるものを除く) の先頭の 10 行を出力する例を以下に示します。

```
example% head *
```

環境 | head の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 — 正常終了

>0 — エラーが発生した

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | cat(1), more(1), pg(1), tail(1), attributes(5), environ(5), largefile(5)

名前 | ckdate, errdate, helpdate, valdate – 日付の入力要求とその検証

形式 | **ckdate** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errdate [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helpdate [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valdate [-f *format*] *input*

機能説明 | ckdate ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに日付の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザーの応答は、date コマンドで定義されている書式に一致したものでなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckdate コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errdate (エラーメッセージを書式化して表示する)、helpdate (ヘルプメッセージを書式化して表示する)、および valdate (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errdate および helpdate の各モジュールに format が定義されている場合、メッセージには、指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は、どのような書式も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。指定できる書式とその定義は、次のとおりです。

- %b = 省略された月の名前 (jan、feb、mar)
- %B = 完全な月の名前 %d = 日 (01 ～ 31)
- %D = %m/%d/%y の形式の日付 (デフォルトの書式)
- %e = 日 (1 ～ 31、1 桁の数字の前にはブランクが挿入されます)

`%h` =	省略形の月の名前。`%b`%と同じ
`%m` =	月 (01 〜 12)
`%y` =	西暦中の年 (たとえば、89)
`%Y` =	CCYY の形式の年 (たとえば、1989)

`-h` *help* — `help` をヘルプメッセージとして定義します。

`-k` *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

`-p` *prompt* — *prompt* をプロンプトメッセージとして定義します。

`-Q` — 有効な応答として終了 (quit) を使用できないように指定します。

`-s` *signal* — 終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル `signal` を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-W` *width* — プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、`width` の行長に書式化します。

オペランド

次のオペランドを指定できます。

input — 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0` — 正常な終了

`1` — 入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`3` — ユーザー終了 (quit)

`4` — 不正な書式引数

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckdate` のデフォルトのプロンプトは、次のとおりです。

```
Enter the date [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter a date.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter a date. Format is <format>.
```

終了 (quit) オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valdate` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckgid, errgid, helpgid, valgid – グループ ID の入力要求とその検証

形式 | **ckgid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errgid [-W *width*] [-e *error*]

/usr/sadm/bin/helpgid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valgid *input*

機能説明 | ckgid は、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに既存のグループ ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckgid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errgid (エラーメッセージを書式化して表示する)、helpgid (ヘルプメッセージを書式化して表示する)、および valgid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような書式基準も満たす必要はありません。

-e *error* — *error* エラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するように指定します。

-m — ユーザーがヘルプを要求した場合、あるいはユーザーの入力が不正な場合は、グループ名の一覧を表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/group` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckgid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the name of an existing group [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

デフォルトのヘルプメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valgid` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckint, errint, helpint, valint – 整数値の入力要求とその検証

形式 | **ckint** [-Q] [-W *width*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errint [-W *width*] [-b *base*] [-e *error*]

/usr/sadm/bin/helpint [-W *width*] [-b *base*] [-h *help*]

/usr/sadm/bin/valint [-b *base*] *input*

機能説明 | ckint ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白（復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckint コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errint (エラーメッセージを書式化して表示する)、helpint (ヘルプメッセージを書式化して表示する)、および valint (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errint と helpint の各モジュールに *base* が定義されている場合、メッセージには入力する整数の基数が含まれます。

オプション | 次のオプションを指定できます。

-b *base* — 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。

-d *default* — *default* をフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

`-s` *signal*　終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-W` *width*　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　基数 *base* の基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　正常な終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckint` のデフォルトのプロンプトは、次のとおりです (基数は 10)。

```
Enter an integer [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
ERROR - Please enter an integer.
```

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
Please enter an integer.
```

基数を 10 以外の数に設定した場合は、上記のメッセージは "integer" から "base *base* integer" に変更されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valint` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckitem, erritem, helpitem – メニューの作成、およびメニュー項目の入力要求とその検証

形式 | **ckitem** [-Q] [-W *width*] [-uno] [-f *filename*] [-l *label*] [[-i *invis*] [,...]] [-m *max*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]] [*choice* [...]]

/usr/sadm/bin/erritem [-W *width*] [-e *error*] [*choice* [..]]

/usr/sadm/bin/helpitem [-W *width*] [-h *help*] [*choice* [..]]

機能説明 | ckitem ユーティリティは、メニューを作成し、ユーザーにプロンプトを出力してメニュー項目から 1 つの項目を選択するように促します。さらに、ユーザーの応答入力を検証します。このコマンドのオプションでは、プロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) を定義します。

デフォルトでは、メニューは書式化されており、各項目の前に番号が付けられ、端末上に複数の列で出力されます。列の長さは、選択した最大長の項目によって決まります。項目はアルファベット順に表示されます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckitem コマンドには、2 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erritem (エラーメッセージを書式化して表示する) と helpitem (ヘルプメッセージを書式化して表示する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。これらのモジュールに *choice* が定義されている場合、メッセージには利用可能なメニューの選択項目が含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *filename* — *filename* を表示されるメニュー項目のリストを含むファイルとして定義します。(このファイルの書式は、token<tab>description です。ポンド記号 (#) で始まる行は注釈として指定され、無視されます。)

-h *help* — *help* をヘルプメッセージとして定義します。

-i *invis* — 非表示メニュー項目 (メニューに表示されない選択項目) を定義します。(たとえば、非表示項目として使用される "all" は、有効なオ

プションですが、メニューには表示されません。任意の数の非表示項目を定義できます。) 非表示項目があることを、プロンプトメッセージまたはヘルプメッセージのどちらかで、ユーザーに知らせるようにします。

`-k` *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

`-l` *label* メニューの上に表示するラベル、*label* を定義します。

`-m` *max* ユーザーが選択できるメニュー項目の最大数を定義します。デフォルト値は 1 です。

`-n` メニュー項目をアルファベット順で表示しないようにします。

`-o` 1 つのメニュートークンだけが返されるようにします。

`-p` *prompt* *prompt* をプロンプトメッセージとして定義します。

`-Q` 有効な応答として終了 (quit) を使用できないようにします。

`-s` *signal* 終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-u` メニュー項目に番号を付けずに表示するようにします。

`-W` *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

choice メニュー項目を定義します。項目は空白か復帰改行で区切ります。

終了ステータス

次の終了ステータスが返されます。

`0` 正常な終了

`1` 入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、`-f` オプションでファイルが開けない、あるいは使用法に誤りがあった

`3` ユーザー終了 (quit)

`4` 選択すべき項目がない

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

ユーザーは、メニュー項目に番号が付いている場合はその項目の番号を、あるいは、その項目を一意に識別するのに必要な長さの文字列を入力できます。長いメニューはページに分割され、各ページには 10 個の項目が表示されます。

メニュー項目が、`-f` オプションで指定したファイルとコマンド行の両方に定義されている場合、メニュー項目は通常、アルファベット順に表示されます。ただし、アルファベット順での表示を抑制する `-n` オプションが使用されている場合は、ファイルに定義された項目が最初に表示され、次にコマンド行に定義されたオプションが表示されます。

`ckitem` のデフォルトのプロンプトは次のとおりです。

```
Enter selection [?,??,q]:
```

1 つの疑問符はヘルプメッセージを表示してから、プロンプトを再表示します。2 つの疑問符は、ヘルプメッセージを表示してから、メニューラベル、メニュー、およびプロンプトを再表示します。

番号を入力した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Bad numeric choice specification
```

文字列を入力した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Entry does not match available menu selection.
Enter the number of the menu item you wish to select,
the token which is associated with the menu item,
or a partial string which uniquely identifies the token
for the menu item. Enter ?? to reprint the menu.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Enter the number of the menu item you wish to select,
the token which is associated with the menu item,
or a partial string which uniquely identifies the token for
the menu item. Enter ? to reprint the menu.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。

名前 | ckpath, errpath, helppath, valpath – パス名の入力要求とその検証

形式 | **ckpath** [-Q] [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errpath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-e *error*]

/usr/sadm/bin/helppath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-h *help*]

/usr/sadm/bin/valpath [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] *input*

機能説明 | ckpath ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーにパス名の入力を促すプロンプトメッセージ、ヘルプメッセージおよびエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

パス名は、最初のオプショングループで指定した基準に従わなければなりません。基準が定義されていない場合、パス名はまだ存在していない通常のファイルに対するものでなければなりません。-a (絶対) と -l (相対) のいずれも指定されていない場合は、どちらかが有効であるとみなされます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckpath コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errpath (標準出力上にエラーメッセージを書式化して表示する)、helppath (標準出力上にヘルプメッセージを書式化して表示する)、および valpath (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-a　パス名は絶対パスでなければなりません。

-b　パス名はブロック型特殊ファイルでなければなりません。

-c　パス名は文字型特殊ファイルでなければなりません。

-d *default*　*default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

`-e` *error*	*error* をエラーメッセージとして定義します。
`-f`	パス名は通常ファイルでなければなりません。
`-h` *help*	*help* をヘルプメッセージとして定義します。
`-k` *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
`-l`	パス名は相対パスでなければなりません。
`-n`	パス名が存在していてはなりません (新規のものでなければなりません)。
`-o`	パス名が存在していなければなりません (既存のものでなければなりません)。
`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (`quit`) を使用できないようにします。
`-r`	パス名は読み取り可能でなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-t`	パス名は作成可能 (処理可能) でなければなりません。パス名が、まだ存在していない場合には作成されます。
`-w`	パス名は書き込み可能でなければなりません。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。
`-x`	パス名は実行可能でなければなりません。
`-y`	パス名はディレクトリでなければなりません。
`-z`	パス名には、`0` バイトを超えるサイズのファイルがなければなりません。

オペランド

次のオペランドを指定できます。

input	検証オプションと照合される入力

使用例

`ckpath` のデフォルトメッセージのテキストは、使用されている基準オプションによって異なります。

例 1 デフォルトのプロンプト

`ckpath` (`-a` オプションを使用) のデフォルトプロンプトの例は、次のとおりです。

```
example% ckpath -a
Enter an absolute pathname [?,q]
```

例 1 デフォルトのプロンプト　(続き)

例 2 デフォルトのエラーメッセージ

デフォルトのエラーメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/errpath -a
ERROR: A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 3 デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/helppath -a
A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 4 終了オプション

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード 3 と共に q が返されます。終了を入力すると、後に復帰改行が続きます。

例 5 valpath モジュールの使用

valpath モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valpath
usage: valpath [-[a|l][b|c|f|y][n|[o|z]]rtwx] input
    .
    .
    .
```

終了ステータス

次の終了ステータスが返されます。

0	正常終了
1	入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった
2	同時に指定できないオプション
3	ユーザー終了 (quit)
4	同時に指定できないオプション

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 face(1), signal(3HEAD), attributes(5)

名前 | ckrange, errange, helprange, valrange – 整数の入力要求とその検証

形式 | **ckrange** [-Q] [-W *width*] [-l *lower*] [-u *upper*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errange [-W *width*] [-e *error*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/helprange [-W *width*] [-h *help*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/valrange [-l *lower*] [-u *upper*] [-b *base*] *input*

機能説明 | ckrange ユーティリティは、ユーザーに指定範囲内の整数の入力を要求して、ユーザーの応答が有効かどうかを検証します。このユーティリティでは、ユーザーに指定範囲内の整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

また、このコマンドは、有効な入力範囲も定義します。下限または上限のいずれかが定義されていない場合、範囲は一方の限界だけに制限されます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckrange コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errrange (標準出力上にエラーメッセージを書式化して表示する)、helprange (標準出力上にヘルプメッセージを書式化して表示する)、および valrange (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

注：*input* 引数として負の値を指定すると、valrange 内の getopt で混乱が生じます。引数の前に - を付けると、getopt は処理を停止します。getopt のパラメータ処理については、getopt(1) および intro(1) のマニュアルページを参照してください。getopt は、位置指定パラメータを解析して、有効なオプションかどうかを検査するために使用されます。

オプション | 次のオプションを指定できます。

-b *base* — 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。基数を変換するには strtol(3C) を使用します。出力は常に、基数を 10 として実行されます。

-d *default*	*default* をデフォルト値として定義します。*default* は strtol(3C) を使用して指定した基数に変換できます。指定した基数に無効な文字があると、strtol はエラーを示すことなく変換を終了します。
-e *error*	*error* をエラーメッセージとして定義します。
-h *help*	*help* をヘルプメッセージとして定義します。
-k *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
-l *lower*	*lower* を範囲の下限として定義します。デフォルト値は、マシンの負の最大整数です。
-p *prompt*	*prompt* をプロンプトメッセージとして定義します。
-Q	有効な応答として終了 (quit) を使用できないようにします。
-s *signal*	終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。
-u *upper*	*upper* を範囲の上限として定義します。デフォルト値は、マシンの正の最大整数です。
-W *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定します。

input	上限、下限、および基数と照合される入力

使用例

例 1 デフォルトのプロンプト

ckrange のデフォルトのプロンプトは、次のとおりです (基数は 10)

```
example% ckrange Enter an integer between lower_bound and
upper_bound [lower_bound–upper_bound,?,q]:
```

例 2 デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/errange
ERROR: Please enter an integer between lower_bound \
     and upper_bound.
```

例 3 デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/helprange
Please enter an integer between lower_bound and upper_bound.
```

例 **3** デフォルトのヘルプメッセージ　*(続き)*

例 **4** 基数を 10 以外の数に変更した場合のメッセージ

基数を 10 以外の数に設定した場合、メッセージは "integer" から "base *base* integer" に変更されます。次に例を示します。

```
example% /usr/sadm/bin/helprange -b 36
```

例 **5** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **6** valrange モジュールの使用

`valrange` モジュールは、標準エラー出力に使用法に関するメッセージを作成します。正常終了したな場合は `0`、失敗した場合には `0` 以外の値を返します。

```
example% /usr/sadm/bin/valrange
usage: valrange [-l lower] [-u upper] [-b base] input
```

終了ステータス

次の終了ステータスが返されます。

`0`　正常終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`2`　使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`intro`(1), `face`(1), `getopt`(1), `strtol`(3C), `attributes`(5), `signal`(3HEAD)

名前 | ckstr, errstr, helpstr, valstr – 文字列の入力要求とその検証

形式 |

ckstr [-Q] [-W *width*] [[-r *regexp*] [...]] [-l *length*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [- s *signal*]]

/usr/sadm/bin/errstr [-W *width*] [-e *error*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/helpstr [-W *width*] [-h *help*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/valstr [-l *length*] [[-r *regexp*] [...]] *input*

機能説明 | ckstr ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、文字列の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

このコマンドから返される応答は、定義済みの正規表現に一致していなければならず、指定された長さ以内でなければなりません。正規表現が指定されていない場合、有効な入力は、内部に空白を含まず、また先行や後続する空白がない、指定された長さ以内の文字列でなければなりません。長さが指定されていない場合、長さは検査されません。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckstr コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errstr (標準出力上にエラーメッセージを書式化して表示する)、helpstr (標準出力上にヘルプメッセージを書式化して表示する)、および valstr (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-l *length* — 入力の最大長を指定します。

`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (quit) を使用できないようにします。
`-r` *regexp*	入力を検証するときの基準となる正規表現、 *regexp* を指定します。空白を含めることができます。複数の式が定義されている場合、応答はその内のいずれかに一致しなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを *width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input	書式の長さや正規表現の基準に対して検証される入力

使用例

例 **1** デフォルトのプロンプト

`ckstr` のデフォルトのプロンプトは、次のとおりです。

```
example% ckstrEnter an appropriate value [?,q]:
```

例 **2** デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、適用される妥当性検査のタイプによって異なります。ユーザーには、長さまたはパターン照合のいずれかが失敗したことが通知されます。デフォルトのエラーメッセージは、次のとおりです。

```
example% /usr/sadm/bin/errstr
ERROR: Please enter a string which contains no embedded,
leading or trailing spaces or tabs.
```

例 **3** デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージも、適用される妥当性検査のタイプによって異なります。正規表現が定義されている場合、メッセージは次のようになります。

```
example% /usr/sadm/bin/helpstr -r regexp
Please enter a string which matches the following pattern:
regexp
```

他のメッセージは、文字列の長さの要件と定義を指定します。

例 **4** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **5** valstr モジュールの使用

valstr モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valstr
usage: valstr [-l length] [[-r regexp] [ . . . ]] input
```

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは、使用法に誤りがあった

2 無効な正規表現

3 ユーザー終了 (quit)

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

face(1), signal(3HEAD), attributes(5)

名前 | cktime, errtime, helptime, valtime – 時刻の入力要求とその検証

形式 | **cktime** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errtime [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helptime [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valtime [-f *format*] *input*

機能説明 | cktime ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに時刻の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザー応答は、時刻について定義されている書式に一致しなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

cktime コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errtime (エラーメッセージを書式化して表示する) と helptime (ヘルプメッセージを書式化して表示する) と、valtime (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errtime および helptime の各モジュールに format が定義されている場合、メッセージには指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。次の書式と定義を指定できます。

```
%H  =  hour (00 - 23)
%I  =  hour (00 - 12)
%M  =  minute (00 - 59)
%p  =  ante meridian or post meridian
%r  =  time as %I:%M:%S %p
%R  =  time as %H:%M (the default format)
%S  =  seconds (00 - 59)
%T  =  time as %H:%M:%S
```

`-h` *help*	*help* をヘルプメッセージとして定義します。
`-k` *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (quit)を使用できないようにします。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`	正常終了
`1`	入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった
`3`	ユーザー終了 (quit)
`4`	無効な書式引数

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`cktime` のデフォルトのプロンプトは、次のとおりです。

```
Enter a time of day [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter the time of day.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the time of day.  Format is <format>.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。`valtime` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckuid, erruid, helpuid, valuid – ユーザー ID の入力要求とその検証

形式 | **ckuid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erruid [-W *width*] [-e *error*]

/usr/sadm/bin/helpuid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valuid *input*

機能説明 | ckuid ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに既存のユーザー ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckuid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erruid (エラーメッセージを書式化して表示する) と helpuid (ヘルプメッセージを書式化して表示する) と、valuid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-m — ヘルプが要求された場合、またはユーザーがエラーを犯した場合は、すべてのログインのリストを表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/passwd` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、または使用法に誤りがあった

`2`　　使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、 `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckuid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the login name of an existing user [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR:  Please enter one of the following login names:  <List>
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのヘルプメッセージは、次のとおりです。

```
Please enter one of the following login names:  <List>
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。 `valuid` モジュールは出力を何も生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckyorn, erryorn, helpyorn, valyorn – yes/no の入力要求とその検証

形式 | **ckyorn** [-Q] [-W *width*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erryorn [-W *width*] [-e *error*]

/usr/sadm/bin/helpyorn [-W *width*] [-h *help*]

/usr/sadm/bin/valyorn *input*

機能説明 | ckyorn は、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、「はい (yes)」または「いいえ (no)」の応答を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckyorn コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erryorn (エラーメッセージを書式化して表示する) と helpyorn (ヘルプメッセージを書式化して表示する) と、valyorn (応答を検証する) です。これらのモジュールは、FACE オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* デフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

-W *width* — プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input y、yes、または n、no (大文字小文字の任意の組み合わせ) に対して検証される入力

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった

2 使用法に誤りがあった

3 ユーザー終了 (quit)

属性

次の属性については、 attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

attributes(5)

注意事項

ckyorn のデフォルトのプロンプトは、次のとおりです。

```
Yes or No [y,n,?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter yes or no.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
To respond in the affirmative, enter y, yes, Y, or YES.
To respond in the negative, enter n, no, N, or NO.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード 3 と共に q が返されます。valyorn モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前 | history, fc – コマンドの履歴リストの処理

形式 | **/usr/bin/fc** [*first* [*last*]]

/usr/bin/fc -l [-nr] [*first* [*last*]]

/usr/bin/fc -s [*old* = *new*] [*first*]

csh | **history** [-hr] [*n*]

ksh | **fc** -e - [*old* = *new*] [*command*]

fc [-e *ename*] [-nlr] [*first* [*last*]]

/usr/bin/fc

fc ユーティリティは、以前に対話型 sh に入力されたコマンドの一覧表示、または 編集と再実行を行います。

コマンドの履歴リストは、番号によってコマンドを参照します。 リスト中の最初の番号は任意に選択できます。 コマンド番号とコマンドとの関係は変わることはありません。 ただしユーザーがログインしたときに、どのプロセスも リストをアクセスしていない場合を除きます。 この場合には、システムはコマンド番号をリセットするので、 保存されている最も古いコマンドに新たな番号 (通常は 1) が与えられます。コマンド番号が、HISTSIZE の値と 128 の大きい方の値に到達したとき、シェルは番号を循環させるので、次のコマンドが再び最小値 (通常は 1) から始まります。このように番号が循環しても、fc は時間の順序は把握しています。 たとえば 4 つのコマンドにそれぞれ 32 766 、32 767、 1 (循環後)、 2 という番号が与えられたとき、 32 767 は 1 よりも大きい値であっても 1 以前に実行されたものであると認識しています。

コマンドが編集されたとき (-l オプション省略時) 、結果として生成された行が履歴リストの最後尾に記録され、sh によって再実行されます。編集作業を起動した fc コマンド自身は履歴リストには記録されません。 エディタがゼロ以外の終了ステータスを返した場合、 履歴リストへの記録や再実行は抑止されます。fc コマンドに指定したコマンド行変数割り当てや リダイレクト演算子は、fc コマンド自身および生成されるコマンドの両方に対して有効です。次の例を見てください。

fc -s -- -1 2>/dev/null この例は以前のコマンドを再度呼び出すものですが、 fc および以前のコマンドの両方に対して標準エラー出力を抑止します。

csh

履歴のリストを表示します。 *n* を指定すると、 *n* 個の最新の履歴を表示します。

-r　出力を、古い履歴からではなく 最近の履歴から順に並べ変えます。

-h　先頭に番号を付加せずに履歴の リストを出力します。 -h オプションを使用することにより、 csh 組み込みコマンドの source(1) の入力に適したファイルを出力することができます。

履歴置換: 履歴置換を使用すれば、 以前入力したコマンド行のワードを、 これから入力するコマンド行で使用できます。 これにより、 綴りの訂正、 複雑なコマンドや引数の繰り返し入力が簡単になります。 コマンド行は履歴リストに保存されます。 履歴リストのサイズは history 変数によって変更できます。 history シェル変数は履歴ファイルに保存されるコマンド行の 最大数に設定されます。 たとえば

```
set history = 200
```

は、履歴リストに最新の 200 コマンド行のトラックを保管します。 設定を行わない場合、 C シェルは最新のコマンドだけを保存します。

履歴置換の指定は ! で始まり (histchars 変数により他の文字に変更可能)、 コマンド行のどこに現われてもかまいません。 ただし履歴置換のネストはできません。 ! を \ でエスケープすれば、 その特別な意味を抑制できます。

履歴置換を含んだ入力行は、 展開された後で、 他の置換が起こる前または コマンドが実行される前に、 端末上に表示されます。

イベント指示子

イベント指示子は、 履歴リスト内の コマンド行エントリを参照するものです。

!	次の文字が空白文字、 タブ、 復帰改行、 =、 または (でなければ、 履歴置換を開始します。
!!	直前のコマンドを指します。 他の文字を加えずにこれだけを入力すると、 直前のコマンドを繰り返します。
!*n*	*n* 番のコマンド行を指します。
!-*n*	入力中のコマンドから *n* 個前のコマンドを指します。
!*str*	*str* で始まる最新のコマンドを指します。
!?*str*?	*str* を含む最新のコマンドを指します。
!?*str*? *additional*	*str* を含む最新のコマンドを指し、 その参照コマンドに *additional* を追加します。
!{command} *additional*	command で始まる最新のコマンドを指し、 その参照コマンドに *additional* を追加します。
^*previous_word*^*replacement*^	文字列 *previous_word* を文字列 *replacement* に置き換えて、 直前のコマンド行を繰り返します。 これは履歴置換と同じ動作です。 !:s/*previous_word*/*replacement*/ .以前の特定のコマンドを再実行し、さらに同じような置換を行う場合、たとえば 6 番目のコマンドを再実行するには、次のようにします。 !:6s/*previous_word*/*replacement*/ .

ワード指示子:

A ':' (コロン) はイベント指示子とワード指示子とを区切ります。 ワード指示子が ^、 $、 *、 -、 または % で始まるときは省略できます。 直前のコマンドからワードを選択する場合、 イベント指示子の 2 番目の ! は省略できます。 たとえば !!:1 と !:1 は、 両方とも直前のコマンドの最初のワードを指します。 また、 !!$ と !$ は、 両方とも直前のコマンドの最後のワードを指します。 ワード指示子には以下のものがあります。

#	今までに入力したすべてのコマンド行
0	最初に入力したワード (コマンド)
n	*n* 番目の引数
^	最初の引数。 1 と同じ
$	最後の引数
%	最新の ?*s* による検索で一致したワード
x-*y*	ワードの範囲。 -*y* は 0-*y* の省略形
*	すべての引数。 イベント中に 1 ワードしか存在しないときは NULL
*x**	*x*-$ の省略形
x-	*x** と同じだが、 ワード $ を省略

修飾子:

オプションのワード指示子の後に、 : で始まる 1 つ以上の修飾子を追加できます。

h	パス名の後部分の構成要素を除去して、 前部分を残します。
r	'.*xxx*' の形の接尾辞を除去して、 ベース名を残します。
e	接尾辞以外はすべて除去して、 拡張部分を残します。
s/*oldchars*/*replacements*/	*replacements* を *oldchars* に置換します。 *oldchars* は組み込まれた空白文字を含む文字列ですが、 イベント指示子、 ^*oldchars*^*replacements*^ にある *previous_word* は空白文字を含みません。
t	パス名の前部分の構成要素を除去して、 後部分を残します。
&	前の置換を繰り返します。
g	各ワード内の最初の一致が発生した箇所を、 上記のオプションに接頭辞を付けて変更します (たとえば g&) 。
p	新しいコマンドを表示するだけで、 実行はしません。

q	置換されたワードをクォートして、 それ以上の置換をエスケープします。
x	q と同じですが、 空白文字、 タブ、 復帰改行文字 (NEWLINE) ごとにワードに分割します。

g を先頭に付加しないと、*oldchars* に一致する最初の文字列だけが変更されます。 一致する文字列がなければ、 エラーとなります。

置換部分の左側は正規表現ではなく文字列です。 / の箇所には、 区切り文字としてどのような文字でも使用できます。 区切り文字用の文字はバックスラッシュで囲まれます。 右側の & 文字は、 左側のテキストで置換されます。 & はバックスラッシュでクォートすることができます。 *oldchars* が NULL のとき、 直前の文字列における *oldchars* 、または !?*s*. における文脈検索文字列 *s* を使用します。 同様に *replacements* の直後に復帰改行がある場合、 文脈検索の最右にある ? は省略できます。

イベントが指定されないと、 履歴リファレンスは前のコマンドか、 (もしあれば) そのコマンド行上での 前の履歴リファレンスを参照します。

ksh fc -e- [*old=new*] [*command*] の形式で fc を指定すると、 *old=new* の置換を行なった後で *command* が再実行されます。*command* 引数を省略すると、最後に行なったコマンドが実行されます。

fc [-e *ename*] [-nlr] [*first* [*last*]] の形式で fc を指定すると、端末から最近入力された HISTSIZE 個のコマンドの中から、*first* から *last* までの範囲のコマンドを選択します。*first* と *last* の両引数は、 数値または文字列で指定できます。 文字列の場合、その文字列で始まる最新のコマンドを見つけます。 負の数値は、 現在のコマンド番号からのオフセットとなります。-l オプションを指定すると、標準出力上にコマンドを一覧表示します。 -l を指定しないと、これらのキーボードコマンドの入ったファイル上で -e *ename* で示すエディタプログラムを起動します。 *ename* が省略されていると、 変数 FCEDIT (デフォルトは /bin/ed) の値をエディタとして使用します。 編集が完了すると、 編集されたコマンドを実行します。*last* を省略すると、*first* と同一値に設定されます。*first* を省略すると、デフォルトは、編集については直前のコマンドに、 一覧表示については –16 になります。-r オプションはコマンドの順序を逆にします。-n オプションは一覧表示時にコマンド番号の出力を抑止します (コマンド行編集の詳細については ksh(1) を参照) 。

HISTFILE シェル起動時にこの変数が設定されていると、その値はコマンド履歴 を格納するために使用されるファイルのパス名になります。

HISTSIZE シェル起動時にこの変数が設定されていると、このシェルで使用可能な入力済みコマンドの数が、この値以上になります。デフォルト値は 128 です。

コマンド再入力:

端末装置から最近入力された HISTSIZE が示す個数 (デフォルトは 128 個) のコマンドのテキストは、履歴ファイルに保存されています。 $HOME/.sh_history というファイルは、HISTFILE 変数が設定されていない場合、または変数が示すファイルが書き込み不可能な場合に使用されます。シェルは、同じ名前の HISTFILE を使用する

対話型シェルすべてのコマンド履歴を使用できます。fc という特殊コマンドは、 このファイルの一部をリスト表示または編集するときに使用します。 編集またはリスト表示すべきファイルの部分は、 番号か、 またはコマンドの最初の文字 を 指定することによって選択できます。 単一のコマンドを指定することも、コマンドの範囲を指定することも可能です。fc の引数としてエディタプログラムが指定されていないと、FCEDIT という変数の値が使用されます。FCEDIT が未定義の場合は、/bin/ed が使われます。編集されたコマンドは、 エディタを終了した時点で 表示および再実行されます。エディタ名に – を指定すると、 編集段階が省かれ、コマンドが再実行されます。 この場合、 *old=new* という形式の代入パラメタを使用すれば、 実行前にコマンドを変更できます。 たとえば、 r が 'fc -e -' の別名として定義されているとき 'r bad=good c' と入力すると、c という文字で始まるコマンドのうち最新のものが、その記述中の最初の bad という 文字列を good に置き換えられて再実行されます。

複合コマンドの中に fc 組み込みコマンドを指定すると、 すべてのコマンドが履歴ファイルから 削除されます。

オプション

以下のオプションを指定できます。

-e *editor*
: *editor* が示すエディタを使ってコマンドを編集します。文字列 *editor* はユーティリティ名で、 PATH 変数の値に従って検索されます。-e を省略すると、 FCEDIT 変数の値がデフォルトとして用いられます。FCEDIT の値が NULL または未設定のときは、エディタとして ed が使用されます。

-l
: (文字のエル) エディタを呼び出して編集する代わりに、コマンドを一覧表示します。 *first* と *last* の両オペランドで指定した範囲のコマンドを、 -r オプションがあればそれに従って、順番にコマンド番号付きで表示します。

-n
: -l オプションによる一覧表示において、コマンド番号を出力しません。

-r
: コマンドの一覧表示 (-l 指定時) または編集 (-l および -s 省略時) において、順序を逆にします。

-s
: エディタを呼び出さずにコマンドを再実行します。

オペランド

以下のオペランドを指定できます。

first

last
: 表示または編集するコマンドを選択します。いくつまでさかのぼってコマンドをアクセスできるかは、HISTSIZE 変数の値により決まります。*first* と *last* の値は、それぞれ以下のいずれかです。

  [+]*number*
  : コマンド番号を表す正の整数。過去に実行した各コマンドの番号は、 -l オプションを使えば確認できます。

  –*number*
  : いくつ前のコマンドかを示す負の整数。 たとえば直前に実行したコマンドなら –1 となります。

string 指定した文字列で始まっていたコマンドのうち、 最後に実行したコマンド。*old=new* オペランドが `-s` オプションなしで指定された場合、文字列形式の *first* オペランドを使用するなら、その文字列中に等記号を含めることはできません。

「形式」の項で示した形式で、`-s` を指定する場合、

- *first* を省略すると、直前のコマンドが使用されます。

「形式」の項で示した形式で、`-s` を指定しない場合、

- *last* を省略すると、デフォルト値は `-l` 指定時は直前のコマンドとなり、`-l` 省略時は *first* の値となります。
- *first* と *last* の両方を省略すると、`-l` 指定時は直前の 16 個のコマンドの表示、`-l` 省略時は直前の 1 つのコマンドの編集となります。
- *first* と *last* の両方を指定すると、 *first* から *last* までのすべてのコマンドが表示 (`-l` 指定時) または編集 (`-l` 省略時) されます。複数のコマンドを一度に、 各々を新たな行で開始してエディタに渡せば、複数コマンドの編集が可能です。*first* が示すコマンドが *last* が示すものより新しい場合、`-r` 指定時と同じように逆の順序で表示または編集されます。次の例を見てください。1 行目の 2 つのコマンドは、2 行目のそれぞれ真下にあるコマンドと同じ意味を持ちます。

```
fc -r 10 20     fc    30 40
fc   20 10      fc -r 40 30
```

- 一連のコマンドを範囲指定する場合、 履歴リストに存在していない値を *first* や *last* に指定してもエラーとはなりません。 `fc` は、存在している最も古いまたは新しい番号を、その代わりに使用します。 たとえば、履歴リスト中に 10 個のコマンドが記録されていて、そのコマンド番号が 1 から 10 となっているとします。このとき、以下のコマンドはいずれも 10 個のコマンドすべてを表示または編集することになります。

```
fc -l
fc 1 99
```

old=new 再実行対象のコマンド中に最初に現れた文字列 *old* を、他の文字列 *new* に置き換えます。

出力 `-l` オプションを使ってコマンドを表示する場合、 その出力形式は次のとおりです。

`"%d\t%s\n"`, <*line number*>, <*command*>

`-l` と `-n` の両オプションを指定すると、各コマンドの出力形式は 次のようになります。

"\t%s\n", <*command*>

command が複数の行で構成されている場合、 2 行目以降は以下のように表示されます。

"\t%s\n", <*continued-command*>

使用例

例 **1** history と fc の使用例

```
            csh                                     ksh

% history                                 $ fc -l
  1   cd /etc                               1   cd /etc
  2   vi passwd                             2   vi passwd
  3   date                                  3   date
  4   cd                                    4   cd
  5   du .                                  5   du .
  6   ls -t                                 6   ls -t
  7   history                               7   fc -l

% !d                                      $ fc -e - d
  du .                                      du .
  262   ./SCCS                              262   ./SCCS
  336   .                                   336   .

% !da                                     $ fc -e - da
  Thu Jul 21 17:29:56 PDT 1994              Thu Jul 21 17:29:56 PDT 1994

%                                         $ alias \!='fc -e -'

% !!                                      $ !
  date                                      alias ='fc -e -'
  Thu Jul 21 17:29:56 PDT 1994
```

環境

fc の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

FCEDIT　この変数は、シェルによって展開されると、 e *editor* オプションの *editor* のデフォルト値を定義します。 FCEDIT が NULL または未設定の場合、エディタとして ed が使用されます。

HISTFILE　コマンドの履歴ファイル名を表すパス名を定義します。 HISTFILE 変数が設定されていないと、シェルはユーザーのホームディレクトリ内で .sh_history というファイルをアクセスまたは生成しようとします。この履歴ファイルへの読み込みおよび書き込みアクセスができない、または生成できない場合、シェルは何らかのメカニズムを使って 履歴が正しくとれるようにします。 (この項で述べる履歴ファイルの参照とは、 このメカニズムが使用される場合があることを前提としています。) 履歴ファイルの初期化時にのみ fc をアクセスするようにすることもできます。初期化は、 fc または sh がユーザーからのコマンドに従って、最

初にこのファイル、または ENV 変数が指定するファイル、または /etc/profile のようなシステム起動 ファイルからのエントリ検索もしくはエントリの追加を試みたときに 発生します。 なお履歴ファイル用の初期化処理は、 システム起動ファイルの内容に依存させることもできます。 つまり、ユーザーが設定した HISTFILE や HISTSIZE 値を効果的に上書きするようなコマンドを、このファイルに記述することが可能です。たとえば set -o nolog オプションが設定されていなければ、関数定義コマンドが 履歴ファイルに記録されます。 ENV ファイルの前に呼び出されるシステムスタートアップファイル中に、 システム管理者が関数定義を記述しておけば、 ユーザーが履歴ファイルの属性を変更するような動作が可能になる前に、 履歴ファイルが初期化されます。 シェルが呼び出されると、最初に HISTFILE 変数が参照されます。 他のシェルが呼び出されるまでは、HISTFILE に対する変更は反映されません。

HISTSIZE　以前のコマンドを最大いくつまでさかのぼってアクセスできるかを 10 進数を使って定義します。この変数が設定されていないと、128 以上の不定の値がデフォルトとして用いられます。 シェルが呼び出されると、最初に HISTSIZE 変数が参照されます。 他のシェルが呼び出されるまでは、HISTSIZE に対する変更は反映されません。

終了ステータス

以下の終了ステータスが返されます。

0　一覧表示の正常終了

>0　エラーが発生した。

上記以外の場合、 fc により実行されたコマンドの終了ステータスがそのまま返されます。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ed(1), ksh(1), set(1), set(1F), sh(1), source(1), attributes(5), environ(5)

名前 | iconv – コードセット変換ユーティリティ

形式 | **iconv** -f *fromcode* -t *tocode* [*file...*]

機能説明 | iconv ユーティリティは、*file* で示すファイル中にある文字群または一連の文字群に対し、コードセットを変換してその結果を標準出力に書き出します。出力側のコードセットが対応する文字を持っていない文字は、下線 _ に変換されます。

サポートされる変換処理の一覧と変換テーブルの場所は、 iconv(5) のマニュアルページに記載されています。

オプション | 以下のオプションを指定できます。

-f *fromcode* 入力側のコードセットを指定します。

-t *tocode* 出力側のコードセットを指定します。

オペランド | 以下のオペランドを指定できます。

file 変換する入力ファイルのパス名。*file* を 1 つも指定しないと、標準入力が用いられます。

使用例 | 例 **1** ファイルを変換して保存する

次の例は、ファイル mail1 の内容を 8859 から 646fr へコードセット変換し、その結果をファイル mail.local に書き出します。

```
example% iconv -f 8859 -t 646fr mail1 > mail.local
```

環境 | iconv の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 正常終了

1 エラーが発生した

ファイル

/usr/lib/iconv/*.so 変換モジュール

/usr/lib/iconv/*.t 変換テーブル

/usr/lib/iconv/iconv_data 変換テーブルでサポートしている変換のリスト

/usr/lib/iconv/geniconvtbl/binaries/*.bt 変換バイナリテーブル

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | geniconvtbl(1), iconv(3C), geniconvtbl(4), attributes(5), environ(5), iconv(5), iconv_unicode(5)

注意事項 | iconv ユーティリティは、変換モジュール (/usr/lib/iconv/*.so)、変換テーブル (/usr/lib/iconv/*.t)、または変換バイナリテーブル (/usr/lib/iconv/geniconvtbl/binarytables/*.bt) を使って、コードセット変換を行うことができます。iconv ユーティリティは、iconv_open(3C) を使用してiconv(3C) 関数中で特定のコードセット変換が可能であるかどうかを調べます。iconv_open(3C) はまず変換バイナリテーブルを検索し、次に変換モジュールを検索します。システム上にどちらも存在しない場合、 iconv_open(3C) はエラーコードを返します。最後に、iconv が変換テーブルを検索します。

コードセット変換をサポートしているアジア系言語に関する情報は、 アジア地区ローカル版リリースの /usr/share/man/man5/iconv_locale.5 のマニュアルページを参照してください。 コマンドの例を以下に示します。

```
% man -s 5 iconv_ja
```

このコマンドにより、日本語ロケール用にサポートしている コードセット変換を説明するマニュアルページが表示されます。

なお iconv_locale.5 のマニュアルページがシステム上に存在しない場合があります。オペレーティングシステムのインストール時に選択したロケールによっては、このマニュアルページが存在しないことがあります。

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | jobs, fg, bg, stop, notify – プロセスの実行の制御

sh

jobs [-p | -l] [% *job_id*...]

jobs -x *command* [*arguments*]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

csh

jobs [-l]

fg [% *job_id*]

bg [% *job_id*...]

notify [% *job_id*...]

stop % *job_id*...

stop *pid*...

ksh

jobs [-lnp] [% *job_id*...]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

sh

ジョブ制御が有効なとき、Bourne シェルに組み込まれた jobs は、停止中または バックグラウンドで実行中の すべてのジョブを表示します。 %*job_id* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが表示されます。次のオプションを使って、ジョブに関する表示を変更できます。

-l　ジョブのプロセスグループ ID および作業ディレクトリを表示します。

-p　ジョブのプロセスグループ ID のみを表示します。

-x　*command* または *argument* 中に見つかった *job_id* を、対応するプロセスグループ ID に置き換え、*command* に *argument* を渡して実行します。

シェルを jsh として呼び出すと、sh の説明で述べたすべての機能に加えて、ジョブ制御が可能になります。通常、ジョブ制御は対話型シェルに対してだけ可能です。通常、非対話型シェルは、ジョブ制御の機能 を使用しません。

ジョブ制御が可能なとき、ユーザーが端末から入力したコマンドまたはパイプラインは、すべて *job_id* と呼ばれます。どのジョブも、必ずフォアグラウンド、バックグラウンド、または停止のいずれかの状態にあります。 これらの用語の定義を次に示します。

1. フォアグラウンド状態にあるジョブは、 制御している端末への読み取りおよび 書き込みアクセス権を持っています。
2. バックグラウンド状態にあるジョブは、 制御している端末への読み取りアクセスを拒否されていますが、 条件付き書き込みアクセス権は持っています (`stty`(1) を参照)。
3. 停止ジョブは保留状態に置かれたジョブであり、通常は `SIGTSTP` シグナルにより、この状態になります (`signal`(3HEAD) を参照)。

シェルが起動するすべてのジョブには、 ジョブ番号 (*job-id number*) と呼ばれる正の整数が割り当てられます。シェルはこの番号を把握し、 特定のジョブを示す識別子として使用します。また、シェルは現在 (*current*) および前回 (*previous*) のジョブも把握しています。現在のジョブとは、最後に起動または再起動されたジョブです。 前回のジョブとは、その直前のジョブです。

ジョブ識別子の正しい構文は次のような形式です。

`%`*job_id*

job_id は、次のいずれかの形式で指定できます。

`%` または `+`	現在のジョブ
–	前回のジョブ
`?`<*string*>	*string* を含むコマンド行 (一意に表す) に対応したジョブ
n	ジョブ番号が *n* のジョブ
pref	コマンド名の先頭が *pref* のコマンド。 たとえば `ls -l` *name* がバックグラウンドで実行中だった場合、`% ls` と指定すればこのコマンドを示すことができます。*pref* は、引用符で囲まない限り、空白文字を含めることができません。

ジョブ制御が有効なとき、`fg` は中断しているジョブの実行を フォアグラウンドで再開します。 またバックグラウンドで実行中のジョブを フォアグラウンドに移動します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

ジョブ制御が有効なとき、`bg` は中断されているジョブの実行を バックグラウンドで再開します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

`stop` は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (`ps`(1) を参照) 。

csh C シェルに組み込まれた `jobs` は、引数なしでジョブ制御下で活動中のジョブを一覧表示します。

`-l` 通常の情報の他に、プロセス ID を表示します。

シェルは、番号の付いた *job_id* を各コマンドシーケンスと対応付けて、バックグラウンドで動作中のコマンド、または TSTP シグナル (通常は Control-Z) によって停止したコマンドの動作を追跡します。コマンドまたはコマンドシーケンス (セミコロンで区切られたリスト) をメタキャラクタ & を使用してバックグラウンドで起動した場合、 シェルは角括弧で囲まれたジョブ番号と 関連するプロセス番号のリストを表示します。以下に例を示します。

[1] 1234現在のジョブリストを見るには、組み込みコマンド jobs を使用します。最後に停止したジョブ (停止したジョブがない場合は、 最後にバックグラウンドに投入されたジョブ) を「現在のジョブ」といい、 '+' で示します。前のジョブは '-' で示します。 現在のジョブが終了したりフォアグラウンドに移された場合、 前のジョブが新しく現在のジョブになります。

ジョブの操作方法については、組み込みコマンド bg、 fg、 kill、 stop、 % の説明を参照してください。

ジョブの参照は '%' で始まります。 パーセント記号だけの指定は、現在のジョブを示します。

% %+ %%	現在のジョブ
%-	前のジョブ
%*j*	'kill -9 %*j*' のようにジョブ *j* を参照します。 *j* はジョブ番号、またはジョブを起動した コマンド行を一意に表す文字列です。 たとえば 'fg %vi' は、停止した vi ジョブをフォアグラウンドに移します。
%?*string*	*string* を含むコマンド行 (一意に表す) に対応したジョブを指定します。

バックグラウンドで動作中のジョブは、端末からの読み取り時に停止します。 バックグラウンドジョブは、通常出力を生成しますが、'stty tostop' コマンドを使用して抑止することも可能です。

fg は現在のジョブまたは指定された *job_id* をフォアグラウンドへ移します。

bg はバックグラウンドで、現在のジョブ または指定されたジョブを実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

notify は現在のジョブまたは指定されたジョブの状態が変わったとき、 その旨非同期にユーザーに知らせます。

ksh jobs は、現在のシェル環境で開始されたジョブの状況を表示します。 jobs がジョブの終了を報告したとき、シェルはそのジョブのプロセス ID を、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除します。

特定のジョブの報告だけが必要なら、*job_id* を使ってジョブを指定します。*job_id* を 1 つも指定しないと、全ジョブに関する情報が出力されます。

以下のオプションは、 `jobs` の出力を変更または拡張するために使用します。

`-l` (文字のエル) 個々のジョブに関して詳細な情報を出力します。具体的には、 ジョブ番号、現在のジョブ、プロセスグループ ID 、状態、ジョブを生成したコマンドを出力します。

`-n` 前回通知を受けた後に停止または終了した ジョブだけを表示します。

`-p` 選択されたジョブのプロセスグループリーダのプロセスグループ ID だけを出力します。

デフォルトでは、`jobs` は、停止しているすべてのジョブの状態、実行中のバックグラウンドジョブの 状態、そして状態が変わったのにシェルによりまだ報告されていない すべてのジョブの状態を表示します。

`set` コマンドの `monitor` オプションを有効にすると、対話型シェルが `job` を各パイプラインと関連付けます。 このオプションは、`jobs` コマンドが表示する現在のジョブのテーブルを維持し、これらのジョブに整数番号を割り当てます。ジョブを `&` で非同期に起動すると、シェルは、`[1] 1234` という形式の行を表示します。 非同期に起動されたジョブはジョブ番号 1 であり、プロセス ID が 1234 であるプロセスが 1 つ (トップレベル) あることを示します。実行中のジョブがあるが、 別に実行したいジョブがある場合、`^Z` (Control-Z) キーを押せば、 現在のジョブに `STOP` シグナルが送信されます。 そうするとシェルは通常、 ジョブが「停止」されたことを示し (後述の「出力」の項を参照)、 プロンプトを表示します。これで、このジョブの状態を `bg` コマンドでバックグラウンドで処理するか、または他のコマンドを実行してから、 `fg` というコマンドで ジョブをフォアグラウンドに移すことができます。`^Z` は直ちに有効になります。つまり `^Z` は、 保留中の出力や読み取られていない入力が 直ちに中止されるという点で、割り込みに似ています。

シェル内のジョブを参照する方法はいくつかあります。そのジョブのいずれかのプロセスの ID を使っても、また以下のいずれかを使っても参照できます。

`%`*number* *number* が示す番号のジョブ

`%`*string* コマンド行が *string* で始まっていたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%?`*string* コマンド行が *string* を含んでいたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%%` 現在のジョブ

`%+` `%%` と同じ

`%-` 直前のジョブ

シェルは、 プロセスの状態が変更すると、 直ちにそれを検出します。 ジョブがブロックされてそれ以上進めない状態になると、 シェルはその旨をユーザーに通知します。 ただし、ユーザーの作業の妨げにならないよう、 この通知はプロンプトを発行

する直前にだけ行います。 モニタモードが有効なとき、 完了した各バックグラウンドジョブは、 CHLD に設定されているトラップを起こします。 ジョブの実行中または停止中にシェルを終了しようとすると、 「停止中 (実行中) のジョブがある (‘You have stopped (running) jobs.’) 」旨の警告を受けます。 jobs コマンドを使用すれば、 どのジョブが該当するのかを確認できます。これを実行するか、 または直ちにシェルを再終了しようとすると、 シェルは 2 度目の警告は出さず、 停止中のジョブは終了します。

fg は、バックグラウンドジョブを、現在の環境からフォアグラウンドへ移します。 fg を使ってジョブをフォアグラウンドへ移した場合、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除されます。 fg コマンドが使えるのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をフォアグラウンドで実行します。*job_id* が指定されないと、 現在のジョブをフォアグラウンドで実行します。

bg は、現在の環境で中断されたジョブを、バックグラウンドジョブとして 実行することにより再開します。 *job_id* が示すジョブがすでにバックグラウンドジョブを実行している場合、bg は何も行わず正常に終了します。bg を使ってジョブをバックグラウンドへ移した場合、あたかも非同期リストから起動されたかのように、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID 」の 1 つとなります。 bg コマンドが使えるのは、 ジョブ制御をサポートするシステム上だけです。*job_id* が省略された場合は、 現在のジョブをバックグラウンドで実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

出力

-p オプションを指定すると、各プロセス ID に対して次に示す 1 行の情報が出力されます。

"%d\n", *"process ID"*

-p を省略すると、 -l オプションも省略されていれば、以下の形式の一連の行が出力されます。

"[%d] %c %s %s\n", *job-number*, *current*, *state*, *command*

各フィールドの意味を以下に説明します。

current

文字 + は、 fg および bg コマンド用のデフォルトとして使用するジョブを表します。 このデフォルトジョブは、 *job_id* %+ または %% を使って指定することもできます。文字 – は、現在のデフォルトジョブが終了してしまった場合に デフォルトとなるジョブを表します。このジョブは、 *job_id* %– を使って指定することもできます。 その他のジョブに関しては、このフィールドは空白文字として出力されます。 + や – を使って表せるジョブの数は、どちらも最大 1 つです。停止中の

ジョブがあれば、現在のジョブも停止ジョブとなります。 停止中のジョブが 2 つ以上あれば、以前のジョブも停止ジョブとなります。

job-number
: `wait` 、 `fg` 、 `bg` 、 `kill` の各ユーティリティ用にプロセスグループを識別するのに使用する番号。 これらのユーティリティを使うと、ジョブはジョブ番号の後に `%` を付加することにより識別できます。

state
: 以下の文字列 (POSIX ロケールにある) のいずれかです。

`Running`	ジョブはシグナルによって中断されておらず、 終了もしていないことを表す。
`Done`	ジョブは終了して、ゼロの終了ステータスを返したことを表す。
`Done(`*code*`)`	ジョブは正常に終了し、指定されたゼロ以外の終了ステータス (*code* が示す 10 進数) を返したことを表す。
`Stopped`	
`Stopped (SIGTSTP)`	`SIGTSTP` シグナルがジョブを停止したことを表す。
`Stopped (SIGSTOP)`	`SIGSTOP` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTIN)`	`SIGTTIN` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTOU)`	`SIGTTOU` シグナルがジョブを停止したことを表す。

利用者側で、文字列 `Stopped` の代わりに `Suspended` を使うよう定義することができます。 ジョブをシグナルが終了した場合、 `state` の形式は不定ですが、ここに示した他の `state` 形式とは明確に区別できるものです。その出力上で、 ジョブを終了させたシグナルの名前または説明が与えられます。

command
: シェルに与えられた関連コマンド。

`-l` オプションを指定すると、プロセスグループ ID を示すフィールドが `state` フィールドの前に挿入されます。さらに、プロセスグループ内の より多くのプロセスが別の行に出力されることがあります。 その内容は、プロセス ID と `command` フィールドだけです。

環境

jobs 、 fg 、 bg の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

jobs 、 fg 、 bg は、以下の終了ステータスを返します。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), kill(1), ksh(1), ps(1), sh(1), stop(1), shell_builtins(1), stty(1), wait(1), signal(3HEAD), attributes(5), environ(5)

名前 | join – リレーショナルデータベース演算子

形式 | **join** [-a *filenumber* | -v *filenumber*] [-1 *fieldnumber*] [-2 *fieldnumber*] [-o *list*] [-e *string*] [-t *char*] *file1 file2*

join [-a *filenumber*] [-j *fieldnumber*] [-j1 *fieldnumber*] [-j2 *fieldnumber*] [-o *list*] [-e *string*] [-t *char*] *file1 file2*

機能説明 | join コマンドは、*file1* および *file2* を行単位で結合し、標準出力上にその結果を出力します。

一致するフィールドを持つ *file1* と *file2* の各行について 1 行が出力されます。出力行は通常、共通フィールド、*file1* の行の残り、*file2* の行の残りの順で構成されます。 ただしこの出力形式は、後述する -o オプションを使って変更できます。また -a オプションを使えば、一致しなかった行も出力できます。-v オプションは、一致しなかった行だけを出力するために使用します。

デフォルトの入力フィールド区切り文字は、空白文字、タブ、 または復帰改行です。この場合、連続する区切り文字は 1 つのフィールド区切り文字とみなします。 先行する区切り文字は無視されます。デフォルト出力フィールド区切り文字は空白文字です。

入力行が正しい照合順序でならんでいない場合、コマンドの結果は予測できません。

オプション | 次のオプションには *filenumber* という引数を使用するものがあります。この引数の値は 1 または 2 で、それぞれ *file1* または *file2* を表します。

-a*filenumber* — 通常の出力に加え、 ファイル *filenumber* 内の対にならなかった行についても出力します。 *filenumber* の値は 1 または 2 です。 -a 1 と -a 2 の両方を指定すると、対にならなかった行がすべて出力されます。

-e *string* — -o オプションで選択したリスト内にある空の出力フィールドを *string* 文字列で置き換えます。

-j *fieldnumber* — -1*fieldnumber* -2*fieldnumber* と同じ意味です。

-j1 *fieldnumber* — -1*fieldnumber* と同じ意味です。

-j2 *fieldnumber* — -2*fieldnumber* と同じ意味です。 フィールドには 1 から始まる番号が付きます。

-o *list* — 各出力行は、*list* に指定されたフィールドを含みます。*list* で選択されたフィールドのうち入力中に現れないものは、出力側では空のフィールドとして扱われます (-e オプションの説明を参照)。 *list* の各要素の形式は *filenumber*.*fieldnumber* または 0 で、後者は join フィールドを表します。 特に指定しないかぎり、共通フィールドは出力しません。

-t *char* — *char* で示す文字を区切り文字として使用します。 1 つの行に *char* が複数個あるとき、それらはすべて有効です。文字 *char* は、入力と出力の両方においてフィールド区切り文字として使われます。

このオプションを指定する場合、照合される語は -b オプションなしで sort を実行した場合にも同一とみなされる語でなければなりません。

-v *filenumber*　デフォルト形式の出力ではなく、*filenumber* 内の対にならなかった行だけを出力します。*filenumber* の値は 1 または 2 です。-v 1 と -v 2 の両方を指定すると、対にならなかった行がすべて出力されます。

-1 *fieldnumber*　*file1* の何番目のフィールドで結合するかを、フィールド番号で指定します。フィールド番号は 1 から始まる整数です。

-2 *fieldnumber*　*file2* の何番目のフィールドで結合するかを、フィールド番号で指定します。フィールド番号は 1 から始まる整数です。

オペランド

以下のオペランドを指定できます。

file1

file2　結合するファイルのパス名。*file1* と *file2* のどちらかに – を指定すると、そのファイルの代わりに標準入力が使用されます。

file1 と *file2* は、結合するフィールド (通常は各行の最初のフィールド) 上で LC_COLLATE によって決められた 照合順序で昇順にソートされていなければなりません (sort(1) を参照)。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の join の動作については、largefile(5) を参照してください。

使用例

例 1 パスワードファイルとグループファイルの結合

下記のコマンド行は、パスワードファイルとグループファイルを結合し、数値グループ ID のマッチングを行い、ログイン名、グループ名、 ログインディレクトリを出力します。ファイルは、グループ ID フィールド上において ASCII の照合順序でソートされていると仮定します。

```
example% join -j1 4-j2 3 -o 1.1 2.1 1.6 -t:/etc/passwd /etc/group
```

例 2 -o オプションの使用

-o 0 は、基本的には結合フィールドを統合したフィールドを表します。たとえば、phone という名のファイルは次のような内容だとします。

```
!Name           Phone Number
Don             +1 123-456-7890
Hal             +1 234-567-8901
Yasushi         +2 345-678-9012
```

また、fax というファイルは次のような内容だとします。

```
!Name           Fax Number
Don             +1 123-456-7899
```

```
Keith           +1 456-789-0122

Yasushi         +2 345-678-9011
```

この 2 つのファイルで、長い空白部分は 1 つのタブ文字を表しているものとします。 このとき次のようなコマンドを実行します。

```
example% join -t"<tab>" -a 1 -a 2 -e '(unknown)' -o 0,1.2,2.2 phone fax
```

すると以下のような出力が生成されます。

```
!Name           Phone Number           Fax Number
Don             +1 123-456-7890         +1 123-456-7899
Hal             +1 234-567-8901         (unknown
Keith           (unknown)               +1 456-789-012
Yasushi         +2 345-678-9012         +2 345-678-9011
```

環境

join の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、LC_COLLATE、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	入力ファイルはすべて正常に出力された
>0	エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

awk(1), comm(1), sort(1), uniq(1), attributes(5), environ(5), largefile(5)

注意事項

デフォルトでフィールドを分離する場合、照合順序は -b を付けた sort コマンドに対応した順序です。 -t の場合、照合順序は -b を付けない sort コマンドの順序になります。

join 、 sort 、 comm 、 uniq 、 awk コマンドの規則は一律ではありません。

名前 | sh, jsh – 標準シェルとジョブ制御シェルおよびコマンドインタプリタ

形式 | **/usr/bin/sh** [-acefhiknprstuvx] [*argument*...]

/usr/xpg4/bin/sh [± abCefhikmnoprstuvx] [± o *option*...] [-c *string*] [*arg*...]

/usr/bin/jsh [-acefhiknprstuvx] [*argument*...]

機能説明 | /usr/bin/sh ユーティリティは、端末またはファイルからコマンドを実行する コマンドプログラミング言語です。

/usr/xpg4/bin/sh ユーティリティは標準に準拠したシェルです。このユーティリティは ksh(1) のすべての機能を提供します。ただし、ksh(1) で説明しているように、動作が異なる場合を除きます。

jsh ユーティリティは、sh のすべての機能を備えた、かつジョブ制御を可能にするシェルへのインタフェースです (下記の「ジョブ制御」を参照) 。

シェルへの引数については、後述の「呼び出し」にリストされています。

定義 | ブランク (*blank*) はタブ (tab) または空白文字 (space) です。名前 (*name*) は ASCII 英文字、 数字、 または下線の並びで、 先頭文字は英文字または下線です。パラメタ (*parameter*) は、 名前、 複数、 または *、 @、 #、 ?、 –、 $、 ! などの文字のいずれかです。

コマンド | 単純コマンド (*simple-command*) は、ブランクで区切られた、ブランクでないワードの並びです。先頭のワードは、実行すべきコマンドの名前を指定します。残りのワードは、以下に述べる場合を除き、呼び出されたコマンドに引数として渡されます。コマンド名は引数 0 として渡されます (exec(2) を参照) 。単純コマンドの値 (*value*) は、正常終了した場合は終了ステータス、異常終了した場合は 200+*status* (8 進数) です。ステータス値の一覧表については、signal(3HEAD) を参照してください。

パイプライン (*pipeline*) は、パイプ (|) で区切られた 1 つ以上のコマンドの並びです。最後のコマンドを除き、各コマンドの標準出力は pipe(2) によってその次のコマンドの標準入力と結合されます。各コマンドは、別々のプロセスとして実行されます。シェルは最後のコマンドが終了するのを待ちます。最後のコマンドの終了ステータスが パイプライン全体の終了ステータスとなります。

リスト (*list*) は、; 、&、&&、または | | で区切られた 1 つ以上のパイプライン の並びです。その並びの終わりに ; または & を記述することもできます。これら 4 つの記号の中で、; と & の優先度は同じで、&& と | | の優先度より低くなります。&& と | | の優先度は同じです。セミコロン (;) によって、直前のパイプラインが順次実行されます。つまりシェルはパイプラインが終了するのを待ってから、セミコロンの後のコマンドを実行します。アンパサンド記号 (&) によって、直前のパイプラインが非同期的に実行されます。つまりシェルはパイプラインが終了するのを待ちません。&& という記号は、直前のパイプラインの終了ステータスが 0 の場合にだけ、後続のリストを実行するものです。反対に (| |) は、終了ステータスが 0 以外の場合にだけ、後続のリストを実行します。リスト中のコマンドを区切るのに、セミコロンの代わりに 任意の数の復帰改行 (newline) を指定することもできます。

コマンド (*command*) は、単純コマンドまたは以下のいずれかです。特に断わりのない限り、コマンドが返す値は、そのコマンド中で最後に実行された単純コマンドの値です。

`for` *name* [`in` *word* . . .] `do` *list* `done`
コマンドが実行されるたびに、 *name* は `in` *word* リストから次に得られる *word* に設定されます。`in` *word* . . . を省略すると、`for` コマンドは、設定された各定位値パラメタに対して、`do` *list* を 1 回実行します (後述の「パラメタ置換」を参照) 。リストの *word* がなくなると、実行は終了します。

`case` *word* `in` [*pattern* [| *pattern*]) *list* `;;`] . . . `esac`
`case` コマンドは、*word* に一致する最初の *pattern* に対応した *list* を実行します。パターンの形式は、ファイル名生成に使用される形式と同じです (「ファイル名の生成」の項を参照) 。ただしスラッシュ、先行するドット、およびスラッシュ直後のドットは、明示的に一致しなくてもかまいません。

`if` *list* `;` `then` *list* `;` [`elif` *list* `;` `then` *list* `;`] . . . [`else` *list* `;`] `fi`

`if` の後の *list* を実行後、*list* が 0 の終了ステータスを返すと、最初の `then` の後の *list* を実行します。それ以外の場合、`elif` の後の *list* を実行します。この値が 0 の場合、次の `then` の後の *list* を実行します。これが失敗すると、`else` *list* を実行します。`else` *list* も `then` *list* も実行しない場合、`if` コマンドは 0 の終了ステータスを返します。

`while` *list* `do` *list* `done`	`while` コマンドは、`while` *list* を繰り返し実行し、*list* 中の最後のコマンドの終了ステータスが 0 の場合、`do` *list* を実行します。それ以外の場合、ループは終了します。`do` *list* 中のコマンドを実行しない場合、`while` コマンドは 0 の終了ステータスを返します。ループ終了条件の判定を逆にするには、`while` の代わりに `until` を使用します。
(*list*)	サブシェル内の *list* の実行
{ *list* ; }	現在の (つまり、親) シェル内での *list* の実行。記号 { の後には空白が必要です。
name ( ) { *list* ; }	*name* が参照する関数を定義します。{ と } の間のコマンド群 (*list*) が関数の本体となります (後述の「関数」参照) 。{ の後には空白が必要です。関数の実行については「実行」の項で後述します。関数の本体が、上記の「コマンド」の項で 定義したようなコマンドの場合、{ および } は必要ではありません。

下記のワードは、コマンドの最初に現れたとき、およびクォートされずに記述されたときに認識されます。

`if then else elif fi case esac for while until do done { }`

注釈行

でワードを始めると、そのワードおよび以降の 復帰改行までの文字がすべて無視されます。

コマンド行置換

シェルは、2 つの逆引用符 (`...`) で囲まれた文字列から コマンドを読み取ります。これらのコマンドからの標準出力は、ワードの一部または全体として使用できます。標準出力上で最後につく復帰改行は削除されます。

文字列は、読み取られる前にはいっさい解釈されません。ただし例外として、文字のエスケープに使用されるバックスラッシュ (\) の削除だけは行われます。バックスラッシュは、逆引用符 (`) または別のバックスラッシュ (\) をエスケープするためにも使用され、コマンド文字列の読み取り前に削除されます。逆引用符をエスケープすることにより、コマンド置換のネストが可能になります。コマンド置換が、一対の二重引用符に囲まれている場合 (" . . . ` . . . ` . . . ") 、二重引用符をエスケープしているバックスラッシュ (\") は削除されますが、それ以外のバックスラッシュはそのまま残されます。

復帰改行文字のエスケープにバックスラッシュを用いた場合は (\newline)、バックスラッシュと復帰改行の両方が削除されます (後述の「クォート」の項の後半を参照)。さらに、ドル記号 (\$) をエスケープしているバックスラッシュも削除されます。コマンド文字列に対するパラメタの置換は 読み取り前には行われないので、バックスラッシュでドル記号をエスケープしようとしても 無意味です。\、`、"、復帰改行 (newline)、および $ 以外の文字の前に付くバックスラッシュは、コマンド文字列の読み取り時にもそのまま残ります。

パラメタ置換

文字 $ は、置換可能なパラメタ (*parameters*) を示します。パラメタには、定位置パラメタとキーワードパラメタの 2 種類があります。パラメタが数字の場合、これは定位置パラメタです。定位置パラメタには、set コマンドによって値を割り当てられます。キーワードパラメタ (変数とも呼ばれる) には、以下の記述により値を代入することもできます。

name=*value* [*name*=*value*] . . .

value に対しては、パターンマッチングは行われません。同じ *name* を持つ関数と変数が存在することはできません。

${*parameter*}
: パラメタの値 (もしあれば) は置換されます。中括弧が必要となるのは、パラメタの後に、その名前の一部として解釈すべきでない文字、数字、または下線を指定するときだけです。*parameter* が * または @ の場合、$1 で始まる定位値パラメタはすべて (空白で区切られて) 置換されます。パラメタ $0 は、シェルが呼び出されたときに、引数 0 から設定されます。

${*parameter*:–*word*}
: *parameter* が NULL 以外の値に設定されている場合、その値に置き換えられます。その他の場合、*word* に置き換えられます。

${*parameter*:=*word*}	*parameter* が設定されていない場合、または NULL に設定されている場合、それを *word* に設定します。次に、パラメタの値を置き換えます。この方法で定位値パラメタに値を代入することはできません。
${*parameter*:?*word*}	*parameter* が NULL 以外の値に設定されている場合、その値に置き換えられます。その他の場合、*word* を出力しシェルを終了します。*word* を省略すると、メッセージ “parameter null or not set” が出力されます。
${*parameter*:+*word*}	*parameter* が NULL 以外の値に設定されている場合、*word* に置き換えられます。その他の場合は置換を行いません。

上記にあるように、*word* は、代入用文字列として使用する場合にだけ評価されます。たとえば次の例では、pwd が実行されるのは、d が設定されていないかあるいは NULL に設定されている場合だけです。

```
echo  ${d:-`pwd`}
```

上記の式からコロン (:) を省略すると、シェルは *parameter* が設定されているかどうかだけをチェックします。

次のパラメタは、シェルが自動的に設定します。

#	定位置パラメタ数 (10 進数)
-	呼び出し時にまたは set コマンドによってシェルに与えられたフラグ
?	最後に同期実行されたコマンドが返した 10 進数
$	このシェルのプロセス番号
!	最後に呼び出されたバックグラウンドコマンドのプロセス番号

次に挙げるパラメタはシェルが使用するもので、環境変数とも呼ばれるものです。

HOME	cd コマンドのデフォルト引数 (ホームディレクトリ) 。login(1) によって、パスワードファイルから ユーザーのログインディレクトリに設定されます (passwd(4) を参照) 。
PATH	コマンド用検索パス (後述の「実行」を参照) 。
CDPATH	cd コマンドの検索パス
MAIL	このパラメタにメールファイルの名前がセットされていて、かつ MAILPATH パラメタが設定されていない場合、シェルは指定されたファイルにメールが到着すると ユーザーに通知します。
MAILCHECK	このパラメタは、MAILPATH または MAIL パラメタで指定されたファイルへメールが到着したか否かを、シェルが何秒ごとにチェックするかを指定します。デフォルト値は 600 秒 (10 分) で

す。このパラメタの値として 0 が設定された場合、シェルは次のプロンプトを出す前に チェックを行います。

MAILPATH	コロン (:) で区切ったファイル名のリスト。このパラメタが設定されると、指定されたいずれかのファイルに メールが到着するたびに、シェルはユーザーに通知します。各ファイル名の後には、% および更新時刻の変更時に出力されるメッセージを 指定することができます。デフォルトのメッセージは you have mail です。
PS1	1 次プロンプト文字列。デフォルトは " $ " です。
PS2	2 次プロンプト文字列。デフォルトは " > " です。
IFS	内部フィールドセパレータ。通常は空白文字、タブ、および復帰改行です (「ブランクの解釈」を参照) 。
SHACCT	このパラメタにユーザーが書き込み可能な ファイル名が設定された場合、シェルは、実行された各シェルプロシージャごとの アカウンティングレコードをこのファイルに書き込みます。
SHELL	シェルは、呼び出されると、このパラメタが示す名前が環境中に存在するかを 確かめます (「環境」の項を参照) 。sh の実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES についての詳細は、environ(5) を参照してください。シェルは、PATH、PS1、PS2、MAILCHECK、および IFS にデフォルト値を割り当てます。HOME および MAIL のデフォルト値は login(1) で設定します。

ブランクの解釈

パラメタとコマンドの置換後、置換の結果内でフィールドセパレータ文字 (IFS で発見されるもの) を検索し、その文字が現れた位置で分割します。分割された各々が引数となります。明示的な NULL 引数 ("" または '') は保持されます。暗示的な NULL 引数 (値を持たないパラメタにより生ずるもの) は削除されます。

入出力のリダイレクト

シェルが解釈する特殊表記法によって、入出力をリダイレクションできます。以下は、単純コマンド内の任意の位置および コマンドの前後に指定することができ、起動されたコマンドには引き渡されません。なお *word* または *digit* を使用する前にコマンドとパラメタの置換が発生するので 注意してください。

<*word*	*word* というファイルを標準入力 (ファイル記述子 0) として使用します。
>*word*	*word* というファイルを標準出力 (ファイル記述子 1) として使用します。ファイルが存在しない場合は作成します。ファイルが存在していれば、ファイルの長さを 0 にします。
>>*word*	*word* というファイルを標準出力として使用します。ファイルが存在する場合、(EOF までシークした後) そのファイルに出力を追加します。ファイルが存在しない場合は、ファイルを作成します。
< >*word*	*word* というファイルを標準入力として読み書き用にオープンします。

<<[−]*word*

word に対するパラメタおよびコマンド置換が行われた後、その結果得られた *word* と文字どおり一致する最初の行が現れるまで、または EOF に達するまでシェルへの入力が読み取られます。ただし << に − が付加されて指定された場合は、以下のようになります。

1. まず、シェルへの入力の読み取り前に (ただしパラメタとコマンドの置換後) 、先行するタブを *word* から取り除きます。
2. 読み取ったシェルへの入力の各行を *word* と比較する前に、その行から先行するタブを取り除きます。そして、
3. その結果得られた *word* と文字どおり一致する最初の行が現れるまで、または EOF に達するまでシェルへの入力が読み取られます。

word 中のいずれかの文字がクォートされている場合 (後述の「クォート」を参照) 、シェルへの入力に対して追加処理は行われません。*word* 中のどの文字もクォートされていない場合は、以下のようになります。

1. まずパラメタとコマンドの置換を実施します。
2. エスケープされた復帰改行 (`\newline`) を削除します。
3. 文字 \、$、および ` に対しては、 \ を使ってクォートしなければなりません。

その結果得られるドキュメントが標準入力となります。

<&*digit*

ファイル記述子 *digit* に対応するファイルを標準入力として使用します。同様に、標準出力の使用には >&*digit* を指定します。

<&−

標準入力をクローズします。同様に、標準出力については >&− を使用します。

上記のいずれかの前に数字が付く場合、その値が (デフォルトの `0` または `1` の代わりに) 該当ファイルに対応した ファイル記述子となります。

```
... 2>&1
```

この例では、現在ファイル記述子 1 に関連しているファイルに、ファイル記述子 2 を関連付けます。

リダイレクションを指定する場合、記述する順序が重要になります。シェルは、リダイレクション記述を左から右へ評価します。

```
... 1>xxx 2>&1
```

上記の例では、まず *xxx* というファイルにファイル記述子 1 を関連付けます。次に、ファイル記述子 1 に関連するファイル (つまり *xxx*) に、ファイル記述子 2 を関連付けます。リダイレクションの向きが逆であれば、まずファイル記述子 2 を端末に関連付け (ファイルを記述子 1 が既に端末に関連付けられているとみなし) 、次にファイル記述子 1 をファイル *xxx* に関連付けます。

最初のページの「コマンド」の項で述べた用語を使って説明すると、以下のようになります。コマンドが複数の単純コマンドで構成される場合、リダイレクションは、個々の単純コマンドに対して行う前に、コマンド全体に対して評価されます。すなわち、シェルはまずリスト 全体に対してリダイレクションを評価し、次にリスト内の各パイプラインに対して評価し、次にパイプライン内の各コマンドに対して評価し、最後にコマンド内の各単純コマンドに対して評価します。

コマンドの後に & を指定すると、コマンドにおけるデフォルトの標準入力は /dev/null という空ファイルになります。その他の場合、コマンドを実行するための環境には、起動側シェルのファイル記述子 (入出力指定で変更可能) が含まれます。

ファイル名の生成

コマンド実行に先立ち、各コマンドワードは、*、?、および [を含んでいないかチェックされます。これらの文字のいずれかがあると、そのワードはパターンとみなされます。このワードは、パターンと一致する、辞書編集方式の順にソートされたファイル名に置換されます。パターンと一致するファイル名が見つからない場合、ワードは変更されません。ファイル名の先頭のピリオド (.) または スラッシュ (/) 直後のピリオドは、明示的に一致しなければなりません (後者の場合はスラッシュ自体をも含む) 。

*	NULL 文字列を含め、任意の文字列と一致します。
?	任意の単一文字と一致します。
[...]	囲まれた文字のいずれかと一致します。2 つの 文字を - で区切ると、その間にある任意の文字 (その 2 つの文字も含む) に一致します。先頭の [の次の文字が ! である場合、 で囲まれていない任意の文字と一致します。

クォートされているすべての文字 (下記「クォート」を参照) は、ファイル名において明示的に一致しなければなりませんので、ご注意ください。

クォート

次の文字はシェルに対しては特別な意味を持ち、クォートしない (後述の説明を参照) 限り ワードの終わりを表します。

; & ( ) | ^ < > 復帰改行 空白文字 タブ

これらの文字をクォートする、つまり文字自身を表すには、文字の前にバックスラッシュ (\) を付けるか、または一対の引用符 (' ' または "") で囲みます。シェルは、特定の文字をクォートして、それらが特別の意味を持たないようにすることがあります。単一の文字をクォートするのに用いるバックスラッシュは、コマンド実行前にワードから取り除かれます。\ と復帰改行との組み合せは、コマンドとパラメタの置換前にワードから取り除かれます。

一対の単一引用符 (' ') で囲まれたすべての文字 (ただし単一引用符は除く) は、シェルによってクォートされます。バックスラッシュは、一対の単一引用符で囲まれていれば特殊な意味を持ちません。単一引用符は、一対の二重引用符で囲めばクォートされますが (例 " '") 、一対の単一引用符で囲んでもクォートされません。

一対の二重引用符 ("") の中では、パラメタとコマンドの置換が実施され、シェルは、その結果をクォートして、ブランクの解釈とファイル名の生成が行われないようにします。$* が一組の二重引用符で囲まれている場合、定位置パラメタは置換され、クォートされ、クォートされた空白で分けられます ("$1 $2 ...") 。しかし $@ が一組の二重引用符で囲まれている場合は、定位置パラメタは置換され、クォートされ、クォートされていない空白 ("$1" "$2" ...) で分けられます。\ は \、'、"、, (コンマ)、$ といった文字をクォートします。\ と復帰改行との組み合せは、コマンドとパラメタの置換前にワードから取り除かれます。バックスラッシュが \、'、"、, (コンマ)、$、および 復帰改行以外の文字の前に付く場合は、バックスラッシュ自体がシェルによってクォートされます。

プロンプト

シェルは、対話的に使用すると、コマンドを読み取る前に PS1 の値によるプロンプトを発行します。復帰改行を入力したあとで、コマンドを完了するためにさらに入力が必要な場合は、2 次プロンプト (PS2 の値) が出力されます。

環境

環境 (*environment*) は、通常の引数リストが 実行されるプログラムに引き渡される場合と同様の方法で引き渡される、名前と値の対の集まりです (environ(5) を参照) 。シェルが環境と対話する方法はいくつかあります。シェルは、呼び出されると、環境を走査して、見つけた名前ごとに変数を作成し、対応する値を設定します。ユーザーがこれらのパラメタの値を変更したり 新しいパラメタを作成したときには、export コマンドを用いてシェルのパラメタを環境に関連付けなければ、これらのパラメタは環境に何の影響も与えません (set -a の説明を参照) 。環境からパラメタを削除するには、unset コマンドを使用します。したがって、実行されるコマンドが参照する環境は、シェルが最初に引き継いだ「名前 = 値」の対のうち 変更されていないものから、unset によって削除された対を引き、変更または追加した対をくわえたものです。これらはいずれも export コマンドで指定する必要があります。

単純コマンドの環境は、いくつかのパラメタ代入指定を 先頭に付加すれば拡張できます。

以下は、*command* が特殊コマンドでなければ、*command* の実行に関するかぎり同じことを意味します。

TERM=450 *command*

および

(export TERM; TERM=450; *command*

command が特殊コマンドの場合、次の指定は、現在のシェル内の TERM 変数を修正します。

TERM=450 *command*

-k フラグを設定すると、すべてのキーワード引数は 環境に格納されます。これらの引数がコマンド名の後に指定された場合も同様です。以下の例では、まず a=b c を、次に c を表示します。

```
echo a=b  c

a=b  c
```

```
set  –k

echo a=b  c

c
```

シグナル　起動されたコマンドに対する INTERRUPT シグナルと QUIT シグナルは、コマンドの後に & が付く場合には無視されます。その他の場合、シグナルは、シェルが親から引き継いだ値を持ちます。ただし、シグナル 11 だけは例外です (後述の trap コマンドの説明を参照) 。

実行　コマンド実行のたびに、前述の コマンドの置換、パラメタの置換、ブランクの解釈、入出力のリダイレクション、およびファイル名の生成が行われます。コマンド名が定義済みの関数名と一致する場合、その関数がシェルプロセスで実行されます (これと実行時にサブシェルを要求する シェルスクリプトファイルの実行の違いに注意) 。定義済み関数名とは一致しないが、後述の特殊コマンドの いずれかと一致するコマンド名の場合、そのコマンドがシェルプロセスで実行されます。

定位置パラメタの $1、 $2、 ... は関数の引数に設定されます。コマンド名が特殊コマンド とも定義済み関数の名前とも一致しない場合、新しいプロセスが作成され、exec(2) を用いてそのコマンドの実行が試みられます。

PATH というシェルパラメタは、コマンドを含んでいる ディレクトリの検索パスを定義します。2 つのディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは /usr/bin です。現在のディレクトリは NULL パス名によって指定されます。これは等号の直後、パスリスト内の任意の場所にある 2 つの区切り文字のコロンの間、またはパスリストの最後に記述できます。コマンド名に / が含まれている場合は、検索パスは使用されません。/ が含まれていなければ、パスにおける各ディレクトリに実行可能ファイルがあるか検索します。ファイルが実行権を有するが、それが a.out ファイルでない場合、シェルコマンドの入ったファイルとみなされます。そのファイルを読み取るときは、サブシェルが生成されます。括弧で囲まれたコマンドもサブシェル内で実行されます。

シェルは、(あとで不必要な exec を行わなくてもいいように) 検索パス内のコマンドの位置を記憶します。コマンドが相対ディレクトリにあった場合、その位置を 現在のディレクトリの変更のたびに再決定しなければなりません。シェルは、PATH 変数が変更されるか hash -r コマンドが実行されるたびに、記憶していたすべての位置を忘れてしまいます (下記参照) 。

特殊コマンド　以下に示す特殊コマンドに対しては、入出力のリダイレクションが可能です。ファイル記述子 1 は、デフォルトの出力位置です。ジョブ制御が可能なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます (「ジョブ制御」の項を参照) 。

:
コマンドは何もせず、実行による影響は何もありません。終了ステータス 0 が返されます。

. *filename*
filename からコマンドを読み取り実行し、戻ります。PATH によって指定された検索パスを用いて、*filename* を含むディレクトリを探します。

bg [%*jobid* . . .]
ジョブ制御が可能なとき、bg コマンドはジョブの操作用に ユーザー環境に追加されます。停止状態のジョブをバックグラウンドで再び実行します。%*jobid* を省略すると、現在のジョブとみなされます。(詳細については後述の 「ジョブ制御」の項を参照)

break [*n*]
for または while ループがあれば抜け出します。*n* を指定すると、*n* レベル分ブレークします。

cd [*argument*]
現在のディレクトリを *argument* に変更します。シェルパラメタ HOME は、*argument* のデフォルト値です。シェルパラメタ CDPATH は、*argument* を含むディレクトリの検索パスを定義します。2 つのディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定) 。なお現在のディレクトリは 空のパス名で指定します。このパス名は、等号の直後か、パスリスト内の区切り文字のコロンの間に指定します。*argument* の先頭文字が / の場合、検索パスは使用しません。それ以外の場合は、パス中の各ディレクトリで *argument* を検索します。

chdir [*dir*]
chdir はシェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。引数を指定しないと、そのユーザーのホームディレクトリに変更します。*dir* が現在のディレクトリからは見つからない相対パス名の場合、変数 CDPATH 環境内のディレクトリリストを検索します。*dir* が / で始まる値を持つシェル変数の名前である場合、その変数の値が示すディレクトリに変更します。

continue [*n*]
for または while ループの次の繰り返しを実行します。*n* を指定すると、*n* 番目のループから実行します。

echo [*arguments* . . .]
arguments の文字列が 空白文字に区切られて、シェルの標準出力に書かれます。引数をエコーバックします。使用法と説明については echo(1) を参照してください。

eval [*argument* . . .]
引数をシェルへの入力として読み取り、生成されるコマンドを実行します。

exec [*argument* . . .]
このシェルの代わりに、引数で指定されたコマンドを (新規プロセスは生成せずに) 実行します。入出力引数が指定可能で、その他の引数が指定されない場合は、これによってシェルの入出力が変更されます。

exit [*n*]
呼び出し元のシェルまたはシェルスクリプトを *n* で指定した終了ステータスで終了させます。*n* を省略すると、最後に実行されたコマンドの終了ステータス がシェルの終了ステータスになります。EOF によっても、シェルは終了します。

`export` [*name* . . .]
指定された *name* 群に対し、後で実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。引数を省略すると、現在のシェル実行中にエクスポートのマークが付けられた 変数名を一覧表示します。親シェルからエクスポートされた変数名は、現在のシェル実行中に再びエクスポートされた場合にだけ一覧表示されます。関数名はエクスポートされません。

`fg` [`%`*jobid* . . .]
ジョブ制御が可能なとき、`fg` はジョブの操作用に ユーザー環境に追加されます。このコマンドは、フォアグラウンド状態の停止ジョブを再び実行します。また、実行中のバックグラウンドジョブをフォアグラウンドへ移します。`%`*jobid* を省略すると、現在のジョブとみなされます。(詳細については後述の 「ジョブ制御」 の項を参照)

`getopts`
コマンドシンタクス標準のサポート用に、シェルスクリプト内で使用されるコマンドです (`intro`(1) を参照) 。このコマンドは、定位置パラメタを構文解析し、オプションの正当性をチェックします。使用法と説明については、`getoptcvt`(1) を参照してください。

`hash` [`-r`] [*name* . . .]
シェルは、各 *name* ごとに、それが示すコマンドの検索パス内の位置を決定し、記憶します。`-r` オプションを指定すると、シェルは記憶したすべての位置を忘れます。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。この出力表示において、*hits* はシェルプロセスによってコマンドが呼び出された回数を表します。*cost* は、検索パスのコマンドを見つけるのに必要な作業です。コマンドが検索パスの「相対」ディレクトリにある場合、そのディレクトリの変更後に そのコマンドが格納された位置が再計算されます。この再計算が行われる予定のコマンドに対しては、*hits* 情報の隣にアスタリスク (`*`) が示されます。*cost* の値は、再計算が行われるたびに増加されます。

`jobs` [`-p`|`-l`] [`%`*jobid* ...]

`jobs -x` `command` [*arguments*]
停止中またはバックグラウンドで実行中の すべてのジョブを報告します。`%`*jobid* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが報告されます (詳細については後述の 「ジョブ制御」 の項を参照) 。

`kill` [*–sig*] `%`*job* . . .

`kill -l`
`TERM` (終了) シグナルまたは指定されたシグナルのいずれかを、指定されたジョブまたはプロセスに送信します。シグナルは、番号または名前 (`signal`(3HEAD) の場合と同様に “SIG” という接頭辞を取り除いたもの (`CHLD` と名付けられた `SIGCHD` 以外)) で指定します。送信するシグナルが `TERM` (終了) または `HUP` (ハングアップ) の場合、停止中のジョブまたはプロセスには `CONT` (継続) シグナルを送信します。*job* という引数は、活動中のジョブではないプロセスのプロセス ID を指定することもできます。後述の「ジョブ制御」の項を参照してください。第 2 の形式の `kill –l` は、シグナル番号とシグナル名をリスト表示します。(`kill`(1) 参照)

`login` [*argument* . . .]
'`exec login` *argument*. . . .' と同機能です。使用法と説明については、`login`(1) を参照してください。

`newgrp` [*argument*]
`exec newgrp` *argument*. と同機能です。使用法と説明については、`newgrp`(1) を参照してください。

`pwd`
現在の作業用ディレクトリを表示します。使用法と機能説明については、`pwd`(1) を参照してください。

`read` *name* . . .
標準入力から 1 行を読み取り、内部フィールドセパレータの `IFS` (通常は空白文字またはタブ) を用いてワード境界を区切り、最初のワードを最初の *name* に割り当て、2 番目のワードを 2 番目の *name* に割り当て、続くワードも順次割り当てます。残ったワードは最後の *name* に割り当てます。\ に続いて復帰改行を入力すれば、行を継続できます。復帰改行以外の文字の前にバックスラッシュを付加すれば、その文字をクォートできます。このバックスラッシュは、ワードが *name* に割り当てられる前に削除され、バックスラッシュの後に位置する文字は解釈されません。EOF に到達した場合を除き、リターンコードは `0` となります。

`readonly` [*name* . . .]
指定された *name* に読み取り専用のマークを付け、これらの名前が後続の割り当てでは変更できないようにします。引数を省略すると、読み取り専用と指定された名前がすべて一覧表示されます。

`return` [*n*]
関数を、*n* が示すリターンステータスで終了させます。*n* を省略すると、リターンステータスは最後に実行された コマンドのリターンステータスになります。

`set` [`- -aefhkntuvx` [*argument* . . .]]

`-a` エクスポート用に修正または作成された変数にマークを付けます。

`-e` コマンドが 0 以外の終了ステータスで終了した場合、直ちに終了します。

`-f` ファイル名を生成しないようにします。

`-h` 関数の定義時に、関数コマンドを検索しその位置を記憶します (通常、関数コマンドは関数実行時に検索されます) 。

`-k` コマンド名の前にあるキーワード引数だけでなく、すべてのキーワード引数がコマンド用の環境に置かれます。

`-n` コマンドを読み取るが、実行しません。

`-t` 1 つのコマンドを読み取り、実行し、終了します。

`-u` 未設定の変数を置換時にエラーとして扱います。

`-v` シェルへの入力行の読み取り時に、その内容を表示出力します。

`-x` コマンドおよび引数の実行時に、その内容を表示出力します。

– どのフラグも変更しません。$1 に – を設定する際に便利です。

– の代わりに + を使用すると、これらのフラグがオフになります。これらのフラグはシェル起動時にも使用できます。現在セットされているフラグは、$– で見つけられます。残りの引数は定位置パラメタで、$1、 $2、 ... へ順に割り当てられます。引数が指定されない場合、 すべての名前の値が出力されます。

shift [*n*]
$*n*+1 . . . から始まる一連の定位置パラメタを $1 . . . に再命名 (リネーム) します。*n* を省略すると、1 とみなされます。

stop *pid* . . .
pid (プロセス ID 番号) の実行を停止します (ps(1) 参照) 。

suspend
現在実行中のシェルを停止します。(ログインシェルの場合は停止しません。)

test
条件式を評価します。使用法と説明については、test(1) を参照してください。

times
シェルから実行されたプロセスの ユーザーおよびシステム時間の累積値を出力します。

trap [*argument n* [*n2* . . .]]
argument が示すコマンドを、シェルが数値または記号シグナル (*n*) を受信した時に読み取り実行します。なお *argument* は、トラップ設定時に 一度、トラップ取り出し時に 一度検索されます。トラップコマンドは、シグナル番号または 対応するシンボリック名の順序で実行されます。現在のシェルで無視されているシグナルにトラップを 設定しようとしても無効となります。シグナル 11 (メモリフォールト) にトラップを指定しようとすると、エラーになります。*argument* を省略すると、すべてのトラップ *n* は元の値に再設定されます。*argument* が NULL 文字列の場合、シェルおよびシェルが呼び出したコマンドは、このシグナルを無視します。*n* が 0 の場合、*argument* が示すコマンドはシェル終了時に実行されます。引数なしの trap コマンドは、コマンドの一覧を 各々が対応しているシグナル番号とともに表示します。

type [*name* . . .]
各 *name* ごとに、コマンド名として使用される場合に どのように解釈されるかを指示します。

ulimit [–[HS] [a | cdfnstv]]

ulimit [–[HS] [c | d | f | n | s | t | v]] limit
ulimit は、資源の強い制限値または弱い制限値を 表示または設定します。これらの限界については getrlimit(2) の説明を参照してください。

limit 引数を省略すると、ulimit は指定されている限界値を表示します。限界値は一度にいくつでも表示できます。-a オプションは限界値すべてを表示します。

limit 引数を指定すると、ulimit は指定されたフラグに対応する限界値をその引数の値に設定します。*limit* 引数の値として unlimited という文字列を指定すると、有効な最大値に設定されます。一度に資源 1 つについてだけ限界値を設定できま

す。ユーザーは誰でも、弱い限界値を強い限界値を超えない 任意の値に設定できます。ユーザーは誰でも、強い限界値を下げることができます。強い限界値を上げることができるのはスーパーユーザーだけです。詳しくは su(1M) を参照してください。

-H オプションはハード限界を表し、-S オプションはソフト限界を表します。どちらのオプションも指定しない場合、ulimit は両方の限界値を設定し、弱い限界値を表示します。

以下のオプションは、限界値を表示または設定する資源を指定します。オプションをいっさい指定しないと、ファイルサイズ限界値を表示または設定します。

-c 最大 core ファイルサイズ (512 バイトブロック単位)

-d データセグメントまたはヒープの最大サイズ (K バイト単位)

-f 最大ファイルサイズ (512 バイトブロック単位)

-n 最大ファイル記述子プラス 1

-s スタックセグメントの最大サイズ (K バイト単位)

-t 最大 CPU 時間 (秒単位)

-v 仮想メモリの最大サイズ (K バイト単位)

システムで利用可能な最大上限値を 調べるには sysdef(1M) コマンドを実行してください。表示される値は 16 進数ですが、bc(1) ユーティリティを使って 10 進数に変換できます。swap(1M) を参照してください。

たとえば、0 メガバイトに コアファイルダンプのサイズを 制限するには次のように入力します。

```
ulimit -c 0
```

umask [*nnn*]
ユーザーファイル作成マスクを *nnn* が示す値に設定します (umask(1) を参照) 。*nnn* を省略すると、マスクの現在の値を出力します。

unset [*name* . . .]
name ごとに、対応する変数または関数値を削除します。変数 PATH、 PS1、 PS2、MAILCHECK、 および IFS は設定解除できません。

wait [*n*]
当該ユーザーのバックグラウンドプロセスのうち ID が *n* のプロセスを待ち、その終了ステータスを報告します。*n* が省略された場合、当該ユーザーの現在活動中のすべての バックグラウンドプロセスを待ち、リターンコードは 0 になります。

呼び出し

exec(2) を介してシェルが呼び出される場合で、引数 0 の先頭文字が – のとき、コマンドは、まず /etc/profile から読み込まれ、次に $HOME/.profile から読み込まれます (これらのファイルがある場合) 。その後、コマンドは後述のように読み込まれます。シェルが /usr/bin/sh として呼び出される場合にも、このようになりま

す。以下に述べるフラグは、呼び出し時にのみ、シェルによって解釈されます。なお -c または -s フラグが指定されないかぎり、先頭引数はコマンドを含むファイルの名前であるとみなされ、残りの引数は定位置パラメタとして そのコマンドファイルに引き渡されます。

`-c` *string*	このフラグが指定されると、*string* からコマンドを読み取ります。
`-i`	このフラグが指定された場合あるいは シェル入出力が端末に接続されている場合、このシェルは対話型となります。この場合、`kill 0` が対話型シェルを終了しないように TERM を無視し、`wait` が割り込み可能になるように INTERRUPT を捕え、無視します。いずれの場合も、シェルは QUIT を無視します。
`-p`	このフラグが指定されると、シェルは実効ユーザーおよびグループ ID に、実ユーザーおよびグループ ID を設定しません。
`-r`	このフラグを指定すると、シェルは制限付きシェルになります (`rsh`(1M) を参照) 。
`-s`	このフラグが指定された場合または引数が残っていない場合、標準入力からコマンドを読み取ります。引数が残っていれば、それらは定位置パラメタを指定します。前述の特殊コマンドの出力を除くシェル出力は、ファイル記述子 2 に書き出されます。

他のフラグと引数については、前述の `set` コマンドの箇所で説明されています。

ジョブ制御 (jsh)

シェルを `jsh` として呼び出すと、`sh` の説明で述べたすべての機能に加えて、ジョブ制御が可能になります。通常、ジョブ制御は対話型シェルに対してだけ可能です。非対話型シェルは、ジョブ制御の機能が追加されても、その恩恵を受けないのが通常です。

ジョブ制御が可能な場合、ユーザーが端末から入力したコマンドまたはパイプラインは、すべてジョブ (*job*) と呼ばれます。どのジョブも、必ずフォアグラウンド、バックグラウンド、または停止のいずれかの状態にあります。これらの用語の定義を次に示します。

1. フォアグラウンド状態にある ジョブは、制御している端末への読み取り および書き込み権を持っています。
2. バックグラウンド状態にある ジョブは、制御している端末への読み取り 権を拒否されていますが、条件付き 書き込み権は持っています (`stty`(1) を参照) 。
3. 停止ジョブは保留状態に置かれた ジョブであり、通常は `SIGTSTP` シグナルにより、この状態になります (`signal`(3HEAD) を参照) 。

シェルが起動するすべてのジョブには、ジョブ番号 (*job number*) と呼ばれる 正の整数が割り当てられます。シェルはこの番号を把握し、特定のジョブを示す識別子として使用します。また、シェルは現在 (*current*) および直前 (*previous*) のジョブも把握しています。現在のジョブとは、最後に起動または再起動されたジョブです。前回のジョブとは、その直前のジョブです。

ジョブ識別子の正しい構文は次のような形式です。

%*jobid*

このうち *jobid* は、次のいずれかの形式で指定できます。

%	または + 現在のジョブ
–	前回のジョブ
?<*string*>	コマンド行が *string* を含んでいるジョブ
n	ジョブ番号が *n* のジョブ
pref	コマンド名の先頭が *pref* のコマンド。たとえば `ls –l name` がバックグラウンドで実行中だった場合、`%ls` と指定すればこのコマンドを示すことができます。*pref* は、クォートしない限り、ブランクを含めることができません。

ジョブ制御が可能なとき、ジョブの操作用に 次のコマンドがユーザー環境に追加されます。

`bg` [`%`*jobid . . .*]
停止状態のジョブをバックグラウンドで再び実行します。`%`*jobid* を省略すると、現在のジョブとみなされます。

`fg` [`%`*jobid . . .*]
停止状態のジョブをフォアグラウンドで再び実行します。また、実行中のバックグラウンドジョブをフォアグラウンドへ移します。`%`*jobid* を省略すると、現在のジョブとみなされます。

`jobs` [`-p`|`-l`] [`%`*jobid . . .*]

`jobs -x command` [*arguments*]
停止中またはバックグラウンドで実行中の すべてのジョブを報告します。`%`*jobid* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが報告されます。次のオプションを使って、ジョブの出力を変更できます。

`-l`	ジョブのプロセスグループ ID および作業ディレクトリを報告します。
`-p`	ジョブのプロセスグループ ID のみを報告します。
`-x`	コマンドまたは引数中に見つかった *jobid* を、対応するプロセスグループ ID に置き換え、コマンドに引数を渡して実行します。

`kill` [`–`*signal*] `%`*jobid*
`kill` コマンドの組み込みバージョン。*jobid* で示すプロセスに対して `kill` コマンドの機能を提供します。

`stop %`*jobid . . .*
バックグラウンドジョブの実行を停止します。

`suspend`
現在のシェルの実行を停止します (ただし、ログインシェルの場合は停止しません)。

`wait` [`%`*jobid* . . .]
`wait` コマンドの組み込みバージョンで、ジョブ識別子の指定を受け入れます。`%`*jobid* が省略された場合、`wait` は、前述の「特殊コマンド」で説明したように動作します。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `sh` と `jsh` の動作については、`largefile`(5) を参照してください。

終了ステータス

構文エラーなどのエラーを検出すると、シェルは 0 以外の終了ステータスを返します。シェルを非対話型で使用している場合、シェルファイルの実行は中止されます。対話型で使用している場合は、シェルは最後に実行されたコマンドの 終了ステータスを返します (上記の `exit` コマンドの説明を参照) 。

jsh のみ

シェルが `jsh` として呼び出された場合、停止ジョブがあるのにシェルを終了させようとすると、シェルは次のような警告を出します。

```
There are stopped jobs.
```

これが唯一のメッセージです。もう一度終了が試みられ、停止ジョブがまだ存在している場合、これらのジョブにカーネルから `SIGHUP` シグナルが送られ、シェルは終了します。

ファイル

`$HOME/.profile`

`/dev/null`

`/etc/profile`

`/tmp/sh*`

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/jsh

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/sh

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

`intro`(1), `bc`(1), `echo`(1), `getoptcvt`(1), `kill`(1), `ksh`(1), `login`(1), `newgrp`(1), `ps`(1), `pwd`(1), `set`(1), `shell_builtins`(1), `stty`(1), `test`(1), `umask`(1), `wait`(1), `rsh`(1M), `su`(1M), `swap`(1M), `sysdef`(1M), `dup`(2), `exec`(2), `fork`(2), `getrlimit`(2), `pipe`(2), `ulimit`(2), `setlocale`(3C), `signal`(3HEAD), `passwd`(4), `profile`(4), `attributes`(5), `environ`(5), `largefile`(5), `XPG4`(5)

警告 | シェルスクリプトを `setuid` して使用することは避けてください。

注意事項 | 入出力のリダイレクションでファイル名に使用されたワードは、ファイル名生成では解釈されません (上記の「ファイル名の生成」の項を参照) 。たとえば `cat file1 >a*` という指定は、`a*` という名前のファイルを生成します。

パイプラインにあるコマンド群は それぞれ別個のプロセスとして稼動するので、パイプラインに設定された変数は親シェルには何の影響も与えません。

`cannot fork,too many processes` というエラーメッセージを受け取った場合には、`wait`(1) コマンドを用いてユーザーのバックグラウンドプロセスを クリーンアップしてください。それでも効果がない場合には、おそらくシステム・プロセステーブルが満杯であるか、または活動中のフォアグラウンドプロセスがありすぎるためです。ユーザーログインに結合するプロセス ID の数、およびシステムが把握できる数には 限度があります。

パイプラインの最後のプロセスだけが、待つ対象になり得ます。

あるコマンドを実行し、その後で同一名のコマンドが、検索パスにおいて元のコマンドがあるディレクトリの 前に位置するディレクトリにインストールされた場合、シェルは元のコマンドの方を実行し続けます。新しい方のコマンドを実行させたければ、`hash` コマンドを使用してください。

Bourne シェルにはプロセスの実効ユーザー ID に対して制限があります。このユーザー ID が 100 よりも小さい (さらにプロセスの実ユーザー ID と同等ではない) 場合には、ユーザー ID はプロセスの実ユーザー ID にリセットされます。

同じプロセスグループで フォアグラウンドジョブと バックグラウンドジョブの両方を シェルが実行しているため、ジョブは同じシグナルを受け取り、予期しない結果を招くことがあります。したがって、特に対話型のシェルを動作している場合は、他のジョブ制御のシェルを使用することを おすすめします。

存在しないコマンドのインタプリタを実行しようとするシェルスクリプトを、シェルが処理した場合、シェルはシェルスクリプトが存在しないという間違った診断メッセージを 返します。

名前 | kill – プロセスの終了またはシグナル送出

形式 | **/usr/bin/kill** -s *signal_name pid...*

/usr/bin/kill -l [*exit_status*]

/usr/bin/kill [-*signal_name*] *pid...*

/usr/bin/kill [-*signal_number*] *pid...*

機能説明 | kill ユーティリティは、各 *pid* オペランドによって指定されたプロセス (1 つまたは複数) にシグナルを送信します。

pid オペランドのそれぞれについて、kill ユーティリティは以下の引数で呼び出された kill(2) 関数と等価のアクションを実行します。

1. *pid* オペランドの値は、*pid* 引数として使用されます。
2. *sig* 引数は、-s オプション、-*signal_name* オプション、または -*signal_number* オプションで指定された値です。あるいは、これらのオプションが何も指定されていない場合は、SIGTERM で指定された値です。

シグナルが送信されるプロセスは、ユーザーがスーパーユーザーでない限り 現在のユーザーに属していなければなりません。

kill のシェル組み込みバージョンの説明については、「注意事項」を参照してください。

オプション | 以下のオプションを使用できます。

-l (文字エル) 。オペランドが何もない場合、当該システムでサポートされている *signal_name* のすべての値を書き出します。*exit_status* オペランドが指定され、それがシェル特殊パラメータ値 ? およびシグナルが終了したプロセスに対応する wait の場合は、そのプロセスを終了するシグナルに対応する *signal_name* が書き出されます。 *exit_status* オペランドが指定され、それがシグナル番号を示す符号なしの 10 進整数値なら、そのシグナルに対応する *signal_name* が書き出されます。それ以外の場合、結果は不定です。

-s *signal_name* <signal.h> に定義されている記号名の 1 つを使って、送信するシグナルを指定します。 *signal_name* の値は、 SIG 接頭辞なしで、大文字／小文字を区別しない方法で認識されます。 さらに記号名 0 は、ゼロのシグナル値を表すものと認識されます。SIGTERM の代わりに、対応するシグナルが送信されます。

-*signal_name* -s *signal_name* と同じです。

-*signal_number* SIGTERM の代わりに使用するシグナルを表す、負でない 10 進整数 *signal_number* を指定します。kill(2) への有効な呼び出しにおける *sig* 引数と同じです。

オペランド | 次のオペランドを指定できます。

pid 以下の 1 つです。

1. シグナルが送信されるプロセスまたはプロセスグループを指定する 10 進の整数。 *pid* オペランドの正の値、負の値、またはゼロで選択されたプロセスは、 kill 関数で説明されているものと同じです。プロセス番号 0 が指定されると、プロセスグループ内のすべての プロセスにシグナルが送信されます。最初の *pid* オペランドに負の値を指定する場合、 オプションとして解釈されないように、前に – – を付加しなければなりません。
2. シグナルが送信されるバックグラウンドプロセスグループを識別する ジョブ制御のジョブ ID 。ジョブ制御ジョブ ID 表記が適用できるのは、現在のシェル実行環境で `kill` を呼び出すときだけです。

注：*pid* のジョブ制御ジョブ ID タイプが使用できるのは、ジョブ制御オプションを サポートしているシステムだけです。

exit_status シグナル番号を指定する 10 進の整数または シグナルが終了したプロセスの終了ステータス。

使用法 プロセス番号は `ps`(1) を使って知ることができます。

`kill` が独自のユーティリティ実行環境で動作するときは、ジョブ制御ジョブ ID 表記は、前述のような働きをする必要はありません。以下の例の `kill` はどちらも異なる環境で動作し、 各ジョブ番号については関連性はありません。

```
example% nohup kill %1 &
example% system( "kill %1");
```

出力 `-l` オプションが指定されなければ、標準出力は使用されません。

`-l` オプションが指定されると、 各シグナルの記号名が以下のフォーマットで書き出されます。

`"%s%c", <`*signal_name*`>, <`*separator*`>`

この <*signal_name*> は、 *SIG* 接頭辞を付けずに大文字で指定します。 <*separator*> は復帰改行文字 (NEWLINE) または空白文字です。最後に書き出されるシグナルの場合、 <*separator*> は復帰改行文字になります。

`-l` オプションと *exit_status* オペランドの両方が指定されると、 対応するシグナルの記号名が以下のフォーマットで書き出されます。

`"%s\n", <`*signal_name*`>`

使用例

例 1 強制終了シグナルを送信する

送信側プロセスが指定されたプロセスへのシグナル送信を許可されていて、 指定されたプロセスが存在するという条件で、以下のコマンドはどれも、100 という ID を持つプロセスと、165 というプロセスグループ ID を持つすべてのプロセスに SIGKILL シグナルを送信します。

```
example% kill -9 100 -165
example% kill -s kill 100 -165
example% kill -s KILL 100 -165
```

例 2 最初の負の引数のあいまいさを避ける

シグナル番号またはプロセスグループを指定する最初の負の引数のあいまいさを 避けるため、最初の負の数はいつもシグナルとします。そのため、プロセスグループ (たとえば 123) にデフォルトシグナルを送信する場合、アプリケーションは以下のようなコマンドを使用しなければなりません。

```
example% kill -TERM -123
example% kill -- -123
```

環境

kill の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　各 *pid* オペランドに対して一致するプロセスが 1 つ以上見つかり、さらに 1 つ以上のプロセスに対して指定されたシグナルが処理された。

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

csh(1), jobs(1), ksh(1), ps(1), sh(1), shell_builtins(1), wait(1), kill(2), signal(3C), signal(3HEAD), attributes(5), environ(5)

sh

sh には kill コマンドに、 *jobid* でプロセスを識別する機能を追加した、 組み込みコマンドが用意されています。 sh の構文を以下に示します。

```
kill [ -sig ] [ pid ] [ %job ]...
kill -l
```

csh

C シェルと csh にも kill の組み込みコマンドが用意されています。構文は以下のとおりです。

```
kill [-sig][pid][%job]...
kill -l
```

csh における kill の組み込みコマンドは TERM (終了) シグナル (デフォルトの場合) または指定されたシグナルを、指定されたプロセス ID、指定されたジョブ、または現在のジョブへ送信します。 シグナルは番号または名前で指定します。 シグナルを送るプロセスまたはジョブにデフォルトはなく、 kill だけを入力しても現在のジョブにはシグナルを送信しません。 送信中のシグナルが TERM (終了) または HUP (ハングアップ) の場合、そのジョブまたはプロセスには CONT (継続) シグナルも送られます。

-l　　送信可能なシグナル名の一覧を表示します。

ksh

ksh における kill の構文は以下の通りです。

```
kill [-sig][pid][%job]...
kill -l
```

ksh における kill は TERM (終了) シグナルまたは指定されたシグナルのいずれかを、 指定されたジョブまたはプロセスに送信します。シグナルは、 番号または名前 (signal(3HEAD) の場合と同様に "SIG" という接頭辞を取り除いたもの) で指定します。 送信するシグナルが TERM (終了) または HUP (ハングアップ) の場合、停止中のジョブまたはプロセスには CONT (継続) シグナルを送信します。引数 *job* には、活動中のジョブの 1 つのメンバーではないプロセスのプロセス ID を指定することもできます。第 2 の形式の kill -l は、シグナル番号とシグナル名をリスト表示します。

名前 | ksh, rksh – Korn シェル。標準/制限付きコマンドとプログラミング言語

形式 | **/usr/bin/ksh** [± abCefhikmnoprstuvx] [± o *option*...] [*arg*...]

/usr/bin/ksh -c [± abCefhikmnoprstuvx] [± o *option*...] *command_string* [*command_name* [*arg*...]]

/usr/xpg4/bin/sh [± abCefhikmnoprstuvx] [± o *option*...] [*arg*...]

/usr/xpg4/bin/sh -c [± abCefhikmnoprstuvx] [± o *option*...] *command_string* [*command_name* [*arg*...]]

/usr/bin/rksh [± abCefhikmnoprstuvx] [± o *option*...] [*arg*...]

/usr/bin/rksh -c [± abCefhikmnoprstuvx] [± o *option*...] *command_string* [*command_name* [*arg*...]]

機能説明 | /usr/xpg4/bin/sh ユーティリティは標準に準拠したシェルです。このユーティリティは /usr/bin/ksh のすべての機能を提供します。ただし、後述するように、動作が異なる場合を除きます。詳細については、「算術展開」の項を参照してください。

/usr/bin/ksh は、端末またはファイルから読み取られたコマンドを 実行するコマンドおよびプログラミング言語です。rksh は、標準コマンドインタプリタである ksh と比べて、機能の一部が制限されており、標準シェルよりも制限された機能を持った ログイン名や実行環境を設定します。シェルへの引数の意味については、後述の「呼び出し」を参照してください。

定義 | メタキャラクタ (*metacharacter*) は、次の文字のいずれかです。

; & ( ) | < > 復帰改行(NEWLINE) 空白文字 タブ

ブランク (*blank*) とは、タブ (TAB) または空白文字 (SPACE) のことです。識別子 (*identifier*) とは、英文字、数字、または下線の並びのことで、その先頭の文字は英文字または下線です。識別子は関数 (*function*) や変数 (*variable*) の名前として使用します。ワード (*word*) とは、引用符がつかない 1 つまたは複数のメタキャラクタで区切られた、文字の並びのことです。

コマンド (*command*) とは、シェル言語の文法にそった文字の並びのことです。シェルは各コマンドを読み取り、指定された動作を直接実行するか、または動作を実行するユーティリティを起動します。特殊コマンド (*special-command*) とは、個別のプロセスを作成しなくても、シェルが実行してくれるコマンドです。本書で述べる副作用が発生する場合を除き、ほとんどの特殊コマンドは それぞれ個別のユーティリティとして実装できます。

コマンド | 単純コマンド (*simple-command*) は、ブランクで区切られたワードの並びで、その前に変数代入リストを指定できます (後述の「環境」を参照) 。先頭のワードは、実行するコマンド名を指定します。残りのワードは、以下に述べる場合を除き、呼び出されたコマンドに引数として渡されます。コマンド名は引数 0 として渡されます (exec(2) を参照) 。単純コマンドの値 (*value*) は、正常終了した場合は終了状態の値、シグナルを

受け取って異常終了した場合は、シグナル番号に 128 を足した値になります。シグナルの値については、signal(3HEAD) のリストを参照してください。なお、正常な終了状態の値 129 ～ 255 と、シグナル番号 1 ～ 127 を受け取って異常終了した場合の値を見分けることはできません。

パイプライン (*pipeline*) は、パイプ (|) で区切られた 1 つ以上のコマンドの並びです。最後のコマンドを除き、各コマンドの標準出力は pipe(2) によってその次のコマンドの標準入力と結合されます。各コマンドは、別々のプロセスとして実行されます。シェルは最後のコマンドが終了するのを待ちます。最後のコマンドの終了状態がパイプライン全体の終了状態となります。

リスト (*list*) は、;、&、&&、または || で区切られた 1 つ以上のパイプラインの並びです。その並びの終わりに ;、&、または |& を記述することもできます。この 5 つの記号の中で、;、&、および |& の優先度は同じで、&& と || の優先度より低くなります。&& と || の優先度は同じです。セミコロン (;) によって、直前のパイプラインが順次実行されます。アンパサンド記号 (&&) によって、直前のパイプラインが非同期的に実行されます (つまりシェルはパイプラインが終了するのを待ちません) 。|& という記号によって、親シェルに対して双方向パイプが確立された、直前のコマンドまたはパイプラインが非同期的に実行されます。

生成されたコマンドの標準入出力は、親シェルが特殊コマンドの read および print (後述「特殊コマンド」を参照) の -p オプションを使用して書き込み、読み取ることができます。&& (または ||) という記号によって、直前のパイプラインが 0 (またはゼロ以外) の終了ステータスを返した場合にだけ、その後のリストが実行されます。コマンドの区切りとして、セミコロンの代わりに任意の数の復帰改行を *list* に指定できます。

コマンドは、単純コマンドまたは以下のいずれかです。特に断わりのないかぎり、コマンドが返す値は、そのコマンド中で最後に実行された単純コマンドの値です。

for *identifier* [in *word* . . .] ; do *list* ; done
: for コマンドが実行されるたびに、*identifier* は in *word* リストから次に得られる *word* に設定されます。in *word* . . . を省略すると、for コマンドは、設定された各定位値パラメータに対して、do *list* を 1 回実行します (後述の「パラメータ置換」を参照)。in *word* リストの *word* がなくなると、実行は終了します。

select *identifier* [in *word* . . .] ; do *list* ; done
: select コマンドは、標準エラー (ファイル記述子 2) に、一群のワードを各々の前に番号を付けて出力します。in *word* . . . を省略すると、定位値パラメータが使用されます (後述の「パラメータ置換」を参照) 。PS3 プロンプトが出力され、標準入力から行が読み取られます。この行が、リストに示された *word* のいずれかの番号からなる場合、*identifier* が示す変数の値はこの番号に該当する *word* に設定されます。この行が空の場合は、再び選択リストを出力します。空でない場合は、*identifier* 変数の値を NULL に設定します (後述の「ブランクの解釈」を参照) 。標準入力から読み取った行の内容は、REPLY というシェル変数に保存します。break またはファイルの終わり (EOF) に行き当たるまで、選択が発生するたびに *list* が実行されます。*list* の実行によって REPLY 変数が NULL に設定されると、2 番目の選択をプロンプトする PS3 の表示前に選択リストが出力されます。

`case` *word* `in` [*pattern* [| *pattern*]) *list* `;;`] ... `esac`
`case` コマンドは、*word* に一致する最初の *pattern* に対応した *list* を実行します。*pattern* の形式は、ファイル名生成に使用される形式と同じです (「ファイル名生成」の項を参照) 。

`if` *list* `;` `then` *list* `;` [`elif` *list* `;` `then` *list* `; ...`] [`else` *list* `;`] `fi`
`if` の後の *list* を実行後、*list* が `0` の終了ステータスを返すと、最初の `then` の後の *list* を実行します。それ以外の場合、`elif` の後の *list* を実行します。この値が `0` の場合、2 番目の `then` の後の *list* を実行します。これが失敗すると、`else` *list* を実行します。`else` *list* も `then` *list* も実行しない場合、`if` コマンドは `0` の終了ステータスを返します。

`while` *list* `;` `do` *list* `;` `done`
`until` *list* `;` `do` *list* `;` `done`
`while` コマンドは、`while` *list* を繰り返し実行し、*list* 中の最後のコマンドの終了ステータスが `0` の場合、`do` *list* を実行します。それ以外の場合、ループは終了します。`do` *list* 中のコマンドを実行しない場合、`while` コマンドは `0` の終了ステータスを返します。ループ終了条件の判定を逆にするには、`while` の代わりに `until` を使用します。

(*list*)
別の環境で *list* を実行します。なお、入れ子において 2 つの開いた括弧を連続して記述する場合、後述の算術評価を避けるために空白を 挿入する必要があります。

{ *list* }
単に *list* を実行します。なお、メタキャラクタの (と) とは異なり、{ と } は「予約語」なので、認識されるためには行の始めまたは ; の後に現れる必要があります。

`[[` *expression* `]]`
expression が示す式を評価し、その値 が真のとき `0` の終了ステータスを返します。*expression* の説明については、後述の「条件式」を参照してください。

`function` *identifier* { *list* `;` }
identifier `(` `)` { *list* `;` }
identifier で参照される関数を定義します。{ と } の間のコマンド群 (*list*) が関数の本体となります (後述の「関数」参照) 。

`time` *pipeline*
pipeline を実行し、標準エラーに経過時間、ユーザー時間、およびシステム時間を出力します。

下記のワードは、コマンドの最初に現れたとき、および引用符をつけずに記述されたときに 「予約語」として認識されます。

```
!          if       then     else    elif    fi      case
esac       for      while    until   do      done    {   }
function   select   time     [[  ]]
```

コメント

でワードを始めると、そのワードおよび以降の 復帰改行までの文字がすべて無視されます。

別名

各コマンドの最初のワードに別名 (alias) が定義されている場合、そのワードは別名のテキストに置き換えられます。別名は任意の数の文字で構成されます。別名に使用できない文字は、メタキャラクタ、引用符、ファイル展開文字、パラメータ置換文字、コマンド置換文字、= です。代入する文字列としては、前述のメタキャラクタを含む有効な シェルスクリプトを指定できます。置換されたテキスト内にある各コマンドの最初のワードは、置換対象のものを除き、別名についてチェックされます。別名の最後の文字がブランクの場合、別名の後のワードも別名置換についてチェックされます。別名を使用すれば、特殊組み込み型コマンドを再定義できますが、前述の予約語の再定義はできません。別名は `alias` コマンドで作成、表示、エクスポートでき、`unalias` コマンドで削除できます。エクスポートされた別名は、名前指定で起動されたスクリプトに対して引き続き有効ですが、シェルを起動 (後述の「呼び出し」を参照) するたびに初期化し直す必要があります。再帰的に別名をつける際に無限ループの発生を防ぐため、シェルが現在その同じ名前の別名を処理中でなければ、ワードは別名の値に置換されます。処理中であれば置換されません。

別名化はスクリプト読み取り時に実行されますが、スクリプト実行中には行われません。したがって別名を有効にするには、別名を参照するコマンドの読み取り前に `alias` コマンドで定義しておく必要があります。

別名は、完全パス名の省略形としてよく使用されます。別名機能のオプションを使えば、別名の値を、該当するコマンドの完全パス名に自動的に設定できます。これらの別名を検索済み (*tracked*) 別名と呼びます。検索済み別名の値は、対応するコマンドを最初に検索するときに定義され、`PATH` 変数を再設定するたびに未定義になります。これらの別名は検索済みのままとなり、次回の参照時に値が再定義されます。複数の検索済み別名がシェル中にコンパイルされます。`set` コマンドの `-h` オプションは、各々の参照されたコマンド名を 検索済み別名にします。

以下に示す「エクスポート済み別名」 はシェルにコンパイルされ組み込まれますが、設定解除または再定義が可能です。

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

後方のブランク文字と予約語に関する例を示します。ユーザーが次のように入力したとします。

```
$ alias foo="/bin/ls "
$ alias while="/"
```

ここで次のコマンドを実行します。

```
$ while true
> do
```

```
> echo "Hello, World"
> done
```

すると Hello, World という文字列が画面上に無限に現れます。一方、次のように入力すると、結果は / の ls 出力となります。

```
$ foo while
```

foo に対する別名置換は空白文字で終わりとなるため、次のワードが置換用にチェックされます。次のワード while も別名化されているので、置換が行われます。つまり while の位置がコマンド名として正しくないために、予約語とは認識されないのです。

以下のように入力すれば、while は通常どおり予約語と見なされます。

```
$ foo; while
```

チルド置換

別名置換を実行すると、各ワードは引用符なしの ~ で始まっているかどうかチェックされます。結果が真なら、/ までのワードに一致するユーザー名が 存在するかどうかチェックされます。存在すれば、~ および一致したワード (ログイン名) は、一致したユーザーのログインディレクトリに置き換えられます。これをチルド置換と呼びます。一致するユーザー名が見つからない場合、元のテキストは変更を受けません。~ が単独で指定された場合、または後ろに / を伴って指定された場合には、$HOME に置き換えられます。~ のあとに + または - を指定すると、それぞれ $PWD または $OLDPWD に置き換えられます。

また変数に代入する値が ~ で始まるときにも、チルド置換が試されます。

チルド展開

チルド接頭辞 (*tilde-prefix*) はワードの先頭の引用符なしのチルド文字と、それに続く文字列からなります。この文字列とは、ワード中に引用符がつかないスラッシュが 含まれていれば、その最初のスラッシュ直前までのすべての文字です。スラッシュが含まれていなければ、ワード中のすべての文字です。代入においては複数のチルド接頭辞を使用できます。つまりワードの先頭 (代入を示す等号の直後) 、引用符がつかない コロンのあと、およびその両方にも指定できます。代入におけるチルド接頭辞の終わりは、引用符がつかないコロンまたは スラッシュが最初に現れた地点です。チルド接頭辞中に引用符付きの文字が 1 つもなければ、先頭のチルドを除いた部分文字列は、ユーザーデータベース中に 登録されているログイン名の可能性があると見なされます。

移植性のあるログイン名には、環境変数 LOGNAME の項に記載されている文字以外の文字を含めることはできません。ログイン名が NULL のとき、すなわちチルド接頭辞がチルドだけからなるとき、チルド接頭辞は変数 HOME の値に置き換えられます。HOME が設定されていないと、その結果は予測できません。ログイン名が NULL でなければ、チルド接頭辞は getpwnam 関数を使って得られるログイン名に対応したホームディレクトリの パス名に置き換えられます。システムがログイン名を認識できない場合、結果は未定義です。

チルド展開は、通常はワードの先頭でのみ発生しますが、従来からの使用法に基づいた例外的な使い方もあります。次の例を見てください。

```
PATH=/posix/bin:~dgk/bin
```

これもチルド展開の対象となります。チルドがコロンの直後にあり、それに続く文字が 1 つも引用符で囲まれて いないためです。以下のような置換も発生しうるため、このようなチルド展開は抑止する方向も検討されました。

```
PATH=$(printf %s ~karels/bin : ~bostic/bin)
for Dir in ~maart/bin ~srb/bin .
do
     PATH=${PATH:+$PATH:}$Dir
done
```

最初のコマンドでは、各ディレクトリにコロンが明示的に指定されています。いずれの場合も、シェルはすべてのディレクトリ名に対してチルド展開を実行します。どの名前も別個のワードとして認識するためです。

次にオペランド中の式の例を見てみましょう。

```
make -k mumble LIBDIR=~chet/lib
```

このような代入式は、シェル変数の代入と認識される条件を 満たしていないので、チルド展開は行われません。ただし、コマンドが展開を自身で行う場合は除きます (make は行いません) 。

将来的な仕様向けに、どのワード中でもチルド展開を強制的に実行する方法として、$~ という特殊な文字列が提供されています。

ワードは引用符をつけてはいけない、という規則なので、以下の指定は同等にはなりません。チルド展開が行われるのは、最後の指定だけです。

```
\~hlj/   ~h\lj/   ~"hlj"/   ~hlj\/   ~hlj/
```

Korn シェルの ~+ と ~– 構造は、この規則を利用しているため、未定義のログイン名に チルドを付加した場合、処理の結果は予測できません。なお、一般に、誤ったログイン名にチルドを付加するとエラーになります。また HOME が未設定の場合、従来のシェルの中にはそれをエラーとするものもあるため、結果は予測できません。

コマンド置換

コマンドが、ドル記号に続く括弧で囲まれている場合 (つまり $(*command*) の形式)、または一対の逆引用符 (`) で囲まれている場合には、その標準出力をワードの一部または全体として使用できます。その場合に標準出力に含まれる復帰改行は削除されます。2 番目の引用符で囲む形式では、コマンドの実行前に、引用符間の文字列に含まれる特殊引用符文字が処理されます (後述の「クォート」を参照) 。$(<*file*) というコマンド置換は、同じ動作で実行速度の速い $(<*file*) で代用することもできます。入出力リダイレクトを行わない特殊コマンドの多くは、別のプロセスを作成せずに実行されるからです。

コマンド置換を使うと、コマンド名をコマンドの出力で置き換えて しまうこともできます。コマンド置換は、コマンドを以下のように囲んで 記述すると行われます。

$(*command*)

または逆引用符を使って 次のように 指定します。

`` `command` ``

シェルは *command* をサブシェル環境で実行し、コマンド置換指定 (*command* のテキストと、それを囲んでいる `$()` または逆引用符) をコマンドの標準出力で置き換え、置換指定の最後にあるいくつかの連続した復帰改行文字を除去する ことにより、コマンド置換を展開します。出力の途中に埋め込まれている復帰改行文字は除去されません。ただし、IFS の値および現在有効な引用符によっては、それらの復帰改行文字はフィールド区切り文字とみなされて、フィールド分割時に削除されることもあります。

逆引用符形式のコマンド置換においては、バックスラッシュは文字そのものとしての意味を保持します。ただし以下に示す文字、つまりドル記号、逆引用符、バックスラッシュが直後に続く場合を除きます。

```
$     `     \
```

対の相手となる逆引用符の検索は、直前の文字がバックスラッシュでない 逆引用符が見つかれば満たされます。この検索において、エスケープのない逆引用符がシェルコメント中、here-document (各種シェルで << を使用する機能) 、$(*command*) 形式のコマンド置換中、または引用符つきの文字列中で検出された場合、結果は未定義となります。先頭が ‘. . .‘ の並びで、終わりがこの並びではない、単一引用符または二重引用符で囲まれた文字列がある場合、結果は未定義となります。

$(*command*) 形式の指定では、左括弧の次の文字から、対応する右括弧までの文字が *command* を構成します。この *command* には、シェルのスクリプトとして正しいものであれば自由に使用できます。ただし、

- 入出力先のリダイレクト指定だけからなるスクリプトの場合、結果は予測できません。
- 単一のサブシェルの場合には、後述するような制限があります。

コマンド置換によって得られた結果は、あとでチルド展開、パラメータ展開、コマンド置換、または 算術展開を行うためにフィールド分割やパス名展開されることはありません。二重引用符に囲まれた内部でコマンド置換が発生しても、二重引用符の中で通常起こる 置換の結果に対しては行われません。

コマンド置換は入れ子にできます。逆引用符形式の内部で入れ子を 指定する場合には、内側の両方の逆引用符の直前にバックスラッシュを 記述する必要があります。以下に例を示します。

`` `\`command \` ``

逆引用符を使ったときの動作が一律でない、という問題は、$() 形式のコマンド置換を使えば解決できます。次の例を見てください。

コマンド	出力
echo '\$x'	\$x
echo `echo '\$x'`	$x
echo $(echo '\$x')	\$x

また、逆引用符形式は、埋め込まれたコマンドの内容に対して、従来から制限を持っていました。新しい $() 形式がどんな種類のスクリプトでも処理できるのに対し、逆引用符形式は、逆引用符を含んだ正しいスクリプトを扱えないことがあります。たとえば以下に示す例は、いずれも正しいスクリプトですが、右側は正しく処理されるのに対し、左側は処理されません。

echo `	echo $(
cat <<eeof	cat <<eeof
a here-doc with `	a here-doc with)
eof	eof
`	)
echo `	echo $(
echo abc # a comment with `	echo abc # a comment with)
`	)
echo `	echo $(
echo '`'	echo ')'
`	)

このように逆引用符形式のコマンド置換は動作に一貫性がないため、コマンド置換を入れ子にしたり、複雑なスクリプトを埋め込もうとするアプリケーションには、使わない方がいいでしょう。

コマンド置換が以下のように単一のサブシェルからなる場合、

$((*command*))

移植性のあるアプリケーションであれば、$(と (を 2 つのトークンに (空白を使って) 分離させる必要があります。これは算術展開と混同するのを避けるためです。

算術展開

ドル記号に続く二重括弧で囲まれた算術式 (つまり ($ ((*arithmetic-expression*))) の形式) は、二重括弧内の算術式の値に置き換えられます。算術展開は、算術式を評価して値を代入するための メカニズムを提供します。算術展開の形式は次のとおりです。

$((*expression*))

式すなわち *expression* は、あたかも二重引用符で囲まれているかのように扱われます。ただし、式の内部の二重引用符は特別な意味を持つとは見なされません。シェルは、パラメータ置換、コマンド置換、引用符削除のために、式の中のトークンをすべて展開します。

次にシェルはこの式を算術式と見なし、その値を置き換えます。算術式は、以下に述べる例外を除き、ISO C の規則に従って処理されます。

- 整数の算術だけが必須です。
- `sizeof()` 演算子、および先頭と後尾の ++ と – – 演算子は必須ではありません。
- 選択、繰り返し、ジャンプの各ステートメントは サポートされていません。
- `/usr/bin/ksh` は先頭の 0 から 9 までを 10 進定数として扱います。次の例を参照してください。

コマンド	**/bin/ksh** の結果	**/usr/xpg4/bin/sh** の結果
`echo $((010+10 ))`	20	18
`echo $((010+10 ))`	29	エラー
`[ 10 –le $((011)) ]`	真	偽

拡張機能として、上記リストに挙がっている算術式でも シェルが認識できる場合があります。式が不正な場合、展開は失敗に終わり、シェルはそのことを示す メッセージを標準エラー出力に書き出します。

算術展開を行う簡単な例を以下に示します。

```
# repeat a command 100 times
x=100
while [ $x –gt 0 ]
do
     command
     x=$(($x–1))
done
```

プロセス置換

この機能は SunOS が提供するもので、UNIX オペレーティングシステムのバージョンが、オープンしたファイルに名前をつけるための `/dev/fd` ディレクトリをサポートしている 場合にだけ使用できます。< (*list*) または > (*list*) 形式の各コマンド引数は、*list* が示すプロセスを実行します。このプロセスは、`/dev/fd` 中のファイルに非同期的に接続されています。このファイルの名前が当該コマンドの引数となります。> が

付いた形式を使用した場合、このファイルに書き出すことにより *list* への入力が可能となります。< の形式を用いると、引数として渡されたファイルには *list* プロセスからの出力が含まれます。次の例を見てください。

```
paste <(cut -f1 file1) <(cut -f3 file2) | tee >(process1) >(process2)
```

このコマンドは *file1* からフィールド 1 をカットし、*file2* からフィールド 3 をカットし、その両者をペーストし、その結果を *process1* と *process2* に送り、さらに標準出力上に書き出します。なおこのファイルは、引数としてコマンドに渡されますが、UNIX の `pipe`(2) になっているので、ファイル上で `lseek`(2) を行おうとするプログラムは動作できません。

パラメータ置換

パラメータ(*parameter*) は、識別子 (*identifier*)、1 つまたは複数の数字、または `*`、`@`、`#`、`?`、`-`、`$`、`!` の文字のいずれかです。変数 (*variable*) は識別子が示すパラメータで、1 つの値 (*value*) といくつかの属性 (*attributes*) を持ちます。属性がない場合もあります。特殊コマンド `typeset` を使用すれば、変数に値と属性を代入できます。シェルがサポートする属性については、`typeset` 特殊コマンドの項で後述します。エクスポートされた変数は、値と属性を環境に渡します。

シェルは一次元配列機能をサポートします。配列変数の要素は、添字 (*subscript*) によって参照されます。添字を指定するには、最初に `[` 、次に算術式 (後述の「算術評価」を参照) 、その次に `]` を記述します。配列に値を割り当てるときは、`set -A` *name value* `. . . .` を使用してください。添字の値は常に 0 から 4095 の範囲内で指定してください。配列を宣言する必要はありません。有効な添字を伴う変数参照は正当であり、必要に応じて配列が作成されます。添字なしで配列を参照するのは、`0` 番目の要素を参照することと同じ意味です。配列の *identifier* に添字として `*` または `@` を用いると、個々の要素の値が置換されます (フィールド区切り文字で区切られます)。

以下の記述により変数に値を 代入することもできます。

name=*value* [*name*=*value*] . . .

name に `-i` という整数属性を設定すると、*value* は後述の算術評価を受けます。

定位値パラメータは数値により指定されるパラメータで、`set` 特殊コマンドで値を代入できます。`$0` パラメータには、シェル起動時に 0 番目の引数から値を設定します。1 つまたは複数の数字からなるパラメータは、定位値パラメータになります。複数の数字からなる定位値パラメータは中括弧で囲む必要があります。

パラメータの展開

パラメータの展開は以下の形式で記述します。

${*expression*}

ここで *expression* は、対応する } の直前までのすべての文字を含みます。なおバックスラッシュによりエスケープされている } や引用符で囲まれた文字列内の }、および埋め込まれた算術展開、コマンド置換、変数展開の中の文字は、対応する } の検索の対象とはなりません。

パラメータ展開の最も単純な形式は次のとおりです。

${*parameter*}

parameter が値を持っていれば置き換えられます。

パラメータの名前と記号は、中括弧 ({}) で囲むこともできます。この中括弧は、複数の数字からなる位置パラメータ、および名前の一部として 解釈される恐れのあるような文字が *parameter*．．．に続く場合を除いては、省略可能です。対となる閉じる中括弧は、括弧のレベルを数えながら、引用符で囲まれた文字列やコマンド置換をスキップして見つけられます。

パラメータの名前や記号が中括弧で囲まれていない場合、正当な名前となりうる最長のものが展開に使われます。その名前が表す記号が存在しているかどうかは問いません。シェルが名前の境界を決定するために入力を走査するとき、どの名前がすでに定義されているかをシェルが知っていても、それに影響されることはありません。たとえば F というシェル変数が定義されているとき、以下のコマンドを実行したとします。

```
echo $Fred
```

このとき、$F のあとに red が表示されるようなことはありません。正しい名前となりうる最長の文字列は Fred であり、そのような名前は定義されていないためです。

二重引用符内でパラメータ展開が発生した場合、

- 展開結果に対してはパス名展開は行われません。
- 展開結果に対しては、@ の場合を除きフィールド分割は行われません。

また、以下に示す形式を使って、パラメータ展開を変更することが可能です。*word* の値が必要となる場合 (後述するように *parameter* の状態による) には、*word* チルド展開、パラメータ展開、コマンド置換、および算術展開の対象となります。*word* が必要でなければ、展開は行われません。なおパラメータ展開の変更指定を区切る文字 } は、先ほど述べた方法、さらに dquote でも述べるような方法で決定されます。たとえば、${foo-bar}xyz} という指定は、foo が設定済みであれば foo の展開のあとに文字列 xyz} が続き、foo が設定されていなければ文字列 barxyz} となります。

${*parameter*:–*word*}	「デフォルト値の使用」。*parameter* が未設定または NULL の場合、*word* の展開結果が代入されます。それ以外の場合には *parameter* の値が代入されます。
${*parameter*:=*word*}	「デフォルト値の割当」。*parameter* が未設定または NULL の場合、*word* の展開結果が *parameter* に割り当てられます。どのような場合でも、*parameter* の最終的な値が代入されます。この割当方法は変数だけに使用可能で、位置パラメータや特殊パラメータには使用できません。
${*parameter*:?[*word*]}	「未設定または NULL のときエラー表示」。*parameter* が未設定または NULL の場合、*word* の展開結果 (また

は *word* 省略時は未設定を表すメッセージ) が標準エラー出力に書き出され、シェルは ゼロ以外の終了ステータスで終了します。それ以外の場合には、*parameter* の値が代入されます。対話型シェルでは終了しません。

${*parameter*:+[*word*]} 「代替値の使用」。*parameter* が未設定または NULL の場合、NULL が代入されます。それ以外の場合には、*word* の展開結果が代入されます。

前述のパラメータ展開では、形式中にコロンを使うとパラメータが未設定または NULL であることのテストとなり、コロンを省略すると パラメータが未設定であることのテストとなります。コロンの役割を以下の表にまとめます。

	パラメータが NULL以外に設定	パラメータが NULLに設定	パラメータが 未設定
${*parameter*:-*word*}	*parameter* に置換	*word* に置換	*word* に置換
${*parameter*–*word*}	*parameter* に置換	NULL に置換	*word* に置換
${*parameter*:=*word*}	*parameter* に置換	*word* を代入	*word* を代入
${*parameter*=*word*}	*parameter* に置換	*parameter* に置換	NULL を代入
${*parameter*:?*word*}	*parameter* に置換	エラー 終了	エラー 終了
${*parameter*?*word*}	*parameter* に置換	NULL に置換	エラー 終了
${*parameter*:+*word*}	*word* に置換	NULL に置換	NULL に置換
${*parameter*+*word*}	*word* に置換	*word* に置換	NULL に置換

この表で、「．．．に置換」は、式が表に示すものに置き換えられることを表します。また「．．．を代入」は、パラメータにその値が代入され、さらに式を置き換えることを表します。

${#*parameter*}　「文字列の長さ」。パラメータの値の長さを文字単位で示します。*parameter* が * または @ のとき、$1 を始めとしてすべての定位置パラメータ (フィールド区切り文字で区切られている) が置き換えられます。

以下に示す 4 種類のパラメータ展開は、いずれも部分文字列処理用のものです。どの場合でも、パターンの評価には、正規表現ではなく パターンマッチング表現 (patmat を参照) が使われます。*parameter* が * または @ のとき、$1 を始めとしてすべての定位置パラメータ (フィールド区切り文字で区切られている) が置き換えられます。完全パラメータ展開文字列を二重引用符で囲んでも、以下の 4 種類のパターン文字列は引用符付きとはなりません。ただし中括弧の内部で文字列を囲んだ場合には、引用符付きとなります。

${*parameter*%*word*}　「最小の接尾辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接尾辞中の *pattern* と一致する最小部分が削除されます。

${*parameter*%%*word*}　「最大の接尾辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接尾辞中の *pattern* と一致する最大部分が削除されます。

${*parameter*#*word*}　「最小の接頭辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接頭辞中の *pattern* と一致する最小部分が削除されます。

${*parameter*##*word*}　「最大の接頭辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接頭辞中の *pattern* と一致する最大部分が削除されます。

使用例:

${*parameter*:–*word*}

次の例では、x が NULL または未設定の場合にのみ ls が実行されます。なお $(ls) コマンド置換の表現方法については、前述の 「コマンド置換」の項で説明してあります。

```
${x:-$(ls)}
```

${*parameter*:=*word*}

```
unset X
echo ${X:=abc}
abc
```

${*parameter*:?*word*}

```
unset posix
echo ${posix:?}
sh: posix: parameter null or not set
```

${*parameter*:+*word*}

```
set a b c
echo ${3:+posix}
posix
```

${#*parameter*}

```
HOME=/usr/posix
echo ${#HOME}
10
```

${*parameter*%*word*}

```
x=file.c
echo ${x%.c}.o
file.o
```

${*parameter*%%*word*}

```
x=posix/src/std
echo ${x%%/*}
posix
```

${*parameter*#*word*}

```
x=$HOME/src/cmd
echo ${x#$HOME}
/src/cmd
```

${*parameter*##*word*}

```
x=/one/two/three
echo ${x##*/}
three
```

シェルが設定するパラメータ

次のパラメータは、シェルが自動的に設定します。

#	定位置パラメータ数 (10 進数)
-	呼び出し時に、または `set` コマンドによってシェルに与えられたフラグ
?	最後に実行されたコマンドが返した 10 進数
$	このシェルのプロセス番号

_	初期設定時、_ というパラメータは、実行中のシェルまたはスクリプトの絶対パス名で、環境 (*environment*) に引き渡される値。そのあと、このパラメータには直前のコマンドの最後の引数が代入される。このパラメータは、非同期式のコマンドに関しては設定されない。このパラメータは、メールのチェック時にも、一致する MAIL ファイルの名前を保持するために使用する
!	最後に呼び出されたバックグラウンドコマンドのプロセス番号
ERRNO	最後に失敗したシステムコールにより設定された *errno* の値。この値はシステムに依存し、デバッグ目的で使用される
LINENO	実行中のスクリプトまたは関数内での現在行の行番号
OLDPWD	cd コマンドで設定された、直前の作業ディレクトリ
OPTARG	getopts 特殊コマンドで処理された最後のオプション引数の値
OPTIND	getopts 特殊コマンドで処理された最後のオプション引数のインデックス
PPID	シェルの親のプロセス番号
PWD	cd コマンドで設定された、現在の作業ディレクトリ
RANDOM	この変数を参照するたびに、0 から 32767 間に均一に分散した乱整数を生成する。乱数の並びは、RANDOM に数値を代入すれば初期化できる
REPLY	この変数は、引数指定のない select 文または read 特殊コマンドにより設定される
SECONDS	この変数を参照するたびに、シェルを起動してからの秒数が返される。この変数に値を代入すると、その値と代入処理実行時からの秒数との合計値が 参照時に返される

シェルが用いる変数

次の変数はシェルによって使用されます。

CDPATH	cd コマンドの検索パスを指定します。
COLUMNS	この変数を設定すると、シェル編集モードと select リストの出力用の編集ウィンドウの幅がその値によって、定義されます。
EDITOR	この変数の値が emacs、gmacs、または vi で終わり、VISUAL 変数が設定されていないと、該当するオプション (後述の「特殊コマンド」中の set の項を参照) が有効になります。
ENV	この変数は、シェルが呼び出されたときに シェルによるパラメータ展開の対象となり、それにより得られた値が、現環境で実行するシェルコマンドを含んでいるファイルのパス名として用いられます。このファイルは実行可能形式でなくてもかまいません。ENV を展開した値が絶対パス名でない場合、結果は予測できませ

	ん。ユーザーの実ユーザー ID と実効ユーザー ID が異なっていたり、実グループ ID と実効グループ ID が異なっていたりするとき、ENV は無視されます。 この変数を使って、そのシェル呼び出しだけに有効な 別名や他の項目を設定することができます。ENV が参照するファイルは $HOME/.profile とは異なります。つまり .profile が通常はセッション立ち上げ時に実行されるのに対し、ENV ファイルはシェル呼び出しの度に冒頭で実行されます。ENV の値はドットスクリプトと同じように解釈されます。つまりコマンドは現環境で実行され、ファイルは実行可能でなくてもかまいませんが、読み取り可能で ある必要があります。なお、ドットスクリプトでは PATH の検索が行われますが、ENV では行われません。これは「トロイの木馬」型セキュリティ侵入から保護するために使われます。
FCEDIT	fc コマンドのデフォルトのエディタ名を指定します。
FPATH	関数定義の検索パスを指定します。デフォルトでは、PATH 変数に設定されているディレクトリを検索したあと、FPATH に設定されているディレクトリを検索します。実行可能ファイルを見つけると、現在の環境中で実行します。-u 属性を持つ関数を参照した場合、PATH 変数の前に FPATH のディレクトリを検索します。あらかじめ設定されている別名の autoload を使えば、-u 属性を持つ関数を生成できます。
IFS	内部フィールド区切り文字。通常は空白文字、タブ、復帰改行です。コマンドまたはパラメータの置換によって生じる コマンドワードを区切るときと、read 特殊コマンドでワードを区切る場合に使用します。$* 置換において引数を区切るときは、IFS 変数の最初の文字を使用します (後述の 「クォート」 を参照) 。
HISTFILE	シェル起動時にこの変数が設定されていると、その値はコマンド履歴 (後述の「コマンド再入力」を参照) を格納するために使用されるファイルのパス名になります。
HISTFILE	シェル起動時にこの変数が設定されていると、このシェルで使用可能な入力済みコマンドの数が、この値以上になります。デフォルト値は 128 です。
HOME	cd コマンドのデフォルト引数 (ホームディレクトリ) を指定します。
LC_ALL	LC_* 変数のデフォルト値を提供する変数を指定します。
LC_COLLATE	この変数は、パターンマッチングにおける範囲式、同等クラス、および 複数バイト文字照合要素の動作を定義します。
LC_CTYPE	シェルが文字を処理する方法を決定します。LC_CTYPE に有効な値が設定されていると、シェルは、そのロケールに合った文字を含む テキストやファイル名を表示および処理できます。環境に

LC_CTYPE (environ(5) を参照) が設定されていなければ、シェルの動作は環境変数 LANG によって決定されます。LC_ALL が設定されていれば、その内容が LANG 変数やその他の LC_* 変数より優先されます。

LC_MESSAGES　メッセージをどの言語で出力するかを表す変数を指定します。

LANG　未設定もしくは NULL の国際化変数に対するデフォルト値を指定します。国際化変数のいずれかが不正な値に設定されていると、ユーティリティは変数が 1 つも定義されていないように動作します。

LINENO　この変数は、各行のコマンド実行前に、スクリプトまたは関数内での現在の 行番号 (1 から始まる連続した番号) を表す 10 進数に、シェルにより設定されます。ユーザーが LINENO を設定解除もしくは再設定すると、現在のシェルの動作中は、この変数が持つ特殊な意味は失われます。シェルが現在スクリプトも関数も実行していない場合、LINENO の値は予測できません。

LINES　この変数を設定すると、その値は選択 (select) リスト出力用のカラム長の決定に使用されます。選択リストは、LINES が示す行数の 3 分の 2 が一杯になるまで垂直に出力されます。

MAIL　この変数をメールファイルの名前に設定し、さらに MAILPATH 変数を設定しない場合、シェルは指定されたファイルにメールが到着すると ユーザーに通知します。

MAILCHECK　この変数は、MAILPATH 変数または MAIL 変数で指定されるファイルが更新されたかどうかを、シェルが何秒ごとにチェックするかを指定します。デフォルト値は 600 秒です。この時間が経過すると、シェルは次のプロンプトを出す前に チェックします。

MAILPATH　コロン (:) で区切ったファイル名のリストを指定します。この変数を設定すると、シェルは直前の MAILCHECK 秒間に発生した、指定ファイルに対する変更について ユーザーに通知します。各ファイル名のあとには ? と、出力されるメッセージを指定できます。メッセージは、出力に先立ち、$_ 変数に変更されたファイルの名前を代入する パラメータ置換処理を受けます。デフォルトメッセージは you have mail in $_ です。

NLSPATH　LC_MESSAGES 処理用のメッセージカタログの場所を示す変数を指定します。

PATH　コマンド用の検索パスを指定します。(後述の「実行」を参照) rksh の下で実行する場合 (.profile の場合を除く) 、PATH を変更することはできません。

PPID　この変数は、シェルを呼び出したプロセスのプロセス ID (10 進数) に対して、シェルにより設定されます。サブシェル中では、PPID は現在のシェルの親の値と同じ値に設定されます。たとえば echo $PPID と (echo $PPID) は同じ値を生成します。

PS1	この変数の値はパラメータ置換用に展開され、一次プロンプト文字列を定義します。デフォルトは $ です。一次プロンプト文字列内の文字 ! は、コマンド番号で置換されます (後述の「コマンド再入力」を参照) 。! を 2 つ連続して指定すると、プロンプト文字列の出力時に ! が 1 つ表示されます。
PS2	二次プロンプト文字列を指定します。デフォルトは > です。
PS3	select ループ内で使用する選択プロンプト文字列を指定します。デフォルトは #? です。
PS4	この変数の値は、パラメータ置換用に展開され、実行トレースの各行の前に出力されます。省略すると、実行トレースプロンプトは + になります。
SHELL	シェルのパス名は環境内に保持されます。起動時に、この変数のベース名が rsh、rksh、または krsh の場合、シェルの機能は制限されます。
TMOUT	0 より大きい値に設定すると、PS1 プロンプトの発行後、指定された秒数以内に コマンドが入力されない場合、シェルは終了します。シェルのコンパイル時に、この値がユーザーが 超えられないような大きい値に設定されていることも あります。
VISUAL	この変数の値が emacs、gmacs、または vi で終わる場合、該当するオプション (後述の「特殊コマンド」中の set の項を参照) が有効になります。

シェルは、PATH、PS1、PS2、PS3、PS4、MAILCHECK、FCEDIT、TMOUT、および IFS にデフォルト値を割り当てますが、HOME、SHELL、ENV、および MAIL は、シェルによって設定されることはありません。ただし HOME は login(1) により設定されます。また一部のシステムでは、MAIL や SHELL も login で設定されることがあります。

ブランクの解釈

パラメータとコマンドの置換後、置換の結果内でフィールド区切り文字 (IFS で発見されるもの) を検索し、その文字が現れた位置で分割します。分割された各々が引数となります。明示的な NULL 引数 ("" または '' は保持されます。暗示的な NULL 引数 (値を持たないパラメータにより生じるもの) は削除されます。

ファイル名の生成

置換後、各コマンドワードは、*、?、[を含んでいないかチェックされます。ただし -f オプションを設定していない場合にかぎります。これらの文字のいずれかがあると、そのワードはパターンとみなされます。このワードは、パターンと一致する、辞書編集方式の順にソートされたファイル名に置換されます。パターンと一致するファイル名が見つからない場合、ワードは変更されません。パターンをファイル名の生成に使用する場合、ファイル名の先頭のピリオド (.) または スラッシュ (/) 直後のピリオドは、明示的に一致させる必要があります (後者の場合はスラッシュ自体をも含む)。ピリオドで始まるファイル名は 括弧の中でピリオドのある次のようなパターンとは 一致しません。

```
ls .@(r*)
```

これは .restore というファイル名を検出しますが、ls @(.r*) は検出しません。パターンマッチングの他のケースでは、/ に続く . は特殊文字とはみなされません。

*	NULL 文字列を含め、任意の文字列と一致します。
?	任意の単一文字と一致します。
[...]	囲まれた文字のいずれかと一致します。2 つの文字を - で区切ると、その間にある任意の文字 (その 2 つの文字も含む) に一致します。先頭の [の次の文字が ! である場合、[ ] で囲まれていない任意の文字と一致します。- は、最初の文字または最後の文字として 文字セットに挿入できます。

pattern-list は、| で区切られた 1 つまたは複数のパターンのリストです。複合パターンは、次のうちの 1 つまたは複数で形成することができます。

?(*pattern-list*)	指定されたパターンのいずれかと任意に一致します。
*(*pattern-list*)	指定されたパターンの 0 回以上の発生と一致します。
+(*pattern-list*)	指定されたパターンの 1 回以上の発生と一致します。
@(*pattern-list*)	指定されたパターンのうち 1 つだけと一致します。
!(*pattern-list*)	指定されたパターンのうち 1 つだけを除き、あらゆるものと一致します。

クォート

前述の「定義」の項で示したメタキャラクタ の各々は、シェルに対して特別の意味を持ち、クォートしないかぎり、ワードの終わりを表します。文字は、その前に \ を指定すればクォートされます。つまりその文字自身を 示すことができます。\ NEWLINE の対は削除されます。一対の単一引用符 (' ') で囲まれた文字はすべてクォートされます。単一引用符内にさらに単一引用符を指定することはできません。一対の二重引用符 ("") で囲まれた文字列内では、パラメータとコマンドの置換が発生し、\ は \ 、 ` 、 " 、 $ をクォートします。$* と $@ の意味は、クォートされていないとき、あるいはパラメータの代入値もしくはファイル名として 使用されるときは同一です。ただしコマンド引数として使用するとき、$* は $1*d* $2*d* . . . と同じになります ( *d* は IFS 変数の最初の文字です)。一方、$@ は $1 $2 . . . と同じになります。一対の逆引用符 ( ` ` ) で囲まれた中では、\ は \ 、 ` 、 $ といった文字をクォートします。逆引用符を二重引用符内で指定すると、\ も " という文字をクォートします。

予約語や別名中のいずれかの文字をクォートすれば、その予約語が持つ特別な意味はなくなります。なお以下に説明する関数や特殊コマンドに関しては、その名前をクォートしても、関数名またはコマンド名としての 認識を変えることはできません。

算術評価

let という特殊コマンドには、整数演算を実行する機能が提供されています。評価は *long* 演算を使用して実行します。定数は [*base*#] *n* 形式とします。ここで *base* は底を表す 2 から 36 の範囲の 10 進数で、*n* はその底における数です。*base* を省略すると、底は 10 となります。

算術式は、C 言語での式と同一の構文、優先度、および結合規則を使用します。++、-- -、?:、, 以外のすべての整数演算子がサポートされています。算術式内では、パラメータ置換構文を使用しなくても、名前で変数を参照できます。変数を参照すると、その値は算術式として評価されます。

変数の内部整数表記は、typeset 特殊コマンドの -i オプションで指定できます。算術評価は、-i 属性を備えた変数に対する各代入値について実行されます。底を指定しないと、変数に対する最初の代入が底を決定します。この底は、パラメータ置換が発生する際に使用されます。

算術演算子の多くはクォートしなければならないので、代替形式の let コマンドが提供されています。((で始まるコマンドについては、対応する)) までの文字はすべてクォートされた表現とみなします。具体的に言うと、((...)) は let "..." と同じ意味です。

プロンプト

シェルは、対話的に使用すると、コマンドを読み取る前に PS1 のパラメータ展開値によるプロンプトを発行します。復帰改行を入力したあとで、コマンドを完了するためにさらに入力が必要な場合は、2 次プロンプト (つまり PS2 の値) が出力されます。

条件式

条件式 (*conditional expression*) は、ファイルの属性をテストしたり文字列を比較するときに、複合コマンドの [[とともに使用します。[[と]] の間のワードについてはワード分割とファイル名生成を実行しません。各条件式は、次の単項式または 2 項式をいくつか組み合わせて構築できます。

-a *file*	*file* が存在する場合、真です。
-b *file*	*file* が存在し、ブロック型特殊ファイルである場合、真です。
-c *file*	*file* が存在し、文字型特殊ファイルである場合、真です。
-d *file*	*file* が存在し、ディレクトリである場合、真です。
-e *file*	*file* が存在していれば、真です。
-f *file*	*file* が存在し、通常ファイルである場合、真です。
-g *file*	*file* が存在し、setgid ビットがセットされている場合、真です。
-k *file*	*file* が存在し、スティッキ・ビットがセットされている場合、真です。
-n *string*	*string* の長さが 0 ではない場合、真です。
-o *option*	*option* という名前のオプションが有効の場合、真です。
-p *file*	*file* が存在し、FIFO 特殊ファイルまたはパイプである場合、真です。
-r *file*	*file* が存在し、現在のプロセスで読み取り可能な場合、真です。

`-s` *file*	*file* が存在し、サイズが 0 より大きい場合、真です。
`-t` *fildes*	*fildes* が示すファイル記述子番号がオープンされていて、端末デバイスに対応している場合、真です。
`-u` *file*	*file* が存在し、setuid ビットがセットされている場合、真です。
`-w` *file*	*file* が存在し、現在のプロセスで書き込み可能な場合、真です。
`-x` *file*	*file* が存在し、現在のプロセスで実行可能な場合、真です。*file* が存在し、ディレクトリである場合、現在のプロセスにはそのディレクトリに対する 検索権があります。
`-z` *string*	*string* の長さが 0 の場合、真です。
`-L` *file*	*file* が存在し、シンボリックリンクである場合、真です。
`-O` *file*	*file* が存在し、このプロセスの実効ユーザー ID が所有している場合、真です。
`-G` *file*	*file* が存在し、そのグループがこのプロセスの実効グループ ID と一致する場合、真です。
`-S` *file*	*file* が存在し、ソケットである場合、真です。
file1 `-nt` *file2*	*file1* が存在し、*file2* より新しい場合、真です。
file1 `-ot` *file2*	*file1* が存在し、*file2* より古い場合、真です。
file1 `-ef` *file2*	*file1* と *file2* が存在し、同一のファイルを参照する場合、真です。
string	*string* が NULL 文字列でない場合、真です。
string `=` *pattern*	*string* が *pattern* と一致する場合、真です。
string `!=` *pattern*	*string* が *pattern* と一致しない場合、真です。
string1=*string2*	*string1* と *string2* が同じ場合、真です。
string1`!` =*string2*	*string1* と *string2* が同じでない場合、真です。
string1 `<` *string2*	カテゴリ `LC_COLLATE` に設定されるロケールに適切であると 解釈された文字列を比較し、*string1* が *string2* より小さい場合、真です。
string1 `>` *string2*	カテゴリ `LC_COLLATE` に設定されるロケールに適切であると 解釈された文字列を比較し、*string1* が *string2* より大きい場合、真です。
exp1 `-eq` *exp2*	*exp1* が *exp2* と等しい場合、真です。
exp1 `-ne` *exp2*	*exp1* が *exp2* と等しくない場合、真です。

exp1 `-lt` *exp2*	*exp1* が *exp2* 未満である場合、真です。
exp1 `-gt` *exp2*	*exp1* が *exp2* より大きい場合、真です。
exp1 `-le` *exp2*	*exp1* が *exp2* 以下の場合、真です。
exp1 `-ge` *exp2*	*exp1* が *exp2* 以上の場合、真です。

上記の式の各々において、*file* が `/dev/fd/`*n* という形式の場合 (*n* は整数)、記述子番号が *n* でオープンされているファイルでテストします。

以下のいずれかを使用すれば、これらの基本式から複合式を構築することができます。優先度の高いものから順に並べてあります。

(*expression*)	*expression* が真の場合、真です。式のグループ化に使用します。
`!` *expression*	*expression* が偽の場合、真です。
expression1 `&&` *expression2*	*expression1* と *expression2* が両方とも真の場合、真です。
expression1 `||` *expression2*	*expression1* と *expression2* のどちらかが真の場合、真です。

入出力

コマンド実行前に、シェルが解釈する特殊表記法によって入出力先を変更 (リダイレクト) できます。以下は、単純コマンド内の任意の位置および コマンドの前後に指定することができ、起動されたコマンドには引き渡されません。以下に示す場合を除き、*word* または *digit* を使用する前にコマンドとパラメータの置換が発生します。ファイル名生成が発生するのは、パターンが 1 つのファイルだけと一致し、さらにブランク解釈が実行されない場合だけです。

`<`*word*	*word* というファイルを標準入力 (ファイル記述子 0) として使用します。
`>`*word*	*word* というファイルを標準出力 (ファイル記述子 1) として使用します。ファイルが存在しない場合は作成します。ファイルが存在し、かつ `noclobber` オプションが有効の場合、エラーになります。その他の場合は、ファイルの長さが 0 になります。
`>|`*word*	`>` と同一です。ただし、これは `noclobber` オプションを無視します。
`>>`*word*	*word* というファイルを標準出力として使用します。ファイルが存在する場合、(EOF までシークしたあと) そのファイルに出力を追加します。ファイルが存在しない場合は、ファイルを作成します。
`<>`*word*	*word* というファイルを読み取りおよび書き込み用に 標準入力としてオープンします。
`<< [-]`*word*	シェルへの入力として *word* と同一の行まで、または EOF まで読み取ります。*word* に対しては、パラメータ

	置換やコマンド置換や ファイル名生成を実行しません。*here-document* が生成されて標準入力になります。*word* のいずれかの文字がクォートされている場合、ドキュメントの文字はいっさい解釈されません。クォートされている文字がなければ、パラメータとコマンドの置換が発生し、\NEWLINE が無視されます。また \ を使用して、\ 、$、`、および *word* の最初の文字をクォートす必要があります。- を << のあとに付加すると、*word* とドキュメントから先行タブをすべて取り除きます。
<&*digit*	*digit* が示すファイル記述子から複製したものを 標準入力として使用します (dup(2) を参照) 。同様に、標準出力への複製には >&*digit* を使用します。
<&-	標準入力をクローズします。同様に、標準出力については >&- を使用します。
<&p	並行プロセスからの入力を標準入力へ移動します。
>&p	並行プロセスへの出力を標準出力へ移動します。

上記のいずれかの前に数字が付く場合、その値が (デフォルトの 0 または 1 のかわりに) 該当ファイルに対応した ファイル記述子となります。

```
. . . 2>&1
```

ファイル記述子 2 をファイル記述子 1 から複製して、書き込み用にオープンします。

リダイレクトを指定する場合、記述する順序が重要になります。シェルは、リダイレクト指定を左から右へ評価します。

```
. . . 1>fname 2>&1
```

上記の例では、まず *fname* というファイルにファイル記述子 1 を関連付けます。次に、ファイル記述子 1 に関連するファイル (つまり *fname*) に、ファイル記述子 2 を関連付けます。リダイレクトの向きが逆であれば、まずファイル記述子 2 を端末に関連付け (ファイルを記述子 1 がすでに端末に関連付けられているとみなし) 、次にファイル記述子 1 をファイル *fname* に関連付けます。

ジョブ制御が有効でない場合に コマンドのあとに & を指定すると、コマンドにおけるデフォルトの標準入力は /dev/null という空ファイルになります。その他の場合、コマンドを実行するための環境には、起動側シェルのファイル記述子 (入出力指定で変更可能) が含まれます。

環境

環境は、通常の引数リストが 実行されるプログラムに引き渡される場合と同様の方法で引き渡される、名前と値の対です (environ(5) を参照) 。名前は識別子、値は文字列である必要があります。シェルが環境と対話する方法はいくつかあります。シェルは、起動されると、環境を走査して、見つけた名前ごとに変数を作成し、対応する値

を設定し、さらに export というマークを付けます。実行されるコマンドは環境を引き継ぎます。ユーザーがこれらの変数の値を変更したり 新しい変数を作成したときには、export コマンドまたは typeset -x コマンドを使用すれば それらの値が環境の一部になります。したがって、実行されるコマンドが参照する環境は、シェルが最初に引き継いだ「名前 = 値」の対 (その値は現在のシェルで変更可能) に、export コマンドまたは typeset -x コマンドで指定したものを加えたものになります。

1 つまたは複数の変数代入を先頭に付加すれば、単純コマンドおよび関数の環境を 拡張できます。変数代入引数は、*identifier=value* という形式のワードです。

TERM=450 *cmd args*

および

(export TERM; TERM=450; *cmd args)*

上記 2 つの例は同じことを意味します。これは *cmd* の実行に関してだけにかぎります。ただし、*cmd* が「特殊コマンド」の項で示す * 印の付いたコマンドの場合を除きます。

-k フラグを設定すると、変数代入引数はすべて環境に格納されます。これらの引数がコマンド名のあとに指定された場合も同様です。以下の例では、まず a=b c を、次に c を表示します。

```
echo a=b c
set –k echo
a=b c
```

この機能は、シェルの初期バージョン用に作成された スクリプトで使用するためのものです。新しいスクリプトには使用しないでください。将来、この機能がなくなる可能性があるからです。

関数

前述の 「コマンド」 の項で説明した function は予約語であり、シェル関数の定義に使用します。シェル関数は内部で読み取られ、保存されます。別名は、関数を読み取る際に解釈されます。関数はコマンドと同様に実行され、引数は定位置パラメータとして渡されます (後述の「実行」を参照) 。

関数は、呼び出し側と同一のプロセスにおいて実行され、すべてのファイルと現在の作業ディレクトリを呼び出し側と共有します。呼び出し側が受け取るトラップは、関数内部でそのデフォルトの動作に再設定されます。関数が受け取らないかあるいは無視するトラップ条件があると、関数は終了し、その条件は呼び出し側に引き渡されます。

関数内部で設定された EXIT に基づくトラップは、関数が呼び出し側の環境で完了したあとに実行されます。これは、非 POSIX スタイルの関数についてのみ真です。非 POSIX スタイルの関数は次のように宣言されます。

```
function func
```

一方、POSIX スタイルの関数は次のように宣言されます。

```
func()
```

通常、変数は呼び出し側プログラムと関数間で共有されます。ただし、関数内で使用される typeset 特殊コマンドは、適用範囲として 現在の関数とそれが呼び出す関数すべてを含む 局所変数を定義します。

return という特殊コマンドは、関数の呼び出しから戻るときに使用します。関数の中でエラーが起こると、呼び出し側に制御が戻ります。

全関数の名前を一覧表示するには typeset +f と入力します。全関数の名前とともに関数のテキストも表示するには typeset -f と入力します。特定の関数のテキストだけを表示するには typeset -f *function-names* と入力します。unset 特殊コマンドの -f オプションを使えば、関数を未定義状態にすることができます。

通常、シェルがシェルスクリプトを実行するとき、関数は設定を解除されます。typeset コマンドの -xf オプションによって、別個にシェルを起動しなくても 実行されるスクリプトに関数をエクスポートできます。シェルを起動して定義する必要がある関数は、typeset の -xf オプションで ENV ファイルに指定する必要があります。

関数定義コマンド

関数とは、一群のコマンド (compound command) を、新たな定位置パラメータを指定して 単純コマンドとして呼び出すために、ユーザーが 定義した名前です。関数は「関数定義コマンド」を使って定義します。

関数定義コマンドの形式を以下に示します。

fname() *compound-command*[*io-redirect . . .*]

fname は名前でなければならず、これが関数名となります。実装によっては拡張機能として、他の文字を関数名に使うことを 許しているものもあります。その場合、関数と変数とを別個の名前領域で管理します。

関数定義コマンド中の () は、2 つの演算子で構成されます。したがって、fname、(、および) をブランク文字で区切ることもできますが、省略も可能です。

引数 *compound-command* は、関数呼び出しにより実行する一群のコマンドです。

関数が宣言されたとき、wordexp の展開は *compound-command* や *io-redirect* に対しては行われません。すべての展開処理は、通常のように、関数が呼び出されるたびに行われます。同様に、*io-redirect*(省略可能) が示す入出力のリダイレクトや、*compound-command* 中の変数割当は、関数定義時ではなく関数の実行時に行われます。

関数を実行すると、特殊組み込みユーティリティの項で説明するように、構文エラーや変数割当用の機能が与えられます。

関数名が単純コマンドとして指定されると、それに対応した *compound-command* が実行されます。その単純コマンドに指定したオペランドが、*compound-command* の実行中、一時的に位置パラメータとなります。特殊パラメータ # も、オペランド数を示す値に変更されます。特殊パラメータ 0 は変更されません。関数の実行が終了すると、

定位置パラメータや # の値は、関数実行前の値に復元されます。*compound-command* 中で特殊組み込みコマンド return が実行された場合、関数実行は終了し、関数呼び出しの次のコマンドから 処理が再開されます。

単純コマンドが記述できるところであれば、関数定義を記述できます。以下に例を示します。

```
# If variable i is equal to "yes",
# define function foo to be ls −l
#
[ "$i" = yes ] && foo() {
      ls −l
}
```

関数定義コマンドの終了ステータスは、関数が正常に宣言されれば 0 で、そうでなければゼロより大きい値です。関数呼び出しの終了ステータスは、関数によって最後に実行されたコマンドの 終了ステータスです。

ジョブ

set コマンドの monitor オプションを有効にすると、対話型シェルが job を各パイプラインと関連付けます。このオプションは、jobs コマンドが表示する現在のジョブのテーブルを維持し、これらのジョブに整数番号を割り当てます。ジョブを & で非同期に起動すると、シェルは次の形式の行を表示します。

```
[1] 1234
```

非同期に起動されたジョブはジョブ番号 1 であり、プロセス ID が 1234 であるプロセスが 1 つ (トップレベル) あることを示します。

ジョブを実行中であり、別に実行したいジョブがある場合、^Z (CTRL-Z) キーを押せば、現在のジョブに STOP シグナルが送信されます。そうするとシェルは通常、ジョブが「停止」されたことを示し、プロンプトを表示します。これで、このジョブの状態を bg コマンドでバックグラウンドで処理するか、あるいは他のコマンドを実行してから、fg というコマンドで ジョブをフォアグラウンドに移すことができます。^Z は直ちに有効になります。つまり ^Z は、保留中の出力や読み取られていない入力が 直ちに中止されるという点で、割り込みに似ています。

バックグラウンドで実行中のジョブは、端末から読み取ろうとすると停止します。通常バックグラウンドジョブは出力を生成できますが、stty tostop というコマンドを指定すればこの出力生成も抑止することができます。この tty オプションを設定すると、バックグラウンドジョブは、入力の読み取り時と同様に出力を生成しようとすると停止します。

シェル内のジョブを参照する方法はいくつかあります。ジョブは、そのジョブのプロセス ID または以下のいずれかで参照できます。

%*number*	*number* が示す番号のジョブ
%*string*	コマンド行が *string* で始まっていたジョブ
%?*string*	コマンド行が *string* を含んでいたジョブ

%%	現在のジョブ
%+	%% と同じ
%-	直前のジョブ

シェルは、プロセスの状態が変更すると、直ちにそれを検出します。ジョブがブロックされてそれ以上進めない状態になると、シェルはそのことをユーザーに通知します。ただし、ユーザーの作業の妨げにならないよう、この通知はプロンプトを発行する直前にだけ行います。

モニターモードが有効のとき、完了した各バックグラウンドジョブは、CHLD に設定されているトラップを起こします。

ジョブの実行中または停止中にシェルを終了しようとすると、「停止中 (実行中) のジョブがある (You have stopped（running）jobs.)」という警告を受けます。jobs コマンドを使用すれば、どのジョブが該当するのかを確認できます。これを実行するか、あるいは直ちにシェルを再終了しようとすると、シェルは 2 度目の警告は出さず、停止中のジョブは終了します。nohup で起動したジョブの実行中に ログアウトを行うと、次のような警告メッセージを受けます。

```
You have jobs running.
```

実際にログアウトを行うには、ログアウトを 2 回行う必要がありますが、その場合でもバックグラウンドジョブは 実行し続けます。

シグナル

起動されたコマンドに対する INT シグナルと QUIT シグナルは、コマンドの後ろに & が指定され、ジョブの monitor オプションが有効でない場合、無視されます。その他の場合、シグナルは、シェルが親から引き継いだ値を持ちます (ただし、後述の trap コマンドの説明を参照) 。

実行

コマンドが実行されるたびに、上記の置換が実行されます。コマンド名は、後述の「特殊コマンド」 のいずれかと一致する場合、現在のシェルプロセス内で実行されます。次に、コマンド名がユーザー定義関数のいずれかと 一致するかどうかチェックされます。一致する場合、定位置パラメータが保存され関数呼び出しの引数に再設定されます。関数が完了するか return を発行すると、定位置パラメータリストが復元され、関数内の EXIT に設定されているトラップが実行されます。関数の値は、最後に実行されたコマンドの値です。関数は現在のシェルプロセスでも実行されます。コマンド名が特殊コマンドやユーザー定義関数を示していない場合、プロセスが作成され、exec(2) を介してコマンドが実行されます。

PATH というシェル変数は、コマンドが置かれている ディレクトリの検索パスを定義します。ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは /bin:/usr/bin: です (/bin、/usr/bin、および現在のディレクトリの順で指定) 。現在のディレクトリは、複数のコロンを連続して記述するか、パスリストの始めか終わりにコロンを付ければ指定できます。コマンド名に / が含まれている場合は、検索パスは使用されません。/ が含まれていなければ、パスにおける各ディレクトリに実行可能ファイルがあるか検索します。ファイルが実行権を持っているが、ディレクトリや a.out ファイルでない場合、シェルコマンドの入ったファイルとみなされま

す。そのファイルを読み取るときは、サブシェルが生成されます。この場合、エクスポートされていない別名、関数、および変数すべてが削除されます。括弧で囲まれたコマンドは エクスポートされていないものを削除することなく、サブシェルで実行されます。

コマンド再入力

端末装置から最近入力された、HISTSIZE が示す個数 (デフォルトは 128 個) のコマンドのテキストは、履歴ファイルに保存されています。$HOME/.sh_history というファイルは、HISTFILE 変数が設定されていない場合、または変数が示すファイルが書き込み不可能な場合に使用されます。シェルは、同じ名前の HISTFILE を使用する対話型シェルすべてのコマンド履歴を使用できます。fc という特殊コマンドは、このファイルの一部をリスト表示または編集するときに使用します。編集またはリスト表示すべきファイルの部分は、番号か、またはコマンドの最初の文字 (1 つまたは複数) を 指定することによって選択できます。単一のコマンドを指定することも、コマンドの範囲を指定することも可能です。fc の引数としてエディタプログラムが指定されていないと、FCEDIT という変数の値が使用されます。FCEDIT が未定義の場合は、/bin/ed が使われます。編集されたコマンドは、エディタを終了した時点で 表示および再実行されます。エディタ名に - を指定すると、編集段階がスキップされ、コマンドが再実行されます。この場合、*old=new* という形式の代入パラメータを使用すれば、実行前にコマンドを変更できます。たとえば、r が 'fc -e -' の別名として定義されているとき 'r bad=good c' と入力すると、c という文字で始まるコマンドのうち最新のコマンドが、その記述中の最初の bad という 文字列が good に置き換えられて再実行されます。

インライン編集オプション

通常、端末装置から入力されるコマンド行は、単に復帰改行 (RETURN または LINEFEED) をあとに伴います。emacs、gmacs、vi のいずれかのオプションが有効な場合、ユーザーはコマンド行を編集できます。いずれかのオプションを set すれば、対応する編集モードになります。編集オプションは、オプション名のいずれかで終了する値を VISUAL 変数または EDITOR 変数に代入するたびに、自動的に選択されます。

編集機能では、ユーザーの端末が RETURN を改行のないキャリッジリターンとして扱うことができ、空白文字 がスクリーン上の現在の文字を上書きする必要があります。

これらの編集モードは、ユーザーがウィンドウごしに 現在の行を見るという概念を実現します。ウィンドウ幅は、COLUMNS が定義されていればその値に、未定義の場合は 80 になります。ウィンドウ幅が小さすぎて、プロンプトを表示すると入力用に 8 カラム以上残すことができなくなる場合には、プロンプトは左端から切り捨てられます。行がウィンドウ幅から 2 引いたものより長いと、ウィンドウの終わりにマークを表示してユーザーに通知します。カーソルが移動し、ウィンドウの境界に達すると、ウィンドウはカーソルを中心としてセンタリングされます。行がウィンドウの右端を超えている場合、表示されるマークは > となります。同様に、左端を超えていれば < が、左右両端を超えていれば * が表示されます。

各編集モードでは検索コマンドにより 履歴ファイルを使用できます。パターンではなく文字列だけがマッチングされます。ただし文字列の先頭に ^ があると、マッチング開始位置は行の先頭に限定されます。

emacs 編集モード

emacs または gmacs オプションを有効にすると、それぞれに対応する編集モードに入ります。この 2 つのモードは、^T の扱い方が異なるだけです。編集を行うには、訂正が必要な位置にカーソルを移動し、文字やワードを挿入または削除します。編集用のコマンドは、制御文字またはエスケープシーケンスで 入力します。本項の説明では、山型記号 (^) を使って制御文字を示してあります。たとえば ^F というのは CTRL–F を表します。つまり、CTRL (コントロール) キーを押した状態で f キーを入力します。このときシフトキーは押しません。なお、^? は DEL (削除) キーを表します。

エスケープシーケンスの入力は M- を使って示してあります。たとえば M-f(メタ f と呼ぶ) は、ESC キー (ASCII コード 033) のあとに f を入力することを表します。同様に M-F は、ESC キーのあとに大文字の F を入力することを表します。

編集コマンドは、行のどこからでも実行できます。行の先頭だけではありません。特に断わりのないかぎり、編集コマンドの終わりを表すために改行キー (RETURN キーまたは LINEFEED キー) を入力する必要はありません。

^F	カーソルを 1 文字分前進 (右方向) します。
M-f	カーソルを 1 ワード分前進 します。emacs エディタにおける「ワード」とは、英文字、数字、下線からなる文字列を意味します。
^B	カーソルを 1 文字分後退 (左方向) します。
M-b	カーソルを 1 ワード分後退 します。
^A	カーソルを現在の行の先頭に移動します。
^E	カーソルを現在の行の終端に移動します。
^] *char*	現在の行において文字 *char* が次に現れる位置にカーソルを移動します。
M-^] *char*	現在の行において文字 *char* が前に現れた位置にカーソルを移動します。
^X^X	カーソルとマークを入れ替えます。
erase	(ユーザーが stty(1) で定義した削除キー。通常は ^H または #) 直前の文字を削除します。
^D	現在の文字を削除します。
M-d	現在のワードを削除します。
M-^H	(メタ –Back Space) 直前のワードを削除します。
M-h	直前のワードを削除します。
M-^?	(メタ –DEL) 直前のワードを削除します。なお、割り込み文字を ^?(つまりデフォルトの DEL) と定義している場合、このコマンドは動作しません。

^T	emacs モードでは、現在の文字と次の文字とを入れ換えます。gmacs モードでは、直前の 2 つの文字を入れ換えます。
^C	現在の文字を大文字にします。
M-c	現在のワードを大文字にします。
M-l	現在のワードを小文字にします。
^K	現在カーソルが置かれている文字から 行の終端までをすべて削除します。なお、^K の前に数値パラメータを入力した場合には処理が異なります。その値が現在の位置の番号より小さければ、その値が示す位置の文字から現在の文字までを削除します。その値が現在の位置の番号より大きければ、現在の文字からその値が示す位置の文字までを削除します。
^W	カーソルのある文字からマークまでを抹消します。
M-p	カーソル位置からマークまでの 領域をスタックにプッシュします。
kill	(ユーザーが stty(1) で定義した抹消キー。通常は ^G または @) 現在の行を抹消します。2 つの kill 文字を連続して入力すると、そのあとの kill 文字は、改行を意味することになります。この機能は、プリンタ端末 (印字式端末) を使用している場合に便利です。
^Y	最後に削除されたデータを復元します。
^L	改行して、現在の行を印刷します。
^@	(NULL 文字) マークを設定します。
M–*space*	(メタ–空白文字) マークを設定します。
J	(NEWLINE) 現在の行を実行します。
M	(RETURN) 現在の行を実行します。
eof	現在の行が NULL の場合のみ、end-of-file 文字 (ファイルの終わりを示す。通常は ^D) を end-of-file として処理します。
^P	直前のコマンドを取り出します。^P を入力するたびに、1 つずつ逆方向に コマンドの履歴を取り出します。複数行にわたるコマンドで最初の行でない場合は、1 行戻ります。
M-<	最も古いコマンドの履歴を取り出します。
M->	最も新しいコマンドの履歴を取り出します。
^N	次のコマンド行を取り出します。^N を入力するたびに、1 つずつ順方向に コマンドの履歴を取り出します。
^R*string*	履歴を逆上って、*string* が示す文字列を含むコマンドを検索します。パラメータとして 0 を指定すると、順方向に検索します。*string* の終わりは、復帰改行 (RETURN または NEWLINE) で示

します。*string* の先頭に ^ を付加すると、その文字列で始まっている コマンドだけが検索されます。*string* を省略すると、直前に指定した文字列を指定したものとして 検索します。この場合、パラメータとして 0 を指定すると検索方向が逆になります。

`^O` 現在の行を実行し、現在の行に関連する次の行を 履歴ファイルから取り出します。

`M-`*digits* (エスケープ) 数値パラメータを定義します。*digits* は次のコマンドに対するパラメータと見なされます。パラメータを受け付けるコマンドは、`^F`、`^B`、*erase*、`^C`、`^D`、`^K`、`^R`、`^P`、`^N`、`^]`、`M-.`、`M-^]`、`M-_`、`M-b`、`M-c`、`M-d`、`M-f`、`M-h`、`M-l`、および `M-^H` です。

`M-`*letter* (ソフトキー) ユーザーの別名リスト中で、*_letter* という名前の別名を検索します。その別名が定義されていると、その値が入力待ち行列に挿入されます。*letter* は、前述のメタ関数に用いられている文字であってはなりません。

`M-[`*letter* (ソフトキー) ユーザーの別名リスト中で、*_letter* という名前の別名を検索します。この名前の別名が定義されていれば、処理待ちの入力待ち行列上にその値を挿入します。この機能は、多くの端末において、ファンクションキーをプログラムするために 使用できます。

`M-.` 直前のコマンドの最後のワードを 行に挿入します。数値パラメータを指定すると、最後のワードではなく その数値が表すワードが挿入されます。

`M-_` `M-.` と同じです。

`M-*` アスタリスクがワードの最後に付加され、ファイル名が展開されます。

`M-ESC` ファイル名の補完を行います。現在のワードにアスタリスクを付加したものと 一致するすべてのファイル名の最長の 前方一致部分で、現在のワードを 置き換えます。一致するものが 1 つしかない場合、ファイルがディレクトリなら / を付加し、ファイルがディレクトリでないときは空白を付加します。

`M-=` 現在のワードのあとにアスタリスクを付加すれば一致する ファイル名をリスト表示します。

`^U` 次のコマンドの数値パラメータを 4 倍します。

`\` 次の文字をエスケープします。編集用文字、ユーザーが設定した消去文字、抹消文字、割り込み文字 (通常は `^?`) は、その前に \ を指定すれば、コマンド行または検索文字列に入力できます。\ は、次の文字の編集機能 (もしあれば) を無効にします。

`^V` シェルのバージョンを表示します。

`M-#`	# を行の先頭に挿入後、行を実行します。これによりコメントを履歴ファイルに挿入することができます。

vi 編集モード

2 つの処理モードがあります。初期状態では、コマンドを入力すると、入力モードになります。編集するときは、エスケープキー (ESC(`033`)) を入力すれば制御モードになるので、訂正が必要な箇所にカーソルを移動し、必要に応じて文字やワードを挿入または削除することができます。大半の制御コマンドでは、コマンドの先頭に繰り返し数を指定できます。

多くのシステムでは、`vi` モードにいる時ははじめに標準処理モード (行入力モード) になっています。端末の回線速度が 1200 ボー以上であり、しかも制御文字を含んでいるか、または プロンプトが表示されてから 1 秒以下の経過時間であれば、コマンドは再び表示されます。ESC 文字を入力することにより、コマンドの残りについての標準処理を終了します。このときユーザーはコマンド行を変更できます。この機構により、標準処理モードは、raw モード (文字入力モード) の先行入力表示という利点を持ちます。

`viraw` オプションも設定すると、端末は標準処理モードを常に無効にすることができます。このモードは、行の終わりの区切りを示す代替記号を 2 つサポートしていないシステムでは暗に設定されています。ある種の端末においては便利な機能です。

入力編集コマンド

デフォルトでは、エディタは入力モードになります。

erase	(`stty`(1) コマンドで定義するユーザー定義消去文字、通常は `^H` または `#`) 直前の文字を削除します。
`^W`	直前の、ブランクで区切られたワードを削除します。
`^D`	シェルを終了します。
`^V`	次の文字をエスケープします。編集用の文字、ユーザーの定義した消去文字または抹消文字は、その前に `^V` を入力すれば、コマンド行または検索文字列に入力できます。`^V` は、次の文字の編集機能 (もしあれば) を無効にします。
`\`	次の *erase* または末消文字をエスケープします。

移動編集コマンド

次のコマンドはカーソルを移動させます。

[*count*]`l`	カーソルを 1 文字分右に移動します。
[*count*]`w`	1 つ先の英数字のワードに カーソルを移動します。
[*count*]`W`	ブランクがあとに続く次のワードの 先頭にカーソルを移動します。
[*count*]`e`	カーソルを ワードの終わりに移動します。
[*count*]`E`	ブランクで区切られている直前のワード の終わりにカーソルを移動します。
[*count*]`h`	カーソルを 1 文字分左に移動します。
[*count*]`b`	カーソルを 1 つ前のワードに移動します。

[*count*]B	ブランクで区切られている 直前のワードにカーソルを移動します。
[*count*]|	カーソルを *count* で示すカラムへ移動します。
[*count*]f*c*	現在の行において文字 *c* が次に現れる位置にカーソルを移動します。
[*count*]F*c*	現在の行において文字 *c* が前に現れる位置にカーソルを移動します。
[*count*]t*c*	f と h を連続して実行した場合と同じ結果です。
[*count*]T*c*	F と l を連続して実行した場合と同じ結果です。
[*count*];	直前の単一文字検索コマンドの f、F、t、または T を *count* の数だけ繰り返します。
[*count*],	直前の単一文字検索コマンドを *count* の数だけ逆方向に行います。
0	カーソルを、行の始めまで移動します。
^	カーソルを、行における最初のブランク以外の文字まで移動します。
$	カーソルを、行の終わりまで移動します。
%	現在の位置にある括弧記号（、）、{、}、[、] に対応する括弧記号にカーソルを移動します。現在の文字がいずれの括弧でもない場合、現在の行を前方向に検索し、最初に現れた括弧に対応する括弧の位置に移動します。

検索編集コマンド　以下のコマンドはコマンド履歴を使用します。

[*count*]k	直前のコマンドの履歴を取り出します。k を入力するたびに、1 つずつ逆方向に コマンドの履歴を取り出します。
[*count*]-	k と同じです。
[*count*]j	次のコマンドを取り出します。j を入力するたびに、1 つずつ順方向に コマンドの履歴を取り出します。
[*count*]+	j と同じです。
[*count*]G	*count* という番号のコマンドの履歴を取り出します。省略時は最も古いコマンドの履歴を取り出します。
/*string*	履歴をさかのぼって、*string* が示す文字列を含むコマンドを検索します。*string* の終わりは復帰改行 (RETURN または NEWLINE) で示します。*string* の先頭に ^ を付加すると、その文字列で始まっている コマンドだけを検索します。*string* が NULL の場合は、直前に指定された文字列を使用します。
?*string*	/ と同じです。ただし、検索は順方向になります。

`n`	直前の / または ? コマンドで指定した文字列と 次に一致するコマンドの履歴を検索します。
`N`	直前の / または ? コマンドで指定した文字列と 次に一致するコマンドの履歴を逆方向に検索します。

テキスト変更編集コマンド

以下のコマンドは行を変更します。

`a`	入力モードにして、現在の文字のあとにテキストを入力します。
`A`	行の終わりにテキストを追加します。`$a` と同じです。
`c`[*count*]*motion*	[*count*]`c`*motion* 現在の文字から *motion* によりカーソルが移動する先までの文字を削除し、入力モードにします。*motion* が `c` の場合、行全体を削除し入力モードにします。
`C`	現在の文字から行の終わりまでを削除し、入力モードにします。`c$` と同じです。
[*count*]`s`	*count* で指定した数の文字を削除して 入力モードにします。
`S`	`cc` と同じです。
`D`	現在の文字から行の終わりまでを削除します。`d$` と同じです。
`d`[*count*]*motion*	[*count*]`d`*motion* 現在の文字から *motion* によりカーソルが移動する先までの文字を削除します。*motion* が `d` の場合、行全体を削除します。
`i`	入力モードにして、現在の文字の前にテキストを挿入します。
`I`	行の先頭にテキストを挿入します。`0i` と同じです。
[*count*]`P`	カーソルの前に、直前のテキスト変更を挿入します。
[*count*]`p`	カーソルのあとに、直前のテキスト変更を挿入します。
`R`	入力モードにして、スクリーン上の文字を重ね打ちした文字に 置き換えます。
[*count*]`r`*c*	現在位置から始まる *count* 個の文字を *c* に置き換え、カーソルを前進させます。
[*count*]`x`	現在の文字を削除します。
[*count*]`X`	カーソル直前の文字を削除します。
[*count*]`.`	直前のテキスト変更コマンドを繰り返します。

[*count*]≈	現在のカーソル位置から始まる *count* 個の文字を、大文字の場合は小文字に、小文字の場合は大文字に変換して、カーソルを前進させます。
[*count*]_	直前のコマンドの *count* 個のワードを追加し、入力モードにします。*count* を省略すると、最後のワードを使用します。
*	* を現在のワードのあとに付けたしたものと見なし、ファイル名を生成しようとします。一致するものが見つからない場合、ベルを鳴らします。見つかった場合は、ワードを一致した文字列で置換し、入力モードにします。
\	ファイル名の補完を行います。現在のワードにアスタリスクを付加したものと 一致するすべてのファイル名の最長前方一致部分で、現在のワードを置き換えます。一致するものが 1 つしかない場合、ファイルがディレクトリであれば / を付加し、ファイルがディレクトリでなければ 空白を付加します。

他の編集コマンド　その他のコマンドを以下に説明します。

[*count*]y*motion* y[*count*]*motion*	現在の文字から *motion* によりカーソルが移動する先までの文字を、削除用バッファに入れます。テキストとカーソルは変わりません。
Y	現在の位置から行の終わりまでの文字をバッファに入れます。y$ と同じです。
u	直前のテキスト変更コマンドを取消 (undo) します。
U	現在の行で実行されたテキスト変更コマンドすべてを取消 (undo) します。
[*count*]v	`fc -e ${VISUAL:-${EDITOR:-vi}}` *count* コマンドを入力バッファに戻します。*count* を省略すると、現在の行を使用します。
^L	復帰改行して現在の行を表示します。制御モードでのみ有効です。
J	復帰改行 (NEWLINE) 。モードと無関係に現在の行を実行します。
M	復帰改行 (RETURN) 。モードと無関係に現在の行を実行します。
#	コマンドの先頭文字が # であれば、その # およびそのあとの 復帰改行 (RETURN) に続く # を削除します。先頭文字が他の文字であれば、# を各行の先頭に挿入後、行を送ります。現在の行を注釈として履歴に挿入

したり、履歴ファイル中にある以前の注釈付きコマンドから 注釈を削除したりする際に便利です。

= 現在のワードのあとにアスタリスクを付加すれば、一致する ファイル名をリスト表示します。

@*letter* 別名リストで _*letter* という名前の別名を検索します。この名前の別名が定義されていれば、処理待ちの入力待ち行列上にその値を挿入します。

特殊コマンド

以下の単純コマンドは、シェルプロセス中で実行されます。入出力のリダイレクトが可能です。特に断わりのないかぎり、出力はファイル記述子 1 上に書き込まれ、構文エラーがなければ終了ステータスは 0 です。1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、以下のような特殊な処理を受けます。

1. コマンドが完了しても、コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、それを含むスクリプトは中止されます。
4. 変数代入形式で、** を先頭に持つコマンドに続くワードは、変数代入と同一の規則で展開されます。つまり、チルド置換は = 符号のあとに実行され、ワード分割とファイル名生成は実行されません。

* : [*arg* . . .]
パラメータの展開だけを行います。

* . *file* [*arg* . . .]
file 全体を読み取ってからコマンドを実行します。コマンドは現在のシェル環境において実行されます。PATH で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。引数の *arg* は (指定されていれば) 定位置パラメータになります。引数を指定しないと定位置パラメータは変更されません。終了状態は、最後に実行されたコマンドの終了状態です。

** alias [-tx] [*name*[=*value*]] . . .
引数なしの場合、このコマンドは標準出力上に *name=value* という形式の別名のリストを表示します。*value* が指定された名前に対しては別名を定義します。*value* の後方に空白があると、次のワードが別名置換指定かどうかをチェックします。-t フラグは、検索済みの別名を設定またはリスト表示します。検索済み別名の値は、指定した *name* に対応する完全パス名になります。PATH の値を再設定すると この値は未定義になりますが、別名は検索済みのままです。-t フラグを省略すると、*value* が指定されていない引数リスト内の各 *name* について、別名の名前と値を表示します。-x フラグは、エクスポートされた別名を設定または表示します。エクスポートされた別名は、名前で起動されるスクリプト用に定義されます。*name* が指定されているが、*value* は指定されておらず、*name* に対しての別名も定義されていない場合は、終了ステータスは ゼロ以外になります。

bg [%*job*. . .]
このコマンドが有効なのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をバックグラウンドで実行します。*job* が省略された場合は、現在のジョブをバックグラウンドで実行します。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。

* break [*n*]
for ループ、while ループ、until ループ、または select ループがあれば終了します。*n* を指定すると、*n* レベル分だけ ループを終了します。

* continue [*n*]
for ループ、while ループ、until ループ、または select ループの次の繰り返しを実行します。*n* を指定すると、*n* 番目のループから実行します。

cd [*arg*]
cd *old new*
このコマンドは上記 2 つの形式の いずれかで入力します。第 1 の形式は、現在のディレクトリを *arg* に変更します。*arg* が – の場合、ディレクトリを直前のディレクトリに変更します。シェル変数 HOME の値がデフォルトの *arg* になります。PWD 変数は、現在のディレクトリに設定されます。CDPATH というシェル変数は、*arg* を含むディレクトリの検索パスを定義します。ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定) 。なお、現在のディレクトリは 空のパス名で指定します。このパス名は、等号の直後か、パスリスト内にある区切り文字のコロンの間に指定します。*arg* の先頭文字が / の場合、検索パスは使用しません。それ以外の場合は、パス中の各ディレクトリで *arg* を検索します。

cd の第 2 の形式は、PWD 中の現在のディレクトリ名における *old* という文字列を *new* という文字列に置換し、この新規のディレクトリへ変更しようとします。cd コマンドは rksh では実行できません。

command [-p] [*command_name*] [argument . . .]
command [-v -V] *command_name*
command ユーティリティは、シェルに対して、シェル関数検索を抑止し 引数を単純コマンドとして扱うようにします。-p フラグは、全標準ユーティリティを検索できる PATH のデフォルト値を使って、コマンド検索を行います。-v フラグは、標準出力に文字列を書き出します。この文字列は、現在のシェル実行環境でシェルが *command_name* を呼び出すために使用するパス名またはコマンドを表します。-V フラグも標準出力に文字列を書き出します。この文字列は、*command_name* オペランドに指定された名前を、シェルが現在のシェル実行環境で どのように解釈するかを表します。

echo [*arg* . . .]
このコマンドの使用法と説明については、echo(1) を参照してください。

* eval [*arg* . . .]
引数をシェルへの入力として読み取り、生成されるコマンドを実行します。

* exec [*arg* . . .]
arg を指定すると、このシェルの代わりに、引数で指定されたコマンドを (新規プロセスは生成せずに) 実行します。入出力引数が指定可能で、現在のプロセスに影

響を及ぼす場合があります。引数を指定しない場合は、ファイル記述子が、入出力リダイレクトリストの指定どおりに 変更されることになります。この場合、この機能によりオープンされた 2 より大きい番号のファイル記述子は、別のプログラムを起動するとクローズされます。

* `exit` [*n*]
呼び出し元のシェル、またはシェルスクリプトを *n* で指定した終了ステータスで終了させます。具体的には、指定した値の最下位 8 ビットが終了ステータスの値となります。*n* を省略すると、最後に実行されたコマンドの終了ステータス がシェルの終了ステータスになります。トラップ実行中に `exit` が発生した場合、ここで言う最後に実行されたコマンドとは、トラップ呼び出し直前に実行されたコマンドを指します。なお、`ignoreeof` オプション (後述の `set` を参照) が有効になっているシェルを除き、EOF を検出した場合もシェルが終了します。

** `export` [*name*[=*value*]] . . .
指定された *name* に対し、あとで実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

`fc` [`-e` *ename*] [`-nlr`] [*first* [*last*]]
`fc -e -` [*old*=*new*] [`command`]
第 1 の形式は、端末から最近入力された `HISTSIZE` 個のコマンドの中から、*first* から `last` までの範囲のコマンドを選択します。*first* と `last` の両引数は、数値または文字列で指定できます。文字列の場合、その文字列で始まる最新のコマンドを見つけます。負の数値は、現在のコマンド番号からのオフセットとなります。`-l` フラグを指定すると、標準出力上にコマンドをリスト表示します。`-l` を指定しないと、これらのキーボードコマンドの入ったファイル上で *ename* というエディタプログラムを起動します。*ename* が省略されていると、変数 `FCEDIT` (デフォルトは `/bin/ed`) の値をエディタとして使用します。編集が完了すると、編集されたコマンドを実行します。`last` を省略すると、*first* と同一値に設定されます。*first* を省略すると、デフォルトは、編集については直前のコマンドに、リスト表示については –16 になります。`-r` フラグはコマンドの順序を逆にします。`-n` フラグはリスト表示時にコマンド番号の出力を抑止します。第 2 の形式では、*old=new* 置換実行後に `command` が再実行されます。`command` 引数を指定しない場合、一番最後に入力したコマンドが実行されます。

`fg` [`%`*job*. . .]
このコマンドが有効なのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をフォアグラウンドで実行します。*job* が指定されないと、現在のジョブをフォアグラウンドで実行します。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。

`getopts` *optstring name* [*arg* . . .]
arg が正当なオプションを示しているかをチェックします。*arg* を省略すると、定位置パラメータが使用されます。オプション引数は + または – で始まります。+ または – 以外の文字で始まっているオプション、あるいは – – 引数があると、オプションの終わりとみなされます。*optstring* には、`getopts` が認識する文字を記述します。文字のあとに : が続く場合、そのオプションには引数があるとみなされます。オプションと引数とはブランクで区切ることができます。次の *arg* のインデックスは `OPTIND` に格納されます。オプション引数がある場合は `OPTARG` に格納されます。*optstring* 内で先頭に : がある場合は、`getopts` は無効なオプション文

字を OPTARG に格納し、*name* を ? (未定義のオプションが指定された場合) または : (必要なオプション引数が省略されている場合) に設定します。: が先頭にない場合には、getopts はエラーメッセージを表示します。オプションがなくなると、終了ステータスはゼロ以外になります。使用法と説明については、getoptcvt(1) を参照してください。

hash [*name* . . .]
シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。-r オプションを指定すると、シェルは記憶したすべての位置を破棄します。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。*Hits* は、シェルプロセスによってコマンドが呼び出された回数を表します。*Cost* は、検索パスのコマンドを見つけるのに必要な作業領域です。コマンドが検索パスの「相対」ディレクトリにある場合、そのディレクトリの変更後に そのコマンドが格納された位置が再計算されます。この再計算が行われる予定のコマンドに対しては、*Hits* 情報の隣にアスタリスク (*) が示されます。*Cost* の値は、再計算が行われるたびに増加されます。

jobs [-lnp] [%*job* . . .]
指定された各々のジョブに関する情報を一覧表示します。*job* 引数を省略すると、活動中のジョブすべてに関する情報を一覧表示します。-l フラグは、通常の情報に加えてプロセス ID も表示します。-n フラグは、前回通知を受けたあとに停止または終了した ジョブだけを表示します。-p フラグは、プロセスグループだけを表示します。*job* の記述形式についての説明は、「ジョブ」の項と jobs(1) を参照してください。

kill [-*sig*] %*job* . . .
kill [-*sig*] *pid* . . .
kill -l
TERM (終了) シグナルまたは指定されたシグナルのいずれかを、指定されたジョブまたはプロセスに送信します。シグナルは、番号または名前 (signal(3HEAD) の場合と同様に SIG という接頭辞を取り除いたもの。ただし、SIGCHD は CHLD という名前となる) で指定します。送信するシグナルが TERM (終了) または HUP (ハングアップ) の場合、停止中のジョブまたはプロセスには CONT (継続) シグナルを送信します。*job* という引数は、活動中のジョブではないプロセスのプロセス ID を指定することもできます。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。第 2 の形式の kill -l は、シグナル番号とシグナル名をリスト表示します。

let *arg* . . .
各 *arg* は、評価の対象となる個々の算術式を表します。評価の方法については、前述の「算術的評価」の項を参照してください。終了ステータスは、最後の式の値が 0 の場合には 1 で、0 以外の場合は 0 です。

login *argument* . . .
exec /bin/newgrp *arg* . .. と同じです。使用法と機能説明については login(1) を参照してください。

* newgrp [*arg* . . .]
exec /bin/newgrp *arg* . . . と同じです。

`print` [`-Rnprsu`[*n*]] [*arg* . . .]
シェルの出力機構です。フラグを省略した場合、あるいは - または - - フラグを指定した場合には、`echo`(1) で述べるように標準出力上に引数を表示します。出力ファイルが書き込み用にオープンされて いない場合を除いて、終了ステータスは `0` となります。

`-n`	復帰改行 (NEWLINE) の出力を抑止します。
`-R` | `-r`	(raw モード) `echo` のエスケープ規則を無視します。`-R` オプションは、`-n` を除く後続の引数およびオプションすべてを表示します。
`-p`	標準出力の代わりに `|&` で生成されたプロセスのパイプ上に引数を出力します。
`-s`	標準出力の代わりに 履歴ファイル上に引数を書き込みます。
`-u` [*n*]	出力を格納するファイル記述子番号を、1 桁の数値 *n* で指定します。デフォルトは `1` です。

`pwd`
`print -r - $PWD` と同じです。

`read` [`-prsu`[*n*]] [*name*`?`*prompt*] [*name* . . .]
シェルの入力機構です。1 つの行を読み取り、IFS が示す文字を区切り文字として使用して、行の内容をいくつかのフィールドに分割します。エスケープ文字 (\) は、次の文字の特別な意味または 行の継続に関する意味を取り除くために使用します。`-r` で指定する raw モードでは、\ が持つこの特殊な意味は無視されます。第 1 フィールドを 1 番目の *name* に、第 2 フィールドを 2 番目の *name* に、という順番で割り当てていき、余ったフィールドがあれば最後の *name* に割り当てます。`-p` オプションは、シェルが `|&` を使用して生成したプロセスの入力パイプから入力行を取り出します。`-s` フラグは、入力をコマンドとして履歴ファイルに保存します。`-u` フラグは、読み取り元となるファイル記述子番号を 1 桁の数値 *n* で指定します。ファイル記述子は、`exec` という特殊コマンドでオープンできます。*n* のデフォルト値は `0` です。*name* を省略すると、`REPLY` の値をデフォルトとして使用します。入力ファイルが読み込み用にオープンされていない場合と EOF に到達した場合を除き、終了ステータスは `0` です。`-p` オプションが指定されていて EOF を検出すると、このプロセスをクリアし別のプロセスを作成可能にします。最初の引数が `?` を含んでいると、シェルが対話型のとき、このワードの残りを標準エラーに対する プロンプトとして使用します。EOF に到達しないかぎり、終了ステータスは `0` です。

** `readonly` [*name*[=*value*]] . . .
name に「読み取り専用」のマークを付け、これらの名前が後続の割り当てでは変更できないようにします。

* `return` [*n*]
シェル関数または　. スクリプトを、*n* で指定された戻り値で 呼び出し側スクリプトに戻します。*n* で指定した値の最下位 8 ビットが戻り値となります。*n* を省略すると、戻り値は最後に実行された コマンドの戻り値になります。関数や　. スクリプト実行中以外で `return` を起動すると、結果は `exit` と同一になります。

set [±abCefhkmnopstuvx] [±o *option*]. . . [±A *name*] [*arg* . . .]
このコマンドのフラグの意味は以下のとおりです。

-A　配列の代入。*name* で示される変数の設定を解除し、*arg* リストから順々に値を割り当てます。+A は、最初の変数設定を解除しません。

-a　定義される後続の変数すべてを自動的にエクスポートします。

-b　シェルに対し、バックグラウンドジョブの終了を ユーザーに非同期に通知するよう要求します。以下のメッセージが標準エラー出力に書き出されます。

"[%d]%c %s%s\n", <*job-number*>, <*current*>, <*status*>, <*job-name*>

個々のフィールドの意味は次のとおりです。

<*current*>　文字 + は、fg または bg ユーティリティ用のデフォルトとして用いるジョブを表します。このジョブは、*job_id* %+ または %% を使って指定することもできます。文字 – は、現在のデフォルトジョブが終了したときに デフォルトとなるジョブを表します。このジョブは、*job_id* %– を使って指定することもできます。その他のジョブには、このフィールドは空白文字となります。+ が識別できるジョブは最大 1 つで、– が識別できるジョブも最大 1 つです。中断中のジョブがあると、現在のジョブも中断中となります。複数のジョブが中断中だと、1 つ前のジョブも中断中になります。

<*job-number*>　wait、fg、bg、kill の各ユーティリティ用にプロセスグループを識別するのに用いる番号です。これらのユーティリティを使用する際、ジョブ番号の先頭に % を付加してジョブを識別できます。

<*status*>　未定義です。

<*job-name*>　未定義です。

シェルがジョブの終了をユーザーに伝えるとき、そのジョブのプロセス ID を現在のシェル実行環境内のリストから削除することがあります。非同期の通知をデフォルトにすることはできません。

-C
シェルのリダイレクト演算子 > によって既存のファイルが上書きされないようにします。リダイレクト演算子 >| は、個々のファイルに対し、noclobber オプションに優先して有効となります。

-e
コマンドの終了ステータスが 0 でない場合、ERR トラップ (設定されていれば) を実行し、終了します。このモードは、プロファイル読み取り時は無効です。

-f
ファイル名の生成を無効にします。

-h
各コマンドは、最初に検出された時点で、検索済み別名になります。

-k
コマンド名に先行するものだけでなく、すべての変数代入引数をコマンドの環境に格納します。

-m
バックグラウンドジョブを個別のプロセスグループで実行し、完了時にメッセージを 1 行表示します。バックグラウンドジョブの終了ステータスは 完了メッセージで報告されます。ジョブ制御を備えたシステムでは、このフラグは 対話型シェルに対して自動的に有効になります。

-n
コマンドを読み取り、構文エラーがないかチェックします。実行は行いません。対話型シェルに対しては無視されます。

-o
このフラグのあとに指定する引数は、以下のオプション名のいずれかです。

allexport	-a と同じです。
errexit	-e と同じです。
bgnice	バックグラウンドジョブをすべて低い優先度で実行します。これはデフォルトモードです。
emacs	コマンド入力用に、emacs 形式のインラインエディタを起動します。
gmacs	コマンド入力用に、gmacs 形式のインラインエディタを起動します。
ignoreeof	EOF を検出してもシェルは終了しません。終了させるには exit コマンドを使用する必要があります。
keyword	-k と同じです。
markdirs	ファイル名生成によって生成されるディレクトリ名には、すべて最後に / を付加します。
monitor	-m と同じです。
noclobber	> によるリダイレクトが存在するファイルを切り捨てないようにします。このオプションが有効なとき、ファイルを切り捨てるには >| 指定が必要です。このオプションは -C と同等です。
noexec	-n と同じです。
noglob	-f と同じです。
nolog	履歴ファイルに関数定義を保存しません。
notify	-b と同等です。
nounset	-u と同じです。

privileged	-p と同じです。
verbose	-v と同じです。
trackall	-h と同じです。
vi	vi 形式のインラインエディタの挿入モードになります。033 というエスケープ文字を押すと、挿入モードから制御モードに変わります。リターンキーで行を送信します。
viraw	各文字を vi モードで入力されたときと同様に処理します。
xtrace	-x と同じです。

オプション名を指定しない場合、-o は現在のオプション設定を表示します。

-p
$HOME/.profile ファイルを処理しないようにし、ENV ファイルの代わりに /etc/suid_profile ファイルを使用します。このモードは、実効ユーザー ID が実ユーザー ID と等しくないとき、また実効グループ ID が実グループ ID と等しくないときには 必ず有効になります。このモードを無効にすると、実効ユーザー ID が実ユーザー ID に、実効グループ ID が実グループ ID にそれぞれ設定されます。

-s
定位置パラメータを辞書編集方式の順にソートします。

-t
コマンド 1 つを読み取って実行し、終了します。

-u
置換を行う際に、設定されていないパラメータをエラーとして扱います。

-v
シェルへの入力行を読み取り時に表示します。

-x
コマンドとその引数を実行時に表示します。

-
-x フラグと -v フラグを無効にし、フラグに対する引数の検査を停止します。

- -
どのフラグも変更しません。このフラグは、$1 を - で始まる値に設定する際に便利です。このフラグのあとに引数がない場合、定位置パラメータが設定解除されます。

- の代わりに + を使用すると、これらのフラグが無効になります。これらのフラグはシェル起動時にも使用できます。現在設定されているフラグは、$- で見ることができます。-A を指定しないかぎり、残りの引数は定位置パラメータとなり、$1 $2 ... に順番に割り当てられます。引数を 1 つも指定しない場合には、すべての変数の名前と値を標準出力上に表示します。

* `shift [ n ]`
  `$n+1 . . .` の定位置パラメータを `$1 . . .` という名前に変更します。*n* のデフォルト値は 1 です。*n* に指定できる値は、評価結果が `$#` 以下の負でない数になる算術式です。

`stop%jobid . . .`

`stop pid . . .`
  `stop` は、*jobid* (ジョブ ID 番号) を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (`ps`(1)参照) 。

`suspend`
  現在のシェルがログインシェルでない場合、その実行を中断します。

`test` *expression*
  条件式を評価します。使用法と機能説明については 前述の「条件式」の項と `test`(1) を参照してください。

* `times`
  シェルおよびシェルから実行されたプロセスの、ユーザー時間およびシステム時間の累計値を表示します。

* `trap [` *arg sig . . .* `]`
  arg は、*sig* が示すシグナルをシェルが受信したときに読み取られ、実行されるコマンドです。*arg* は、トラップ設定時と トラップ取り出し時に 1 度ずつ検索されます。*sig* は、シグナル番号またはシグナル名を指定します。`trap` コマンドは、シグナル番号の順序で実行されます。現在のシェルで無視されているシグナル番号にトラップを 設定しようとしても無効となります。*arg* が – の場合、シェルは各 *sig* デフォルト値を再設定します。*arg* が NULL 文字列の場合、シェルは指定された各 *sig* が発生してもそれを無視します。ただし 対応する *sig* が 1 つでも発生した場合、*arg* はシェルにより実行されます。トラップのアクションは、以前のアクション (デフォルトまたは明示的に設定されたもの) より優先して用いられます。トラップのアクション完了後、`$?` の値はトラップが呼び出されたときの値となります。

  sig は `EXIT` または `0` (`EXIT` と同義) 、またはシンボル名を使って指定したシグナルから接頭辞 `SIG` を除いたものです。たとえば `HUP`、`INT`、`QUIT`、`TERM` などです。*sig* が `0` または `EXIT` で、`trap` 文がある関数内部で実行された場合、関数終了後に *arg* の示すコマンドが実行されます。*sig* が `0` または `EXIT` で、トラップが関数の外側で設定されている場合、シェルの終了時に *arg* の示すコマンドが実行されます。*sig* が `ERR` の場合、コマンドが ゼロ以外の終了ステータスで終わると必ず *arg* が実行されます。*sig* が `DEBUG` の場合、各コマンドのあとに *arg* が実行されます。

  シェルが `EXIT` に対してトラップを実行する環境は、`EXIT` のトラップを得る以前に実行された最後の コマンドの直後の環境と同じです。

  トラップが呼び出されるたびに、*arg* 引数は以下のものと同じように処理されます。

```
eval "$arg"
```

非対話型シェルの開始時に無視されたシグナルは、トラップもリセットもできません。ただしトラップやリセットを 試みても、エラーは報告されません。対話型のシェルは、呼び出し時に無視されたシグナルを リセットしたりキャッチしたりできます。トラップは、そのシェルの動作中は、他の trap コマンドにより明示的に変更されないかぎり、有効であり続けます。

サブシェルを立ち上げたときは、トラップはデフォルトに設定されます。ただしこれは、サブシェル内では コマンドを使って新たなトラップを設定できない、ということではありません。

引数なしの trap コマンドは、各シグナルに対応したコマンドの一覧を標準出力に書き出します。その形式は次のとおりです。

```
trap — %s %s  . . . <arg>, <sig> . . .
```

シェルは、この出力の形式 (引用符の使用法も含む) を直し、同様のトラップ結果をもたらすようなコマンドとしてシェルに再入力できる形式にします。次の例を見てください。

```
save_traps=$(trap) . . . eval "$save_traps"
```

トラップの名前や番号が正しくないと、ゼロ以外の終了ステータスが 返されます。正しければ 0 が返されます。対話型のシェルも非対話型のシェルも、シグナル名やシグナル番号が誤りでも構文エラーとはならず、シェルも異常終了しません。

ジョブがフォアグラウンドプロセスを待っている間は、トラップは処理されません。このため CHLD に対するトラップは、フォアグラウンドジョブが終了するまで実行されません。

type *name* . . .
name をコマンド名として使用した場合に どのように解釈されるかを表示します。

** typeset [±HLRZfilrtux[*n*]] [*name*[=*value*]] . . .
シェル変数と関数の属性と値を設定します。関数内で typeset を実行すると、*name* が示す変数の新しいインスタンスが生成されます。関数が完了すると、その変数の値と型が復元されます。このコマンドには、以下の属性を指定できます。

-H
このフラグは UNIX 以外のマシン上で、UNIX とホスト名ファイルとのマッピング情報を提供します。

-L
左詰めを行い、先行するブランクを *value* から取り除きます。*n* は、ゼロ以外であればフィールドの幅を定義します。0 の場合、フィールドの幅は最初に代入される値の幅で決定されます。変数に値を代入したとき、フィールド幅より短ければ右側にブランクが詰められ、長ければ切り捨てられます。-Z フラグも指定されていれば、先行する 0 を削除します。-R フラグは無効になります。

-R
右詰めを行い、先行するブランクを挿入します。*n* は、ゼロ以外であればフィールドの幅を定義します。0 の場合、フィールドの幅は最初に代入される値の幅で決定されます。変数に値を代入したとき、フィールド幅より短ければ左側にブランクが詰められ、長ければ終端が切り捨てられます。-L フラグは無効になります。

-Z 最初の、ブランクでない文字が数字で、さらに -L フラグが設定されていない場合、右詰めを行い先頭に 0 を詰めます。*n* は、ゼロ以外であればフィールドの幅を定義します。0 の場合、フィールドの幅は最初に代入される値の幅で決定されます。

-f 名前は、変数名ではなく関数名を指します。代入は行われません。このフラグと共に指定できる他のフラグは、-t、-u、-x だけです。-t フラグは、この関数の実行トレースを有効にします。-u フラグは、この関数に「未定義」を示すマークを付けます。関数が参照されると、関数定義を見つけるために FPATH 変数が検索されます。-x フラグを指定すると、名前で呼び出されるシェル手続き全体で関数定義が有効になります。

-i パラメータを整数とします。これにより算術演算が高速化されます。*n* は、ゼロ以外であればその値を底として定義します。0 の場合、最初の代入で底が決定されます。

-l 大文字をすべて小文字に変換します。大文字への変換を示す -u フラグを無効にします。

-r 指定された *name* を読み取り専用にします。あとの代入でこれらの名前を変更できないようにします。

-t 変数にタグを付けます。タグはユーザーが定義可能で、シェルに対して特別の意味を持ちません。

-u 小文字をすべて大文字に変換します。小文字への変換を示す -l フラグを無効にします。

-x 指定された *name* に対し、あとで実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

-i 属性は、-R、-L、-Z、-f と同時に指定することはできません。

– の代わりに + を使用すると、これらのフラグは無効になります。*name* 引数をまったく指定せずにフラグを指定すると、これらのフラグが設定されている変数の名前 (および選択により値も) が一覧表示されます。具体的には – を付加すれば名前と値が、+ を付加すれば名前だけが表示されます。*name* 引数とフラグを 1 つも指定しないと、すべての変数の名前と属性が表示されます。

ulimit [-HSacdfnstv] [limit]
資源の制限を表示または設定します。使用可能な資源の制限は以下に説明します。システムによっては、以下に挙げたすべての 資源の制限を提供していないこともあります。limit 引数を指定すると、制限値が設定されます。limit の値は、各資源に対応した単位 (後述) の数値、または unlimited という文字列です。H と S の両フラグは、資源に対して 強い制限と弱い制限のどちらを 設定するかを表します。強い制限値は、いったん設定したら あとで増加させることはできません。弱い制限値は、強い制限値を超えない範囲で 増加させることが可能です。H も S も省略すると、指定した制限値が強い制限と弱い制限の 両方に適用されます。limit 引数を省略すると、現在の資源制限値が表示されます。このとき、H が指定

された場合を除き、表示されるのは弱い制限値です。複数の資源を指定すると、値の前に制限する資源名と単位とが表示されます。

`-a` 現在の資源制限値をすべて表示します。

`-c` コアダンプ時の コアファイルのサイズをブロック (512 バイト) 単位で表します。

`-d` データ領域のサイズを K バイト単位で表します。

`-f` 子プロセスが書き込むファイルのサイズをブロック (512 バイト) 単位で表します。読み込むファイルにはサイズの制限はありません。

`-n` 最大ファイル記述子に 1 を加えた値を表します。

`-s` スタック領域のサイズを K バイト単位で表します。

`-t` 各プロセスが使用する秒数を表します。

`-v` 仮想記憶のサイズを K バイト単位で表します。

オプションをすべて省略すると、`-f` が指定されたものとみなします。

`umask` [`-S`] [*mask*]
ユーザーファイルの作成時のマスクを *mask* 引数が示す値に設定します (`umask`(2) を参照)。*mask* には、`chmod`(1) で説明する記号値または 8 進数を指定できます。記号値を指定すると、新しい `umask` 値は、*mask* を直前の `umask` 値の補数に適用した結果の補数になります。*mask* 引数を省略すると、マスクの現在の値を表示します。`-S` フラグは、シンボリック形式の出力を生成します。

`unalias` *name*`. . .`
name が示す別名を別名リストから削除します。

`unset` [`-f`] *name* . . .
name が示す変数の設定を解除します。つまり、それらの変数の値と属性を消去します。読み取り専用の変数は設定を解除できません。`-f` フラグが指定されていると、*name* 引数は関数名を表します。`ERRNO`、`LINENO`、`MAILCHECK`、`OPTARG`、`OPTIND`、`RANDOM`、`SECONDS`、`TMOUT`、および `_` の設定を解除すると、これらの変数の特殊な意味が削除されます。あとでこれらの変数に値を代入しても、特殊な意味は持ちません。

* `wait` [*job*]
job 引数で指定されたジョブの終了を待ち、その終了ステータスを報告します。*job* を指定しないと、現在実行中の子プロセスすべてを待ちます。待つ対象のプロセスの終了ステータスが、このコマンドの終了ステータスになります。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。

`whence` [`-pv`] *name* . . .
指定された各 *name* について、それをコマンド名として使用した場合 どのように解釈されるかを示します。

`-v` フラグは、より詳細に表示します。

-p フラグは、*name* が別名、関数名、予約語の場合でも、それに対するパス検索を行います。

呼び出し

シェルを exec(2) で呼び出し、0 番目の引数 ($0) の最初の文字が – である場合、シェルをログインシェルとみなし、/etc/profile からコマンドを読み取り、次に、現在のディレクトリ内に .profile が存在するか、または $HOME/.profile がある場合、そのいずれかのファイルからコマンドを読み取ります。次に、環境変数 ENV に設定されている値をパラメータ置換することによって 指定されるファイルが存在すれば、そのファイルからコマンドを読み取ります。-s フラグが省略され、*arg* 引数が指定されている場合、最初の *arg* に対してパス検索を実行し、実行すべきスクリプトの名前を判別します。*arg* が示すスクリプトには読み取り権が必要で、setuid 設定と setgid 設定は無視されます。パス上でスクリプトが見つからない場合は、*arg* は組み込みコマンドまたは組み込み関数の名前を示しているものとして処理されます。次にコマンドは後述する方法で読み取られます。以下のフラグは、起動時にシェルによって解釈されます。

-c　このフラグが指定されると、*command_string* からコマンドを読み取ります。特殊パラメータ 0 の値は、*command_name* オペランドの値と残りの *arg* オペランドにある定位置パラメータ ($1、$2 など) から設定　されます。標準入力から読み取られるコマンドはありません。

-s　このフラグが指定された場合または引数が残っていない場合、標準入力からコマンドを読み取ります。前述の特殊コマンドの出力を除くシェル出力は、ファイル記述子 2 に書き出されます。

-i　このフラグが指定された場合または シェル入出力が端末に接続されている場合 (ioctl(2) で説明) 、このシェルは対話型となります。この場合、kill 0 が対話型シェルを終了しないように TERM を無視し、wait が割り込み可能になるように INTR を捕えて 無視します。いずれの場合も、シェルは QUIT を無視します。

-r　このフラグを指定すると、シェルは制限付きシェルになります。

他のフラグと引数については、前述の set コマンドの箇所で説明されています。

rksh の特記事項

rksh が設定するログイン名と実行環境の機能は、標準シェルの機能よりも制限を受けることになります。rksh の機能は、以下の動作ができない点を除き ksh と同じです。

- ディレクトリの変更 (cd(1) を参照)
- SHELL、ENV、または PATH の値の設定
- / を含むパス名またはコマンド名の指定
- 出力のリダイレクト (>、>|、<>、>>)
- グループの変更 (newgrp(1) を参照)

これらの制限は、.profile ファイルと ENV ファイルの解釈後に有効となります。

実行すべきコマンドがシェル手続きであることがわかると、rksh は ksh を起動し実行します。したがって一般ユーザーに対して、限られたコマンドのメニューを提供しながら、標準シェルの全機能を利用するシェル手続きを提供することが可能になります。この機構は、一般ユーザーが 同じディレクトリへの書き込み権と 実行権の両方を持ってはいないことを前提としています。

つまり、.profile の作者が、確実な設定処理を実行し ユーザーを適切なディレクトリ (おそらく、ログインディレクトリではない) に置くことにより、ユーザーの動作を完全に制御できるという点が、これらの規則の実際の効果となります。

システム管理者は、rksh で安全に起動できるコマンドのディレクトリ (つまり /usr/rbin) を設定することがよくあります。

エラー

構文エラーなどのエラーを検出すると、シェルは ゼロ以外の終了ステータスを返します。エラーがなければ、シェルは、最後に実行されたコマンドの終了ステータスを返します (前述の exit コマンドの説明を参照) 。シェルを非対話型で使用している場合、シェルファイルの実行は中止されます。シェルが検出する実行時エラーは、コマンド名と関数名、およびエラー状態を表示することにより報告されます。エラーが発生した行の番号が 1 より大きい場合、コマンド名または関数名のあとに角括弧 ([]) で囲んで行番号も表示します。

非対話型のシェルの場合、特殊組み込みユーティリティや他の種類のユーティリティがエラー状態を検出すると、シェルは診断メッセージを書き出し、以下の表に示すように終了します。

エラー	特殊組み込みユーティリティ	他のユーティリティ
シェル言語の構文エラー	終了する	終了する
ユーティリティの構文エラー		
(オプションまたはオペランド)	終了する	終了しない
リダイレクトのエラー	終了する	終了しない
変数割当のエラー	終了する	終了しない
展開エラー	終了する	終了する
コマンドが見つからない	該当せず	終了の場合あり
ドットスクリプトが見つからない	終了する	該当せず

展開エラーとは、シェルの展開時に発生するものです (たとえば ${x!y} のようなとき ! は演算子として正しくないのでエラー) 。ただし実装によっては、これらのエラーを展開時ではなく トークン化時に検出できるのであれば、構文エラーとして 扱うことも可能です。

上記の表で「終了する」 (または「終了の場合あり」) と示されているエラーが サブシェル内で発生した場合、サブシェル自身はゼロ以外のステータス で終了します (または終了する場合があります) が、サブシェルを含んでいるスクリプト自体は終了しません。

上記の表のすべての場合において、対話型のシェルは、診断メッセージを標準エラー出力に書き出すだけで終了はしません。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `ksh` と `rksh` の動作については、`largefile`(5) を参照してください。

終了ステータス

各コマンドには、他のシェルコマンドの動作に影響を与える可能性のある 終了ステータスが定義されています。ユーティリティを除く コマンドの終了ステータスは、本項内で説明します。また標準ユーティリティの終了ステータスは、それぞれの 対応する項で説明されています。

コマンドが見つからない場合、終了ステータスは `127` となります。コマンド名は見つかったが実行可能なユーティリティでは ない場合、終了ステータスは `126` となります。シェルを使わないでユーティリティを呼び出すアプリケーションは、これらの終了ステータスコードを使って同様なエラーを 報告してください。

ワードの展開中またはリダイレクト中にコマンドが失敗すると、その終了ステータスはゼロより大きい値となります。

特殊パラメータ `?` 付きの終了ステータスを報告する際、シェルは得られた終了ステータスの 8 ビットすべてを報告します。シグナルを受け取ったために終了したコマンドの終了ステータスは、`128` より大きな値となります。

ファイル

`/etc/profile`

`/etc/suid_profile`

`$HOME/.profile`

`/tmp/sh*`

`/dev/null`

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/rksh

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/ksh

属性タイプ	属性値
使用条件	SUNWxcu4

CSI	対応済み

関連項目 cat(1), cd(1), chmod(1), cut(1), echo(1), env(1), getoptcvt(1), jobs(1), login(1), newgrp(1), paste(1), ps(1), shell_builtins(1), stty(1), test(1), vi(1), dup(2), exec(2), fork(2), ioctl(2), lseek(2), pipe(2), ulimit(2), umask(2), wait(2), rand(3C), signal(3C), a.out(4), profile(4), attributes(5), environ(5), largefile(5), signal(3HEAD), XPG4(5)

Morris I. Bolsky and David G. Korn, *The KornShell Command and Programming Language*, Prentice Hall, 1989

警告 シェルスクリプトを setuid して使用することは避けてください。

注意事項 検索済み別名であるコマンドを実行し、そのあとで同一名のコマンドが、検索パスにおいて元のコマンドがあるディレクトリの前のディレクトリにインストールされた場合、シェルは元のコマンドの方を実行します。新しい方のコマンドを実行させたければ、alias コマンドの -t オプションを使用してください。

きわめて古いシェルスクリプトには、パイプ文字として | の他に ^ を許すものもあります。

複合コマンド内で fc 組み込みコマンドを使用すると、履歴ファイルからコマンド全体が消えます。

. file という組み込みコマンドは、いずれのコマンドを実行する場合でもその前に必ずファイル全体を読み取ります。したがって、ファイル内の alias コマンドと unalias コマンドは、ファイル内に定義されたどの関数にも適用されません。

存在しないコマンドのインタプリタを実行しようとするシェルスクリプトを、シェルが処理した場合、シェルは、シェルスクリプトが存在しないという間違った診断メッセージを返します。

名前 | ld – オブジェクトファイル用リンカー

形式 |

```
/usr/ccs/bin/ld [-64] [-a | -r] [-b] [-c name] [-C] [-G] [-i] [-m]
     [-s] [-t] [-V] [-B direct] [-B dynamic | static] [-B group]
     [-B local] [-B eliminate] [-B reduce] [-B symbolic] [-d y | n]
     [-D token,...] [-e epsym] [-F name | -f name] [-h name] [-I name]
     [-L path] [-l x] [-M mapfile] [-N string] [-o outfile] [-p auditlib]
     [-P auditlib] [-Q y | n] [-R path] [-S supportlib] [-u symname]
     [-Y P,dirlist] [-z absexec] [-z allextract
     | defaultextract | weakextract ] [-z combreloc] [-z defs
     | nodefs] [-z endfiltee] [-z finiarray=function] [-z groupperm
     | nogroupperm] [-z ignore | record] [-z initarray=function] [-z
     initfirst] [-z interpose] [-z lazyload | nolazyload] [-z
     ld32=arg1,arg2,...] [-z ld64=arg1,arg2,...] [-z loadfltr] [-z muldefs]
     [-z nodefaultlib] [-z nodelete] [-z nodlopen] [-z nodump] [-z
     nopartial] [-z noversion] [-z now] [-z origin] [-z
     preinitarray=function] [-z redlocsym] [-z rescan] [-z text
     | textwarn | textoff] [-z verbose] filename...
```

機能説明 | ld コマンドは、複数の再配置可能オブジェクトファイルの結合、再配置の実行、外部シンボルの解釈処理を行います。ld は 2 つのモード (静的モードと動的モード) で動作します。モードの指定には、-d オプションを使用します。いずれの場合も、ld の実行結果はデフォルトで a.out ファイルに出力されます (「注意事項」を参照)。

静的モード (-dn) では、引数として指定した再配置可能オブジェクトファイルが結合され、実行可能なオブジェクトファイルが生成されます。同時に -r オプションを指定すると、再配置可能オブジェクトファイルが結合され、1 つの再配置可能オブジェクトファイルが生成されます。

デフォルトのモードである動的モード (-dy) では、再配置可能オブジェクトファイル (引数として指定) が結合され、共有オブジェクトファイル (引数として指定) と実行時にリンクする実行可能なオブジェクトファイルが生成されます。同時に -G オプションを指定すると、再配置可能オブジェクトファイルが結合され、共有オブジェクトが生成されます。

ライブラリを含む複数の引数を指定した場合、デフォルトでは、このライブラリの検索は、引数リスト内でライブラリが検出されたときに 1 度だけ実行されます。ライブラリとして指定できるのは、再配置可能アーカイブか共有オブジェクトです (ar(3HEAD) を参照)。

アーカイブライブラリの場合、未解決の外部参照を定義するルーチンだけが読み込まれます。ld は、ライブラリメンバーによって外部参照が解決されるまで、複数のパスで、アーカイブライブラリのシンボルテーブルを順番に検索します 。このため、1 つの外部シンボルが複数のライブラリメンバーによって定義されている場合を除けば、ライブラリ内でのメンバーの順序は、機能上は重要ではありません。相互依存の関係にあるアーカイブライブラリには、複数のコマンド行定義を指定するか、-z rescan オプションを使用する必要があります。

共有オブジェクトは不可分の完全なユニットで、1 つ以上の入力ファイルをリンクすることによって生成されます。リンカーが共有オブジェクトを処理すると、共有オブジェクトの内容すべてが、結果の出力ファイルイメージの論理部分になります。共有オブジェクトが実際に取り込まれるのはプロセスの実行時であるため、リンク処理中に物理的なコピー処理が行われることはありません。リンク時の共有オブジェクトの論理的な取り込みによって、このオブジェクト内に定義されているすべてのシンボルのエントリがリンク処理可能になります。

64 ビットオブジェクトをリンクするためのオプションは不要です。リンカーは、コマンド行に最初に指定されている再配置可能オブジェクトファイルの ELF クラスによって、32 ビットまたは 64 ビットのどちらにリンクを実行するかを決定します。32 ビットオブジェクトと 62 ビットオブジェクトを混在させることはできません。`-64` オプションと `LD_NOEXEC_64` の記述も参照してください。

オプション

次のオプションを指定できます。

`-64`
64 ビットオブジェクトを作成します。デフォルトでの生成されるオブジェクトのクラスは、コマンド行から処理される最初の ELF オブジェクトによって決まります。このオプションは、*mapfile* (`-M` の説明を参照) からの入力のみで `ld` を使用してオブジェクトを直接作成する時に便利です。

`-a`
静的モード専用です。実行可能なオブジェクトファイルを生成し、未定義の参照があるとエラーを発行します (静的モードのデフォルトの動作)。`-r` オプションと同時に使用することはできません。

`-b`
動的モード専用です。実行可能ファイルを作成する際、共有オブジェクト内のシンボルを参照するような特殊な再配置処理を行いません。`-b` オプションを指定しない場合、リンカーは、位置に依存しない特殊な再配置を作成し、これを使って共有オブジェクト内に定義されている関数を参照します。また、共有オブジェクト内に定義されているデータオブジェクトが実行時リンカーによって実行可能ファイルのメモリーイメージにコピーされるようにします。

このオプションは、特殊化された動的オブジェクトを対象としているため、一般的な用途に使用することはお勧めしません。このオプションを使用すると、オブジェクトに共有性を持たせるために必要なすべての特殊処理が無効になります。また、64 ビット実行可能オブジェクトの再配置を妨げることもあります。

`-B direct`
各シンボル参照と定義を提供している依存関係との関連を記録することによって、直接結合の情報を確立します。実行時リンカーは、デフォルトのシンボル検索を行わずに、この情報を使用して、関連付けられたオブジェクト中のシンボルを直接検索します。直接結合の情報は、リンク時に指定された依存関係に対してのみ確立されるため、リンク時に `-z defs` オプションを使用する必要があります。直接結合中のシンボルよりも優先させたいオブジェクトは、`-z interpose` オプションを使用して指定する必要があります。`-B direct` オプションを使用すると、すべての依存関係に対して `-z lazyload` が有効になります。

-B dynamic | static
ライブラリの取り込みを制御するオプションです。-B dynamic は、動的モードでのみ有効です。-B dynamic オプション、-B static オプションは、コマンド行のトグルとして何度でも指定できます。たとえば、-B static オプションを指定すると、次に -B dynamic オプションが指定されるまで共有オブジェクトは受け付けられなくなります。-l オプションも参照してください。

-B eliminate
バージョン定義に割り当てられていない大域シンボルを、シンボルテーブルから削除します。このオプションでは、*mapfile* バージョン定義の一部として使用する *auto-elimination* 指令と同じようなシンボル削除処理が行われます。

-B group
共有オブジェクトとその依存オブジェクトを 1 つのグループにまとめます。グループ内のオブジェクトは、実行時に、グループ内の他のメンバーに結合されます。このフラグを持つオブジェクトの実行時プロセスは、RTLD_GROUP モードで dlopen(3DL) を使用してプロセスにオブジェクトを追加した場合と同じように動作します。あるグループに属しているオブジェクトに対して明示的な依存関係を持つオブジェクトは、自分自身もそのグループのメンバーになります。

グループは自己完結している (依存関係がグループ内で解決されている) 必要があるため、-z defs オプションを追加して、グループが自己完結していることを確認してください。

-B local
バージョン定義に割り当てられていない大域シンボルの割り当てを変更して、ローカルシンボルにします。バージョン定義とは、生成されたオブジェクト内で外部から参照可能なままにしておく必要がある大域シンボルを指定するもので、*mapfile* を介して提供されます。このオプションでは、*mapfile* バージョン定義の一部として使用する *auto-reduction* 指令と同じようなシンボルの割り当て変更処理を実行できます。この機能は、バージョン化された再配置可能オブジェクトとバージョン化されていないオブジェクトを統合する場合に便利です。

-B reduce
再配置可能オブジェクトを生成する際、バージョン定義で指定されているシンボル情報を縮約します。バージョン定義は、生成されたオブジェクト内で外部から参照可能なままにしておく必要がある大域シンボルを指定するもので、*mapfile* を介して提供されます。再配置オブジェクトが生成されると、バージョン定義は出力イメージに単純に転記されます (デフォルト)。シンボル情報の縮約は、動的実行可能ファイルまたは共有オブジェクトの構築にオブジェクト自体が使用されるまで、実際には行われません。動的実行可能ファイルや共有オブジェクトを生成する時には、自動的にこのオプションが適用されます。

-B symbolic
動的モード専用です。共有オブジェクトを作成する際、可能であれば、大域シンボルへの参照を共有オブジェクト内の参照定義に結合します。通常、共有オブジェクト内の大域シンボルへの参照は、定義が使用可能でも実行時まで結合されません。このため、実行可能オブジェクト内または他の共有オブジェクト内に定義された同

一のシンボルによって、オブジェクト自体の定義が無効になる可能性があります。`-z defs` オプションによって無効にしない限り、`ld` はシンボルが未定義であることを知らせる警告を出力します。

このオプションは、特殊化された動的オブジェクトを対象としているため、一般的な用途に使用することはお勧めしません。オブジェクトの実行時に再配置のオーバーヘッドを軽減するには、バージョン定義を作成することをお勧めします。

`-c` *name*
実行時に使用する構成ファイル *name* を記録します。構成ファイルは、デフォルト検索パスの変更に使用されます。また、ディレクトリキャッシュや代替オブジェクトの依存関係を提供します (`crle`(1) を参照)。

`-C`
診断メッセージに表示される C++ シンボル名を復号化します。

`-d y | n`
`-d y` が指定されている場合 (デフォルト)、`ld` は動的リンクを使用します。`-d n` が指定されている場合、`ld` は静的リンクを使用します。`-B dynamic | static` の項目も参照してください。

`-D` *token,token, . .*
token に指定したデバッグ情報を標準エラーに出力します。`help` は、使用可能なトークンをすべて指定した場合と同じ意味を持つ特殊なトークンです。

`-e` *epsym*
出力ファイルのエントリポイントのアドレスとして、シンボル *epsym* のアドレスを設定します。

`-f` *name*
共有オブジェクトの作成時に使用するオプションです。このオプションを指定すると、共有オブジェクトのシンボルテーブルが、*name* に指定した共有オブジェクトのシンボルテーブルで、補助フィルタとして使用されるようになります。このオプションは複数回指定できますが、`-F` オプションと同時に指定することはできません。

`-F` *name*
共有オブジェクトの作成時に使用するオプションです。このオプションを指定すると、共有オブジェクトのシンボルテーブルが、*name* に指定した共有オブジェクトのシンボルテーブルで、フィルタとして使用されるようになります。このオプションは複数回指定できますが、`-f` オプションと同時に指定することはできません。

`-G`
動的モード専用です。共有オブジェクトを生成します。未定義のシンボルも許容されます。

`-h` *name*
動的モード専用です。共有オブジェクトの作成時、*name* をオブジェクトの動的セクションに記録します。*name* は、オブジェクトのシステムファイルではなく、このオブジェクトにリンクされた動的オブジェクトに記録されます。このため、実行時リンカーは、実行時に検索する共有オブジェクト名として *name* を使用します。

-i
LD_LIBRARY_PATH の設定を無視します。LD_LIBRARY_PATH の設定がリンク処理の妨げになる場合は、このオプションを使用して、実行時のライブラリ検索への影響を防ぎます。

-I *name*
実行可能オブジェクトの作成時、プログラムヘッダーに書き込まれるインタプリタのパス名として *name* を使用します。静的モードのデフォルトの設定では、インタプリタは使用されません。動的モードのデフォルトの設定では、実行時リンカー ld.so.1(1) が使用されます。どちらの場合も、-I *name* でデフォルトの設定を無効にできます。exec(2) は、a.out を読み込む際にこのインタプリタを読み込み、a.out ではなくインタプリタに制御を渡します。

-l *x*
ライブラリ lib*x*.so (共有オブジェクトの慣例的な名前) または lib*x*.a (アーカイブライブラリの慣例的な名前) を検索します。動的モードでは、-B static オプションが有効になっている場合を除けば、ライブラリ検索パスに指定された各ディレクトリ内で、lib*x*.so ファイルまたは lib*x*.a ファイルが検索されます。ディレクトリ検索は、どちらかのファイルが入っているディレクトリを検出した時点で終了します。-l*x* が、lib*x*.so、lib*x*.a という形式の名前を持つ 2 つのファイルに展開される場合、.so ファイルが選択され、lib*x*.so が見つからない場合は lib*x*.a が選択されます。静的モードを使用している場合や -B static オプションが有効になっている場合は、.a ファイルだけが選択されます。ld は、ライブラリの名前を検出した時点でライブラリの検索を実行するので、-l の位置は重要な意味を持ちます。

-L *path*
path をライブラリ検索ディレクトリに追加します。ld のライブラリ検索は、まず -L オプションで指定したディレクトリで行われ、次に標準ディレクトリで行われます。このオプションは、同一コマンド行の -l オプションより前に指定した場合のみ有効です。環境変数 LD_LIBRARY_PATH を使用してライブラリ検索パスを追加することもできます (LD_LIBRARY_PATH を参照)。

-m
メモリーマップ (入出力セクションのリスト) と、致命的ではない多重定義シンボルを生成し、標準出力に出力します。

-M *mapfile*
マップファイル *mapfile* を、ld への指令が記述されているテキストファイルとして読み取ります。このオプションは複数回指定できます。*mapfile* がディレクトリの場合、stat(2) で定義されるように、そのディレクトリ内のすべての通常ファイルが処理対象になります。 マップファイルの詳細については、『リンカーとライブラリ』を参照してください。/usr/lib/ld ディレクトリには、プログラムのデフォルトの配置が指定されているマップファイル、4 ギガバイト以上または 4 ギガバイト未満の 64 ビット対応プログラムをリンクするためのマップファイル、および、アプリケーション内に実行不可能なスタックを確立するためのマップファイルがあります。後述の「ファイル」の項を参照してください。

-N *string*
このオプションを指定すると、作成されるオブジェクトの *.dynamic* セクションに DT_NEEDED エントリが追加されます。DT_NEEDED 文字列の値は コマンド行で指

定した *string* です。このオプションは位置に依存します。このため、DT_NEEDED *.dynamic* エントリはリンク行にある他の動的な依存オブジェクトに対して相対的になります。デバイスドライバの再配置可能オブジェクト間の依存関係を指定するときに、このオプションをオプション -dy および -r と組み合わせて使用すると便利です。

-o *outfile*
outfile という名前の出力オブジェクトファイルを生成します。デフォルトのオブジェクトファイル名は、a.out です。

-p *auditlib*
実行時のオブジェクトの監査に使用する監査ライブラリ *auditlib* を識別します。自分自身の監査を行うような共有オブジェクトと依存関係にあるオブジェクトは、この共有オブジェクトの性質を継承し、自分自身の監査を行います (-P オプションを参照)。

-P *auditlib*
指定のオブジェクトと依存関係にあるオブジェクトの実行時監査用ライブラリ *auditlib* を識別します。この監査は、このオブジェクトと依存関係にあり、監査を行う必要があるオブジェクトから継承される場合もあります (-p オプションを参照)。

-Q y | n
-Q y を指定すると、出力ファイルの作成に使用されたリンカーのバージョンを識別する ident 文字列が、出力ファイルの *.comment* セクションに追加されます。このため、複数のリンク手順 (ld -r を使用した場合など) を経て作成されたファイルには、複数の ld ident があることになります。これは、cc コマンドのデフォルト時の動作と同じです。-Q n は、バージョンの識別を抑止します。

-r
再配置可能オブジェクトファイルを結合して、1 つの再配置可能オブジェクトファイルを生成します。ld は、未解決の参照があってもメッセージを出力しません。このオプションは、-a と同時には使用できません。

-R *path*
複数のディレクトリをコロンで区切って指定します。このリストは、実行時リンカーにライブラリ検索ディレクトリを指定する際に使用されます。NULL 以外の文字列は、出力オブジェクトファイルに記録され、実行時リンカーに渡されます。このオプションを複数回指定する場合と、コロンで区切って *path* を指定する場合の結果は同じになります。

-s
出力ファイルからシンボル情報を取り除きます。デバッグ情報 (*.debug*、*.line*、*.stab* の各セクション) と、これらに関連する再配置エントリは、すべて削除されます。再配置可能ファイルと共有オブジェクト以外のシンボルテーブルと文字列テーブルのセクションも、出力オブジェクトファイルから削除されます。

-S *supportlib*
リンカーによって共有オブジェクト *supportlib* が読み込まれ、このオブジェクトにリンクのプロセスに関する情報が提供されます。環境変数 SGS_SUPPORT を使用すると、サポートされている共有オブジェクトにも同じ情報が提供されます。詳細については、『リンカーとライブラリ』を参照してください。

-t
サイズの異なる多重定義シンボルを検出した場合の警告の出力を抑止します。

-u *symname*
symname を、未定義シンボルとしてシンボルテーブルに入力します。このオプションは、すべてのルーチンをアーカイブライブラリから読み込む場合に便利です。これは、最初は空の状態になっているシンボルテーブルに、最初のルーチンを読み込むために、未解決の参照が必要になるからです。コマンド行内でのこのオプションの位置は重要な意味を持ち、シンボルを定義するライブラリより前に配置する必要があります。

-V
使用する ld のバージョン情報を示すメッセージを出力します。

-Y P,*dirlist*
ライブラリ検索に使用するデフォルトのディレクトリを変更します。*dirlist* には、複数のパスをコロンで区切って指定します。

-z absexec
動的な実行可能オブジェクトを構築する場合にのみ効果を発揮します。このオプションは、外部の絶対的シンボルへの参照を、実行時ではなく今すぐに解決するよう指示します。これによって、ある条件が整った場合には、動的オブジェクトがスワップ領域を大量に消費する可能性のあるテキストの再配置を行わなくなります。

-z allextract | defaultextract | weakextract
後続のすべてのアーカイブのオブジェクトの抽出条件を変更します。デフォルトでは、アーカイブメンバーは、未定義の参照を解決し、データ定義による仮の定義を行うために抽出されます。弱いシンボル参照は抽出を行いません。-z allextract を指定すると、すべてのアーカイブメンバーがアーカイブから抽出されます。-z weakextract を指定すると、弱い参照によってアーカイブの抽出が行われます。-z defaultextract を指定すると、既に指定されている抽出オプションを無効にし、デフォルトの状態に戻すことができます。

-z combreloc
複数の再配置セクションを結合します。通常、再配置セクションは、再配置の対象となるセクションと 1 対 1 の対応で保持されています。実行可能オブジェクトまたは共有オブジェクトを作成する際に ld は、データ再配置セクションのエントリを各エントリのシンボル参照によってソートして、実行時のシンボル参照を最低限に抑えます。複数のデータ再配置セクションを結合することで、このソート処理を最適化できるため、複数のオブジェクトをメモリーに読み込むときの再配置オーバーヘッドを最低限に抑えることができます。

-z defs | nodefs
-z defs オプションは、リンク終了時に未定義のシンボルがあると、致命的エラーを強制的に出力します。これは、実行可能オブジェクトを作成する場合のデフォルトの設定ですが、歴史的経緯から、共有オブジェクトを作成する場合にはデフォルトではありません。共有オブジェクト作成時に、そのオブジェクトが自己完結していること (シンボル参照がそのオブジェクト内またはそのオブジェクトの依存関係の中で解決されていること) を確認できるため、-z defs オプションを使用することをお勧めします。

-z nodefs オプションは、未定義のシンボルを許可します。歴史的経緯から、共有オブジェクトを作成する場合には、この動作がデフォルトになっています。このオプションを実行可能ファイルに対して指定した場合の、未定義のシンボルに対する参照の動作は不定です。-z nodefs オプションを使用することは、お勧めしません。

-z endfiltee
フィルタ処理の対象オブジェクトに印をつけます。フィルタによる検索処理は、このオブジェクトを検出した時点で終了します。

-z finiarray=*function*
構築するオブジェクトの .finiarray セクションにエントリを追加します。.finiarray セクションが存在しない場合は、新たに作成します。追加されたエントリは *function* を指すように初期化されます。詳細については『リンカーとライブラリ』を参照してください。

-z groupperm | nogroupperm
一意のグループにつながる依存関係の割り当て、または割り当ての解除を行います。グループに対して依存関係の割り当てを行うと、-B group オプションを使って依存関係を構築した場合と同じ効果が得られます。

-z ignore | record
リンク処理の一部として参照されない動的依存関係を無視または記録します。デフォルトでは -z record が有効です。

-z initarray=*function*
構築するオブジェクトの .initarray セクションにエントリを追加します。.initarray セクションが存在しない場合は、新たに作成します。追加されたエントリは *function* を指すように初期化されます。詳細については『リンカーとライブラリ』を参照してください。

-z initfirst
オブジェクトの実行時初期設定が完了してから、他のオブジェクトの実行時初期設定を同時処理します。また、同時プロセスから除去された他のすべてのオブジェクトの実行時最終設定が完了してから、オブジェクトの実行時最終設定が行われるようにします。このオプションは共有オブジェクトの作成時にのみ有用です。

-z interpose
直接結合よりも優先して検索するオブジェクトを指定します。直接結合が有効な時 (-B direct の説明を参照してください)、実行時リンカーは、直接結合に関連付けられているオブジェクトよりも先に、このオブジェクト中のシンボルを検索します。

-z lazyload | nolazyload
遅延して読み込まれる動的な依存オブジェクトの指定を有効または無効にします。lazyload と指定された動的な依存オブジェクトは、初期プロセスの起動では読み込まれません。オブジェクトの読み込みは、はじめてこのオブジェクトへの結合が行われたときに行われます。

-z ld32=*arg1,arg2,...*
-z ld64=*arg1,arg2,...*

リンカーのクラスは、作成される出力ファイルのクラスおよびリンカーが実行されているオペレーティングシステムの機能によって変わります。このオプションを使用すると、指定した引数が、32 ビットリンカーのクラスとしてのみ、または、 64 ビットリンカーのクラスとしてのみ解釈されます。

たとえば、サポートするライブラリがクラスに固有のものである場合に、そのライブラリに正しいクラスを指定するには、次のようにします。

```
ld ... -z ld32=-Saudit32.so.1 -z ld64=-Saudit64.so.1 ...
```

注: 起動されたリンカーのクラスは、コマンド行に入力ファイルとして最初に指定されている再配置可能ファイルの ELF クラスによって、ある程度特定されます。これは、`-z ld[32|64]` による処理の前に行われます。

`-z loadfltr`
フィルタを作成するとき、フィルタ対象が実行時ただちに処理されることを要求するように、オブジェクトに指定します。通常、フィルタ処理は、シンボル参照がフィルタに結合されたときにはじめて行われます。このフラグを持つオブジェクトの実行時プロセスは、環境変数 `LD_LOADFLTR` が設定されている場合と同じように動作します。`ld.so.1`(1) を参照してください。

`-z muldefs`
複数のシンボル定義を許容します。デフォルトでは、再配置可能オブジェクト間で重複したシンボル定義が発生すると、致命的なエラーになります。ところが、このオプションを指定すると、エラーにはならずに、最初の定義が有効な定義として使用されます。

`-z nodefaultlib`
オブジェクトが、実行時デフォルトライブラリ検索パス (`LD_LIBRARY_PATH` や実行パス *runpath* の後に使用) を無視するように指定します。このように指定すると、すべての依存オブジェクトが実行パス *runpath* を使用することになります。

`-z nodelete`
実行時に削除できないように、オブジェクトに指定します。このフラグを持つオブジェクトの実行時プロセスは、`RTLD_NODELETE` モードで `dlopen`(3DL) を使用して、オブジェクトをプロセスに追加する場合と同じように動作します。

`-z nodlopen`
オブジェクトを、`dlopen`(3DL) では利用できないように指定するか、`dlopen()` によって指定されたオブジェクトとして指定します。または `dlopen()` によって指定されたオブジェクトに必要な依存オブジェクトとして指定します。このオプションは、共有オブジェクトを作成している場合にのみ有用です。

`-z nodump`
オブジェクトを、`dldump`(3DL) では利用できないように指定します。

`-z nopartial`
再配置可能な入力オブジェクトファイル内に、部分的に初期化されたシンボルがある場合、このシンボルは、出力ファイルの生成時に展開されます。

`-z noversion`
どのバージョンのセクションも記録しません。バージョンセクションや関連する *.dynamic* セクションエントリは、出力イメージ内に生成されません。

-z now
オブジェクトが実行時リンカーのデフォルトモードを無効にして、遅延のない実行時結合を要求するように指定します。これは、dlopen(3DL) を使用して オブジェクトを RTLD_NOW モードでプロセスに追加したり、環境変数 LD_BIND_NOW を設定したりすることに似ています。ld.so.1(1) を参照してください。

-z origin
実行時ただちに $ORIGIN を処理することを要求するように、オブジェクトに指定します。この動作は現在はデフォルトです。このオプションは、従来機能との互換性のためにのみ残されています。

-z preinitarray=*function*
構築するオブジェクトの .preinitarray セクションにエントリを追加します。.preinitarray セクションが存在しない場合は、新たに作成します。追加されたエントリは *function* を指すように初期化されます。詳細については『リンカーとライブラリ』を参照してください。

-z redlocsym
SHT_SYMTAB シンボルテーブルから、SECT シンボル以外のすべてのローカルシンボルを削除します。ローカルシンボルを参照する配置はすべて更新され、SECT シンボルを参照するようになります。

-z rescan
リンカーに渡されたアーカイブファイルを再走査します。デフォルトでは、アーカイブの検索は、コマンド行に指定されたときに一度だけ処理されます。通常アーカイブは、そのシンボル定義でアーカイブの前に指定されている参照を解決できるように、コマンド行の最後に指定します。しかし実際には、アーカイブ間での相互依存関係を解決するために、アーカイブ自体を複数回指定しなければならないことが多くあります。

-z rescan オプションを指定すると、アーカイブリスト全体を再度処理して、シンボル参照を解決するアーカイブメンバーがあるかどうかを調べます。このアーカイブ再走査は、渡されたアーカイブリストに新しいメンバーが検出されなくなるまで続けられます。

-z text
動的モード専用です。書き込み不可の割り当て可能セクションに対する再配置があると、強制的に致命的エラーを出力します。歴史的理由から、この動作は、実行可能オブジェクトまたは共有オブジェクトを作成するときのデフォルトではありません。作成される動的オブジェクトのテキスト部分が実行中の複数のプロセスによって共有可能であり、オブジェクトをメモリーに読み込むときの再配置オーバーヘッドが最低限に抑えられることを保証するので、このオプションを使用することをお勧めします。

-z textoff
動的モード専用です。書き込み不可の割り当て可能セクションをはじめとするすべての割り当て可能セクションに対して、再配置を許可します。共有オブジェクト作成時のデフォルトの設定です。

`-z textwarn`
動的モード専用です。書き込み不可の割り当て可能セクションに 対する再配置があると、警告を出力します。実行可能オブジェクト作成時のデフォルトの設定です。

環境

`LD_LIBRARY_PATH`
`-l` オプションで指定されたライブラリを検索するディレクトリのリストを指定します。複数のディレクトリを指定する場合は、ディレクトリとディレクトリの間をコロンで区切ります。次のように、2 つのディレクトリをセミコロンで区切る方式が最も一般的です。

dirlist1 **;** *dirlist2*

以下のように `ld` の呼び出し時に `-L` が複数回指定されている場合、

ld . . . -L*path1* **. . . -L***pathn* **. . .**

検索の順番は、次のようになります。

dirlist1 path1 **. . .** *pathn dirlist2* **LIBPATH**

ディレクトリリストにセミコロンが含まれていない場合、*dirlist2* と解釈されます。

`LD_LIBRARY_PATH` 環境変数も、動的な依存関係を検索する実行時リンカーに影響を及ぼします。

この環境変数には、_32 または _64 という接尾辞を指定できます。この接尾辞を追加することで、`LD_LIBRARY_PATH` 環境変数は 32 ビットのプロセスまたは 64 ビットのプロセス専用になり、接尾辞のない `LD_LIBRARY_PATH`環境変数が有効な場合でも、優先的に使用されます。

`LD_OPTIONS`
`ld` のデフォルトのオプションセットです。`LD_OPTIONS` の値は、コマンド行で `ld` の起動コマンドの直後に指定されたものと解釈されます。つまり、次のように指定した場合と同じ結果になります。

ld $LD_OPTIONS . . . *other-arguments* **. . .**

`LD_NOEXEC_64`
64 ビットのリンカーが自動的には実行されないようにします。デフォルトでは、最初に読み取った再配置可能ファイルの ELF クラスが 64 ビットオブジェクトである場合は、自動的に64 ビット版のリンカーが起動されます。

LD_OPTIONS	ld のデフォルトのオプションセットです。LD_OPTIONS の値は、コマンド行で ld の起動コマンドの直後に指定されたものと解釈されます。つまり、次のように指定した場合と同じ結果になります。 ld $LD_OPTIONS . . . *other-arguments . . .*
LD_RUN_PATH	リンカーに実行パスを指定する代替手段です (-R オプションを参照)。LD_RUN_PATH と -R オプションを両方指定すると、-R オプションが優先されます。
SGS_SUPPORT	リンカーによって読み込まれ、リンクプロセスの情報を与えられた共有オブジェクトを、コロンで区切った形式で一覧します。この環境変数には、接尾辞として _32 または _64 を指定できます。こうすることで、この環境変数を、ld の 32 ビットクラスまたは 64 ビットクラスのどちらか専用にすることができます。この指定は、現在有効になっている接尾辞の付いていない環境変数の指定より優先されます。-S オプションも参照してください。

文字列 LD_ で始まる環境変数名は、ld と ld.so.1(1) の拡張用として予約されています。

ファイル

lib*x*.so
　共有オブジェクトライブラリ

lib*x*.a
　アーカイブライブラリ

a.out
　デフォルトの出力ファイル

LIBPATH
　/usr/lib (32 ビットライブラリ用)、または /usr/lib/sparcv9 (64 ビット SPARCV9 ライブラリ用)

/usr/lib/ld/map.bssalign
　bss 整列用のテンプレートを提供するマップファイル

/usr/lib/ld/map.default
　32 ビットプログラムのデフォルトの配置を指定するマップファイル

/usr/lib/ld/map.noexstk
　実行不可能なスタックの定義を指定するマップファイル

/usr/lib/ld/sparcv9/map.default
　64 ビット SPARC V9 プログラムのデフォルトの配置を指定するマップファイル

/usr/lib/ld/sparcv9/map.above4G
　4G バイト以上の 64 ビット SPARC V9 プログラムの望ましい配置を指定するマップファイル

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWtoo

関連項目 | as(1), crle(1), gprof(1), ld.so.1(1), pvs(1), exec(2), stat(2), dlopen(3DL), dldump(3DL), elf(3ELF), ar(3HEAD), a.out(4), attributes(5)

『リンカーとライブラリ』

注意事項 | ld のデフォルト動作を指定するためのオプションは、歴史的理由から残されています。しかし、動的オブジェクトが広く使用されている現在のプログラミング環境では、従来とは異なるデフォルトの方が有効である場合が少なくありません。しかし、従来のデフォルトは、既存のプログラム開発環境との互換性を維持するために残しておく必要があります。このマニュアルでは従来のデフォルトは、可能な限り、そのことを明示する方法で記述します。現在推奨されているオプションについては、『リンカーとライブラリ』の「リンカーのクイックリファレンス」を参照してください。

ld により作成されるたファイルがすでに存在する場合、そのファイルは、すべての入力ファイルが処理された後に破棄され、新しいファイルに置き換えられます。サイズの大きな出力ファイルが複数存在すると、システムのリソースを消費してしまうことが多いため、ld は、リンク編集中に一時ファイルを作成しません。しかし、既存のファイルが実行中のプロセスで使用されている場合は、問題が発生します。既存のファイルを使用していたプロセスは、出力ファイルのイメージが作成された時点で終了させられてしまうことがあります。この問題を回避するには、リンク編集を実行する前に、出力ファイルを削除してください。出力ファイルを削除しても、実行中のプロセスに影響はありません。これは、実際のディスク領域ではなく、ファイルシステムのネームスペースが新規出力ファイル作成用に解放されるためです。削除されたファイルのディスク領域は、そのファイルを参照している最後のプロセスが終了した時点で解放されます。

名前 | ldd – 実行可能ファイルまたは共有オブジェクトの動的依存関係の表示

形式 | **ldd** [-d | -r] [-c] [-e *envar*] [-f] [-i] [-L] [-l] [-s] [-u] [-v] *filename...*

機能説明 | ldd ユーティリティは、実行可能ファイルまたは共有オブジェクトの動的依存関係を表示します。ldd は実行時リンカー ld.so.1 を使用して診断情報を生成します。検査対象のオブジェクトは、実行中のプロセスで使用されるのと同様に扱われます。デフォルトでは、すべてのレイジーな依存関係の読み込みをトリガーします。

filename が実行可能ファイルである場合、ldd は *filename* の読み込み時に読み込まれるすべての共有オブジェクトのパス名を表示します。

filename が共有オブジェクトである場合、ldd は *filename* の読み込み時に読み込まれるすべての共有オブジェクトのパス名を表示します。ldd は、共有オブジェクトに実行権が与えられているものとみなします。実行権が与えられていない場合、ldd は警告を出力してからファイルを処理します。

ldd はファイルを 1 つずつ入力処理します。入力ファイルごとに、ldd は次のいずれかを実行します。

- オブジェクトの依存関係が存在する場合はその依存関係を表示する
- 依存関係が存在しない場合はそのまま続行する
- 正常に処理できなかった場合はエラーメッセージを出力する

オプション | ldd は、*filename* とそこで使用される共有オブジェクト間の互換性を調べることもできます。次のオプションを指定すると、ldd は *filename* の実行時に発生する未解決のシンボル参照について、警告を出力します。

-d　即時参照をチェックします。

-r　即時参照およびレイジー参照の両方をチェックします。

1 回の ldd 呼び出しでは、上記のオプションのいずれか一方だけしか指定できません。即時参照の対象は通常、実行可能または共有オブジェクトが使用するデータ項目、関数へのポインタ、そして、位置に依存する共有オブジェクトからの関数呼び出しです。レイジー参照は通常、位置に依存しない共有オブジェクトからのグローバル関数の呼び出し、または実行可能ファイルからの外部関数呼び出しです。各参照タイプの詳細は、『リンカーとライブラリ』の「再配置処理の実行に関する説明」を参照してください。オブジェクトの読み込みは、再配置処理の影響も受けます。詳細は「レイジー読み込み」を参照してください。

-c　いっさいの構成ファイルの使用を禁止します。構成ファイルは、デフォルトの検索パスの変更、ディレクトリキャッシュの提供、および代替オブジェクト依存関係の提供に利用できます。crle(1) のマニュアルページを参照してください。

-e *envar*　環境変数 *envar* を設定します。このオプションは、ldd そのものに悪影響を与える可能性のある実行時リンカー環境変数を試す場合に便利です。

`-f` 安全でない実行可能ファイルの検査を強制します。スーパーユーザーが呼び出した場合、`ldd` はデフォルトで、安全でないことが判明した実行可能ファイルをいっさい処理しません。実行可能ファイルは、指定されたインタプリタが `/usr/lib` または `/etc/lib` にない場合、あるいは、インタプリタが判別できない場合は、安全でないとみなされます。「セキュリティ」を参照してください。

`-i` 初期化セクションの実行順序を表示します。表示される順序は、`-d` または `-r` オプションの使用によって変わります。「初期化順序」を参照してください。

`-L` レイジー読み込みを有効にします。これは、検査対象のオブジェクトをプロセスの一部として読み込む場合のデフォルトの動作モードです。この場合、レイジーオブジェクト内で定義されているシンボルが参照されたときに、レイジーな依存関係またはフィルタだけがプロセスに読み込まれます。`-L` オプションとともに `-d` または `-r` オプションを使用すると、実行プロセスで発生する依存関係および読み込み順序を調べることができます。

`-l` あらゆるフィルタを強制的に即時処理します。(すべてのフィルタとその依存関係を表示するため)フィルタの即時処理は現在、`ldd` のデフォルトの動作モードです。ただし、このデフォルトモードでは、検出されなかった外部フィルタはそのまま無視されます。`-l` オプションを使用すると、検出されなかった外部フィルタに起因するエラーメッセージが生成されます。

`-s` 共有オブジェクトの依存関係を調べるのに使用した検索パスを表示します。

`-u` 未使用の依存関係を表示します。シンボル参照が依存関係に結合している場合、その依存関係は使用されたとみなされます。したがって、このオプションが有効なのは、シンボル参照を検査しているときだけです。`-r` オプションが無効の場合、`-d` オプションが有効になります。

`-r` オプションを使用したときに、未使用であることが判明したオブジェクトは、依存関係として削除されるべきです。これらのオブジェクトは参照を提供しませんが、*filename* の読み込み時に不要なオーバーヘッドが生じます。`-d` オプションを使用したときに、未使用であることが判明したオブジェクトは、*filename* の読み込み時にすぐに必要なわけではないので、レイジー読み込みの候補のものです (「レイジー読み込み」を参照)。

`-v` *filename* の処理中に発生したすべての依存関係を表示します。このオプションを指定すると、依存関係が要求するバージョン情報も表示します。`pvs`(1) のマニュアルページを参照してください。

セキュリティ

スーパーユーザーは、検査対象の実行可能ファイルが信頼できると判明している場合に限り、-f オプションを使用します。信頼できない実行可能ファイルに -f オプションを使用すると、システムのセキュリティが損なわれる可能性があります。検査対象の実行ファイルが信頼できるものかどうかが不明な場合、ユーザーユーザーは一時的に一般ユーザーになり、一般ユーザーとして ldd を呼び出す必要があります。

dump(1) および adb(1) の :r サブコマンドを使用しないで、adb(1) を使用することにより、信頼できないオブジェクトを安全に検査できます。また、スーパーユーザー以外のユーザーは adb(1) の :r サブコマンドまたは truss(1) を使用することによって、それほどリスクを冒さずに信頼できない実行可能ファイルを検査できます。信頼できない実行可能ファイルで ldd、adb の :r サブコマンド、または truss を使用する場合は、リスクを最小限に抑えるためにユーザー ID nobody を使用してください。

レイジー読み込み

レイジーな依存関係を直接指定することによって (ld(1) の -z lazyload オプションを参照)、あるいは、フィルタを使用することによって (ld(1) の -fおよび -F オプションを参照) レイジー読み込み技法を使用するオブジェクトでは、使用するオプションが原因で ldd の出力が異なることがあります。すべての依存関係がレイジーであるとされたオブジェクトの場合、ldd のデフォルトの動作により、すべての依存関係はそのオブジェクトで記録される順に出力されます。

```
example% ldd main
        libelf.so.1 =>    /usr/lib/libelf.so.1
        libnsl.so.1 =>    /usr/lib/libnsl.so.1
        libc.so.1 =>    /usr/lib/libc.so.1
```

実行時にこのオブジェクトを使用した場合の レイジー読み込み動作は、-L オプションを使用することによって有効にできます。このモードの場合、レイジーな依存関係が読み込まれるのは、レイジーオブジェクト内で定義されているシンボルが参照されたときです。したがって、-L オプションを -d および -r オプションと組み合わせて使用すると、即時参照とレイジー参照のそれぞれを満たすのに必要な依存関係を調べることができます。

```
example% ldd -L main
example% ldd -d main
        libc.so.1 =>    /usr/lib/libc.so.1
example% ldd -r main
        libc.so.1 =>    /usr/lib/libc.so.1
        libelf.so.1 =>    /usr/lib/libelf.so.1
```

この例の場合、出力される依存関係の順序は、オプションなしで実行した ldd の場合と異なります。また、-r オプションを使用した場合とも異なります。レイジー依存関係に対する参照は、実行中のプログラムと同じ順序では発生しません。

レイジー読み込みを調べると、参照を満たす必要のないオブジェクトも明らかになります。このようなオブジェクト (上記の例では libnsl.so.1) は、検査対象のオブジェクトを作成するために使用したリンク行から削除可能な候補です。

初期化順序

必要な依存関係が明示的に定義されていないオブジェクトでは、使用するオプションによって、ldd によって表示される初期化セクションの順序が異なる場合があります。次に、簡単な適用例を示します。

```
example% ldd -i main
        libA.so.1 =>    ./libA.so.1
        libc.so.1 =>    /usr/lib/libc.so.1
        libB.so.1 =>    ./libB.so.1

    init object=./libB.so.1
    init object=./libA.so.1
    init object=/usr/lib/libc.so.1
```

再配置が適用されると、初期化セクションの順序は次のようになります。

```
example% ldd -ir main
        .........

    init object=/usr/lib/libc.so.1
    init object=./libB.so.1
    init object=./libA.so.1
```

この場合、libB.so.1 は /usr/lib/libc.so.1 の関数を参照します。ただし、このライブラリに明示的な依存関係はありません。再配置が検出されてはじめて、依存関係が確立され、その結果、初期化セクションのソート順序が影響を受けます。

通常、アプリケーションの実行時に設定される初期化セクションのソート順序は、ldd に -d オプションを指定した場合と同じです。最適な順序が得られるのは、すべてのオブジェクトでそれぞれの依存関係が完全に定義されている場合です。動的オブジェクトの作成が望ましい場合は、ld(1) でオプション -z defs および -z ignore を使用します。

1 つ以上の動的オブジェクトが相互に参照する場合には、循環型依存関係が生じる可能性があります。循環型依存関係は、各依存関係固有のソート順序を確立できないので、避けなければなりません。

オブジェクトファイルを静的に分析する方が望ましい場合は、dump(1) や elfdump(1) などのツールを使用して依存関係を調べることができます。

ファイル

/usr/lib/lddstub	共有オブジェクトの依存関係を検査するために読み込まれた擬似実行可能ファイル

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWtoo

関連項目

adb(1), crle(1), dump(1), elfdump(1), ld(1), ld.so.1(1), pvs(1), truss(1), dlopen(3DL), attributes(5)

リンカーとライブラリ

診断

ldd は共有オブジェクトのパス名の記録を 標準出力に書き込みます。シンボルの解決問題に関する任意選択のリストは、標準エラー出力に書き込まれます。*filename* が実行可能ファイルまたは共有オブジェクトではない場合、あるいは、読み取り用にオープンできない場合、ゼロ以外の終了ステータスが返されます。

注意事項

ldd は、dlopen(3DL) を使用して明示的に接続された共有オブジェクトを表示しません。

共有オブジェクトで -d または -r オプションを使用すると、誤解を生じるような結果が出力される場合があります。ldd は共有オブジェクトにおける「最悪の場合」の解析を行います。しかし実際には、未解決として報告されたシンボルでも、その一部または全部が共有オブジェクトを参照する実行可能ファイルによって解決されることがあります。実行時リンカーのプリロードメカニズム (LD_PRELOAD) を使用すると、検査対象のオブジェクトに依存関係を追加できます。

ldd は実行時リンカーと同じアルゴリズムを使用して、共有オブジェクトを見つけます。

名前 | let – 1 つ以上の算術式を評価するためのシェル組み込みコマンド

ksh | **`let`** *arg*...

ksh | 各 *arg* は、 評価の対象となる個々の算術式を表します。

終了ステータス | 以下の終了ステータスが返されます。

`0` 最後の式の値が 0 以外

`1` 最後の式の値が 0

属性 | 次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | `ksh`(1), `set`(1), `typeset`(1), `attributes`(5)

名前 | lex – 字句解析プログラムの生成

形式 | **lex** [-cntv] [-e | -w] [-V -Q [y | n]] [*file*...]

機能説明 | lex ユーティリティは、文字入力の字句解析に用いる C プログラムを生成します。このプログラムは、 yacc へのインタフェースとして使用できます。 lex ソースコードから生成された C プログラムは、 ISO C 標準に準拠しています。通常 lex ユーティリティは、生成したプログラムを lex.yy.c ファイルに書き出します。 lex の終了ステータスがゼロでない場合、このファイルの状態は不定です。 lex の入力言語に関する詳細は、「拡張機能説明」の項を 参照してください。

オプション | 以下のオプションを使用できます。

-c　C 言語の動作を示します。本指定がデフォルトです。

-e　EUC 文字を扱えるプログラムを生成します。本オプションは -w と同時に指定することはできません。 yytext[ ] の型は unsigned char[ ] となります。

-n　-v によって出力される合計情報の出力を抑止します。 lex ソースコード中にテーブルのサイズが指定されず、 -v オプションを省略した場合には、この -n が指定されたものと見なされます。

-t　生成したプログラムを、 lex.yy.c ファイルではなく標準出力上に書き出します。

-v　lex 合計情報を標準エラー出力に書き出します(「 lex の定義」の項にある lex テーブルサイズの説明を参照してください)。 lex ソースコード中にテーブルサイズが指定されていて、 -n オプションが省略された場合、 -v オプションが有効となります。

-w　EUC 文字を扱えるプログラムを生成します。本オプションは -e と同時に指定することはできません。 yytext[ ] の型は wchar_t[ ] となります。この点が -e 指定と異なります。

-V　標準エラー出力上にバージョン情報を書き出します。

-Q[y|n]　-Qy は、ファイル lex.yy.c にバージョン情報を書き出します。-Qn はバージョン情報を出力しません。デフォルト指定は -Qn です。

オペランド | 以下のオペランドを指定できます。

file　入力ファイルのパス名。複数のファイルを指定すると、 それらすべてが連結されて 1 つの lex プログラムが生成されます。 このオペランドを 1 つも指定しない場合、およびオペランドとして – を指定した場合、標準入力が用いられます。

Stdout | -t オプションが指定されると、 lex の C ソースコード出力のテキストファイルが標準出力に書き出されます。

Stderr | -t オプションが指定されると、 lex ソースコード入力の内容に関する情報メッセージ、 エラーメッセージ、および警告メッセージが標準エラー出力に出力されます。

-t オプションが指定されなければ、次のようになります。

1. lex ソースコード入力の内容に関する情報メッセージ、 エラーメッセージ、および警告メッセージが 標準出力または標準エラー出力に書き出されます。
2. -v オプションが指定され、 -n オプションが指定されなかった場合は、 lex 統計情報が標準エラー出力に書き出されます。 -n オプションが指定されていない限り、 lex の定義の セクション (「拡張機能説明」を参照のこと) 内にテーブルサイズが % 演算子で指定された場合も、この統計情報が出力されます。

出力ファイル

C ソースコードが入ったテキストファイルが lex.yy.c に、または -t オプションが指定された場合は標準出力に書き出されます。

拡張機能説明

各入力ファイルには lex ソースコードが入っています。これは、 C プログラムフラグメントの形式で対応するアクションを定義した 正規表現のテーブルです。

lex.yy.c がコンパイルされ lex ライブラリにリンクされると (c89 または cc で -l l オペランドを使って)、 生成されたプログラムは標準入力から文字入力を読み込み、 与えられた式と一致する文字列に分割します。

式が一致すると、以下の処理が行われます。

- 一致した入力文字列は、 NULL で終わっている文字列として *yytext* に残されます。 *yytext* は、外部文字配列または文字列へのポインタのどちらかです。 lex の定義の セクション で説明するように、 %array または %pointer 宣言を使えば型を明示選択できますが、 デフォルトは %array です。
- 照合文字列の長さが外部変数 int *yyleng* に設定されます。
- 式に対応するプログラムフラグメントまたはアクションが実行されます。

パターン照合の間、 lex はパターンセットを検索し、一致するもっとも長い文字列を見つけ出します。 同じ文字数の一致する文字列が複数ある場合は、最初のパターンが選択されます。

lex ソースの一般形式は、次のとおりです。

Definitions
%%
Rules
%%
User Subroutines

最初の %% はルール (正規表現とアクション) の開始を示すために必要なもので、 2 番目の %% はユーザーサブルーチンが続く場合にのみ必要なものです。

lex の定義の セクション にある 空白文字で始まる 行は、 C プログラムフラグメントとみなされ、 lex.yy.c ファイルの外部定義域にコピーされます。同じく、 %{ と %} だけの区切り行に囲まれた部分もそのまま、 lex.yy.c ファイルの外部定義域にコピーされます。

他のルールが指定される前に、「lex の定義」の セクションの始めに、このような入力 (空白文字で始まる、 または %{ と %} だけの区切り行に囲まれる) があれば、 yylex 関数の変数宣言のあと、 yylex 内の最初のコード行の前に、 lex.yy.c に書き出されます。そのため yylex に入ったときに実行されるアプリケーションコードだけでなく、 yylex にローカルなユーザー変数をここで宣言できます。

いくつかのルールのあと、 *Rules* セクションに空白文字で始まる、または %{ と %} だけの区切り行に囲まれる入力があったときに lex が実行するアクションは、未定義です。このような入力があったとき、 yylex 関数の定義にエラーが生じることがあります。

lex 内の定義

lex 内の定義 は最初の %% 区切り行より前にあります。このセクション内で %{ と %} 区切り行に囲まれていない行で、しかも空白文字以外で始まっている行は、 lex 置換文字列を定義するものとみなされます。 これらの行の形式は、次のとおりです。

name substitute

ISO C 標準の識別子の条件を *name* が満たしていない場合、結果は未定義です。 *substitute* 文字列は、ルールとして使用されると文字列 (*name*) を置き換えます。この状況で *name* 文字列が認識されるのは、 中括弧が使用されていて、 大括弧や二重引用符で囲まれていないときだけです。

lex 内の定義 セクションでは、 % (パーセント記号) で始まりその後ろに s または S で始まる英数字語が続く行は、開始条件セットを定義します。 % (パーセント記号) で始まりその後ろに x または X で始まる英数字語が続く行は、排他的な開始条件セットを定義します。 生成されたスキャナが %s 状態のときは、状態が指定されていないパターンもアクティブになります。 %x 状態では、このようなパターンはアクティブになりません。 最初の語を除いた行の残りは、 空白文字で区切られた開始条件名とみなされます。 開始条件名は、定義名と同じ方法で作成されます。「lex の正規表現」の項で説明するように、開始条件は、 正規表現の照合をいくつかの状態に制限するときに使用できます。

lex 内の定義 セクションで、次の排他的な宣言の 2 つのうちどちらかを使用します。

%array	*yytext* の型を NULL で終わる文字配列と宣言します。
%pointer	*yytext* の型を NULL で終わる文字配列へのポインタと宣言します。

yytext を変更するために、 %pointer オプションと同時に yyless 関数を使用することはできないので注意してください。

%array はデフォルトです。 %array が指定されている (または %array と %pointer の 2 つとも指定されていない) 場合に、外部参照先を *yyext* にするには以下のように書式を宣言します。

```
extern char yytext[ ]
```

`%pointer` が指定されている場合、 外部参照先の書式は以下の通りです。

```
extern char *yytext;
```

`lex` は、 特定の内部テーブルサイズを設定するため、 lex 内の定義 セクションで宣言することができます。次の表に、宣言を示します。

`lex` でのテーブルサイズ宣言

宣言	説明	デフォルト
`%p`*n*	ポジションの数	2500
`%n`*n*	状態の数	500
`%a`*n*	遷移の数	2000
`%e`*n*	解析ツリーのノード数	1000
`%k`*n*	パック文字クラスの数	10000
`%o`*n*	出力配列のサイズ	3000

`lex` により生成されたプログラムは、補助コードセットの EUC 文字を含む入力データを処理するため、 `-e` または `-w` のいずれかのオプションの指定を必要とします。この両オプションがともに省略されると、 `yytext` の型は `char[ ]` となり、生成されたプログラムは ASCII 以外のコードセットを扱うことはできません。

`-e` オプションを使用すると、 `yytext` の型は `unsigned char[ ]` となり、 `yyleng` は一致した文字列のバイト数を示すことになります。マクロ `input()` 、 `unput(`*c*`)` 、および `output(`*c*`)` は、通常の ASCII `lex` と同様に、バイトを基準とした I/O を実行しなければなりません。 `-e` オプションで使用できる変数は他に 2 つあります。それは `yywtext` と `yywleng` で、それぞれ `-w` オプション指定時の `yytext` および `yyleng` と同じ動きをします。

`-w` オプションを指定すると、 `yytext` の型は `wchar_t[ ]` となり、 `yyleng` には一致した文字列の文字数が記録されます。 ユーザーがこのオプションを使って独自の `input()` 、 `unput(`*c*`)` 、または `output(`*c*`)` マクロを指定するのであれば、それらはワイド文字 (`wchar_t`) の形式で EUC 文字を返したり受け取ったりするよう設計しなければなりません。 これによって、ユーザーのプログラムと `lex` 内部との間に別のインタフェースを設け、 他のプログラムの処理速度を上げることが可能となります。

lex 内のルール

`lex` ソースファイル内のルールは、左のカラムには正規表現、 右のカラムにはその正規表現が認識されたときに実行されるアクション (C プログラムフラグメント) が入ったテーブルです。

ERE action
ERE action
...

拡張正規表現 (ERE) 部分と *action* は、 1 つ以上の空白文字で区切られています。空白文字が入った正規表現は、 以下の条件の 1 つが満たされる場合に認識されます。

- 表現全体が二重引用符で囲まれている
- 二重引用符または大括弧内に空白文字がある
- 各空白文字の前にバックスラッシュがある

lex 内のユーザーサブルーチン

ユーザーサブルーチン内のものはすべて、 `lex.yy.c` の `yylex` のあとにコピーされます。

lex 内の正規表現

`lex` ユーティリティは、 `regex`(5) で記述されている拡張正規表現 (ERE) セットをサポートします。 ただし以下のように追加された構文と、例外となる構文があります。

`. . .` 二重引用符で囲まれた文字列は、二重引用符内の文字を表します。 ただしバックスラッシュエスケープの認識は除きます (次の表を参照してください)。バックスラッシュエスケープシーケンスは、 閉じる引用符で終端されます。たとえば `" \ 01""1"` は 8 進数の 1 と文字 1 からなる 1 個の文字列を表現します。

<state>r

<state1, state2, . . . >r 正規表現 *r* が照合されるのは、プログラムが *state* 、 *state1* 、 *state2* などによって示される開始条件の 1 つに合うときだけです。詳細については、「lex 内のアクション」の項を参照してください (以降本書では、印刷上の規則の例外として、たとえば *<state>* はメタ変数を表さず、記号を囲むリテラル山括弧文字を表します)。 このように開始条件が認識されるのは、正規表現の始めだけです。

r/*x* 正規表現 *r* が照合されるのは、 *r* の後ろに正規表現 *x* が続いているときだけです。 *yytext* で戻されるトークンは、 *r* とだけ一致します。 *r* の末尾部分が *x* の先頭部分と一致する場合の結果は不定です。 *r* 式には末尾コンテキストや `$` (行の終わり) 演算子は含めることができません。 *x* には `^` (行の始まり) 演算子、末尾コンテキスト、 `$` 演算子は含めることができません。つまり `lex` 正規表現には 1 つの末尾コンテキストしか含めず、 `^` 演算子が使用できるのはこのような式の始めだけです。 末尾コンテキスト演算子 / (スラッシュ) は、 括弧でグループ化できないよう制限されています。

{*name*} *name* が *Definitions* セクションからの置換記号の 1 つなら、中括弧も含め、文字列は *substitute* 値で置き換えられます。 拡張正規表現内では、括弧で囲まれているものとして *substitute* 値が扱われます。 {*name*} が大括弧や二重引用符でかこまれている場合、 置換は行われません。

ERE 内では、バックスラッシュ文字 (`\\`, `\ a`, `\ b`, `\ f`, `\ n`, `\ r`, `\ t`, `\ v`) をエスケープシーケンスの始まりとみなします。 さらに以下のエスケープシーケンスが認識されます。

リテラル復帰改行文字 (NEWLINE) は、 ERE 内ではありえません。復帰改行文字を表現するには、エスケープシーケンス `\ n` が使用されます。復帰改行文字は、ピリオド演算子とは照合できません。

`lex` 内のエスケープシーケンス

エスケープシーケンス	説明	意味
\ *digits*	バックスラッシュ文字とそのあとの 1 、 2 、または 3 桁の 8 進数の最長シーケンス (01234567)。すべての桁が 0 なら (つまり NULL 文字なら)、動作は未定義	エンコーディングが 1 、 2 、または 3 桁の 8 進整数を表す文字。システム上のバイトのサイズが 9 ビットよりも大きい場合、バイトを表現するために使用される 有効なエスケープシーケンスは、実装状態に依存する。 複数バイト文字には、バイトごとの先行 \ を含め、このタイプの複数の連結されたエスケープシーケンスが必要
\ x*digits*	バックスラッシュ文字とそのあとの 16 進文字の最長シーケンス (01234567abcdefABCDEF)。すべての桁が 0 なら (つまり NULL 文字なら)、動作は未定義	エンコーディングが 16 進整数で表現される文字
\ *c*	バックスラッシュ文字とそのあとに上記以外の文字があるもの (\ \, \ a, \ b, \ f, \e n, \ r, \ t, \ v)	文字 *c*、そのまま

`lex` 用の拡張正規表現の優先順位は、下の表に示すとおりです。 優先度の高い順に並んでいます。

注: エスケープ文字エントリは、 これらが演算子であることを意味しませんが、真 の演算子との関係を示すため、表に組み込まれています。 この項で説明した配置方法の制限のため、開始条件、末尾コンテキスト、 およびアンカー指定は省略されています。これらは ERE の先頭または末尾にだけ指定できます。

`lex` 内での `ERE` 優先順位

照合関連の括弧記号	`[= =] [: :] [. .]`
エスケープ文字	\<*special character*>
大括弧表現	`[ ]`
引用	`". . ."`
グループ化	`()`

定義	{*name*}
単一文字 *RE* 重複	* + ?
連結	
インターバル表現	{*m,n*}
択一	|

ERE アンカー演算子 (^ と $) は、表にはありません。 lex 正規表現では、これらの演算子の使用が次のように制限されています。 ^ 演算子は正規表現の始めにしか使用できず、 $ 演算子は最後にしか使用できません。 演算子は正規表現全体に適用されます。そのためたとえば、 (^abc)|(def$) などのパターンは未定義です。これは 2 つの別々のルールとして書くことができます。正規表現 ^abc と正規表現 def$ です。この 2 つは、特殊な | アクション (下記を参照のこと) を介して共通なアクションを共有します。 パターンが ^abc|def$ と書かれたら、 abc または def だけの行と一致します。

一般の ERE ルールと異なり、従来のほとんどの lex の実装では組み込み型アンカーは許可されていません。 組み込み型アンカーの例としては、 foo が完全な語として存在するときに foo と照合させるための (^)foo($) などのパターン用があります。これは、以下の既存の lex 機能を使えば可能です。

```
^foo/[ \ n] ｜
" foo"/[ \ n] /* 別の語として見つかった foo */
```

なお $ も末尾コンテキストの形式であり (/\ n と等価)、その演算子の別のインスタンスを含む正規表現では 使用できません (上の末尾コンテキストの説明を参照してください) 。

追加の正規表現末尾コンテキスト演算子 / (スラッシュ) は、二重引用符で囲む (" / ") 、前にバックスラッシュを付ける (\ /) 、または大括弧で囲めば ([/]) 、通常の文字として使用できます。開始条件 < と > 演算子は、正規表現の始めの開始条件においてだけ特別とみなされます。 正規表現のその他の場所では、通常の文字として扱われます。

以下の例を見れば、 lex 正規表現と本書に出てくるその他の正規表現との違いがわかります。 *r*/*x* 形式の正規表現の場合、いつも r と一致する文字列 r が返されます。 *x* の先頭が *r* の末尾部分と一致する場合に混乱が生じることがあります。 たとえば、正規表現 a*b/cc と入力 aaabcc が与えられると、 *yytext* には aaab が入ります。しかし正規表現 x*/xy と入力 xxxy が与えられると、 xxx は x* に一致するため、 xx ではなく xxx が返されます。

ルール ab*/bc では、 *r* の最後の b* が末尾コンテキストの先頭の b までに一致するため 、結果は不定です。ただしこのルールが ab/bc なら、 テキスト ab にテキスト bc が続く場合に一致します。この場合 *r* は *x* の先頭の b までに一致しないので、結果は明確です。

lex 内のアクション

ERE が一致したときに実行されるアクションは、 C プログラムフラグメントまたは以下に説明する特殊なアクションです。 プログラムフラグメントには 1 つ以上の C ステートメントと特殊なアクションが入っています。空の C ステートメント ; は、有効なアクションです。このようなルールのパターン部分に一致する lex.yy.c 入力内の文字列は、実際には無視またはスキップされます。 ただしアクションの不在は無効であり、このような状況で lex が実行するアクションは未定義です。

C ステートメントと特殊アクションも含め、アクションについての仕様は、 中括弧で囲まれていれば複数行にわたることができます。

ERE < 1 つ以上の空白文字 > { プログラムステートメント プログラムステートメント }

lex.yy.c プログラムへの入力内の文字列がどの表現とも一致しないときは、 デフォルトアクションとして、文字列が出力にコピーされます。 lex によって生成されるプログラムのデフォルトの動作は、 入力を読み込んで出力へコピーするだけなので、 %% だけの最小 lex ソースプログラムは、入力をそのまま出力へコピーするだけの C プログラムを生成します。

4 つの特殊アクションが使用可能です。

```
| ECHO; REJECT; BEGIN
```

| アクション | は、次のルールのアクションが このルールのアクションであることを意味します。他の 3 つのアクションとは異なり、 | は中括弧で囲んだりセミコロンを付けたりできません。 | は単独で指定する必要があります。

ECHO; 文字列 *yytext* の内容を出力に書き出します。

REJECT; 1 つの式だけを入力内の文字列と一致させるのが普通です。 REJECT は「現在の入力と一致する次の式まで継続する」という意味で、 現在のルールが実行されたあと、 2 番目のルールが何であれ同じ入力に対して実行されます。そのため、 1 つの入力文字列または重なる入力文字列に対して複数のルールを 一致させて実行することができます。たとえば、正規表現 xyz 、正規表現 xy 、および入力 xyz が与えられると、通常は正規表現 xyz だけが一致します。次の照合は z のあとから開始されます。 xyz ルールの最後のアクションが REJECT なら、このルールと xy ルールの両方が実行されます。 yylex の別の部分への goto と同じように、制御の流れが継続しないような方法で、 REJECT アクションが実装されています。 REJECT を使用すると、スキャナはある程度大きくなり、実行が遅くなります。

BEGIN

BEGIN *newstate*;

このアクションは、状態 (開始条件) を *newstate* に切り替えます。文字列 *newstate* が lex 内の定義 セクションに開始条件として宣言されていない

と、結果は不定です。 初期状態は、数字の 0 または INITIAL トークンで示されます。

以下に説明する関数やマクロは、 lex 入力内のユーザーコードをアクセスできます。 lex の C コード出力に示されるかどうか、および c89 や cc への -l　l オペランドを通してだけアクセス可能かどうかは不定です (lex ライブラリ) 。

int yylex (void)
入力の字句を解析します。これは lex ユーティリティによって生成される主な関数です。 この関数は、入力の終わりに達するとゼロを返します。 その他の場合は、選択されたアクションによって決定された ゼロ以外の値 (トークン) を返します。

int yymore (void)
呼び出されると、次の入力文字列がいつ認識されるかを示します。 置き換えるのではなく、 *yytext* の現在の値に付加されます。これに合わせて *yyleng* の値が調整されます。

int yyless (int *n*)
NULL で終わっている *yytext* 内の最初の *n* 文字を記憶し、残りの文字は読み取っていないものとします。これに合わせて *yyleng* の値が調整されます。

int input (void)
入力から次の文字を返します。ファイルが終わりのときはゼロを返します。 ストリームポインタ *yyin* から、おそらく中間バッファを介して入力を得ます。 そのためスキャニングの開始後に *yyin* の値を変更した場合の影響は未定義です。読み込まれた文字列は、 スキャナの入力ストリームからそのまま取り除かれます。

int unput (int *c*)
文字 c を入力に返します。次の式が一致するまで *yytext* と *yyleng* は未定義です。入力された文字より多い文字に対して *unput* を使用すると、結果は不定です。

以下の関数は -l l オペランドを通してアクセス可能な lex ライブラリ内にだけ出てきます。 そのためこれらの関数は移植性のあるアプリケーションによって再定義可能です。

int yywrap(void)
ファイルの終わりで yylex によって呼び出されます。デフォルトの yywrap は 1 を返します。アプリケーションが yylex に別の入力ソースで処理を継続させたい場合、アプリケーションは関数 yywarp を組み込めます。 yywarp は別のファイルと外部変数 FILE **yyin* を関連付け、ゼロの値を返します。

int main(int *argc*, char **argv*[ ])
字句解析のために yylex を呼び出してから、終了します。 ユーザーコードにはアプリケーションに固有な動作を実行する main を組み込んだり、必要なら yylex を呼び出したりできます。

移植性のあるアプリケーションによって確実に再定義可能なのは libl.a の関数だけだという理由によって、上記の関数は 2 つのグループに分割されています。

lex によって生成される名前で、 input 、 unput 、 main を除くすべての外部およびスタティック名には、接頭辞 yy または YY が付きます。

使用法

lex 内のルール セクションではアクションのない *ERE* は受け付けられないことが、移植性のあるアプリケーションに警告されますが、 `lex` がエラーとして検出する必要はありません。 これはコンパイルまたは実行時エラーを引き起こすことがあります。

`input` の目的は、字句分析に関して、入力ストリームから文字を 取り出し破棄することです。コメントの先頭を検出したら、 コメント全体を破棄するという使い方が一般的です。

`lex` ユーティリティは、 `lex` ソースコードや生成された字句アナライザにおける 正規表現の扱いが完全には国際化されていません。 字句アナライザの実行時に、指定された環境に応じて `lex` ソース内の正規表現を解析することが理想とされますが、現在の `lex` テクノロジではこれは不可能です。さらに `lex` によって生成される字句アナライザの特徴は、 ロケール固有なことが多い入力言語の字句要件に 密接に結びついています (たとえば、フランス語に使用するアナライザを作成しても、 自動的にその他の言語の処理に役立つことはありません) 。

使用例

以下の例は、 Pascal に似た構文用の簡単なスキャナを実装する `lex` プログラムです。

```
%{
/* 以下の atof() を呼び出すのに必要 */
#include <math.h>
/* 以下の printf()、fopen()、stdin を呼び出すのに必要 */
#include <stdio.h>
%}
DIGIT    [0-9]
ID       [a-z][a-z0-9]*
%%
{DIGIT}+    {
    printf("An integer:   %s   (%d)\n",   yytext,
        atoi(yytext));
    }
{DIGIT}+"."{DIGIT}*    {
    printf("A float:   %s   (%g)\n",   yytext,
        atof(yytext));
    }
if|then|begin|end|procedure|function            {
    printf("A keyword:   %s\n",   yytext);
    }
{ID}    printf("An identifier:   %s\n",   yytext);
"+"|"-"|"*"|"/"    printf("An operator:   %s\n",   yytext);
"{"[^}\n]*"}"    /* 1 行コメントを飛ばす */
\t\n]+    /* 空白を読み飛ばすspace */
.    printf("Unrecognized character: %s\n",   yytext);
%%
int main(int argc, char   *argv[ ])
{
    ++argv,   --argc;  /* プログラム名をスキップする */
    if (argc > 0)
        yyin = fopen(argv[0],   "r");
    else
        yyin = stdin;
    yylex();
}
```

環境

lex の実行に影響を与える環境変数 LC_COLLATE、 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWbtool

関連項目

yacc(1), attributes(5), environ(5), regex(5)

注意事項

.l ファイルの yyback()、 yywrap()、 yylock() などのルーチンが外部 C 関数となる場合には、 C++ プログラムをコンパイルするコマンド行で __EXTERN_C__ マクロを定義する必要があります。以下に例を示します。

example% CC -D__EXTERN_C__ ... *file*

名前 | limit, ulimit, unlimit – 現在のシェルとそのシェルから起動されたプロセスで利用できるシステム資源の制限値を設定または取得

形式 | **/usr/bin/ulimit** [-f] [*blocks*]

sh | **ulimit** [- [HS] [a | cdfnstv]]

ulimit [- [HS] [c | d | f | n | s | t | v]] *limit*

csh | **limit** [-h] [*resource* [*limit*]]

unlimit [-h] [*resource*]

ksh | **ulimit** [-HSacdfnstv] [*limit*]

/usr/bin/ulimit | ulimit ユーティリティは、シェルとその子プロセスに課せられる 書き込み時のファイルサイズの制限値を、設定または報告します (ファイルサイズに関係なく 読み取りは可能です) 。適切な特権をもつプロセスのみ 制限値を上げることができます。

sh | Bourne シェルの組み込み関数 ulimit は、資源の強い制限値また弱い制限値を表示または設定します。これらの制限値については getrlimit(2) の説明を参照してください。

limit 引数を省略すると、ulimit は指定されている制限値を表示します。制限値は一度にいくつでも表示できます。-a オプションは制限値すべてを表示します。

limit 引数を指定すると、ulimit は指定されたフラグに対応する制限値をその引数の値に設定します。*limit* 引数の値として unlimited という文字列を指定すると、有効な最大値に設定されます。一度に資源 1 つについてだけ制限値を設定できます。ユーザーは誰でも、弱い制限値を強い制限値を超えない 任意の値に設定できます。また、強い制限値を下げることもできます。ただし、強い制限値を上げることができるのはスーパーユーザーだけです。詳細は su(1M) を参照してください。

-H オプションは強い制限値を表し、-S オプションは弱い制限値を表します。どちらのオプションも指定しない場合、ulimit は両方の制限値を設定し、弱い制限値を表示します。

以下のオプションは、制限値を表示または設定すべき資源を指定します。オプションをいっさい指定しないと、ファイルサイズ制限値を表示または設定します。

-c　最大コアファイルサイズ (512 バイトブロック単位)

-d　データセグメントまたはヒープの最大サイズ (K バイト単位)

-f　最大ファイルサイズ (512 バイトブロック単位)

-n　最大ファイル記述子に 1 を加えたもの

-s　スタックセグメントの最大サイズ (K バイト単位)

-t　最大 CPU 時間 (秒単位)

-v　仮想記憶の最大サイズ (K バイト単位)

csh C シェルの組み込み関数 `limit` は現在のプロセス、またはそれが生成したすべてのプロセスについて、各プロセスが指定された *resource* を *limit* 以上消費しないよう制限します。*limit* を省略すると、現在の制限値を表示します。*resource* を省略すると、すべての制限値を表示します。システムで利用可能な最大上限値を 調べるには `sysdef`(1M) コマンドを実行してください。表示される値は 16 進数ですが、`bc`(1) コマンドを使って 10 進数に変換できます。

`-h` 現在の制限値ではなく強い制限値を使用します。強い制限値は現在の制限値を制限します。強い制限値を上げることができるのは特権ユーザーだけです。

resource として指定できるものは次のとおりです。

`cputime`	プロセス当たりの最大 CPU 使用時間 (秒)
`filesize`	ファイルシステムのサイズに制限された最大の単一ファイル容量 (`df`(1M) 参照)
`datasize`	K バイト単位の プロセスのヒープの最大サイズ
`stacksize`	プロセスの最大スタックサイズ (`swap`(1M) 参照)
`coredumpsize`	最大コアダンプのファイルサイズ ファイルシステムのサイズに制限する
`descriptors`	ファイル記述子の最大数 (`sysdef()` を実行)
`memorysize`	仮想記憶の最大サイズ

limit は数値で、以下の単位を付加して指定することもできます。

n`h`	(`cputime` の) 時間
n`k`	*n*K バイト。これは `cputime` を除くすべての値のデフォルト単位
n`m`	*n*M バイトまたは (`cputime` の) 分
mm`:`*ss*	(`cputime` の) 分と秒

`unlimit` は *resource* に関する制限値を削除します。*resource* が指定されないと、すべての資源の制限値が削除されます。資源名の一覧については、前述の `limit` コマンドの説明を参照してください。

`-h` 対応する強い制限値を削除します。これは特権ユーザーだけが実行できます。

ksh Korn シェルの組み込み関数 `ulimit` は資源の制限を表示または設定します。使用可能な資源の制限は以下に説明します。システムによっては、以下に挙げたすべての 資源の制限を提供していないこともあります。*limit* 引数を指定すると、制限値が設定されます。*limit* の値は、各資源に対応した単位 (後述) の数値、または `unlimited` という文字列です。`-H` と `-S` の両フラグは、資源に対して 強い制限と弱い制限のどちらを設定するかを表します。強い制限値は、いったん設定したら あとで値を上げることはできません。弱い制限値は、強い制限値を超えない範囲で 値を上げることが可能です。`-H` も `-S` も省略すると、指定した制限値が強い制限と弱い制限の 両方に適用され

ます。*limit* 引数を省略すると、現在の資源制限値が表示されます。このとき、-H が指定された場合を除き、表示されるのは弱い制限値です。複数の資源を指定すると、値の前に制限する資源名と単位とが表示されます。

-a　現在の資源制限値をすべて表示します。

-c　コアダンプ時のコアファイルのサイズをブロック (512 バイト) 単位で表します。

-d　データ領域のサイズを K バイト単位で表します。

-f　子プロセスが書き込むファイルのサイズをブロック (512 バイト) 単位で表します。読み取るファイルにはサイズの制限はありません。

-n　最大ファイル記述子に 1 を加えた値を表します。

-s　スタック領域のサイズを K バイト単位で表します。

-t　各プロセスが使用する秒数 (CPU 時間) を表します。

-v　仮想記憶のサイズを K バイト単位で表します。

オプションをすべて省略すると、-f が指定されたものとみなします。

オプション

ulimit では以下のオプションが指定できます。

-f　ブロックごとにファイルサイズの制限を設定 (*blocks* を指定しない場合は報告) します。これはデフォルト値です。

オペランド

ulimit では以下のオペランドが指定できます。

blocks　新しくファイルサイズの制限として使用する 512 バイトごとのブロック数

/usr/bin/ulimit

例 **1** スタックサイズを制限する

スタックサイズを 512K バイトに 制限します。

```
% ulimit -s 512
% ulimit -a
% time(seconds)         unlimited
file(blocks)            100
data(kbytes)            523256
stack(kbytes)           512
coredump(blocks)        200
nofiles(descriptors)    64
memory(kbytes)          unlimited
```

sh/ksh

例 **2** ファイル記述子の数を 12 に制限する

ファイル記述子の数を 12 に制限します。

```
$ ulimit -n 12
$ ulimit -a
time(seconds)            unlimited
file(blocks)             41943
data(kbytes)             523256
```

例 2 ファイル記述子の数を 12 に制限する　(続き)

```
stack(kbytes)             8192
coredump(blocks)          200
nofiles(descriptors)      12
vmemory(kbytes)           unlimited
```

csh

例 3 コアダンプファイルのサイズを制限する

コアダンプファイルのサイズを 0K バイトに 制限します。

```
% limit coredumpsize 0
% limit
cputime                 unlimited
filesize                unlimited
datasize                523256 kbytes
stacksize               8192 kbytes
coredumpsize            0 kbytes
descriptors             64
memorysize              unlimited
```

例 4 コアファイルサイズについての制限を削除する

上記の制限から コアファイルサイズについての制限を削除します。

```
% unlimit coredumpsize
% limit
cputime                 unlimited
filesize                unlimited
datasize                523256 kbytes
stacksize               8192 kbytes
coredumpsize            unlimited
descriptors             64
memorysize              unlimited
```

環境

ulimit の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　正常終了

>0　　要求した制限値が大きいため拒否された、またはエラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | bc(1), csh(1), ksh(1), sh(1), df(1M), su(1M), swap(1M), sysdef(1M), getrlimit(2), attributes(5), environ(5)

名前 | ln – ファイルへのハードリンクまたはシンボリックリンクの作成

形式 | **/usr/bin/ln** [-fns] *source_file* [*target*]

/usr/bin/ln [-fns] *source_file... target*

/usr/xpg4/bin/ln [-fs] *source_file* [*target*]

/usr/xpg/bin/ln [-fs] *source_file... target*

機能説明 | 上記 2 つの形式のうち、第 1 の形式の ln ユーティリティは、*source_file* が示すファイル用に新しいディレクトリエントリ (リンク) を、*target* が示す宛先パスに作成します。*target* が省略されると、カレントディレクトリ内にリンクを作成します。最終オペランドが既存のディレクトリを指していなければ、この第 1 の形式の ln 呼び出しと見なされます。オペランドの数が 3 つ以上で、最終オペランドが既存のディレクトリではない場合、エラーとなります。

第 2 の形式の ln ユーティリティは、*source_file* オペランドで示された各ファイル用に新しいディレクトリを、*target* が示す既存のディレクトリの宛先パスに作成します。

ln ユーティリティは、ハードリンクとシンボリックリンクの両方を作成することができます。ハードリンクはファイルへのポインタで、元のディレクトリエントリと区別されません。ファイルに加えられる変更は、そのファイルの参照にどの名前が使用されたかにかかわらず有効です。ハードリンクは、複数のファイルシステムにまたがることはできません。また、ディレクトリも参照することはできません。

デフォルトでは、ln はハードリンクを作成します。*source_file* は *target* にリンクされます。*target* がディレクトリの場合は、*target* の中に *source_file* という名前の別のファイルが作成され、元の *source_file* にリンクされます。

/usr/bin/ln | *target* がファイルの場合は、そのファイルの内容は上書きされます。/usr/bin/ln は、*target* のモードが書き込み禁止であった場合、そのモード (chmod(1) を参照) を出力し、応答を要求し、標準入力から 1 行を読み取ります。応答が肯定である場合、アクセス権が与えられていればリンクが行われ、アクセス権が与えられていなければコマンドは終了します。

/usr/xpg4/bin/ln | *target* がファイルで -f オプションが指定されていない場合は、/usr/xpg4/bin/ln は標準エラーに診断メッセージを書き込み、現在の *source_file* には何も行わず、他の残りの *source_file* に処理を進めます。

シンボリックリンクはファイルへの間接的なポインタで、そのディレクトリエントリにはリンク先のファイルの名前が書かれています。シンボリックリンクはファイルシステムをまたぐことができ、またディレクトリを参照することもできます。

ハードリンクを作成する時や、リンク元のファイル自体がシンボリックリンクである時、リンク先は、シンボリックリンクのオブジェクト自体 (*source_file*) へのハードリンクではなく、シンボリックリンクによって参照されているファイルへのハードリンクになります。

target のアクセス権は -l を付けた ls(1) コマンドで表示される アクセス権とは、異なる場合があります。*target* のアクセス権を表示する場合は、ls -lL を使用してください。詳細については、stat(2) を参照してください。

オプション　以下のオプションは、/usr/bin/ln と /usr/xpg4/bin/ln で指定できます。

`-f`　*target* のモードで書き込みが禁止されていても、ユーザーに問い合わせることをしないリンクファイル。標準入力が端末でない場合、デフォルトはこの設定になっています。

`-s`　シンボリックリンクを作成します。

`-s` オプションに 2 つの引数を指定する場合、*target* に指定できるのは、既存のディレクトリまたは存在しないファイルです。*target* がすでに存在し、ディレクトリでもない場合は、エラーが返されます。ファイル名 *source_file* には、あらゆるパス名を指定でき、それは既存のものである必要はありません。既存のパス名にする場合は、ファイルとディレクトリのいずれも指定でき、また、*target* とは異なるファイルシステムにあるものを 指定することもできます。*target* が既存のディレクトリである場合は、ディレクトリ *target* の中に *source_file* または *source_file* の最後の構成要素の名前の付いたファイルが作成されます。このファイルは、*source_file* を参照するシンボリックリンクです。*target* が存在しない場合は、*target* という名前のファイルが作成され、*source_file* を参照するシンボリックリンクになります。

`-s` オプションに 3 つ以上の引数を指定する場合は、*target* は既存のディレクトリでなければなりません。既存のディレクトリでない場合は、エラーが返されます。*source_file* ごとに、*target* の中に *source_file* の最後の構成要素の名前の付いたリンクが作成されます。新しい *source_file* は、それぞれ、元の *source_file* へのシンボリックリンクです。ファイルとターゲットは、異なるファイルシステムにあってもかまいません。

/usr/bin/ln　以下のオプションは、/usr/bin/ln でのみ指定できます。

`-n`　リンク名が既存のファイルである場合は、ファイルの内容を上書きしません。`-f` オプションを使用すると、このオプションは無効になります。これは /usr/xpg4/bin/ln のデフォルトの動作であるため、/usr/xpg4/bin/ln に対して指定しても無視されます。

オペランド　以下のオペランドを指定できます。

source_file　リンクされるファイルのパス名。これは、通常ファイル、特殊ファイルのどちらでもかまいません。`-s` オプションが指定された場合には、*source_file* にはディレクトリを指定することもできます。

target　新しいディレクトリエントリのパス名、または 新しいディレクトリエントリが作られる既存ディレクトリのパス名。

使用法　ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `ln` の動作については、`largefile`(5) を参照してください。

環境

ln の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　指定されたファイルはすべて正常にリンクされた

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/ln

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/ln

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

chmod(1)、ls(1)、stat(2)、attributes(5)、environ(5)、largefile(5)、XPG4(5)

注意事項

ディレクトリへのシンボリックリンクは、予想とは異なった動作をする場合があります。シンボリックリンクに対して ls(1) を実行すると、指定したディレクトリ中のファイルが表示されますが、一方 ls -l はリンク自体の情報を表示します。

```
example% ln -s dir link
example% ls link
file1 file2 file3 file4
example% ls -l link
lrwxrwxrwx  1 user            7 Jan 11 23:27 link -> dir
```

cd(1) を使って、シンボリックリンクを介したディレクトリに変更すると、ファイルシステム中の指定位置に移ることになります。つまり新たな作業ディレクトリの親は、シンボリックリンクの親ではなく、指定されたディレクトリの親になります。以下に示す例で、最終的な作業ディレクトリは /home/user/linktest ではなく /usr である点に注意してください。

```
example% pwd
/home/user/linktest
example% ln -s /usr/tmp symlink
example% cd symlink
example% cd . .
example% pwd
/usr
```

C シェルのユーザーは、cd の代わりに C シェルの組み込みコマンドである pushd や popd を使えば、ディレクトリ移動がこのように複雑になるのを回避できます。

名前 | locale – ロケール固有の情報の取得

形式 | **locale** [-a | -m]

locale [-ck] *name...*

機能説明 | locale ユーティリティは現在のロケール環境、または、すべての公開ロケールに関する情報を標準出力に書き込みます。このマニュアルページにおいて、「公開ロケール」とは、アプリケーションからアクセスできる、当該実装上で利用可能なロケールのことです。

引数を指定しないで locale を呼び出すと、環境変数の設定値によって決定されたロケールカテゴリごとに、現在のロケール環境が表示されます。

オペランドを指定して locale を呼び出すと、次のように、ロケールカテゴリでキーワードに割り当てられている値を標準出力に書き込みます。

- キーワード名を指定すると、指定したキーワードとそのキーワードが含まれているカテゴリが選択される。
- カテゴリ名を指定すると、指定したカテゴリとそのカテゴリ内のすべてのキーワードが選択される。

オプション | 次のオプションを指定できます。

-a　利用可能なすべての公開ロケールに関する情報を標準出力に書き込みます。利用可能なロケールには、POSIX ロケールを表す POSIX も含まれます。

-c　選択したロケールカテゴリの名前を標準出力に書き込みます。-c オプションは、複数のカテゴリを選択したときに (たとえば、複数のキーワード名やカテゴリ名を使用したとき)、表示を読みやすくするために使用します。このオプションは、-k オプションの指定に関係なく有効です。

-k　選択したキーワードの名前と値を標準出力に書き込みます。実装によっては、一部のキーワードで値が省略されることがあります。「オペランド」を参照してください。

-m　利用できる文字マッピング (charmap) の名前を標準出力に書き込みます。localedef(1) のマニュアルページを参照してください。

オペランド | 次のオペランドを指定できます。

name　ロケールカテゴリの名前、ロケールカテゴリ内のキーワードの名前、または予約名 charmap。指定したカテゴリまたはキーワードが選択され、出力されます。1 つの *name* が現在のロケールでロケールカテゴリ名とキーワード名の両方を表す場合、指定しないのと同じ結果になります。それ以外の場合、カテゴリ名とキーワード名は、どちらも *name* オペランドとして任意の順序で指定できます。

使用例 | 例 **1** locale ユーティリティの例

以下の例では、ロケール環境変数は次のように設定されているものとします。

例 1 locale ユーティリティの例　(続き)

```
LANG=locale_x
LC_COLLATE=locale_y
```

locale は次の出力を生成します。

```
LANG=locale_x
LC_CTYPE="locale_x"
LC_NUMERIC="locale_x"
LC_TIME="locale_x"
LC_COLLATE=locale_y
LC_MONETARY="locale_x"
LC_MESSAGES="locale_x"
LC_ALL=
```

コマンド LC_ALL=POSIX locale -ck decimal_point は、次の出力を生成します。

```
LC_NUMERIC
decimal_point="."
```

次の例は、locale を使用して、ユーザーの応答が肯定応答であるかどうかを調べるアプリケーションの一部を示します。

```
if printf "%s\n" "$response" | /usr/xpg4/bin/grep -Eq\
        "$(locale yesexpr)"
then
    affirmative processing goes here
else
    non-affirmative processing goes here
fi
```

環境

LANG、LC_ALL、LC_CTYPE、LC_MESSAGES、および NLSPATH については、environ(5) のマニュアルページを参照してください。

LANG、LC_*、およびNLSPATH 環境変数には、出力する現在のロケール環境が指定されていなければなりません。これらの環境変数は、-a オプションが指定されていない場合に使用されます。

終了ステータス

次の終了ステータスが返されます。

0　　要求したすべての情報が見つかり、正常に出力された

>0　　エラーが発生した

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWloc
CSI	有効

関連項目 | localedef(1), attributes(5), charmap(5), environ(5), locale(5)

注意事項 | LC_CTYPE または LC_CTYPE カテゴリのキーワードを指定すると、0x00 ～ 0x7f の範囲内の値だけが書き込まれます。

LC_COLLATE またはLC_COLLATE カテゴリのキーワードを指定すると、実際の値は書き込まれません。

名前 | login – システムへのログイン

形式 | **login** [-p] [-d *device*] [-h *hostname* | [*terminal*] | -r *hostname*] [*name* [*environ...*]]

機能説明 | login コマンドは、 各端末セッションの開始時に使用するコマンドです。このコマンドを使用して、システムにユーザーの識別情報を提示します。 login コマンドは、あるユーザーが exit コマンドを実行してログインシェルを 終了させたあと、次のユーザーが最初に接続を確立するときに、 システムによって呼び出されます。

login がコマンドとして呼び出された場合には、 初期コマンドインタプリタが 置き換えられる必要があります。 この方法で login を呼び出すには、初期シェルから次のように入力します。

```
exec login
```

C シェルと Korn シェルには、それぞれに login の組み込みコマンドがあります。 login の組み込みコマンドと使用方法の詳細については ksh(1) と csh(1) を参照してください。

引数としてユーザー名を指定しないと、login は、ユーザー名とユーザーパスワード (必要な場合) の入力を求めます。 可能な場合、 パスワードの入力中はエコーが抑止されます。このため、 セッションの書き込み記録にパスワードが残ってしまうことはありません。

ログイン手続きの際に入力を誤ると、 次のメッセージが出力され、 新しいログインプロンプトが表示されます。

```
Login incorrect
```

ログイン手続きに 5 回失敗すると、 これがすべて /var/adm/loginlog ファイルに記録され (ファイルが存在しない場合もある)、 TTY 回線が切断されます。

パスワードの有効期限を設定する機能が有効になっている場合に、パスワードが有効期限を過ぎてしまうと、ユーザーはパスワードの変更を求められます (詳細については passwd(1) を参照)。このとき、パスワードの記録場所を特定するために、 /etc/nsswitch.conf ファイルが参照されます (nsswitch.conf(4) を参照) 。 なお、パスワードの更新は、次の 5 つの形式でのみ実行できます。

- `passwd: files`
- `passwd: files nis`
- `passwd: files nisplus`
- `passwd: compat` (==> files NIS)
- `passwd: compat` (==> files NIS+)

  `passwd_compat: nisplus`

いずれの形式にも合っていないと、 passwd(1) が異常終了するため、システムにログインできません。 一定時間内に正常にログインできなかった場合は、 メッセージで通知されることなく接続が解除されることがあります。

正常にログインすると、 アカウンティングファイルが更新され、 デバイスの所有者、グループ、アクセス権が /etc/logindevperm ファイルの内容に従って設定され、 最終ログイン時刻が表示されます (logindevperm(4) を参照) 。

ユーザー ID、 グループ ID、 補助グループリスト、 作業用ディレクトリは初期化され、 コマンドインタプリタ (通常 ksh) が起動されます。

基本環境は、次のように初期設定されます。

HOME=*your-login-directory*
LOGNAME=*your-login-name*
PATH=/usr/bin:
SHELL=*last-field-of-passwd-entry*
MAIL=/var/mail/*your-login-name*
TZ=*timezone-specification*

Bourne シェルや Korn シェルでログインする場合は、それぞれ /etc/profile と $HOME/.profile が実行されます (ファイルが存在しない場合もある)。 一方、C シェルでログインする場合は、/etc/.login 、 $HOME/.cshrc 、 $HOME/.login が実行されます。 デフォルトの /etc/profile ファイルと /etc/.login ファイルは、ディスク容量の確認 (quota(1M) 参照)、/etc/motd の表示、メールの確認を行います。$HOME/.hushlogin ファイルが存在する場合は、メッセージは何も表示されません。コマンドインタプリタ名は、- の後にインタプリタのパス名の最後の構成要素を付けたものになります (例 : -sh)。

パスワードファイルの *login-shell* フィールド (passwd(4) を参照) が空であれば、デフォルト時のコマンドインタプリタ /usr/bin/sh が使用されます。 このフィールドが アスタリスク (*) の場合は、指定のディレクトリがルートディレクトリになります。 この時点で、独自のルート構造を持つ新しいレベルで login が再実行されます。

login の実行時または login がログイン名を要求したときに login に引数を追加すると、 環境を拡張したり変更したりできます。 この引数は、 *xxx* または *xxx=yyy* という形式で指定します。= (等号) を使用しない形式の引数は、 次のように環境設定内に配置します。

L*n*=*xxx*

このとき、*n* は 0 から始まる通し番号になります。この数値は、 新しい変数が必要になるたびに大きくなります。= (等号) を使用した形式の変数は、 そのまま環境設定に取り込まれます。 すでに設定が環境に取り込まれている場合に新しい変数を指定すると、前の値が新しい設定値に置き換えられます。

ただし、例外が 2 つあります。 変数 PATH と変数 SHELL は変更できません。 これは、制限付きのシェル環境にログインしているユーザーが、子プロセスとして制限のない新しいシェルを生成するのを防ぐためです。 login は、単純な単一文字引用規則を認識します。 文字の前に \ (バックスラッシュ) を付けるとその文字が引用項目になるので、 空白文字やタブなどの文字も使用できます。

login に -p フラグをつけることによって、 現在の環境を引き渡すこともできます。このフラグにより、現在定義されているすべての環境変数が、 可能であれば新しい環境に引き渡されます。このオプションを指定しても、変数 PATH と SHELL は変更できません。環境変数の制限を 回避することはできません。 変数が両方の方法で引き渡された場合、login の引数に指定した環境変数が優先されます。

スーパーユーザーによるリモートログインを可能にするには、/etc/default/login ファイルを編集し CONSOLE=/dev/console エントリの前に # (ハッシュ記号) を挿入してください。 これについては、「ファイル」の項を参照してください。

セキュリティ

login コマンドは pam(3PAM) を使って、認証、アカウント管理、セッション管理、パスワード管理を行います。/etc/pam.conf に記述されている PAM 構成ポリシーには、login で使用するモジュールが指定されています。以下に、pam.conf ファイルの抜粋を示します。このファイルには、UNIX 認証、アカウント管理、セッション管理、パスワード管理モジュールを使用する login コマンド用のエントリが含まれています。

```
login  auth       required  pam_authtok_get.so.1
login  auth       required  pam_dhkeys.so.1
login  auth       required  pam_unix_auth.so.1
login  auth       required  pam_dial_auth.so.1

login  account    requisite pam_roles.so.1
login  account    required  pam_projects.so.1
login  account    required  pam_unix_account.so.1

login  session    required  pam_unix_session.so.1
```

パスワード管理スタックは、以下のようになります。

```
other  password   required  pam_dhkeys.so.1
other  password   requisite  pam_authtok_get.so.1
other  password   requisite  pam_authtok_check.so.1
other  password   required  pam_authtok_store.so.1
```

login サービスのエントリがない場合は別のサービスのエントリが使用されます。認証モジュールが複数含まれている場合は、複数のパスワードが必要になることがあります。

rlogind や telnetd で login が呼び出されるときに、 PAM が使用するサービス名はそれぞれ rlogin、telnet になります。

オプション

次のオプションを指定できます。

-d *device*　　login は、 デバイスオプション *device* を受け付けます。*device* は、 login が動作する TTY ポートのパス名とみなされます。デ

バイスオプションを使用すると、 `login` のパフォーマンスの向上が期待できます。 これは `login` が `ttyname`(3C) を呼び出す必要がなくなるからです。`-d` オプションは、ユーザー ID と実効ユーザー ID が root であるユーザーのみが使用できます。 その他のユーザーが `-d` オプションを使用すると、`login` は何も出力しないで終了します。

`-h` *hostname* [*terminal*] — `in.telnetd`(1M) がリモートホストと端末タイプに関する情報を渡すために使用します。

`-p` — ログインシェルに環境変数を渡すために使用されます。

`-r` *hostname* — `in.rlogind`(1M) がリモートホストに関する情報を渡すために使用します。

終了ステータス

次のような終了ステータスが返されます。

`0`	正常終了
`0` 以外	エラー

ファイル

`$HOME/.cshrc`	各 C シェルの初期設定コマンド
`$HOME/.hushlogin`	ログインメッセージの抑制
`$HOME/.login`	C シェルユーザー用のログインコマンド
`$HOME/.profile`	Bourne シェルユーザーと Korn シェルユーザー用のログインコマンド
`$HOME/.rhosts`	ホスト名とユーザー名の組み合わせのリスト
`/etc/.login`	システム全体のログインコマンド
`/etc/issue`	項目またはプロジェクトの識別情報
`/etc/logindevperm`	ログインベースのデバイスアクセス権
`/etc/motd`	本日のメッセージ
`/etc/nologin`	マシンのシャットダウン中にログインしようとするユーザーに対するメッセージ
`/etc/passwd`	パスワードファイル
`/etc/profile`	システム全体の Bourne シェルと Korn シェルのログインコマンド
`/etc/shadow`	ユーザーの暗号化パスワードのリスト
`/usr/bin/sh`	ユーザーのデフォルトのコマンドインタプリタ
`/var/adm/lastlog`	最終ログイン時刻
`/var/adm/loginlog`	失敗したログイン試行の記録
`/var/adm/utmp`	アカウンティング

/var/adm/wtmp	アカウンティング
/var/mail/*your-name*	ユーザー *your-name* 用のメールボックス
/etc/default/login	このファイル中にデフォルト値を設定 (例: TIMEZONE=EST5EDT)。次のフラグを使用

TIMEZONE
シェルの TZ 環境変数を設定 (environ(5) を参照)

HZ
シェルの HZ 環境変数を設定

ULIMIT
ログインのファイルサイズの制限を設定。 設定はディスクブロック単位で、デフォルトはゼロ (制限なし)

CONSOLE
設定されていれば、スーパーユーザーは そのデバイスにのみログイン可能。ただし、rsh(1) によるリモートコマンドの実行は可能。スーパーユーザーでログインできるようにするには、この行をコメントにする

PASSREQ
ログインに NULL 以外のパスワードが必要かどうかを指定

ALTSHELL
ログイン時に SHELL 環境変数を設定するかどうかを指定

PATH
初期シェルの PATH 変数を設定

SUPATH
スーパーユーザー用初期シェルの PATH 変数を設定

TIMEOUT
ログインセッションを終了するまでの待ち時間を秒単位 (0 から 900 の間) で設定

UMASK
初期シェルのファイルモード生成マスクを設定 (umask(1) を参照)

SYSLOG
LOG_NOTICE レベルでのスーパーユーザーによるログインすべてと LOG_CRIT レベルでの失敗したログイン試行を記録するのに syslog(3C) の LOG_AUTH を使うかどうかを指定

DISABLETIME
存在し、かつ、ゼロより大きい場合、ログイン試行が RETRIES 回だけ失敗した後、あるいは、PAM フレームワークが PAM_ABORT を戻した後に login が待つ秒数。デフォルトは 20 秒。最小は 0 秒。最大はない

SLEEPTIME
存在する場合、ログイン失敗メッセージを画面に表示するまでに待つ秒数を設定。これは PAM_ABORT 以外のログイン障害用。RETRIES に到達していない場合、あるいは、PAM フレームワークが PAM_MAXTRIES 回だけ戻されている場合、別のログイン試行が許可される。デフォルトは 4 秒。最小は 0 秒。最大は 5 秒

RETRIES
ログインを再試行する回数を設定 (pam(3PAM) を参照)。 デフォルトは 5

SYSLOG_FAILED_LOGINS
何回ログインに失敗したら、syslog(3C) LOG_NOTICE 機能によってログインに失敗したことを通知するメッセージが記録されるかを指定する。たとえば、変数の値が 0 に設定されている場合、login は失敗したログインをすべて記録する

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), exit(1), ksh(1), mail(1), mailx(1), newgrp(1), passwd(1), rlogin(1), rsh(1), sh(1), shell_builtins(1), telnet(1), umask(1), in.rlogind(1M), in.telnetd(1M), logins(1M), quota(1M), su(1M), syslogd(1M), useradd(1M), userdel(1M), pam(3PAM), rcmd(3SOCKET), syslog(3C), ttyname(3C), auth_attr(4), exec_attr(4), hosts.equiv(4), issue(4), logindevperm(4), loginlog(4), nologin(4), nsswitch.conf(4), pam.conf(4), passwd(4), profile(4), shadow(4), user_attr(4), utmpx(4), wtmpx(4), attributes(5), environ(5), pam_unix_account(5), pam_unix_auth(5), pam_unix_session(5), pam_authtok_check(5), pam_authtok_get(5), pam_authtok_store(5), pam_dhkeys(5), pam_passwd_auth(5), termio(7I)

診断

Login incorrect
ユーザー名またはパスワードが一致しません。

`Not on system console`
スーパーユーザーでのログインが拒否されました。 `/etc/default/login` の `CONSOLE` を確認してください。

`No directory! Logging in with home=/`
`passwd`(4) データベースに指定されているユーザーのホームディレクトリが 見つからないか、アクセス権が正しくありません。システム管理者に問い合わせてください。

`No shell`
`passwd`(4) データベースに指定されているシェルを実行できません。 システム管理者に問い合わせてください。

`NO LOGINS: System going down in` *N* `minutes`
マシンがシャットダウン中で、ログインできません。

警告

ユーザー ID が 76695844 よりも大きいユーザーには、パスワードの有効期限が適用されません。また、このユーザーの最終ログイン時刻も記録されません。 スーパーユーザーによるログインを不能にするために `CONSOLE` を使用する場合は、 スーパーユーザーによるリモートコマンドの実行も不能にする必要があります。 詳細については、`rsh`(1)、`rcmd`(3SOCKET)、 `hosts.equiv`(4) を参照してください。

注意事項

`pam_unix`(5) モジュールは、将来のリリースではサポートされなくなる可能性があります。同様の機能
は、`pam_unix_account`(5)、`pam_unix_auth`(5)、`pam_unix_session`(5)、`pam_authtok_check`(5)、`pam_authtok_get`(5)、`pam_authtok_store`(5)、`pam_dhkeys`(5)、および `pam_passwd_auth`(5) で提供されています。

名前 | logname – ユーザーのログイン名の応答

形式 | **logname**

機能説明 | logname ユーティリティは ユーザーのログイン名を標準出力に書き出します。 ログイン名は getlogin(3C) 関数によって返される文字列です。 getlogin() が失敗した場合、 logname は標準エラーに診断メッセージを書き出して、 ゼロ以外の終了ステータスで終了します。

環境 | logname の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　入力ファイルはすべて、正常に出力された

>0　エラーが発生した

ファイル | /etc/profile　ログイン時のユーザー環境

/var/adm/utmp　ユーザーとアカウンティングの情報

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | env(1), login(1), getlogin(3C), utmpx(4), attributes(5), environ(5)

名前 | logout – ログインのセッションから抜け出すためのシェル組み込み関数

csh | **`logout`**

csh | ログインシェルを終了します。

属性 | 次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | `csh`(1), `login`(1), `attributes`(5)

名前 | lp – 印刷要求の送信

形式 | **lp** [-c] [-m] [-p] [-s] [-w] [-d *destination*] [-f *form-name*] [-H *special-handling*] [-n *number*] [-o *option*] [-P *page-list*] [-q *priority-level*] [-S *character-set* | *print-wheel*] [-t *title*] [-T *content-type* [-r]] [-y *mode-list*] [*file*...]

lp -i *request-ID*... [-c] [-m] [-p] [-s] [-w] [-d *destination*] [-f *form-name*] [-H *special-handling*] [-n *number*] [-o *option*] [-P *page-list*] [-q *priority-level*] [-S *character-set* | *print-wheel*] [-t *title*] [-T *content-type* [-r]] [-y *mode-list*]

機能説明 | lp ユーティリティは印刷要求を宛先に送信します。 lp コマンドには 2 つの形式があります。

最初の形式の lp コマンドは、 ファイル (*file*) (複数可) とそれに関連した情報 (これらをまとめて *print request* と呼ぶ) を印刷します。 *file* 引数を省略した場合は、標準入力と見なされます。 標準入力を指定したい場合は、 *file* の中にハイフン (‘–’) を使用してください。ファイルは、 コマンド行に表示された順序で印刷されます。

2 番目の形式の lp は印刷要求のオプションを変更します。 この形式の lp は Solaris 2.6 オペーレーティング環境またはその互換バージョンの LP 印刷サーバー上に限り使用可能です。 *request-ID* で特定された印刷要求は、指定された印刷オプションによって変更されます。 指定可能なオプションは、 前述の第 1 形式のものと同一です。 指定した要求が すでに印刷処理を終了している場合、変更要求は無視されます。 現在印刷中の場合には、その時点で中断され、始めから (または -P オプションがあれば指定されたページから) 印刷し直します。

宛先の情報を決定するとき、印刷クライアントに関係するコマンドはネームサービススイッチ内にある printers データベースを使用します。詳細については、nsswitch.conf(4)、printers(4)、および printers.conf(4) のマニュアルページを参照してください。

オプション | SunOS 4.*x* または BSD ベースの印刷サーバーを持ったプリンタは、 BSD プロトコルの拡張機能を扱うように設定されていません。 lp はこのように違った宛先に送信する印刷要求を処理します (後述の「注意事項」を参照) 。

次のオプションを指定できます。

-c　印刷前に *file* で指定したファイルのコピーを生成します。

本オプションを省略する場合には、印刷要求の印刷処理が 完全に終了するまで指定した *file* を削除しないでください。 本オプションを省略した場合、印刷要求を送信したあと印刷開始前に *file* の内容を変更すると、変更後の内容が出力されます。 *file* はコピーされないで、リンクされるためです。

-d *destination*　指定した宛先 (*destination*) に file を印刷します。宛先の指定に -d オプションを使用するのは、ジョブが最初に作成されたときだけです。(注: 既存のジョブを異なる宛先に移動する方法については、lpmove(1M) のマニュアルページを参照)。名前、 POSIX スタ

イル名 (*server*:*destination*) 、またはフェデレーテッド・ネーミング・サービス (FNS) 名 (.../service/printer/...) を使用して、 *destination* を指定します。 名前や FNS 名の命名規約については printers.conf(4) を、 POSIX については standards(5) を参照してください。

`-f` *form-name*
: *form-name* 上に file を印刷します。 LP 印刷サービスは、プリンタ上にその用紙がセットされていること を確認します。指定したプリンタが *form-name* で指定した用紙を 扱えない場合、 *form-name* が示す用紙がシステムに対して未定義の場合、 当該ユーザーが *form-name* の使用を許可されていない場合 には、印刷要求は拒否されます (lpforms(1M) を参照)。

`-H` *special-handling*
: *special-handling* の値に従って印刷要求を印刷します。 *special-handling* には以下の値が指定できます。

  `hold`
  : 指示があるまで、印刷要求を印刷しません。すでに印刷が始まっている場合は、ただちに中断されます。 印刷要求が再開されるまで、 保持された要求 (印刷要求の保持) よりも他の要求が先に印刷されます。

  `resume`
  : 印刷要求の保持状態を解除します。 印刷実行中に保持された印刷要求であれば、 現在印刷中の印刷要求の次に再印刷されます。ただし、次に述べる immediate 印刷要求に追い越される場合があります。

  `immediate`
  : 当該要求を現在印刷中の印刷要求の次に印刷します。 複数の印刷要求が与えられると、最後に与えられたものから順番に印刷されます。ある印刷要求を早急に印刷したいが当該プリンタが 他の印刷要求を印刷中の場合には、その印刷中の印刷要求を 保持することにより緊急の印刷要求を先に出力することができます。 immediate 要求は LP の管理者だけが使用可能です。

`-i` *request-ID*
: *request-ID* で指定した印刷要求用のオプションを変更します。 -i と *request-ID* の間にはスペース文字が必要です。このオプションが適用されるのは、印刷サーバー上のローカルの待ち行列内にあるジョブだけです。

`-m`
: *file* を印刷した後でメールを送信します (mail(1) を参照)。デフォルトでは、 印刷要求が正常に終了した場合にはメールは送られません。

`-n` *number*
: 指定した部数の *file* を印刷します。 *number* は数字を指定してください。デフォルトの *number* は 1 です。

`-o` *option*

プリンタ固有のオプションを指定します。 `-o` *option* を複数回記述して (`-o` *option* `-o` *option* `-o` *option*) 、複数のオプションを指定します。 プリンタ固有のオプションは、 `-o` を 1 つだけ使用して、その後にオプション全体を二重引用符で囲んで (`-o "` *option option option*`"`) 指定することもできます。 以下のオプションが指定可能です。

`nobanner`
見出しを示すバナーページの印刷を省略します。 このオプションは LP 管理者によって、 使用を禁止することができます。

`nofilebreak`
複数のファイルを印刷する場合、 ファイルが終わるたびに用紙送りをしません。

`length=`*number*`i` | *number*`c` | *number*
印刷要求を出力するときのページの長さ (インチ、センチメートル、または行数) を指定します。インチの場合は文字 `i` を、センチメートルの場合は文字 `c` を *number* の後に付けます。 *number* だけを指定すると行数を表します。 たとえば `length=66` はページの長さが `66` 行であることを、 `length=11i` は `11` インチであることを、 `length=27.94c` は `27.94` センチであることを表します。

本オプションは `-f` と同時に指定することはできません。

`width=`*number*`i` | *number*`c` | *number*
印刷要求を出力するときのページの幅 (インチ、センチメートル、または行数) を指定します。インチの場合は文字 `i` を、センチメートルの場合は文字 `c` を *number* の後に付けます。たとえば `width=65` はページの幅が `65` カラムであることを、 `width=6.5i` は `6.5` インチであることを、 `width=10c` は `10` センチであることを表します。

本オプションは `-f` と同時に指定することはできません。

`lpi=`*number*
印刷要求を 1 インチ当たり *number* 行に設定した行間隔で印刷します。 *number* は 1 インチ当たりの行数を指定します。

本オプションは `-f` と同時に指定することはできません。

`cpi=`*n* | `pica` | `elite` | `compressed`
文字間隔に 1 インチ当たり *number* 文字を設定して印刷要求を印刷します。 *number* は 1 インチ当たりの文字数を指定します。 `pica` は文字間隔をパイカ (1 インチ当たり `10` 文字) に設定し、 また `elite` は文字間隔をエリート (1 インチ当たり `12` 文字) に設定します。 `compressed` は文字数を プリンタが印字可能な範囲でできるだけ多く設定します。 全プリンタに共通な文字間隔の標準値はありません。 個々のプリンタのデフォルト

値については Terminfo データベースを参照してください (`terminfo`(4) を参照)。 本オプションは `-f` と同時に指定することはできません。

`stty=`*stty-option-list*
`stty` コマンド用のオプションの並びで要求を印刷します (`stty`(1) を参照)。 空白文字が含まれている場合、全体を単一引用符 (`'`) で囲んでください。

`-P` *page-list*
page-list で指定したページを昇順で印刷します。 *page-list* は、ページ番号の範囲、 1 つのページ番号、 またはその両方の組み合わせを指定してください。

`-P` オプションは、これを扱えるフィルタが存在している場合にのみ指定できます。 フィルタがなければ印刷要求は受け付けられません。

`-p`
印刷の終了を知らせます。 通知方法は、ソフトウェアによって異なります。

`-q` *priority-level*
印刷待ち行列において優先順位を印刷要求に割り当てます。 *priority-level* は `0` から `39` までの整数で指定してください。 `0` は優先順位が最も高いことを表し、 `39` は優先順位が最も低いことを表します。 優先順位を省略すると、 LP 管理者が決めた印刷サービス用の デフォルトの優先順位値が用いられます。 LP 管理者はユーザーごとにデフォルトの優先順位も割り当てることができます。

`-s`
`lp` からのメッセージの出力を抑止します。

`-S` *character-set* | *print-wheel*
当該要求を、 *character-set* で示す文字セットまたは *print-wheel* で示す印字ホイールを使って印刷します。使用すべき用紙が、 `-S` オプションで指定したもの以外の文字セットまたは印字ホイールを必要とする場合には、印刷要求は受け付けられません。 セット可能な印字ホイールまたはフォントカートリッジを用いるプリンタは、 `-S` オプションが指定されていなければ、 印刷要求を同時に設定した印字ホイールまたはフォントカートリッジを 使用します。

- 印字ホイールを用いるプリンタ

  `print` *wheel* が、 LP 管理者がプリンタ用のものとして認めたホイールリストに含まれていない場合には、 すでにセットされていない限り、当該要求は受け付けられません。

- ユーザー指定の文字セットまたはプログラム可能な文字セットを 使用するプリンタ

  `-S` オプションが省略された場合、 `lp` は標準の文字セットを使用します。 *character-set* が、当該プリンタ用の `terminfo` データベース中に定義されていない場合や LP 管理者が定義した別名のいずれにも一致しない場合には、 当該要求は受け付けられ

ません。 terminfo に関しては terminfo(4) の説明を参照してください。

-t *title*	見出しを表すバナーページのタイトルを印刷します。 *title* が空白文字を含んでいる場合には、全体を引用符で囲んでください。 *title* を指定しない場合は、バナーページにファイル名が印刷されます。
-T *content-type* [-r]	指定した *content-type* をサポートするプリンタを選択します。 このタイプを直接サポートするプリンタが存在しない場合には、フィルタによりサポート可能なタイプに変換されます。ただし -r オプションを指定すると、フィルタは用いられません。したがって -r を指定したとき、 *content-type* を直接受け付けられるプリンタが存在しなければ、要求は拒否されます。 またフィルタを通しても、その変換後のタイプを受け付けられる プリンタが存在しなければ、やはり要求は拒否されます。
-w	全ファイルを印刷後、ユーザーの端末にメッセージを送信します。 ユーザーがログインしていなければ、メッセージの代わりにメールが送られます。
-y *mode-list*	当該要求を *mode-list* で示したモードで印刷します。 *mode-list* に指定可能な値はローカルに定義されています。 本オプションは、 それを扱えるフィルタが存在している場合に限り使用できます。 フィルタが存在しないと、要求は拒否されます。

オペランド

以下のオペランドを指定できます。

file	出力されるファイル名。 *file* はパス名、または標準入力を表すハイフン (-) を指定します。 *file* を省略した場合、 lp は標準入力を用います。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の lp の動作については、largefile(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
0 以外	エラーが発生した

ファイル

/var/spool/lp/*	LP 印刷待ち行列
$HOME/.printers	ユーザーが構成可能なプリンタデータベース
/etc/printers.conf	システムのプリンタ構成データベース
printers.conf.byname	/etc/printers.conf の NIS バージョン
printers.org_dir	/etc/printers.confの NIS+ バージョン

`fns.ctx_dir.`*domain*	`/etc/printers.conf` の FNS バージョン

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpcu
CSI	対応済み (「注意事項」参照)

関連項目

`cancel`(1), `enable`(1), `lpq`(1B), `lpr`(1B), `lprm`(1B), `lpstat`(1), `mail`(1), `postprint`(1), `pr`(1), `stty`(1), `accept`(1M), `lpadmin`(1M), `lpfilter`(1M), `lpforms`(1M), `lpmove`(1M), `lpsched`(1M), `lpshut`(1M), `lpsystem`(1M), `lpusers`(1M), `nsswitch.conf`(4), `printers`(4), `printers.conf`(4), `terminfo`(4), `attributes`(5), `environ`(5), `largefile`(5), `standards`(5)

注意事項

CSI 機能は、 プリンタ名が ASCII 文字で構成されていると仮定します。

印刷ジョブには 1 種類のデータだけが含まれるものとします。データの種類は、コマンド行で指定されるか、ジョブ内にある最初のファイルの内容に基づいて自動検出されます (単純なデータ、PostScript データなどと識別される)。

SunOS 4.*x* または BSD ベースの印刷サーバーを持つプリンタは、 BSD プロトコルの拡張機能を扱うように構成されません。 `lp` は、次の方法でこのようなプリンタに送られた印刷要求を扱います。

1. 印刷要求が 52 ファイル以上であっても、 52 ファイルで切り捨てます。 この場合、 `lp` は警告メッセージを表示します。
2. `-f` 、 `-H` 、 `-o` 、 `-P` 、 `-p` 、 `-q` 、 `-S` 、 `-T` 、 `-y` オプションには、 印刷サーバーへ引き渡すためにプロトコルの 拡張機能が必要になることがあります。 `lp` が印刷要求を処理できない場合、 警告メッセージが表示されます。

   LP 管理者は `/etc/printers.conf` 中にプリンタの `bsdaddr` エントリを設定して、 プロトコルの拡張機能をつけることができます。 `/etc/printers.conf` の `bsdaddr` エントリを次のように変更します。

   destination`:bsdaddr=`*server*`,`*destination*`,Solaris`すると、 Solaris 印刷サーバーによって処理可能な BSD 印刷プロトコルの拡張機能を有効にします。 この時点では、 `lp` は `Solaris` のプロトコルの拡張機能だけをサポートしています。

名前 | lpr – 印刷要求の提出

形式 | **/usr/ucb/lpr** [-P *destination*] [-# *number*] [-C *class*] [-J *job*] [-T *title*] [-i [*indent*]] [-1 | -2 | -3 | -4 *font*] [-w *cols*] [-m] [-h] [-s] [-*filter_option*] [*file...*]

機能説明 | lpr ユーティリティは印刷要求を宛先に提出して、ファイル (*file*) (複数可) とそれに関連した情報 (これらをまとめて印刷要求と呼ぶ) を印刷します。*file* 引数を省略した場合は、標準入力と見なされます。

宛先の情報を決定するとき、印刷クライアントに関係するコマンドはネームサービススイッチ内にある printers データベースを使用します。詳細については、nsswitch.conf(4)、printers(4)、および printers.conf(4) のマニュアルページを参照してください。

52 ファイルを超える印刷要求が指定された場合、ファイル数は 52 に切り捨てられます。この場合、lpr から警告メッセージが出力されます。

オプション | 次のオプションを指定できます。

-P *destination*
: 指定のプリンタまたはプリンタクラス (lpadmin(1M) 参照) に *file* を出力します。名前、POSIX スタイル名 (*server*:*destination*) 、 またはフェデレーテッド・ネーミング・サービス (FNS) 名 (.../service/printer/...) を使用して、 *destination* を指定します。 名前や FNS 名の命名規約については printers.conf(4) を、 POSIX については standards(5) を参照してください。

-# *number*
: 指定した部数のコピーを出力します。*number* は正の整数で指定してください。デフォルトは 1 です。

-C *class*
: バナーページに出力するジョブクラスとして、 *class* が示すジョブクラスを使用します。 *class* に空白が含まれる場合は 二重引用符で囲んでください。 *class* が指定されなかった場合、hostname によって返される システム名がジョブクラスとして出力されます。hostname(1) のマニュアルページを参照してください。

-J *job*
: バナーページに印刷するジョブ名として、*job* が示す名前を使用します。*job* に空白が含まれる場合は 二重引用符で囲んでください。*job* が指定されなかった場合、*file* (ファイルが複数である場合はコマンド行に指定された最初の *file*) がバナーページに印刷するジョブ名として出力されます。

-T *title*
: バナーページに印刷するタイトルとして、 *title* が示す名前を使用します。*title* に空白が含まれる場合は 二重引用符で囲んでください。*title* が指定されなかった場合、 *file* がバナーページに印刷するジョブ名として出力

されます。-p フィルタオプションを一緒に指定しない場合、-T オプションは無視されます。

-i *indent*
: 出力行のインデントの量として、 指定の数の空白文字を使用します。 *indent* はインデントする空白文字数を指定します。 *indent* は正の整数で指定してください。オプション引数 indent を指定しない場合、デフォルトでは 8 つの空白文字が使用されます。-p フィルタオプションを一緒に指定しない場合、-i オプションは無視されます。

–1|–2|–3|–4 *font*
: フォント位置 1、2、3、4 のマウントするフォントを指定します。 *font* は有効なフォント名を示します。

-w *cols*
: 指定のページ幅で *file* を出力します。 *cols* はカラム幅の数を示します。-p フィルタオプションを一緒に指定しない場合、-w オプションは無視されます。

-m
: *file* が出力されたあと、メールを送信します (mail(1) 参照)。デフォルトでは、 印刷要求が正常に完了した場合にはメールを送信しません。

-h
: バナーページの印刷を抑止します。

-s
: (シンボリックリンクではなく) *file* の完全パス名を使用します。 ファイルのコピーは作成されないため、印刷ジョブ送信後でも 出力が完了するまでは *file* を削除したり更新したりしないでください。 本オプションは、ローカルマシンから作成される ローカルファイルに対してのみコピーの作成を抑止します。 この -s オプションは、指定された *file* にのみ適用されます。つまり lpr コマンドがパイプラインの最後にある場合、*file* はスプールにコピーされます。

- *filter_option*
: プリンタのスプーラに 対して *file* が標準のテキストファイルではないことを知らせます。 スプール処理を行うデーモンは、それに従い、 適切なフィルタを使って *file* を出力します。

  filter_option は 1 文字で指定します。

  filter_option が指定されず、 プリンタが PostScript® を認識できる場合には、*file* の最初の 2 文字に %! を挿入して *file* を PostScript として解釈するようにします。

  次の filter_option が指定できます。

  p
  : pr を使ってファイルをフォーマットします。

  l
  : 制御文字を印刷し、ページブレイクを抑止します。

t	*file* が troff (写植用出力) バイナリデータを含んでいることを表します。
n	*file* が ditroff (デバイスに依存しない troff) のデータを含んでいることを表します。
d	*file* が tex (スタンフォードの DVI 形式) データを含んでいることを表します。
g	*file* が plot(1B) ルーチンが生成した標準プロットデータを含んでいることを表します。
v	*file* がラスタイメージを含んでいることを表します。 イメージを印刷するには、*printer* が PostScript のようなイメージモデルをサポートしている必要があります。
c	*file* が *cifplot* が生成したデータを含んでいることを表します。
f	各行の先頭文字を、標準 FORTRAN キャリッジ制御文字と解釈します。

オペランド

次のオペランドを指定できます。

file 出力されるファイル名。*file* にはパス名を指定します。*file* が指定されない場合には、 lpr は標準入力を使用します。

使用法

ファイルが 2G バイト (2^{31} バイト) 以上ある場合の lpr の動作については、largefile(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
0 以外	エラーが発生した

ファイル

/var/spool/print/.seq	ジョブ ID のシーケンス番号を含むファイル
/var/spool/print/[cd]f*	スプール処理用のディレクトリおよびファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWscplp
CSI	対応済み (「注意事項」参照)

関連項目 | hostname(1), lp(1), lpc(1B), lpq(1B), lprm(1B), lpstat(1), mail(1), plot(1B), pr(1), troff(1), lpadmin(1M), nsswitch.conf(4), printers(4), printers.conf(4), attributes(5), largefile(5), standards(5)

診断 | lpr: *destination* | : unknown destination
destination が LP 構成データベース内に見つかりません。 ただし、通常はコマンドの入力ミスと考えられ、宛先がシステム内に存在しないという場合があります。 lpstat -p を実行して印刷サービスの状況について表示してください。

注意事項 | lpr は *printer* 名を除いて CSI 対応が可能です。

印刷ジョブには 1 種類のデータだけが含まれるものとします。データの種類は、コマンド行で指定されるか、ジョブ内にある最初のファイルの内容に基づいて自動検出されます (単純なデータ、PostScript データなどと識別される) 。

名前 | lpstat – LP 印刷サービスの状態に関する情報の表示

形式 | **lpstat** [-d] [-r] [-R] [-s] [-t] [-a [*list*]] [-c [*list*]] [-f [*list*] [-l]] [-o [*list*]] [-p [*list*] [-D] [-l]] [-S [*list*] [-l]] [-u [*login-ID-list*]] [-v [*list*]]

機能説明 | lpstat ユーティリティは、 LP 印刷サービスの現在の状態に関する情報を標準出力に表示します。

オプションをすべて省略すると、 lp により生成された (lp(1) の項を参照) 当該ユーザーのすべての印刷要求の状況が報告されます。 オプションでない引数はすべて要求 ID (lp により返されるもの) と見なされ、 lpstat はそれらの要求の状況を報告します。 複数のオプションを指定する場合、その記述順序は問いません。 また同一オプションを複数回指定することも、 他の引数と混在させることも可能です。 オプションの一部は、複数の項目を記述した *list* 指定を伴うことができます。この *list* の形式には 2 通りあり、1 つは各項目をコンマで区切る方法、 もう 1 つは全体を引用符で囲んで各項目を空白で区切る方法です。 以下に例を示します。

```
example% lpstat -u "user1 user2 user3"
```

なお項目リストの代わりに all と指定すると、全項目指定と見なされ、そのオプションに関連した すべての情報が出力されます。以下に例を示します。

```
example% lpstat -o all
```

上記コマンドは、すべての出力要求に関する状態情報を表示します。

なおこのような *list* を伴いうるオプションの後に引数を何も指定しない場合も、 やはりすべての情報が出力されます。以下に例を示します。

```
example% lpstat -o
```

上記コマンドも、すべての出力要求に関する状態情報を表示します。

宛先の情報を決定するとき、印刷クライアントに関係するコマンドはネームサービススイッチ内にある printers データベースを使用します。詳細については、nsswitch.conf(4)、printers(4)、および printers.conf(4) のマニュアルページを参照してください。

オプション | 以下のオプションがすべてのプラットフォームで指定できます。

-d　出力要求時のシステムデフォルトの宛先を表示します。

-o [*list*]　出力要求の状態を表示します。 *list* はプリンタ名、クラス名、または要求 ID (混在可) を列挙したものです。オプション文字 -o は省略可能です。 名前、 POSIX スタイル名 (*server*:*destination*) 、 またはフェデレーテッド・ネーミング・サービス (FNS) 名 (.../service/printer/...) を使用して、 プリンタやクラスの

名前を指定します。 名前や FNS 名の命名規約については `printers.conf`(4) を、 POSIX については `standards`(5) を参照してください。

`-r` LP 要求スケジューラの状態を表示します。

`-R` 印刷待ち行列における要求ごとの位置を表す数値を表示します。

`-s` 要約された状態情報を表示します。含まれる内容は LP スケジューラの状況、システムデフォルト宛先、 プリンタとそれに対応した装置の一覧、 印刷サービスを共有しているマシンの一覧、 現在マウントされている形式の一覧、 および認識されている文字セットとプリントホイールの一覧です。

`-t` 全情報を表示します。具体的な内容は、 `-s` オプションによって得られる情報に、全プリンタの受付状況 およびアイドル/ビジー状態を示す情報が付加されたものです。

`-u` [*login-ID-list*] ユーザー用の出力要求の状態を表示します。 *login-ID-list* 引数には、以下に示す項目をいくつでも指定できます。

login-ID	任意のシステム上に存在するあるユーザーを対象とします。
system_name`!`*login-ID*	*system_name* で示したシステム上に存在するあるユーザーを対象とします。
system_name`!all`	*system_name* で示したシステム上に存在する全ユーザーを対象とします。
`all!`*login-ID*	全システム上に存在するあるユーザーを対象とします。
`all`	全システム上に存在する全ユーザーを対象とします。

`-v` [*list*] プリンタの名前とともに、その各々に対応しているデバイスのパス名 もしくはネットワークプリンタのシステム名を表示します。 *list* はプリンタ名を列挙したものです。

次のオプションは Solaris 2.6 オペーレーティング環境またはその互換バージョンの LP プリントサーバーから指定された場合にのみ、 正しい結果を返します。

`-a` [*list*] 印刷の宛先において要求を受け付けているか否かを報告します。 *list* はプリンタ名またはクラス名 (混在可) の並びです。

`-c` [*list*] 全クラスとそのメンバーの名前を表示します。 *list* はクラス名を列挙したものです。

`-f` [*list*] [`-l`] *list* で指定した形式を LP 印刷サービスが認識できるかを確認して報告します。 *list* は形式の列挙で、デフォルトは `all` です。 `-l` オプションは形式に関する記述を出力します。

`-p` [*list*] [`-D`] [`-l`] プリンタの状態を表示します。 *list* はプリンタ名を列挙したものです。 `-D` オプションは、指定された各プリンタについての簡単な記述を出力します。 `-l` オプションを指定すると、ローカルのプリンタに関しては コンフィギュレーションについての詳細な情報が返されます。 この詳細情報にはマウントされている形式、受付可能な内容とプリンタの タイプ、プリンタの説明、使用されているインタフェースが含まれます。

`-S` [*list*] [`-l`] *list* に指定された文字セットまたはプリントホイールが LP 印刷サービスにより認識可能か否かをチェックして報告します。 *list* には文字セットまたはプリントホイールを指定できます。デフォルトは `all` です。 `-l` オプションを指定すると、各行の終わりに プリントホイールあるいは文字セットを扱えるプリンタの一覧が 追加情報として出力されます。またこの追加情報には、 当該文字セットまたはプリントホイールがマウントされているか否か、 およびその文字セットに対応する内蔵文字セットが表示されます。

`-d` 出力要求時のシステムデフォルトの宛先を表示します。

`-o` [*list*] 出力要求の状態を表示します。 *list* はプリンタ名、クラス名、または要求 ID (混在可) を列挙したものです。オプション文字 `-o` は省略可能です。

`-r` LP 要求スケジューラの状態を表示します。

`-R` 印刷待ち行列における要求ごとの位置を表す数値を表示します。

`-s` 要約された状態情報を表示します。含まれる内容は LP スケジューラの状況、システムデフォルト宛先、 プリンタとそれに対応した装置の一覧、 印刷サービスを共有しているマシンの一覧、 現在マウントされている形式の一覧、 および認識されている文字セットとプリントホイールの一覧です。

`-t` 全情報を表示します。具体的な内容は、 `-s` オプションによって得られる情報に、全プリンタの受付状況 およびアイドル/ビジー状態を示す情報が付加されたものです。

`-u` [*login-ID-list*] ユーザー用の出力要求の状態を表示します。 *login-ID-list* 引数には、以下に示す項目をいくつでも指定できます。

login-ID	任意のシステム上に存在するあるユーザーを対象とします。
system_name`!`*login-ID*	*system_name* で示したシステム上に存在するあるユーザーを対象とします。
system_name`!all`	*system_name* で示したシステム上に存在する全ユーザーを対象とします。

	`all!`*login-ID*	全システム上に存在するあるユーザーを対象とします。
	`all`	全システム上に存在する全ユーザーを対象とします。
`-v` [*list*]		プリンタの名前とともに、その各々に対応しているデバイスのパス名 もしくはネットワークプリンタのシステム名を表示します。*list* はプリンタ名を列挙したものです。

終了ステータス

以下の終了ステータスが返されます。

`0`	正常終了
`0` 以外	エラーが発生した

ファイル

`/var/spool/print/*`	LP 印刷待ち行列
`$HOME/.printers`	ユーザーが構成可能なプリンタデータベース
`/etc/printers.conf`	システム構成データベース
`printers.conf.byname`	`/etc/printers.conf` の NIS バージョン
`printers.org_dir`	`/etc/printers.conf` の NIS+ バージョン
`fns.ctx_dir.`*domain*	`/etc/printers.conf` の FNS バージョン

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpcu

関連項目

`cancel`(1), `lp`(1), `lpq`(1B), `lpr`(1B), `lprm`(1B), `nsswitch.conf`(4), `printers`(4), `printers.conf`(4), `attributes`(5), `standards`(5)

名前 | ls – ディレクトリの内容を一覧表示

形式 | **/usr/bin/ls** [-aAbcCdfFgilLmnopqrRstux1@] [*file*...]

/usr/xpg4/bin/ls [-aAbcCdfFgilLmnopqrRstux1@] [*file*...]

機能説明 | *file* オペランドがディレクトリの場合、ls は、そのディレクトリの内容を出力します。*file* が通常ファイルの場合、ls は、そのファイル名と要求された他の情報を出力します。デフォルトでは、アルファベット順にソートして出力します。引数をまったく指定しないと、現在のディレクトリの内容を出力します。引数を複数指定すると、引数は最初に適切にソートされますが、ファイル引数がディレクトリとその内容より先に処理されます。

出力形式には、主に 3 種類あります。端末に対するデフォルトの出力形式は、複数カラムで、ソートしたエントリを縦方向に並べて表示します。-1 オプションを指定すると、単一のカラムで出力します。-m オプションを指定すると、ストリーム出力形式で表示します。-C、 -x、 -m オプションの出力形式を決定するために、ls は環境変数 COLUMNS を使用して、1 行に出力できる文字数を調べます。この環境変数が設定されていない場合は、環境変数 TERM に基づいて、terminfo(4) データベースが使用され、カラム数が決定されます。この情報が得られない場合は、カラム数は 80 カラムと見なされます。

-l オプションで出力されるモードは 10 文字からなります。最初の文字は次のいずれかです。

d　エントリがディレクトリの場合

D　エントリが door の場合

l　エントリがシンボリックリンクの場合

b　エントリがブロック型特殊ファイルの場合

c　エントリが文字型特殊ファイルの場合

p　エントリが FIFO (「名前付きパイプ」と呼ばれる) 特殊ファイルの場合

s　エントリが AF_UNIX アドレスファミリのソケットの場合

–　エントリが通常ファイルの場合

残りの 9 文字は、3 文字ずつの 3 組に分けられます。最初の組は、所有者へのアクセス権を示します。次の組は、同一のグループに属する他のユーザーへのアクセス権を示します。最後の組は、その他のユーザーへのアクセス権を示します。各組においては、3 文字がそれぞれ、ファイルの読み取り、書き込み、実行権を示します。ディレクトリの場合は、「実行」権は、指定したファイルの有無についてディレクトリを検索するアクセス権を意味します。アクセス権に続く文字は ACL について示しています。ファイルに ACL が関連づけられている場合はプラス記号が表示されます。アクセス権だけの場合は 何も表示されません。

ls -l (ロング形式出力) は、POSIX ロケールで次のように出力します。

```
-rwxrwxrwx+  1 smith  dev    10876  May 16 9:42 part2
```

右から左へ見ていくと、現在のディレクトリには part2 というファイルが 1 つあるのがわかります。次に、ファイルの内容を最後に修正したのは、5 月 16 日午前 9 時 42 分です。ファイルには、10,876 文字 (バイト) が含まれています。ファイルの所有者またはユーザーは、グループ dev (おそらく development を示す) に属しており、ログイン名は smith です。この例で 1 と示されている数字は、part2 というファイルへのリンクの数を示します (cp(1) を参照)。プラス記号は、ファイルに関連する ACL があることを示します。注: -@ オプションが指定されている場合、拡張属性が存在していると ACL は無効になり、プラス記号はアット記号 (@) に置き換えられます。さらにダッシュと文字によって、ユーザー、グループ、その他のユーザーが part2 に対する読み取り、書き込み、実行権を持っていることが示されています。

実行権を表す x は、各組の 3 番目の位置に置かれます。3 番目の位置に – がある場合は、実行権を与えられていないことを示します。

各アクセス権の意味は、次のとおりです。

r	ファイルが読み取り可能
w	ファイルが書き込み可能
x	ファイルが実行可能
–	指定されたアクセス権は与えられていない
s	セットユーザー ID ビットまたは セットグループ ID ビットがオンで、対応するユーザーまたはグループ実行権ビットもオン
S	不定ビット状態 (セットユーザー ID ビットがオンで、ユーザー実行権ビットがオフ)
t	1000 (8 進) ビット、つまりスティッキビットがオン (chmod(1) を参照) で、実行権ビットがオン
T	1000 ビットがオンで、実行権ビットがオフ (不定ビット状態)
/usr/bin/ls l	アクセス中に強制ロックが発生する (セットグループ ID ビットがオンで、グループ実行権ビットがオフ)
/usr/xpg4/bin/ls L	アクセス中に強制ロックが発生する (セットグループ ID ビットがオンで、グループ実行権ビットがオフ)

ユーザーやグループへのアクセス権では、3 番目の位置に x や – ではなく、セットユーザー ID ビットあるいはセットグループ ID ビットの状態に応じて s が置かれることがあります。実行中に、ユーザーのユーザー ID をファイルのユーザー ID と同じと見なすこの機能は、たとえば、スーパーユーザーとして起動したユーザーが、ログイン中に普通にログインした場合のユーザー ID を必要とするときなどに使用されます。

グループへのアクセス権においては、3 番目の位置に l が置かれることがあります。l は、強制的なファイルおよびレコードのロックを示しています。これは アクセス中にその他のファイルに対する読み取り または書き込み権をロックする ファイルの機能があることを示しています。

その他のユーザーへのアクセス権においては、3 番目の位置に t または T が置かれることがあります。これらは、スティッキビットと実行権の状態を示しています。

オプション　以下のオプションを指定できます。

`-a`　すべてのエントリを出力します。つまり、ドット (.) で始まるエントリも出力します。

`-A`　すべてのエントリを出力します。つまり、ドット (.) で始まるエントリも出力します。ただし、作業用ディレクトリ (.) や親ディレクトリ (..) は出力しません。

`-b`　出力できない文字を 8 進 *\ddd* の出力形式で強制的に出力します。

`-c`　(`-t` オプションによる) ソート や (`-l` または `-n` オプションによる) 出力において、i ノードを最後に修正した時刻 (ファイル作成、モード変更など) を使用します。

`-C`　複数カラム形式の出力で、エントリを縦方向にソートします。これは、デフォルトの出力形式です。

`-d`　引数がディレクトリの場合、(内容ではなく) その名前だけを出力します。`-l` とともに使用すれば、ディレクトリの状態を知ることができます。

`-f`　強制的に、各引数をディレクトリと解釈し、各引数のディレクトリで見つかる名前を出力します。このオプションは、`-l` 、 `-t` 、 `-s` 、 `-r` を無視し、 `-a` が指定されたものと見なします。エントリは、ディレクトリ内での順序で出力されます。

`-F`　ディレクトリの各内容の末尾に種別を表す記号をつけます。ディレクトリの場合はスラッシュ (/) を、door の場合は大なり括弧 (>) を、実行可能なファイルの場合はアスタリスク (*) を、FIFO の場合は縦棒 (|) を、シンボリックリンクの場合は単価記号 (@) を、`AF_UNIX` アドレスファミリのソケットの場合は等号 (=) を、それぞれ末尾に付けます。

`-g`　所有者名が出力されない点を除いて、`-l` と同じです。

`-h`　すべてのサイズを縮小して、読みやすい形式で出力します。たとえば、`14K`、`234M`、`2.7G`、`3.0T` などのようになります。縮尺は、`1024` を除数として行われます。

`-i`　各ファイルについて、i ノード番号を第 1 カラムに出力します。

`-l`　各ファイルについて、モード、ACL 表示、リンクの数、所有者名、グループ名、サイズ (バイト単位) 、最終修正時刻をロング形式で出力します。ファイルが特殊ファイルの場合、サイズフィールドには、メジャーデバイス番号とマイナーデバイス番号が入ります。最終修正時刻が 6 箇月よりも前である場合には、POSIX ロケールでは「月・日・年」の形式で表示されます。`LC_TIME` ロケールカテゴリが POSIX ロケールに設定されていない場合には、異なった形式で時間フィールドが表示されます。最終修正時刻が 6 箇月以内である場合には、「月・日・時」の形式で表示されます。ファイルがシンボリックリンクの場合は、ファイル名を出力したあとに、→ に続いて参照されるファイルのパス名が出力されます。

`-L` 引数がシンボリックリンクの場合、リンク自身ではなく、そのリンクが参照するファイルまたはディレクトリの情報を出力します。

`-m` ストリーム形式出力を指定します。ファイルはコンマで区切られ横方向に出力されます。

`-n` 所有者名とグループ名の代わりに、所有者のユーザー ID とグループのグループ ID が出力されることを除き、`-l` と同じです。

`-o` グループ名が出力されない点を除き、`-l` と同じです。

`-p` ファイルがディレクトリの場合、各ファイル名の後にスラッシュ (/) を付けます。

`-q` ファイル名中の出力できない文字を疑問符記号 (?) として出力します。

`-r` 他のオプションの指定に応じて、アルファベットの逆順にまたはファイルが古い順にソートします。

`-R` サブディレクトリの内容を再帰的に出力します。

`-s` 各エントリについて、間接ブロックを含むブロック数を出力します。

`-t` ファイル名ではなく、タイムスタンプ (新しい順) でソートします。デフォルトは最終修正時刻です (`-u` と `-c` を参照)。

`-u` (`-t` オプションによる) ソート または (`-l` オプションによる) 出力において、最終修正時刻の代わりに、最終アクセス時刻を使用します。

`-@` `-l` と似ていますが、拡張属性が存在していると ACL は無効になります。拡張属性を持つファイルのアクセス権ビットの後ろにはアット記号 (@) が表示されます。

`-x` 複数カラム形式出力で、エントリを縦方向ではなく横方向にソートします。

`-1` 1 行の出力につき 1 つのエントリを出力します。

/usr/bin/ls オプションのうち `-C` と `-l` (小文字のエル)、`-m` と `-l` (小文字のエル)、 `-x` と `-l` (小文字のエル)、 `-@` と `-l` (小文字のエル) は、同時に指定できません。このような矛盾する組み合わせでオプションを指定してもエラーにはならず、`-l` 指定が有効となります。

/usr/xpg4/bin/ls オプションのうち `-C` と `-l` (小文字のエル)、 `-m` と `-l` (小文字のエル)、 `-x` と `-l` (小文字のエル)、`-@` と `-l` (小文字のエル) は、同時に指定できません。このような矛盾する組み合わせでオプションを指定してもエラーにはならず、あとの方で指定されたものが有効になります。

オペランド 以下のオペランドを指定できます。

file 処理の対象とするファイルのパス名。ここで指定したファイルが見つからない場合、標準エラー出力に診断メッセージが書き出されます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `ls` の動作については、`largefile`(5) を参照してください。

使用例

例 **1** ファイルのアクセス権

```
-rwxr- -r- -
```

これは、ファイルがユーザーに対しては読み取り、書き込み、実行可能で、グループおよびその他のユーザーに対しては読み取り可能であることを示しています。

```
-rwsr-xr-x
```

これは、ファイルがユーザーに対しては読み取り、書き込み、実行可能で、グループおよびその他のユーザーに対しては読み取りおよび実行可能であり、実行時にユーザーがこのファイルのユーザー ID を自分のユーザー ID として使用できることを示しています。

```
-rw-rwl- - -
```

これは、ファイルがユーザーおよびグループによってだけ読み取りおよび書き込み可能で、アクセス中にロックされることを示しています。

例 **2** 全ファイルの名前を出力

以下のコマンドは、現在のディレクトリ中にあるすべてのファイルの名前を出力します。通常は名前が出力されない、先頭にドット . が付いているファイルの名前も出力します。

```
example% ls -a
```

例 **3** ファイルの情報を出力する

```
example% ls -aisn
```

このコマンドは、ドットで始まるものを含む、すべてのファイルについて (`a`) 、`i` 番号 — ファイルに対応する i ノードのメモリーアドレス — をカラムの左側に (`i`)、ファイルの サイズ (ブロック単位) を i 番号の右のカラムに (`s`) 出力します。さらに、コマンドによる出力形式は、ファイルに関連する UID 番号 (ユーザー名の代わり) と GID 番号 (グループ名の代わり) が出力される ロング出力形式の数値バージョン (`n`) になります。

ディレクトリ内のファイルのサイズを出力する際、間接ブロックを含むブロックの総数を出力します。

環境

`ls` の実行に影響を与える環境変数 `LC_COLLATE`、`LC_CTYPE`、`LC_TIME`、`LC_MESSAGES`、`NLSPATH`、`TZ` についての詳細は、`environ`(5) を参照してください。

`COLUMNS` 複数のテキストカラムからなる出力を生成する場合に、ユーザーが希望するカラム幅の値を指定します。この変数の値が 10 進整数を示す文字列の場合、`ls` は書き出すパス名テキストカラムの数

を、その幅の値を元に 計算します (-C の説明を参照)。COLUMNS が設定されていない、または値が不適当な場合には、カラム幅として 80 が用いられます。どのディレクトリのファイル名を出力する場合でも、選択されたカラム幅は一定です。複数テキストカラムを出力する際、ファイル名は長すぎても 切り捨てられません。

終了ステータス

以下の終了ステータスが返されます。

0　　情報はすべて正常に書き出された

>0　　エラーが発生した

ファイル

/etc/group
  ls -l および ls -g 用のグループ ID

/etc/passwd
  ls -l および ls -o 用のユーザー ID

/usr/share/lib/terminfo/?/*
  端末情報データベース

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/ls

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み
インタフェースの安定性	安定

/usr/xpg4/bin/ls

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み
インタフェースの安定性	標準

関連項目

chmod(1), cp(1), setfacl(1), terminfo(4), attributes(5), environ(5), fsattr(5), largefile(5), XPG4(5)

注意事項

ファイル名に出力できない文字があると、カラムの出力オプションが正しく処理されないことがあります。

ハードリンクされたファイルがある場合、総ブロック数は正しく表示されません。

ls の出力のソート順序はロケールの影響を受け、LC_COLLATE 環境変数で変更できます。たとえば、LC_COLLATE が C の場合、ファイルを表示する順序は、ファイル名がドット (.) で始まるファイル、次にファイル名が大文字で始まるファイル、その後にファイル名が小文字で始まるファイルという順番になります。一方、LC_COLLATE が en_US.ISO8859-1 の場合には、ソート順序を決定する際に、先行するドットや大文字と小文字の区別は無視されます。

名前 | mail, rmail – メールの読み取りまたはユーザーへのメールの送信

メールの送信 | **mail** [-tw] [-m *message_type*] *recipient...*

rmail [-tw] [-m *message_type*] *recipient...*

メールの読み取り | **mail** [-ehpPqr] [-f *file*]

デバッグ | **mail** [-x *debug_level*] [*other_mail_options*] *recipient...*

機能説明 | *recipient* は通常、ドメイン形式 (“*user@machine*”) のアドレス、または`login`(1) が認識するユーザー名です。*recipient* を指定すると、`mail` はメッセージを送信中であるとみなします。このコマンドは、標準入力からファイルの終端 (Control-d) まで、あるいは端末デバイスから読み取っている場合は、ピリオドだけの行に至るまで読み取りを続けます。ファイルの終端またはピリオドだけの行のいずれかを受信すると、`mail` は各 *recipient* に対してレターをメールファイルに追加します。

レターは、最初にヘッダー行が数行、次に空行、さらに本文が続きます。レターのヘッダー行セクションは、1 つまたは複数の UNIX ポストマークから構成されます。

`From` *sender date_and_time* [`remote from` *remote_system_name*]

この後に、以下の形式の 1 つまたは複数の標準メッセージヘッダー行が続きます。

keyword-name`:` [*printable text*]

ここで、*keyword-name* は、 コロン (':') 以外の表示可能な空白以外の文字から構成されます。`Content-Length:` ヘッダー行は本文内のバイト数を示し、常に存在します。ただし、レターがメッセージ内容のないヘッダー行だけから構成されている場合は例外です。`Content-Type:` ヘッダー行は本文 (テキスト、バイナリ、マルチパートなど) のタイプを示し、常に存在します。ただし、レターがメッセージ内容のないヘッダー行だけから構成されている場合は例外です。ヘッダー行は、その次の行が空白で始まっている場合、次の行に継続できます。

メールの送信 | 以下のコマンド行引数は、メール送信に影響を与えます。

`-m` *message_type*	`Message-Type:` 行を、*message_type* の値とともにメッセージヘッダーに追加します。
`-t`	対象となる *recipient* の各々について、`To:` 行を、メッセージヘッダーに追加します。
`-w`	リモート送信プログラムの完了を待たずに、レターをリモート受信者に送信します。

レターが配送できないことがわかった場合、そのレターは障害の原因と性質を示す診断とともに送信者に返されます。入力中に `mail` が中断された場合、メッセージを `dead.letter` というファイルにセーブするので、後から編集と再送信を実行することもできます。`dead.letter` は常に追加されるので、以前の内容はすべて保存され

ます。dead.letter の追加 (または作成) は、まず、現在のディレクトリで行われます。これが失敗すると、ユーザーのログインディレクトリに dead.letter を追加 (または作成) しようとします。2 度目の試行も失敗した場合は、dead.letter の処理はいっさい行われません。

rmail は、メールの送信だけ可能です。uucp(1C) は rmail をセキュリティ通知として使用します。メールメッセージを生成するアプリケーションプログラムは、メッセージの送信や配送、またはその両方において、必ず、mailではなく rmail を起動する必要があります。

ローカルシステムに Basic Networking Utilities をインストールしてある場合、ローカルシステムで利用可能な送信機構に応じて、様々な方法でリモートシステム上の受信者にメールを送信できます。最も広く利用されているアドレス指定機構は、ドメイン形式とUUCP 形式の 2 つです。

ドメイン形式のアドレス指定
受信者名に '@' とドメイン (およびサブドメインもあり得る) 情報を追加してリモート受信者を指定します (たとえば、user@sf.att.com)。(ローカルシステム上でどの規則が利用可能かについての詳細は、ローカルシステム管理者に相談する必要があります)。

UUCP 形式のアドレス指定
sysa!user のように、受信者名にリモートシステム名と感嘆符を接頭辞として付けてリモート受信者を指定します。csh(1) がデフォルトシェルの場合、sysa\!user を使用する必要があります。一連のシステム名を感嘆符で区切れば、拡張したネットワークを介してレターを出力することができます (たとえば、sysa!sysb!sysc!user または sysa\!sysb\!sysc\!user)。

メールの読み取り

以下のコマンド行引数は、メールの読み取りに影響を与えます。

-e　メールを表示しません。ユーザーにメールがある場合、0 の終了コードを返します。メールがない場合、1 の終了コードを返します。

-h　最初に、最新のメッセージではなく、ヘッダーのウィンドウを表示します。ディスプレイの後に ? プロンプトを表示します。

-p　配置を示すプロンプトなしに、すべてのメッセージを表示します。

-P　デフォルトの選択したヘッダー行を表示する代わりに、ヘッダー行すべてを表示してから、すべてのメッセージを表示します。

-q　割り込みを受けた後、mail は終了します。通常、割り込みによって発生するのは、表示中のメッセージの終了だけです。

-r　先入れ先出し方式でメッセージを表示します。

-f *file*　デフォルトのメールファイルの代わりに、mail は、*file* (たとえば、mbox) を使用します。

mail は、コマンド行引数の影響を特に受けないかぎり、後入れ先出し方式でユーザーのメールメッセージを表示します。メッセージ表示のデフォルトのモードは、至急必要なヘッダー行だけを表示することです。これらには、UNIX の From ポストマークおよび >From ポストマーク、From: ヘッダー行、Date: ヘッダー行、Subject: ヘッダー行、および Content-Length: ヘッダー行、ならびに To:、Cc:、Bcc: などの受信者ヘッダー行が含まれますが、これに限定されるわけではありません。ヘッダー行を表示した後、mail は、メッセージに表示できない文字がないかぎり、その内容 (本体) を表示します。表示できない文字がある場合、mail は、メッセージの内容がバイナリである旨の警告文を発行し、その内容を表示しません (これは、p コマンドで無効にすることができます。以下を参照)。

各メッセージにおいて、ユーザーは ? というプロンプトを受け、標準入力から行が読み取られます。メッセージの配置を判別するときは、以下のコマンドを利用できます。

#	現在のメッセージの数を表示します。
–	直前のメッセージを表示します。
<復帰改行>、+、または n	次のメッセージを表示します。
!*command*	シェルにエスケープして、*command* を実行します。
a	mail セッション中に到着したメッセージを表示します。
d または dp	現在のメッセージを削除し、次のメッセージを表示します。
d *n*	*n* 番のメッセージを削除します。次のメッセージを表示しません。
dq	メッセージを削除し、mail を終了します。
h	現在のメッセージを中心としてヘッダーのウィンドウを表示します。
h *n*	*n* 番のメッセージを中心としてヘッダーのウィンドウを表示します。
h a	ユーザーのメールファイルにすべてのメッセージのヘッダーを表示します。
h d	削除予定のメッセージのヘッダーを表示します。
m [*persons*]	指定された *persons* に現在のメッセージを送信 (および削除) します。
n	*n* 番のメッセージを表示します。
p	現在のメッセージを再び表示し、バイナリ (つまり、表示不能) の内容の表示を無効にします。
P	デフォルトの省略モードを無効にし、現在のメッセージを再表示して、ヘッダー行をすべて表示します。

q または CTRL-D	削除を解除したメールをメールファイルに戻し、mail を終了します。
r [*users*]	送信者と他の *users* に応答してから、メッセージを削除します。
s [*files*]	メッセージを、指定された *files* (デフォルトは mbox) にメッセージの内容をヘッダー行なしでセーブし、メッセージを削除します。
u [*n*]	*n* 番のメッセージの削除を解除します(デフォルトは直前に読み取ったメッセージ)。
w [*files*]	指定された *files* (デフォルトは mbox) にメッセージの内容をヘッダー行なしでセーブし、メッセージを削除します。
x	メールすべてを変更を加えずにメールファイルに戻し、mail を終了します。
y [*files*]	-w オプションと同じ。
?	コマンド概要を表示します。

通常、ユーザーがログインする際にメールが存在すれば、それが通知されます。また、mail を使用中に新しいメールが到着した場合でも通知されます。

chmod(1) を使用し、2 種類の方法でメールファイルのアクセス権を処理すれば、mail の機能を変更できます。ファイルのその他のアクセス権を、読み取り書き込みともに可能 (0666)、読み取り専用 (0664)、または読み取り書き込みとも不可 (0660) とすることによって、プライバシーをさまざまに調整できます。デフォルト (0660モード) 以外に変更すると、空の場合でもファイルは保存され、希望のアクセス権が永久的になります (管理者は mailcnfg の DEL_EMPTY_MAILFILE オプションを使用すれば、このファイル保存を無効にできます)。

メールファイルのグループ ID を mail として新しいメッセージを配送可能にし、メールファイルをグループ mail で書き込み可能にする必要があります。

デバッグ

以下のコマンド行引数によって、mail は、デバッグ情報を提供します。

-x *debug_level*	mail は、デバッグ情報の入ったトレースファイルを作成します。

-x オプションを指定すると、mail は /tmp/MLDBG*process_id* という名前のファイルを作成します。このファイルには、mail が現在のメッセージを処理した手順に関するデバッグ情報が入っています。*debug_level* の絶対値はデバッグ情報の長さを制御します。0 はデバッグなしを意味します。*debug_level* が 0 より大きい場合、デバッグファイルが保持されるのは、mail のメッセージ処理中に問題が発生した場合だけです。*debug_level* が 0 未満の場合、常にデバッグファイルを保持します。-x で *debug_level* を指定すると、/etc/mail/mailcnfg 内の DEBUG の指定すべてが無効になります。-x オプションの提供する情報は難解であり、役立つのはシステム管理者だけだと思われます。

配送通知

メールの通知には、いくつかの形式があります。以下の行をメッセージヘッダーに挿入することによって実現します。

`Transport-Options:` [*/options*]

`Default-Options:` [*/options*]

`>To:` *recipient* [*/options*]

ここで "*/options*" には、以下のうちの 1 つまたは複数が使用できます。

`/delivery`	メッセージが *recipient* のメールボックスに正常に配送されたことを送信者に通知します。
`/nodelivery`	配送が成功したことを送信者に通知しません。
`/ignore`	配送が失敗したことを送信者に通知しません。
`/return`	メール配送が失敗したか否かを送信者に通知します。送信者に失敗メッセージを返します。
`/report`	`/return` と同じ。ただし、元のメッセージは返しません。

デフォルトは `/nodelivery/return` です。矛盾のあるオプションを使用すると、最初のオプションを認識し、その後の矛盾する条件を無視します。

オペランド

メール送信には次のオペランドがあります。

recipient	ドメイン形式 ("*user@machine*") のアドレス、または`login`(1) が認識するユーザー名

使用法

ファイルが 2G バイト (2^{31}バイト) 以上ある場合の `mail` と `rmail` の動作については、`largefile`(5) を参照してください。

環境

`mail` の実行に影響を与える環境変数 `LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、`environ`(5) を参照してください。

`TZ`	日付と時間の文字列とともに使用する時間帯を指定する

終了ステータス

以下の終了ステータスが返されます。

`0`	そのユーザー宛のメールがあり、処理は正常に終了した
`1`	ユーザー宛のメールがなかった、または初期化時にエラーが発生した
`>1`	初期化の後でエラーが発生した

ファイル

`dead.letter`	メールできなかったテキスト
`/etc/passwd`	送信者の識別および *recipient* の発見用
`$HOME/mbox`	セーブされたメール
`$MAIL`	メールファイルのパス名を含む変数
`/tmp/ma*`	一時ファイル

/tmp/MLDBG*	デバッグ・トレースファイル
/var/mail/*.lock	メールディレクトリのロック
/var/mail/:saved	一時ファイルを保持し、システムクラッシュ時のデータ損失を防ぐためのディレクトリ
/var/mail/*user*	ユーザーに送られたメール (ユーザーに届いたメール)。すなわち、デフォルトのメールファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

chmod(1), csh(1), login(1), mailx(1), uucp(1C), uuencode(1C), vacation(1), write(1), attributes(5), environ(5), largefile(5)

『*OpenWindows* ユーザーズガイド *(*上級編*)*』

注意事項

上記の「配送通知」の節で説明したように、ヘッダー行による解釈と結果的な処置が発生するのは、配送 (または失敗) が行われたシステム上に、このバージョンの mail がインストールされている場合だけです。mail の初期のバージョンはどのタイプの配送通知もサポートしません。

条件によっては、ロックファイルの削除が失敗することがあります。

割り込み後、次のメッセージが表示されないことがあります。表示を強制するときは、p を入力します。

名前

mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式

mailp [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明

`mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション

次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D` 出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。

`-F` メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。

`-h` バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。

`-l` 横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。

`-P` *printer* `-d` オプションを指定した場合と同じです。

`-s` *subject* *subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename 読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`1` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

名前 | man – マニュアルページの表示

形式 | **man** [-] [-adFlrt] [-M *path*] [-T *macro-package*] [-s *section*] *name...*

man [-M *path*] -k *keyword...*

man [-M *path*] -f *file...*

機能説明 | man コマンドはマニュアルページから得た情報を表示します。具体的には、*name* 引数が示すタイトルを持つマニュアルページの全内容を出力します。また -k オプションまたは -f オプションが指定された場合には、それぞれ *keyword* または *filename* が示すキーワードやファイル名により選択された 1 行からなる要約情報を 出力します。指定条件に一致するマニュアルページが見つからない場合には、man はエラーメッセージを出力します。

ソース形式 | 参照用マニュアルページには、nroff (nroff(1) を参照) または SGML (Standard Generalized Markup Language) (sgml(5) を参照) のどちらかのタグがマークアップされています。man コマンドはそのマークアップの種類を認識して、適宜にファイルを処理します。マークアップの種類によって分けられたディレクトリに、個々のソースファイルが置かれています。

マニュアルページの位置 | オンラインの参照用マニュアルページのディレクトリは、通常 /usr/share/man ディレクトリにあります。nroff のソースは、/usr/share/man/man* ディレクトリにあります。また、SGML のソースは、/usr/share/man/sman* ディレクトリにあります。各ディレクトリはマニュアルのセクションごとに対応しています。これらのディレクトリは必須ではないため、ホスト上にインストール されているとは限りません。存在していなければ、他のホストから /usr/share/man をマウントしてください。

プレフォーマットされた最新バージョンが cat* または fmt* ディレクトリ中にあれば、man は単にそれを表示または印刷します。バージョンが古かったり存在しない場合には、man は再フォーマット処理を行なってから出力し、cat* または fmt* が書き込み可の場合にプレフォーマットされたバージョンを 格納します。windex データベースは変更されません (catman(1M) を参照)。プレフォーマットされたバージョン用のディレクトリが提供されていない場合、man は要求に応じてページを再フォーマットし、表示処理中、フォーマットされたテキストは一時ファイルに収納されます。

標準出力が端末ではない場合、または '-' フラグが指定された場合には、man は cat(1) を通じてテキストを出力します。その他の場合、man はページごとの出力や下線付けを画面上で行うために more(1) を通じて出力します。

オプション | 以下にオプションを示します。

-a　MANPATH 検索パス中で *name* に一致したすべてのマニュアルページを出力します。複数個あるときは、見つかった順序で出力します。

-d　デバッグモード。セクションを示す引数に対応する値、検索に用いた方法、および man が検索を行なったパスを表示します。

-f *file. . .*	指定された *file* のいずれかに関連するマニュアルページを探しだします。各 *file* から先頭のパス名部分を取り除き、そのベース名または 名前を含む 1 行の要約情報を出力します。本オプションも windex データベースを使用します。
-F	windex 照合データベースを使う代わりに、MANPATH で指定した全ディレクトリまたは man.cf ファイルを使って検索を行うよう man に指示します。このオプションは、データベースの内容が最新状態にない場合に便利な指定で、man コマンドのデフォルトの動作に戻ります。そのため、通常はこのオプションを起動する必要はありません。参考のためにここに記述しています。
-k *keyword . . .*	windex データベース (目次) を参照し、指定された *keyword* を含んでいるものを選び、その概要情報を出力します。windex データベースは catman(1M) コマンドを使って生成します。
-l	検索パス内にある、*name* に一致するマニュアルページをすべて一覧表示します。
-M *path*	マニュアルページ用に別の検索パスを指定します。*path* には、マニュアルページのディレクトリ・サブツリーを含んだディレクトリの 名前をコロンで区切って記述します。たとえば *path* の値として /usr/share/man:/usr/local/man と指定すると、man は最初に標準のディレクトリである /usr/share/man を検索し、次に /usr/local/man を検索して *name* を探します。なお -M 本オプションを -k または -f と同時に指定する場合、-M の方を始めに記述しなければなりません。*path* 中の各ディレクトリ中には、各セクションにつき 1 つの man* または sman* という名のサブディレクトリが存在しているとみなされます。本オプションで指定した値は、MANPATH 環境変数の値より優先されます。
-r	マニュアルページを再フォーマットするだけで、結果の表示は行いません。本指定は man -t *name* 指定と同等です。
-s *section . . .*	検索を行うセクションを指定します。*section* 引数に対応するディレクトリだけを検索して *name* を探します。*section* の値は 1 桁の数字で、検索対象のマニュアルページのセクション名を伴う場合もあります (たとえば、“3libucb”)。または、1 つの語の場合もあります (たとえば local、new、old、public)。また、*section* は 1 文字の場合もあります。複数のセクションを指定するにはコンマで区切って記述します。本オプションで指定した値は、MANPATH 環境変数や man.cf ファイルの値より優先されます。man がマニュアルページを検索する方法に関しては、後述の「検索パス」の項を参照してください。
-t	指定したマニュアルページが、適当なラスタ出力デバイス用に troff されるようにします (troff(1) を参照)。- と -t の両オプションがともに指定された場合、man は指定された各 *name* の troff バージョンを (必要であれば) 更新しますが、結果の出力は行いません。

`-T` *macro-package*　マニュアルページをフォーマットする際、デフォルトである `/usr/share/lib/tmac/an` に定義されている `-man` マクロの代わりに、*macro-package* が示すパッケージを使用します。デフォルトの検索パスの順序に関しては、後述の「使用法」の「検索パス」の項を参照してください。

オペランド

以下のオペランドを指定できます。

name　標準ユーティリティのキーワード、またはその名前

マニュアルページのセクション

マニュアルページはセクションという単位に分類されています。各セクションの名前は、主セクション名 (通常は 1 桁の数字) と、サブセクション名 (通常は 1 つ以上の文字) からなります (サブセクション名の指定は任意です)。主セクション名 (例: `9`) を、そのセクションのサブセクション名 (例: `9e`、`9f`、`9s` など) の省略名として使用することはできません。サブセクションを検索する場合は、`man -s` で個別に指定する必要があります。各セクションに属するマニュアルページ群は、共通のカテゴリに属しており、その中でさらに細かく分類するためにサブセクションがあります。本リリースにおける分類に関しては、`intro` のマニュアルページを参照してください。

検索パス

指定されたある *name* を検索する前に、`man` はまず候補となりうるディレクトリとセクションのリストを構築します。`man` は環境変数 `MANPATH` が示すディレクトリ群の中で *name* を探します。この環境変数が設定されていなければ、デフォルトとして `/usr/share/man` を検索します。

マニュアルページのディレクトリ内において、`man` は以下に示すセクションだけを以下の順序で検索します。

- コマンド行上で `-s` オプションにより指定されたセクション群
- `MANPATH` 環境変数に埋め込まれているセクション群
- `MANPATH` 環境変数が示す各ディレクトリの `man.cf` ファイル中に指定されているセクション群

上記の指定がいずれも存在しない場合には、`man` はマニュアルページ・パス中の各ディレクトリを検索し、最初に見つかったマニュアルページだけを表示します。

`man.cf` ファイルの形式は次のとおりです。

`MANSECTS=`*section*[`,`*section*] . . .

文字 '`#`' で始まる行および空行は、注釈とみなし無視します。`MANPATH` が示す各ディレクトリには、そのディレクトリの検索順序の デフォルト値を定義するマニュアルページ・コンフィギュレーション ファイルを含めることができます。

マニュアルページのフォーマット

マニュアルページは `nroff`(1) または `sgml`(5) でマークアップされています。nroff のマニュアルページは、`-man` マクロパッケージにより用意されている `nroff`(1) または `troff`(1) によって処理されます。マクロ使用の詳細については `man`(5) を参照してください。SGML タグがついたマニュアルページは、SGML パーサーによって処理され、フォーマッタに引き渡されます。

nroff マニュアルページの前処理

nroff のマニュアルページをフォーマットする際、man は先頭行を参照して特殊な処理が必要かどうかを確かめます。先頭行が

```
' \" X
```

という形式 (*X* と " の間は 1 つの空白文字) で、*X* が以下に示す文字の組合せの場合、man は入力テキストを対応するプリプロセッサを介して troff(1) または nroff(1) に送ります。

e	eqn(1) または、nroff の場合、neqn
r	refer(1)
t	tbl(1)
v	vgrind(1)

eqn および neqn は、呼び出されると自動的に /usr/pub/eqnchar ファイル (eqnchar(5) を参照) を読み込みます。nroff(1) が呼び出されると col(1) が自動的に使用されます。

他の **nroff** マニュアルページへの参照

nroff のマニュアルページの先頭行が、次のパターンで他のマニュアルページを参照している場合、man はその参照されているマニュアルページの方を処理します。

```
.so man*/ sourcefile
```

この参照記述は、マニュアルページのディレクトリのサブツリーのルートに対して相対的なパス名で指定してください。

第 2 行目以降の行が .so で始まっていると man はそれを無視し、troff(1) および nroff(1) は通常の方法でリクエストを処理します。

SGML マニュアルページの処理

マニュアルページはファイル中に <!DOCTYPE という文字列が存在することによって、SGML であると特定されます。ファイルに SHADOW_PAGE という文字列が含まれている場合、そのファイルは別のマニュアルページの内容を参照します。参照はテキストが含まれているマニュアルページに対する ファイル実体の参照です。これは nroff フォーマットのマニュアルページで使用される .so のメカニズムに類似しています。

環境

man の実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES 、NLSPATH についての詳細は、environ(5) を参照してください。

MANPATH　各々をコロンで区切ったディレクトリ名のリストを表します。各ディレクトリ名の後に、各々をコンマで区切ったセクションのリストを 付加することもできます。この環境変数が設定されていると、その値は /usr/share/man (ディレクトリ検索パスのデフォルト) および man.cf (セクション検索パスのデフォルト) よりも優先して用いられます。さらにこの環境変数の値よりも -M および -s オプションの値 (指定されていれば) が優先されます。

PAGER	man の出力を対話モードで画面に送るプログラム名を表します。設定されていないと、プログラムとして 'more -s' が使用されます。more(1) を参照してください。
TCAT	troff で処理されたマニュアルページを表示するプログラム名を表します。
TROFF	-t オプションが指定された場合に使用するフォーマッタ名を表します。設定されていないと、フォーマッタとして troff(1) が使用されます。

終了ステータス

以下の終了ステータスが返されます。

0	入力ファイルはすべて、正常に出力された
>0	エラーが発生した

ファイル

/usr/share/man
  標準マニュアルページのディレクトリ・サブツリーのルート

/usr/share/man/man?/*
  フォーマットされていない nroff のマニュアルのエントリ

/usr/share/man/sman?/*
  フォーマットされていない SGML のマニュアルのエントリ

/usr/share/man/cat?/*
  nroff 処理後のマニュアルのエントリ

/usr/share/man/fmt?/*
  troff 処理後のマニュアルのエントリ

/usr/share/man/windex
  目次とキーワードのデータベース

/usr/share/lib/tmac/an
  標準 –man マクロパッケージ

/usr/share/lib/sgml/locale/C/dtd/*
  SGML ドキュメント型定義ファイル

/usr/share/lib/sgml/locale/C/solbook/*
  SGML のスタイルシートと実体ファイルの定義ディレクトリ

/usr/share/lib/pub/eqnchar
  eqn と neqn の標準定義

man.cf
  セクションの検索順序のデフォルト

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値

使用条件	SUNWdoc
CSI	対応済み

関連項目

apropos(1), cat(1), col(1), eqn(1), more(1), nroff(1), refer(1), tbl(1), troff(1), vgrind(1), whatis(1), catman(1M), attributes(5), environ(5), eqnchar(5), man(5), sgml(5)

注意事項

-f と -k の両オプションは、catman(1M) が生成する windex データベースを使用します。man コマンドは CSI 対応が可能です。ただし、man コマンドによって呼び出されるいくつかのユーティリティ (troff 、eqn 、neqn 、refer 、tbl 、vgrind) は、CSI 対応が実証されていません。このため、-t オプションをつけた man コマンドは、EUC 以外のデータを扱えません。また、eqn 、neqn 、refer 、tbl 、vgrind を通した特殊処理が必要な マニュアルページを表示する man コマンド使用は、CSI 対応ができません。

使用上の留意点

マニュアルページは本来 ASCII 端末や写真植字機上で再出力可能な形式でなければなりません。しかし端末上では、フォントの変更など一部の情報が失われてしまいます。

一部のダム端末では、e 前処理フラグ (eqn(1) を参照) により生成される縦方向の動作を処理できません。このような端末で出力エラーを防ぐには、e フラグを指定するときに t も同時に指定して、暗黙的に col(1) を呼び出してください。ただしこの回避処置をとると、スーパースクリプト (上付き文字) とサブスクリプト (下付き文字) の出力が、たとえ端末がその出力機能を持っていても 不可能になってしまうという欠点があります。端末が eqn(1) の出力により正常に動作しなくなってしまったら、Control-q を使って端末をクリアしてください。

名前 | mesg – メッセージの許可または禁止

形式 | **mesg** [-n | -y | n | y]

機能説明 | mesg ユーティリティは、他のユーザーが write(1) や talk(1) または他のユーティリティを介して 端末装置へメッセージを送信することを 許可すべきかどうかについて制御します。 対象となる端末装置は、標準入力、標準出力、標準エラー出力を順番に検索して最初に見つかったものとなります。 引数をすべて省略すると、 mesg は、現在の状態の報告だけを行います。状態は変更されません。 なお正当な特権を持っているプロセスは、現在の状態に関わらず いつでも端末へメッセージを送信できます。

オプション | 以下のオプションを指定できます。

-n |n　他のユーザーに対し、端末へのメッセージ送信を禁止します。詳細は write(1) を参照してください。

-y|y　他のユーザーに対し、端末へのメッセージ送信を許可します。

環境 | mesg の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　メッセージは受信可能

1　メッセージは受信不可能

2　エラーが発生した

ファイル | /dev/tty*　端末装置

/dev/pts/*　端末装置

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | talk(1), write(1), attributes(5), environ(5)

名前 | mkdir – ディレクトリの作成

形式 | **mkdir** [-m *mode*] [-p] *dir...*

機能説明 | mkdir コマンドは、 777 モード （ファイル生成マスク umask(1) で変更可能） で、指定されたディレクトリを作成します。

ディレクトリ内の標準エントリ (たとえば、そのディレクトリ自身を表す “.” ファイルや、その親ディレクトリを表す “. .” ファイル) は自動的に作成されます。 mkdir は、これらのエントリに名前を指定することはできません。 ディレクトリを作成するためには、 親ディレクトリの書き込み権が必要です。

新しいディレクトリのオーナ ID とグループ ID はそれぞれ、プロセスの実効ユーザー ID と実効グループ ID に設定されます。 mkdir は mkdir(2) システムコールを呼び出します。

setgid と **mkdir** | 新たに作成されたディレクトリの setgid ビットを変更するには、 mkdir を実行後に、 chmod g+s または chmod g-s を使用する必要があります。

setgid ビットの値は、親ディレクトリのものが受け継がれます。

オプション | 以下のオプションを指定できます。

-m *mode*
: 新しいディレクトリにモードを指定します。 モードの選択項目は chmod(1) で表示できます。

-p
: 存在しない親ディレクトリをすべて作成してから、 *dir* を作成します。 中間ディレクトリに与えられるモードの値は、 777 とファイル生成マスクに設定されたビットの値の差です。 ただしこの差は、 少なくとも 300 (ユーザーの書き込み権と実行権) でなければなりません。

オペランド | 以下のオペランドを指定できます。

dir
: 生成されるディレクトリのパス名。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の mkdir の動作については、largefile(5) を参照してください。

使用例 | **例 1** mkdir の使用例

```
example% mkdir -p ltr/jd/jan
```

上記の例は、 ltr/jd/jan というサブディレクトリ構造を作成します。

環境 | mkdir の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0
: 指定されたディレクトリはすべて正常に作成された、 または (-p オプション指定時) 指定された各ディレクトリは存在していたか もしくは正常に作成された。

>0　　エラーが発生した

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目　rm(1), sh(1), umask(1), intro(2), mkdir(2), attributes(5), environ(5), largefile(5)

名前 | mkmsgs – gettxt でアクセスできるメッセージファイルの作成

形式 | **mkmsgs** [-o] [-i *locale*] *inputstrings msgfile*

機能説明 | mkmsgs ユーティリティは、テキスト検索ツール (gettxt(1) 、 srchtxt(1) 、 exstr(1) 、 gettxt(3C) を参照) を使ってアクセスできる、 テキスト文字列からなるファイルを生成します。 入力の対象は、特定のロケール (setlocale(3C) を参照) 用のテキスト文字列からなるファイルです。 出力先は、 gettxt(1) でも gettxt(3C) でも検索できる形式のテキスト文字列からなるファイルです。 -i オプションを使えば、出力ファイルを /usr/lib/locale/locale/LC_MESSAGES ディレクトリに登録することもできます。なお locale は、テキスト文字列の言語に対応したロケールです。

inputstrings 引数には、もとになるテキスト文字列を 含んだファイル名を指定します。 *msgfile* 引数には、 mkmsgs が gettxt(1) や gettxt(3C) で検索可能な形式の文字列を書き出す出力ファイル名を指定します。この *msgfile* 名は、長さが 14 文字以内であること、 \0 (NULL) を含んでいないこと、 スラッシュ (/) またはコロン (:) に該当する ASCII コードを含んでいないことが必要です。

入力ファイルの内容は、特定のロケールに対応した テキスト文字列がいくつか集まったものです。 文字列と文字列とは復帰改行文字で区切られます。 表示不可能な文字は、エスケープシーケンスとして アルファベットで表す必要があります。 メッセージは *inputstrings* から順番に読み込まれ、変換され、 *msgfile* に書き出されます。空のメッセージを *msgfile* 中に生成する場合は、 *inputstrings* 中の適切な位置に空の行を入れておいてください。

文字列の内容を変更する場合は、単に *inputstrings* ファイルを編集してください。新たな文字列は、 ファイルの最後尾にだけ追加できます。 追加を行なったら、新たに *msgfile* を作成して正しい場所に登録しておく必要があります。 この手順を誤ると、検索機能使用時に誤った文字列が 検索されて、ソフトウェアの互換性が損なわれます。

オプション | 以下のオプションを指定できます。

-o	*msgfile* がすでに存在していれば上書きします。
-i *locale*	*msgfile* を /usr/lib/locale/*locale*/LC_MESSAGES ディレクトリ中に登録します。 このディレクトリ中でファイルを生成したり上書きしたりできるのは、 スーパーユーザー、およびグループ bin のメンバーだけです。 /usr/lib/*locale* の下のディレクトリは、存在していなければ生成されます。

使用例 | **例 1** mkmsgs コマンドの使用例

C.str という名の入力メッセージソースファイルの例です。

```
File %s:\t cannot be opened\n
%s: Bad directory\n
    .
    .
    .
write error\n
```

例 **1** mkmsgs コマンドの使用例　(続き)

```
.
.
```

例 **2** ファイルにテキスト文字列を作成するために C.str から入力文字列を使用する例

次のコマンドを実行すると、 C.str から文字列が読み取られ、適切な形式の文字列に変換され、 現在のディレクトリ中の UX というファイルに書き込まれます。

```
example% mkmsgs C.str UX
```

例 **3** ファイルにテキスト文字列を作成するために FR.str から入力文字列を使用する例

次のコマンドは、 FR.str からテキスト文字列を読み取り、適切な形式に変換し、/usr/lib/locale/fr/LC_MESSAGES というディレクトリ中の UX というファイルに書き込みます。

```
example% mkmsgs -i fr FR.str UX
```

このコマンドにより生成されたテキスト文字列は、 環境変数として LC_MESSAGES=fr が設定されていれば、「機能説明」の冒頭で述べた テキスト検索ツールのいずれかを呼び出すことにより アクセスできます。

ファイル

/usr/lib/locale/locale/LC_MESSAGES/*
  mkmsgs が生成するメッセージファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWloc

関連項目

exstr(1), gettxt(1), srchtxt(1), gettxt(3C), setlocale(3C), attributes(5)

名前 | more, page – テキストファイルの表示またはページング

形式 | **/usr/bin/more** [-cdflrsuw] [-lines] [+ *linenumber*] [+/ *pattern*] [*file*...]

/usr/bin/page [-cdflrsuw] [-lines] [+ *linenumber*] [+/ *pattern*] [*file*...]

/usr/xpg4/bin/more [-cdeisu] [-n *number*] [-p *command*] [-t *tagstring*] [*file*...]

/usr/xpg4/bin/more [-cdeisu] [-n *number*] [+ *command*] [-t *tagstring*] [*file*...]

機能説明 | more ユーティリティは、端末の画面上にテキストファイルの内容を 1 画面ずつ表示するフィルタです。 通常、画面が一杯になると休止します。そのとき /usr/bin/more は --More-- を、 /usr/xpg4/bin/more は *file* を、いずれも画面の最下段に表示します。 more がパイプではなくファイルから読み取る場合、 それまでに表示された文字のパーセンテージも併せて表示します。

more ユーティリティは、 RETURN 文字が入力されると、もう 1 行スクロールアップします。 SPACE 文字が入力されると、次の画面を表示します。 その他のコマンドは、以下で説明します。

page ユーティリティは、テキストの次の画面を 表示する前に画面をクリアします。 page は、 more とは異なり、 前画面の最後の行を現画面の最初に重複して表示することはしません。

more ユーティリティは、端末を NOECHO モードに設定し、連続して出力できるようにします。 通常、入力するコマンドは端末上に表示されません。 ただし、 / コマンドと ! コマンドの場合を除きます。

/usr/bin/more ユーティリティは、指定された最後のファイルを 表示して終了します。 /usr/xpg4/bin/more は、指定された最後のファイルの最終行で、コマンド入力を要求する プロンプトを発行します。

標準出力が端末の場合、 more は cat(1) とまったく同じように動作しますが、 複数のファイルが指定された場合、ファイルの前にヘッダが表示されます。

オプション | /usr/bin/more と /usr/xpg4/bin/more の両方で以下のオプションが使用できます。

-c　表示前に画面をクリアします。 より高速に表示するため、画面をスクロールせずに描画し直します。 端末が行の終端までクリアする機能を持たない場合、 このオプションは無視されます。

-d　認識できないコマンドが使用された場合、 端末ベルを鳴らさずにエラーメッセージを表示します。 これは経験の少ないユーザーに役立ちます。

-s　圧縮。 複数の空行を単一の空行と置換します。 これは、 nroff(1) を画面表示させる場合に役立ちます。

/usr/bin/more | 以下のオプションは /usr/bin/more でのみ指定できます。

	`-f`	長い行を折り返しません。 `nroff`(1) 出力を `ul`(1) によりパイプ連結したとき生成されるような非表示文字、またはエスケープシーケンスが行にあるときに役立ちます。
	`-l`	FORMFEED 文字 (CTRL-L) をページの中断とみなしません。 `-l` を使用しない場合、 `more` は休止して、 `^L` 文字 (CTRL-L) の入った行の後のコマンドを受け入れます。 また、ファイルが FORMFEED で始まっている場合、ファイルを表示する前に画面をクリアします。
	`-r`	通常、 `more` は解釈できない制御文字を無視します。 `-r` オプションにより、これらの文字は `^C` として表示されます。
	`-u`	下線引きエスケープシーケンスを生成しません。通常、 `more` は、 `nroff`(1) が生成するような下線引きを、端末に適切な方法で処理します。 端末が下線引きを実行できるか、 あるいは強調モードを備えている場合、 `more` はテキストファイルで呼び出されたものに対応するエスケープシーケンスを提供します。
	`-w`	通常、 `more` はその出力の終端まで来ると終了します。 しかし、 `-w` を用いると、 `more` は終了する前にプロンプトを出し、キーが押されるのを待ちます。
	-lines	デフォルト (端末画面内の行の数から 2 を引いた数) の代わりに、指定された数の *lines* を行数分画面に表示します。
	+linenumber	*linenumber* から表示します。
	+/pattern	*pattern* という正規表現の入った行より上の 2 行から表示します。 注意 : エディタと異なり、この形式は '/' で終了してはいけません。 スラッシュで終了すると、その後のスラッシュは検索パターンの文字と見なされます。
/usr/xpg4/bin/more	以下のオプションは `/usr/xpg4/bin/more` でのみ指定できます。	
	`-e`	引数リストの最後のファイルの最終行を出力すると、 ただちに終了します。
	`-i`	検索時に、大文字と小文字を区別せずにパターンマッチングを行います。
	`-n` *number*	1 画面当たりの行数を指定します。 *number* 引数の値は正の整数です。 `-n` オプションの値は、環境から得られる値に優先して用いられます。
	`-p` *command*	
	+command	検査対象の各ファイルに対し、最初に *command* 引数中の `more` コマンドを実行します。それが行番号や正規表現検索など 位置決め用のコマンドであれば、コマンドの最終結果を表すように現在の位置を設定します。ファイルの中間行は書き出しません。例として次の 2 つのコマンドを見てください。

more -p 1000j file
more -p 1000G fileこれらは、現在の位置を行番号 1000 として表示を開始する、という同じ動作をします。しかも、 `j` がファイルの検査中に呼び出されていれば書き出して画面から 消したであろうと思われる行は飛ばします。 位置決めコマンドが正常終了でなければ、ファイルの先頭行が 現在の位置となります。

`-t` *tagstring*　*tagstring* 引数で指定したタグを持つファイルの内容を 1 画面分表示します。詳細については `ctags`(1) ユーティリティの説明を参照してください。

`-u`　バックスペース文字を印刷可能文字として扱い、 ^H (CTRL-H) と表示します。このとき、ある種の端末では下線付きまたは強調モードテキスト で出力するような特殊な処理や、下線を付ける処理は行いません。 さらにこのオプションが指定されると、 行の終わりのキャリッジリターンを無視しません。

`-t` *tagstring* と `-p` *command* (または旧式の +*command*) オプションの両方が指定された場合、 `-t` *tagstring* が先に処理されます。

環境　`more` は、端末の `terminfo`(4) エントリを使用して、そのディスプレイ特性を判別します。 `more` は `MORE` という環境変数に設定済みオプションがないか調べます。 たとえば、デフォルトで `-c` モードを使用してファイルをページングするときは、 この環境変数の値を `-c` に設定します (通常、この環境変数を設定するコマンドシーケンスは `.login` ファイルまたは `.profile` ファイルに格納します) 。

コマンド　コマンドはただちに有効になります。 コマンドが *file* 、 *command*、 *tagstring* 、 *pattern* のいずれかを必要としない限り、 キャリッジリターンを入力する必要はありません。 コマンド文字自体を指定しないかぎり、行抹消文字を入力すれば、 入力中の数値引数を取り消すことができます。 さらに、消去文字を入力すれば、 '`--More--` (*xx*%)' または *file* メッセージを再表示できます。

以下のコマンドにおいて、 *i* は数値引数です (デフォルトでは `1`) 。

*i*SPACE　次の画面を表示します。 *i* を指定すると、 *i* 行追加表示します。

*i*RETURN　次の行を表示します。 *i* を指定した場合は、 *i* 行追加表示します。

i`b`

i`^B`　(Control-b) 画面を *i* 個、逆にスキップして次の画面を表示します。

i`d`

i`^D`　(Control-d) 画面を半画面分または *i* 行分、順方向にスクロールします。 *i* が指定されていれば、その値が以降の `d` および `u` コマンド用のデフォルトとなります。

i`f`　画面を *i* 個スキップして次の画面を表示します。

`h`　ヘルプ。 `more` のすべてのコマンドの説明を表示します。

^L	(Control-l) 画面の再表示。
*i*n	直前に入力した *pattern* の *i* 番目の一致を検索します。
q	
Q	more を終了します。
*i*s	*i* 行スキップしてから 1 画面分を表示します。
v	現在のファイルの現在行で、 vi エディタに入ります。
*i*z	SPACE と同じ。 ただし *i* を指定した場合、 その値が画面当たりの行数の新しいデフォルト値になります。
=	現在の行番号を表示します。
i/*pattern*	*pattern* という正規表現の *i* 番目の一致を順方向に検索します。 *pattern* という正規表現の *i* 番目の一致またはパイプの終端、どちらか先に見つかった方を含む行の前 2 行から画面を表示します。 more がファイルを表示中であり、一致がない場合、 そのファイルにおける位置は変更されません。 正規表現は、消去文字と抹消文字を使用して編集することができます。 第 1 カラムを越えて消去すると、検索コマンドが取り消されます。
!*command*	シェルを起動し、 *command* を実行します。 % および ! という文字を command 内で使用すると、それぞれ、 現在のファイル名および直前のシェルコマンドに置換されます。 現在のファイル名がない場合、 % は展開されません。 これらの文字の前にバックスラッシュを追加して、 展開をエスケープしてください。
:f	現在のファイル名と行番号を表示します。
i:n	コマンド行に指定された *i* 番目後のファイル名、または *i* が範囲外の場合はリスト内の最後のファイル名までスキップします。
i:p	コマンド行に指定された *i* 番目前のファイル名、または *i* が範囲外の場合はリスト内の最初のファイル名までスキップします。 ファイル内に more を位置決めしている間に指定すると、ファイルの最初に移動します。 more がパイプから読み取り中の場合、 more は単に端末ベルを鳴らすだけです。
:q	
:Q	more を終了します (q または Qと同じ) 。

/usr/bin/more 以下のコマンドは /usr/bin/more でのみ使用できます。

'	単一引用符。 直前の検索が開始された点に移動します。 現在のファイルで検索を実行していない場合、 ファイルの最初に移動します。
.	ドット。 直前のコマンドを繰り返します。
^ \	テキストの一部表示を停止します。 more は出力の送信を停止し、通常の --More-- プロンプトを表示します。 一部の出力は結果的に失われること

があります。

/usr/xpg4/bin/more

以下のコマンドは /usr/xpg4/bin/more でのみ使用できます。

i^F	(Control-f) 画面を *i* 個スキップして次の 1 画面分を表示します (*i*f と同じ) 。
^G	(Control-g) 現在の行番号を表示します (= と同じ) 。
*i*g	ファイル中の行番号 *i* に進みます。デフォルトは先頭行です。
*i*G	ファイル中の行番号 *i* に進みます。デフォルトは最終行です。
*i*j	次の行を表示します。 *i* が指定されていればその行数分を表示します (*i*RETURN と同じ) 。
*i*k	画面を逆方向にスクロールします。 *i* 指定時はその行数、省略時は 1 行です。
m*letter*	*letter* で示す名前で現在の位置をマークします。
N	逆方向に検索します。
r	画面を再表示します。
R	画面を再表示し、バッファ内に入力があればそれを破棄します。
*i*u	
i^U	(Control-u) 画面を逆方向にスクロールします。 *i* 指定時はその行数、省略時は半画面分です。 *i* が指定されていれば、その値が以降の d および u コマンド用のデフォルトとなります。
ZZ	more を終了します (q と同じ) 。
:e *file*	新たなファイルを検査 (表示) します。 file を省略すると、現在のファイルが再表示されます。
:t *tagstring*	*tagstring* 引数が示すタグの位置へ進み、タグを含む行が現在の位置となるように 画面をスクロールします。詳しくは ctags ユーティリティの説明を参照してください。
' *letter*	以前に *letter* という名を付けてマークした位置に戻ります。
''	最新の 1 画面分を超える移動を行なったときの元の位置に戻ります。 デフォルトはファイルの先頭です。
i?[*!*]*pattern*	ファイルを逆方向に検索し、 *pattern* を含んでいる *i* 番目の行を見つけます。 *!* は、 *pattern* を含んでいない *i* 番目の行の検索です。
i/!*pattern*	ファイルを順方向に検索し、 *pattern* を含んでいない *i* 番目の行を見つけます。
![*command*]	シェルまたは指定したコマンドを呼び出します。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の more と page の動作については、 largefile(5) を参照してください。

環境

more の実行に影響を与える環境変数 LC_COLLATE(/usr/xpg4/bin/more のみ)、LC_CTYPE、 LC_MESSAGES、 NLSPATH 、 TERM についての詳細は、 environ(5) を参照してください。

/usr/xpg4/bin/more

以下の環境変数も /usr/xpg4/bin/more の実行に影響を与えます。

COLUMNS　画面の水平方向のサイズとして、 システムが選択した値の代わりに用いる値を指定します。

EDITOR　エディタを選択する際に v コマンドを使用します。

LINES　画面の垂直方向のサイズとして、 システムが選択した値の代わりに用いる値を指定します。 画面当たりの行数を決める際、 -n オプションの値が LINES の値に優先します。

MORE　前述の「オプション」の項で説明した、オプションを指定する文字列です。 コマンド行に記述する場合と同様に、 オプションとオプションの間は空白文字で区切り、個々のオプションは – で始まらなければなりません。 MORE で指定したオプションの後で、コマンド行のオプションが処理されます。 つまり、コマンド行が次に示すように記述されていると見なされます。 more $MORE *options operands*

終了ステータス

次の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

ファイル

/usr/lib/more.help　/usr/bin/more と /usr/bin/page のためだけのヘルプファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/more
/usr/bin/page

属性タイプ	属性値
使用条件	SUNWcsu
CSI	未対応

/usr/xpg4/bin/more

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

cat(1), csh(1), ctags(1), man(1), nroff(1), script(1), sh(1), ul(1), environ(4), terminfo(4), attributes(5), environ(5), largefile(5)

/usr/bin/more
/usr/bin/page

regcomp(3C)

/usr/xpg4/bin/more | `regex`(5), `XPG4`(5)

/usr/bin/more | 逆方向のスキップを大規模なファイルに対して行うと遅くなります。

/usr/xpg4/bin/more | 端末が正しく設定されていないと、このユーティリティも正しく動作しません。

名前 | mp – テキストからプリンタ記述言語 (PDL) へのプリティプリントフィルタ

形式 | **mp** [-A4] [-C] [-D *target_printer_name*] [-F] [-L *localename*] [-P *target_spool_printer*] [-PS] [-US] [-a] [-c *chars*] [-d] [-e] [-ff] [-fp] [-l] [-ll] [-m] [-M] [-n] [-o] [-p *prologue*] [-s *subject*] [-tm] [-ts] [-u *config_file_path*] [-v] [-w *words*] [-z *point_size*] [-?] [*filename*...]

機能説明 | mp プログラムは、-D または -P オプションを指定せずに呼び出された場合は、各 *filename* 順に読み取ってその内容の清書バージョンを PostScript™ フォーマットで作成し、標準出力に送ります。ファイル名引数が指定されていない場合は、mp は標準入力を読み取ります。端末からの標準入力の場合、EOF シグナル (通常では Ctrl-D) で入力を終了します。

-D および -P オプションを指定するには、引数として出力先プリンタ名が必要です。このプリンタ名を指定すると、出力先プリンタのプリンタ記述言語 (PDL) が生成されます。-D オプションを指定すると、 PDL は標準出力に出力されます。-P オプションを指定すると、 PDL はプリンタ用に直接スプールされます。このどちらのオプションも指定されていない場合は、mp はデフォルトの PostScript の出力を生成します。

mp プログラムは、Solaris でサポートされているさまざまなロケールのテキストを受け付け、指定されたロケールに応じて出力を生成します。また mp は、双方向テキストレンダリングや CTL (Complex Text Layout) などをサポートし、適切なテキストレイアウトで出力します。

mp の入力フォーマットとして、メール、ニュース記事、通常の ASCII ファイル、メールフォルダ全体、ダイジェストのすべてを利用できます。出力フォーマットの各ページの上部と下部には、バナー情報を含む、グレーのひし形、またはひし形と同じ寸法の輪郭が含まれます。

オプション | 次のオプションを指定できます。

-a — ニュース記事としてファイルをフォーマットします。上部のバナーには "Article from *newsgroup*" というテキストが書かれています。*newsgroup* は、Newsgroups: 行で最初に見つかった news グループになります。

-A4 — A4 用紙サイズ (8.26 x 11.69 インチ) を使用します。

-c *chars* — ユーザーの /etc/passwd エントリの gecos フィールドから抽出できる最大文字数です。デフォルトは 18 です。

-C — "\nFrom" を使用して新しいメールメッセージの開始を示すのではなく、mp が Content-Length: メールヘッダーの値を探してそれを使用します。Content-Length: で次の "\nFrom" に行かない場合は、その値は正しくありません。mp はメールフォルダ内で後退して次の "\nFrom" を探します。

-d — ダイジェストとしてファイルをフォーマットします。

-D *target_printer_name* — 出力先プリンタ用の PDL を生成し、X 印刷サーバーへの接続を要求します。

`-e` ELM メールが中間ファイルフォーマットにフロントエンドすると仮定します。ELM 内部からのメッセージの印刷 ("p" コマンドを使う)、特にタグ付きメッセージの印刷の場合に使用されます。このオプションは、ELM オプションの設定で指定する必要があります。

`-ff` Filofax のパーソナルオーガナイザによって使用されるファイルをフォーマットします。

`-fp` Franklin Planner のパーソナルオーガナイザによって使用されるファイルをフォーマットします。

`-F` メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。

`-l` 横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。

`-ll` 横モードで出力します。紙 1 枚にテキストの 1 ページ分が印刷されます。このオプションは、1 行が通常より長いファイルを印刷する場合に便利です。

`-L` *localename* 印刷するファイルのロケールを指定します。このオプショ ンを省略すると、`mp` は `MP_LANG` 環境変数のロケールを使用します。`MP_LANG` 環境変数も設定されていない場合は、`LANG` 環境変数を使用します。`LANG` 環境変数も設定されていない場合は、`mp` は現在動作中のロケールを特定しようと試みます。特定できない場合は、C ロケールで動作中であるとみなします。

`-m` メールフォルダとしてファイルをフォーマットし、複数のメッセージを印刷します。

`-M` 対象ロケール用の `prolog.ps` が存在する場合でも、`mp` が `mp.conf` ファイルを使用して印刷するようにします。このオプションは、英語版 PostScript™ プリンタで日本語を印刷する場合に便利です。

`-n` グレーの線と、その前後の情報を、ヘッダーとフッターから取り除きます。このオプションは、`lp` *filename* コマンドのような出力にしたい場合に使用します。

`-o` 通常の ASCII ファイルとしてファイルをフォーマットします。

`-p` *prologue* 指定した *prologue* ファイルを PostScript/Xprt prologue ファイルとして使用します。それ以前に定義したファイル名は無効になります。この prologue ファイルによって、印刷されるファイルの形式が指定されます。PostScript では、*prologue* ファイルの拡張子は `.ps` です。Xprt クライアント (`-D` オプションが指定されている場合) では、このファイルの拡張子は `.xpr` です。これらのファイルの定義については、後述の「提供される prologue ファイル」を参照してください。

`-P` *target_spool_printer*	出力先プリンタ用に PDL をスプールします。標準出力に送られる出力はありません。このオプションを使用するには、X 印刷サーバーへの接続が必要です。
`-PS`	メールやダイジェストメッセージが PostScript をメッセージのテキストとしてだけ持っている場合、通常は印刷されないでそのまま通過してしまいますが、このオプションを指定すると、PostScript もテキストとして印刷されます。
`-s` *subject*	*subject* を、印刷する新しいサブジェクトとして使用します。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。
`-tm`	Time Manager のパーソナルオーガナイザで使用するファイルをフォーマットします。
`-ts`	Time/System International のパーソナルオーガナイザで使用するファイルをフォーマットします。
`-US`	用紙サイズ US (8.5 x 11 インチ) を使用します。これは、デフォルトの用紙サイズです。
`-u` *config_file_path*	デフォルトの構成ファイル `/usr/lib/lp/locale/`*locale_name*`/mp/mp.conf` の代わりに使用するファイルを指定します。ファイル名は絶対パスで指定してください。
`-v`	このリリースの `mp` のバージョン番号を表示します。
`-w` *words*	ユーザーの `/etc/passwd` エントリの gecos フィールドから抽出できる最大文字数です。デフォルトは `3` です。
`-z` *point_size*	*point_size* に指定したサイズでテキストを印刷します。デフォルトのサイズは、縦モードでは `12` ポイントで、横モードでは `9` ポイントです。
`-?`	`mp` の使用方法を表示します (`csh`(1) を使用している場合は `?` 文字をエスケープする必要があります)。

オペランド

次のオペランドを指定できます。

filename	読み取るファイルの名前。

環境

`XPDISPLAY`	`-D` または `-P` オプションが指定され、かつ、環境内で `XPDISPLAY` 変数が設定されていない場合は、印刷サーバー起動スクリプトが、クライアントを実行しているマシンのポート `2100` で Xprt サーバーを起動します。ジョブが終わると、スクリプトは印刷サーバーを終了します。`XPDISPLAY` が設定されている場合、`mp` クライアントは `XPDISPLAY` で実行中の印刷サーバーと通信しようとします。この場合、印刷サーバーが稼動していなくても、起動しようとはしません。

`MP_PROLOGUE` ページフォーマットファイル (`.xpr` または `.ps`) を保存するディレクトリを決定します。これらのファイルは、ページ装飾、各物理ページの論理ページ番号、横フォーマットか縦フォーマットかなどを決定します。`MP_PROLOGUE` がない場合には、デフォルトディレクトリは、 `/usr/lib/lp/locale/C/mp` になります。

`MP_LANG`
`LANG` `-D` または `-P` オプションのどちらも指定されていない場合、prologue ファイルが印刷される出力の先頭に付加されます。prologue ファイルとは、`/usr/openwin/lib/locale/`*localename*`/print/prolog.ps` または `/usr/lib/lp/locale/`*localename*`/mp/prolog.ps` を指します。*localename* は、環境変数 `MP_LANG` または `LANG` の値です。この両方の変数が存在する場合は、下位互換性を保つために `/usr/openwin/lib/locale/`*localename*`/print/prolog.ps` ファイルが優先されます。どちらかのファイルが存在せず、かつ、`-D` オプションが指定されていない場合は、印刷に必要なプロローグ情報の代わりに、該当ロケールの構成ファイル `/usr/lib/lp/locale/`*localename*`/mp/mp.conf` が構成情報として使用されます。`prolog.ps` が存在する場合、下位互換性を保つために `mp.conf` は使用されなくなります。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`1` エラーが発生した

提供される **prologue** ファイル

次の prologue ファイルが提供されています。拡張子 `.ps` を持つファイルは PostScript 出力用のもので、拡張子 `.xpr` を持つファイルは印刷サーバークライアント用のものです。`.xpr` ファイルは 300dpi のプリンタ用に生成されますが、ほかの解像度にも適応可能です。

`mp.common.ps` このディレクトリ中の他のすべてのファイルに共通の `.ps`ファイルです。

`mp.pro.ps`
`mp.pro.xpr` デフォルトで使用されます。

`mp.pro.ff.ps`
`mp.pro.ff.xpr` `-ff` オプションが有効なときに使用されます。

`mp.pro.fp.ps`
`mp.pro.fp.xpr` `-fp` オプションが有効なときに使用されます。

`mp.pro.tm.ps`
`mp.pro.tm.xpr` `-tm` オプションが有効なときに使用されます。

`mp.pro.ts.ps`
`mp.pro.ts.xpr` `-ts` オプションが有効なときに使用されます。

	`mp.pro.alt.ps` `mp.pro.alt.xpr` デフォルトの prologue ファイルの代わりに使用可能なファイル。下部バナーの右隅にページ番号を出力します。
	`mp.pro.l.ps` `mp.pro.l.xpr` 横フォーマット用の prologue ファイル
	`mp.pro.ll.ps` `mp.pro.ll.xpr` 1 行が通常より長いファイルを印刷する場合の、横フォーマット用の prologue ファイル
	`mp.pro.altl.ps` `mp.pro.altl.xpr` 上記の代わりに使用可能な横フォーマット用の prologue ファイル
ファイル	`.cshrc` `csh`(1) の初期化ファイル
	`.mailrc` `mail`(1) の初期化ファイル
	`/usr/bin/mp` 実行可能ファイル
	`/usr/lib/lp/locale/C/mp/mp.conf` デフォルトの構成ファイル
	`/usr/lib/lp/locale/C/mp/mp.common.ps` このディレクトリ中のその他すべての `.ps` ファイル用の共通 prologue ファイル。`.xpr` ファイル用ではありません。

	`/usr/lib/lp/locale/C/mp/mp.pro.ps`
	`/usr/lib/lp/locale/C/mp/mp.pro.xpr`
	メール印刷用のデフォルトの PostScript prologue ファイル

	`/usr/lib/lp/locale/C/mp/mp.pro.l.ps`
	`/usr/lib/lp/locale/C/mp/mp.pro.l.xpr`
	横フォーマット用のデフォルトの prologue ファイル

	`/usr/lib/lp/locale/C/mp/mp.pro.ll.ps`
	`/usr/lib/lp/locale/C/mp/mp.pro.ll.xpr`
	紙 1 枚にテキストの 1 ページ分を印刷する、横フォーマット用のデフォルトの prologue ファイル。1 行が通常より長いファイルを印刷する場合に便利です。

	`/usr/lib/lp/locale/C/mp/mp.pro.alt1.ps`
	`/usr/lib/lp/locale/C/mp/mp.pro.altl.xpr`
	横フォーマット用の代替の prologue ファイル

	`/usr/lib/lp/locale/C/mp/mp.pro.alt.ps`
	`/usr/lib/lp/locale/C/mp/mp.pro.alt.xpr`
	代替のデフォルトの prologue ファイル。各ページの右下にページ番号を挿入。

/usr/lib/lp/locale/C/mp/mp.pro.ff.ps
/usr/lib/lp/locale/C/mp/mp.pro.ff.xpr
Filofax フォーマット用のデフォルトの prologue ファイル

/usr/lib/lp/locale/C/mp/mp.pro.fp.ps
/usr/lib/lp/locale/C/mp/mp.pro.fp.xpr
Franklin Planner フォーマット用のデフォルトの prologue ファイル

/usr/lib/lp/locale/C/mp/mp.pro.tm.ps
/usr/lib/lp/locale/C/mp/mp.pro.tm.xpr
Time Manager フォーマット用のデフォルトの prologue ファイル

/usr/lib/lp/locale/C/mp/mp.pro.ts.ps
/usr/lib/lp/locale/C/mp/mp.pro.ts.xpr
Time/System International フォーマット用のデフォルトの prologue ファイル

/usr/openwin/lib/locale/*localename*/print/prolog.ps
/usr/lib/lp/locale/*localename*/mp/prolog.ps
mp.conf の代わりに使用されるデフォルトのロケール固有 prologue ファイル (詳細は 環境 の説明を参照)。

mp.conf と .xpr ファイルの構造およびフォーマットについては、 国際化対応言語環境の利用ガイド に記述されています。プリンタ常駐フォントなどほかのフォントを使用する必要がある場合、あるいは出力形式を変更したい場合には、この文書を参照してください。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

csh(1), mail(1), mailtool(1), attributes(5)

名前 | mt – 磁気テープの制御

形式 | **mt** [-f *tapename*] *command...* [*count*]

機能説明 | mt ユーティリティは、 指定されたコマンドを磁気テープドライブに送信します。 -f *tapename* が指定されていない場合は、 環境変数 TAPE が使用されます。 TAPE が設定されていない場合は、 デバイス /dev/rmt/0n が使用されます。

オプション | 次のオプションを指定できます。

-f *tapename* raw テープデバイスを指定します。

オペランド | 次のオペランドを指定できます。

count 要求された操作を実行する回数です。 mt は、 デフォルトでは要求された *command* を 1 回だけ実行します。 *command* を指定することにより、 複数の *command* を実行することができます。

command 磁気テープドライブへ送信が可能な コマンドです。 コマンド名の一部 (ただし、コマンドとして一意に認識できる文字数) だけの指定でもかまいません。

eof, weof テープ上の現在の位置に、 *count* で指定された数の EOF マークを書き込みます。

fsf *count* で指定された数の EOF 分だけ進みます。 テープの位置は、 ファイルの第 1 ブロックの位置になります。

fsr *count* で指定された数のレコード分だけ進みます。

bsf *count* で指定された数の EOF 分だけ後退します。 テープの位置は、 EOF マークのテープの先頭側になります。

bsr *count* で指定された数のレコード分だけ後退します。

nbsf *count* で指定された数のファイル分だけ後退します。 テープの位置は、 ファイルの第 1 ブロックの位置になります。 このコマンドは、 *count+1* 回 bsf を行い、 1 回 fsf を行う場合と同じです。

asf *count* で指定された番号のファイルまで移動します。 このコマンドは、 rewind 後に、 fsf *count* を指定した場合と同じです。

count が次のいずれかのコマンドで指定された場合、 *count* は無視され、 コマンドが 1 度だけ実行されます。

eom テープ上の記録済み部分の末尾まで移動します。 このコマンドは、 す

でに書き込んであるテープにファイルを追加するのに便利です。

`rewind`	テープを巻き戻します。
`offline, rewoffl`	テープを巻き戻し、 適切であれば、 テープをアンロードし、 ドライブ装置をオフラインにします。 テープスタッカーに対して実行した場合は、 4 つのすべてのテープを通して行います。
`status`	テープデバイスの状態に関する情報を表示します。
`retension`	カートリッジテープを完全に巻き戻し、 次にリールの終わりまで進め、 テープの初めまで戻して、 テープの張りをなめらかにします。
`reserve`	デバイスをクローズしたあともテープドライブを予約したままにします。 ドライブをリリースする時には、明示的に行う必要があります。
`release`	クローズ時にリリースするデフォルトの動作を確定し直します。
`forcereserve`	他のホストによって保持されているテープドライブの予約を解除して、そのテープドライブを予約します。 このコマンドはスーパーユーザーの特権でのみ実行できます。
`erase`	テープ全体を消去します。 テープの消去は、 装置またはテープ、もしくはその両方によっては時間がかかります。 時間の詳細については、 装置についているマニュアルを参照してください。

終了ステータス

`0`	操作はすべて正常に終了した
`1`	コマンドは認識されなかった。または `mt` は指定された磁気テープドライブを オープンできなかった
`2`	操作は失敗した

ファイル

`/dev/rmt/*`	磁気テープインタフェース

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 tar(1), tcopy(1), ar(3HEAD), environ(4), attributes(5), mtio(7I), st(7D)

使用上の留意点 すべてのデバイスが すべてのオプションをサポートしているわけではありません。 いくつかのオプションは、 ハードウェアに依存します。 該当するデバイスのマニュアルページを参照してください。

mt は、 アーキテクチャに依存します。 異機種間の操作 (つまり、 SPARC から他の機種、またはその逆) はサポートされていません。

名前 | mv – ファイルの移動

形式 | **/usr/bin/mv** [-fi] *source target_file*

/usr/bin/mv [-fi] *source... target_dir*

/usr/xpg4/bin/mv [-fi] *source target_file*

/usr/xpg4/bin/mv [-fi] *source... target_dir*

機能説明 | 第 1 の形式の mv ユーティリティは、*source* オペランドが示すファイルを *target_file* が示す宛先に移動します。*source* と *target_file* に同じ名前を指定することはできません。*target_file* が存在しない場合、mv はその名前のファイルを新たに作成します。*target_file* が存在していれば、その内容は上書きされます。 最終オペランドが既存のディレクトリを示していないとき、 mv はこの第 1 形式であると見なします。

第 2 の形式の mv は、 *source* オペランドで指定された各ファイルを *target_dir* オペランドが示す既存ディレクトリ中のファイルに移動します。 各 *source* 用の宛先パス名は、宛先ディレクトリ名のあとにスラッシュ (/) と *source* の最終パス名部分を付加したものとなります。 最終オペランドが既存のディレクトリを示しているとき、 mv はこの第 2 形式であると見なします。

mv は、 *target_file* のモードが書き込み禁止であると判断すると、モードを表示し (chmod(2) 参照) 、応答を要求して、標準入力を 1 行読み取ります。応答が肯定である場合、使用可能ならば mv が実行されます。 それ以外の場合、このコマンドは終了します。*target_file* が ACL を有効としている場合、 モードはアクセス権を完全には表示しません。 *source* の親ディレクトリが書き込み可能でスティッキ・ビットセット を持っている場合、以下の条件のうち 1 つ以上が真である必要があります。

- ユーザーはファイルを所有している必要がある
- ユーザーはディレクトリを所有している必要がある
- ファイルはユーザーが書き込み可能である必要があるr
- ユーザーは特権ユーザーである必要がある

source がファイルで *target_file* が複数のリンクを持っている別のファイルへのリンクである場合、他のリンクは残存し、 *target_file* は新しいファイルになります。

source と *target_dir* が、異なるファイルシステム上にある場合、 mv はソースファイルをコピーし、もとのファイルまたはディレクトリを削除します。 他のファイルへのハードリンクはすべてなくなります。 mv は 、ソースファイルの特性 (ファイルの所有者 ID やグループ ID、アクセス権モード、修正時間やアクセス時間、および適用可能であれば ACL と拡張属性) も対象ファイルに引き継ごうとします。シンボリックリンクに対しては、リンク自体の所有者 ID やグループ ID だけを保存します。

所有者 ID やグループ ID を保存できなかった場合、mv は対象ファイルの S_ISUID ビットと S_ISGID ビットをクリアします。これらのビットをクリアできなかった場合には、stderr に診断メッセージを出力しますが、終了ステータスには影響を及ぼしません。ターゲットのファイルシステムが拡張属性をサポートしていない場合、mv は拡張属性を保持できません。/usr/xpg4/bin/mv が、ファイルの特性を引き継ごうとして失敗したその他すべてのエラーに付いて、stderr に診断メッセージを出力します。この場合にも、終了ステータスには影響を及ぼしません。

ソースファイルの特性を保存するには、ユーザーが適切なファイルのアクセス権を持っている必要があります。つまり、スーパーユーザーになるか、あるいは対象ファイルと同じ所有者 ID を持つことが必要です。

オプション　以下のオプションを指定できます。

`-f`　`mv` は、既存の *target* に書き込む場合でもプロンプトを出さずにファイルを移動します。 標準入力が端末でない場合これがデフォルトになることに注意してください。

`-i`　`mv` は、 移動により既存の *target* が上書きされる場合は、必ず、確認のプロンプトを出します。 肯定を応答すれば、移動処理は続行されます。 その他の応答では、 `mv` は *target* を上書きしません。

/usr/bin/mv　`-f` と `-i` のオプションを両方指定してもエラーとは見なされません。 `-f` オプションの方が有効となります。

/usr/xpg4/bin/mv　`-f` と `-i` のオプションを両方指定してもエラーとは見なされません。 `mv` の動作は、最後に指定された方に従います。

オペランド　以下のオペランドを指定できます。

source　移動するファイルまたはディレクトリのパス名

target_file　移動するファイルまたはディレクトリの新たなパス名

target_dir　ファイルの移動先となる既存ディレクトリのパス名

使用法　ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `mv` の動作については、`largefile`(5) を参照してください。

環境　`mv` の実行に影響を与える環境変数 `LC_CTYPE` 、 `LC_MESSAGES` 、 `NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス　以下の終了ステータスが返されます。

`0`　入力ファイルはすべて正常に移された

`>0`　エラーが発生した

属性　次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/mv

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み
インタフェースの安定性	安定

/usr/xpg4/bin/mv

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み
インタフェースの安定性	標準

関連項目 cp(1), cpio(1), ln(1), rm(1), setfacl(1), chmod(2), attributes(5), environ(5), fsattr(5), largefile(5), XPG4(5)

注意事項 -- によって、 ユーザーはコマンド行オプションの終端を明示的にマーク付けできるので、 mv は - で始まるファイル名引数を認識できるようになります。 BSD 移行のサポートとして、 mv は - を -- の同義語として受け入れます。 この移行サポートは、将来のリリースではなくなる可能性があります。

名前 | nawk – パターン走査およびパターン処理の言語

形式 | **/usr/bin/nawk** [-F *ERE*] [-v *assignment*]*'program'* | -f *progfile...* [*argument...*]

/usr/xpg4/bin/awk [-F *ERE*] [-v *assignment...*]*'program'* | -f *progfile...* [*argument...*]

機能説明 | /usr/bin/nawk と /usr/xpg4/bin/awk の両ユーティリティは、テキストデータ処理専用の nawk プログラミング言語で作成されたプログラムを実行します。 nawk プログラムは、一連のパターンとそれに対応するアクション (動作) から構成されます。 *program* を指定する文字列は、シェルによって処理されないように、単一引用符 (') で囲まなければなりません。 パターンとアクションの一連の対は、*program* としてコマンド行で指定できます。または -f *progfile* オプションで指定する 1 つ以上のファイル内で指定できます。 パターンに一致する入力が読み込まれたとき、そのパターンに対応するアクションが実行されます。

入力は一連のレコードとして解釈されます。 1 レコードはデフォルトでは 1 行ですが、RS 組み込み変数を使用すれば、変更できます。 入力の各レコードが *program* 内の各パターンと照合されます。 一致したパターンのそれぞれについて、対応するアクションが実行されます。

nawk ユーティリティは、各入力レコードを一連のフィールドとして解釈します。 デフォルトのときフィールドは空白以外の文字列です。 デフォルトの空白フィールド区切り文字 (空白文字、タブ、または両方) は、FS 組み込み変数または -F *ERE* オプションを使用すれば変更できます。 nawk ユーティリティはレコードの最初のフィールドを $1 、2 番目のフィールドを $2 、(以下同様にみなされる) とみなします。記号 $0 はレコード全体を指します。その他のフィールドを設定すると、$0 が再評価されます。 $0 を割り当てると、すべてのフィールドの値と NF 組込み変数の値がリセットされます。

オプション | 以下のオプションを指定できます。

-F *ERE* — 入力が読み込まれる前に、入力フィールドの区切り文字が拡張正規表現 *ERE* であることを定義します (文字も可能) 。

-f *progfile* — nawk プログラムが入ったファイル *progfile* のパス名を指定します。このオプションが複数指定された場合は、それらのファイルが指定された順序で連結されて nawk プログラムになります。nawk プログラムは 1 つの引数としてコマンド行に指定することもできます。

-v *assignment* — *assignment* 引数は *assignment* オペランドと同じ形式でなければなりません。代入は *var=value* の形式です。*var* は以下に説明する変数の 1 つです。指定された代入は、BEGIN パターンがあればそれに対応する動作も含め、nawk プログラムの実行の前に行われます。このオプションは複数個指定できます。

オペランド | 以下のオペランドを指定できます。

program — -f オプションを省略すると、nawk への最初のオペランドは nawk プログラムのテキストです。アプリケーションは nawk への

単一引数として、*program* オペランドを提供します。 テキストが復帰改行文字 (NEWLINE) で終わっていない場合でも、nawk は復帰改行文字で終わっているものと解釈します。

argument 以下の 2 つのタイプの *argument* は混在可能です。

file 読み込まれる入力が入ったファイルのパス名。プログラム内の一連のパターンと照合されます。*file* オペランドが指定されない、または - の場合、標準入力が使用されます。

assignment 下線または移植性のある 文字セットのアルファベットで始まり、その後に下線、移植可能な文字セットのアルファベット、および = 記号が続くオペランドは、パス名ではなく変数の代入を指定するものです。= 記号より前の文字は、nawk 変数の名前を表します。この名前が nawk の予約語の場合、結果は未定義です。= 記号より後の文字は、nawk プログラム内に二重引用符 (") で囲まれているもの、つまり STRING トークンとして解釈されます。 ただし最後の文字は非エスケープバックスラッシュで、シーケンス "\" の最初の文字ではなく、バックスラッシュという文字そのものとして解釈されます。変数にこの STRING トークンの値が代入されます。値が数値列と考えられる場合、変数には数値が代入されます。このような変数割り当ては、*file* 指定があればその処理の直前に実行されます。そのため最初の *file* 引数より前の割り当ては BEGIN アクション (あれば) の後に実行され、最後の *file* 引数より後の割り当ては END アクション (あれば) の前に実行されます。*file* 引数がない場合は、標準入力の処理の前に割り当てが実行されます。

入力ファイル

以下のソースからの nawk プログラムへの入力ファイルは、テキストファイルでなければなりません。

- *file* オペランドまたはそれと同等 (nawk 変数 ARGV と ARGC を変更して得られる)
- 標準入力 (*file* オペランドがない場合)
- getline 関数への引数

これらのファイルについて、変数 RS が復帰改行文字以外の値に設定されるかどうかによって、システムでは指定された区切り文字で終わるレコードを {LINE_MAX} バイトまでサポートしたり、もっと長いレコードをサポートすることができます。

-f *progfile* を指定する場合は、*progfile* オプション引数のそれぞれが示すファイルは nawk プログラムが入ったテキストファイルでなければなりません。

標準入力が使用されるのは、*file* オペランドが指定されない場合、または – の場合だけです。

拡張機能説明

nawk プログラムは、以下の形式のパターンとアクションの対から構成されます。

```
pattern { action }
```

パターンとアクション (中括弧も含む) のどちらかを省略することもできます。パターン-アクション文は、セミコロンまたは復帰改行文字で区切られます。

パターンを省略すると、入力のどのレコードとも一致します。 アクションを省略すると、入力内の一致するレコードが標準出力に書き出されます。

nawk プログラムの実行は、まずすべての BEGIN パターンに対応するアクションをプログラム内の順序で 実行することによって開始されます。次にレコード区切り文字 (デフォルトでは復帰改行文字) が検出されるまで、各 *file* オペランド (ファイルを省略した場合は標準入力) が処理され、ファイルからデータが読み取られ、現在の FS 値を使用して現在のレコードがフィールドに分割され、プログラム内の順序で各パターンが評価され、現在のレコードと一致する各パターンに対応するアクションが実行されます。後続のパターンが評価される前に、一致パターンのアクションが実行されます。 最後に、すべての END パターンに対応するアクションがプログラム内の順序で実行されます。

nawk 内の式

式は、*pattern* と *action* で使用される計算を記述します。以下の表では、有効な式演算をグループごとに、優先度の高い順に並べてあります。 優先度が同じ演算子は上下の横線で囲まれています。 構文があいまいな式は、優先度の高い演算子が 優先度の低い演算子よりも先に評価されます。この表の expr、*expr1*、*expr2*、および *expr3* は任意の式を意味し、*lvalue* は代入される要素 (つまり代入演算子の左側にあるもの) を表しています。

構文	名前	結果のタイプ	結合規則
(*expr*)	グループ分け	*expr* のタイプ	該当なし
$*expr*	フィールド参照	文字列	該当なし
++ *lvalue*	前置インクリメント	数値	該当なし
—*lvalue*	前置デクリメント	数値	該当なし
lvalue ++	後置インクリメント	数値	該当なし
lvalue —	後置デクリメント	数値	該当なし
expr ^ *expr*	べき乗	数値	右
! *expr*	論理否定	数値	該当なし
+ *expr*	単項プラス	数値	該当なし
– *expr*	単項マイナス	数値	該当なし

構文	名前	結果のタイプ	結合規則
expr * *expr*	乗算	数値	左
expr / *expr*	除算	数値	左
expr % *expr*	率	数値	左
expr + *expr*	加算	数値	左
expr – *expr*	減算	数値	左
expr expr	文字列連結	文字列	左
expr < *expr*	より小さい	数値	なし
expr <= *expr*	等しいまたはより小さい	数値	なし
expr != *expr*	等しくない	数値	なし
expr == *expr*	等しい	数値	なし
expr > *expr*	より大きい	数値	なし
expr >= *expr*	等しいまたはより大きい	数値	なし
expr ~ *expr*	ERE 一致	数値	なし
expr !~ *expr*	ERE 不一致	数値	なし
expr in array	配列のメンバー	数値	左
(*index*) in	多次元配列	数値	左
配列	メンバー		
expr && *expr*	論理積	数値	左
expr ｜｜ *expr*	論理和	数値	左
expr1 ? *expr2*	条件式	選ばれた *expr2*	右
: *expr3*		または *expr3* のタイプ	
lvalue ^= *expr*	べき乗	数値	右
	代入		
lvalue %= *expr*	率代入	数値	右
lvalue *= *expr*	乗算	数値	右
	代入		
lvalue /= *expr*	除算代入	数値	右
lvalue += *expr*	加算代入	数値	右

構文	名前	結果のタイプ	結合規則
lvalue –= *expr*	減算代入	数値	右
lvalue = *expr*	代入	*expr* のタイプ	右

各式は 1 つの文字列値、数値、または両方を持ちます。すでに触れた特殊な文脈の場合を除き、式の値は、その式を使う文脈に必要な型に自動的に変換されます。文字列値は、以下の呼び出しと等価なものによって、数値に変換されます。

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

整数値の場合は、`sprintf()` の書式 `%d` で変換されるのと同じ文字列に変換されます。その他の数値は、書式として `CONVFMT` の値を使って変換されるのと同じ文字列に変換されます。

以下の場合、文字列値は数値列とみなされます。

1. 先行および末尾の空白文字が無視されます。
2. 無視されない最初の文字が + または – の場合、それは無視されます。
3. 無視されない残りの文字が `NUMBER` トークンとして認識される場合、文字列は 数値列とみなされます。

上記のステップで – 文字が無視されると、数値列の値は、認識された `NUMBER` トークンの数値の負の値になります。その他の場合、数値列の値は、認識された `NUMBER` トークンの数値です。文字列が 数値列かどうかが問題になるのは、この項でその用語が使用されている文脈でだけです。

ブール型の文脈で式が使用されるとき、式に数値がある場合、ゼロは偽 として、その他の値は真 として扱われます。それ以外の場合、空白文字列の文字列値は偽 として、その他の値は真として扱われます。ブール型の文脈は、以下の 1 つです。

- 条件式の最初の副式 (サブエクスプレッション)
- 論理否定、論理積、または論理和によって演算される式
- `for` 文の 2 番目の式
- `if` 文の式
- `while` 文または `do . . . while` 文の `while` 句の式
- パターンとして使用される式 (Overall Program Structure 内のように)

`nawk` 言語は、数字や文字列を保存するための配列を提供します。 配列は宣言しなければなりません。配列は最初は空で、サイズは動的に変化します。 連想配列機能の一種を提供する添字 (要素識別子) は文字列です。配列名とその後ろにある大括弧内の添字は、構文で説明したように *lvalue* および式として使用できます。添字のない配列名が使用できるのは、以下の文脈でだけです。

- 関数定義または関数呼び出し内のパラメータ
- キーワード `in` の後に続く `NAME` トークン

有効な配列インデックスは、プログラミング言語で多次元配列にインデックスが 付けられる方法と同じように、コンマで区切られた 1 つ以上の式から構成されます。 nawk 配列は 1 次元なので、SUBSEP 変数の値で区切られている個々の式の文字列値を連結することで、コンマで区切られたリストが 1 つの文字列に変換されます。 そのため以下の 2 つのインデックス操作は等しい結果となります。

```
var[expr1, expr2, ... exprn]
var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]
```

in 演算子が付いた多次元 インデックスは、括弧で囲まなければなりません。 in 演算子は特定の配列要素の有無を調べ、無い場合は要素を作成しません。 存在しない配列要素へのその他の参照を行うと、要素は自動的に作成されます。

変数と特殊変数

nawk プログラムでは変数を参照することで変数を使用できます。 関数パラメータ以外は、明示的に宣言されません。 初期設定されていないスカラー変数と配列要素は、数値ゼロと NULL 文字列値の両方を持ちます。

フィールド変数は $ とその後に続く数字または数値式によって指定されます。 負でない整数以外と評価されるフィールド番号 *expression* の動作は未定義です。 初期設定されていない変数や文字列値は、この文脈では 数値に変換される必要はありません。新しいフィールド変数は、値を割り当てることで作成されます。 存在しないフィールド (つまり $NF の後のフィールド) を参照すると、NULL 文字列が生成されます。ただし存在しないフィールドを割り当てると (たとえば $(NF+2) = 5) 、NF の値が増分され、値として NULL 文字列を持つ中間フィールドと $0 の値が再度計算され、OFS の値でフィールドが区切られます。 各フィールド変数には、作成時に文字列値が入ります。 現在のロケールからの小数点文字がピリオドに変更された文字列は、数値列とみなされ (前述の「 nawk 内の式」の項を参照) 、フィールド変数も数値列の値を持つことになります。

nawk は次の特殊変数を設定します。

/usr/xpg4/bin/awk

ARGC — ARGV 配列内の要素の数。

ARGV — オプションと *program* 引数を除いたコマンド行引数の配列。 ゼロから ARGC-1 までの番号がふられます。

ARGV 内の引数は、変更や追加が可能です。 ARGC は変更できます。各入力ファイルが終わると、nawk は現在の ARGC-1 の値まで、ARGV の次の非 NULL 要素を次の入力ファイルの名前として扱います。 ARGV の要素を NULL に設定すると、入力ファイルとして扱われなくなります。名前 - は、標準入力を示します。引数が *assignment* オペランドのフォーマットと一致すると、その引数は *file* 引数ではなく割り当てとして扱われます。

CONVFMT — 数字を文字列に変換するための printf フォーマット (OFMT が使用される出力文は除く) 。デフォルトは %.6g です。

ENVIRON — 変数 ENVIRON は、環境値を表す配列です。 配列のインデックスは環境変数の名前からなる文字列で、各配列要素の値はその変数

	の値からなる文字列です。 環境変数の値が数値列 と考えられる場合、配列要素も数値をとります。 nawk の動作が環境変数の影響を受ける (nawk が system 関数または print 文、printf 文、または getline 関数を使ってパイプライン経由で実行する コマンドの環境も含む) 場合はいずれも、nawk の実行開始時点の環境が使われます。
FILENAME	現在の入力ファイルのパス名。 BEGIN アクション内では、この値は未定義です。 END アクション内では、この値は最後に処理された入力ファイルの名前です。
FNR	現在のファイル内の現在のレコードの序数。 BEGIN アクション内では、この値はゼロです。 END アクション内では、この値は最後に処理されたファイル内の 最後に処理されたレコードの番号です。
FS	入力フィールド区切り文字の正規表現。デフォルトは空白文字です。
NF	現在のレコード内のフィールド数。 BEGIN アクション内では、*var* 引数のない getline 関数がその前に実行されていない限り、NF の使用は未定義です。 END アクション内では、END アクションに入る前に後続の出入力先を変更された *var* 引数のない getline 関数が実行されていない限り、NF は最後に読み取られたレコードの値を維持します。
NR	入力の開始からの現在のレコードの序数。 BEGIN アクション内では、この値はゼロです。 END アクション内では、この値は最後に処理されたレコードの番号です。
OFMT	出力文 "%.6g" 内でデフォルトとして、数字から文字列へ変換する際の printf フォーマット。 OFMT の値が浮動小数点フォーマットでない場合、変換の結果は不定です。
OFS	print 文出力フィールド区切り文字。 デフォルトは空白文字です。
ORS	print 出力レコード区切り文字。デフォルトは復帰改行文字です。
LENGTH	match 関数が照合する文字列の長さ。
RS	RS の文字列値の最初の文字は、入力レコード区切り文字です。 デフォルトは復帰改行文字です。 RS に複数の文字が入っていると、結果は不定です。 RS が NULL の場合、レコードは一連の 1 つ以上の空白行で区切られます。 先行または末尾に空白行があっても、入力の先頭または最後に空のレコードが 作成されることはありません。 FS の値に関係なく、フィールド区切り文字はいつも復帰改行文字です。

RSTART　match 関数が照合する文字列の開始位置。 1 から順に番号が付いています。これは常に、match 関数の戻り値と等価です。

SUBSEP　多次元配列用の添字区切り文字列。 デフォルト値は 1 です。

正規表現

nawk ユーティリティは、拡張正規表現 (ERE) (regex(5) を参照) を活用しています。ただし ERE 内で特殊文字をエスケープするときは、C 言語規約が使用できます。 \\ 、 \a 、 \b 、 \f 、 \n 、 \r 、 \t 、 \v 、および下の表に示したエスケープシーケンスが使用できます。 これらのエスケープシーケンスは、括弧の内でも外でも認識されます。 なお、レコードは復帰改行文字で区切る必要はなく、文字列定数には復帰改行文字を含めるため、nawk ERE では \n シーケンスも有効です。 正規表現内にスラッシュ文字を使う場合は、下の表に示すように、エスケープを使用しなければなりません。

エスケープシーケンス	説明	意味
\"	バックスラッシュ引用符	引用符文字
\/	バックスラッシュスラッシュ	スラッシュ文字
ddd	バックスラッシュの後に 1 から 3 桁の 8 進文字 (01234567) の最長シーケンスが続きます。すべての桁が 0 なら (つまり NULL 文字を表しているなら)、動作は未定義です。	1 から 3 桁の 8 進整数で文字がエンコーディングされています。複数バイト文字には、各バイトごとに先行 \ を含んだ、複数の連結さ れたエスケープシーケンスが必要です。
c	バックスラッシュの後にこの表で説明されていない 文字または特殊文字が続きます (\\, \a, \b, \f, \n, \r, \t, \v)。	未定義

正規表現照合演算子 ~ または ! ~ のどちらかを使って、正規表現を特定のフィールドや文字列と照合させることができます。 これらの演算子は、オペランドの右側を正規表現、オペランドの左側を文字列と解釈します。 正規表現が文字列と一致すると、~ 式は値 1 と評価され、 ! ~ 式は値 0 と評価されます。正規表現が文字列と一致しない場合、 ~ 式は値 0 と評価され、 ! ~ 式は値 1 と評価されます。右辺のオペランドが字句トークン ERE 以外なら、上で説明したエスケープ規約も含め、式の文字列値は拡張正規表現として解釈されます。 なお文字列リテラル (字句トークン STRING) の値を判断するときも、これと同じエスケープ規約が適用され、この文脈で文字列リテラルが使用されるときもう一度適用されます。

~ または ! ~ 演算子の右辺以外の文脈で ERE トークンが使用されている、または下で説明する組み込み関数の引数の 1 つとして使用されているとき、結果の式の値は、

$0 ~ /*ere*/ と等価です。

gsub 、 match 、 sub 関数への *ere* 引数および split 関数への *fs* 引数 (「文字列関数」の項を参照) は、拡張正規表現として解釈されます。これらは ERE トークンまたは任意の式のどちらでもよく、~ または ! ~ 演算子の右辺と同じ方法で解釈されます。

-F *ERE* オプションを使うことで、または式を含んだ文字列を組み込み変数 FS に割り当てることで、フィールドの分割に拡張正規表現を使用できます。 FS 変数のデフォルト値は、1 つの空白文字です。 FS の動作を下に説明します。

1. FS が 1 つの文字の場合
   a. FS が空白文字なら、先行および末尾の空白文字をスキップします。フィールドは 1 つ以上の空白文字で区切られます。
   b. FS か a. で説明した空白以外の文字 *c* なら、フィールドは個々の *c* によって区切られます。
2. それ以外の場合、FS の文字列は拡張正規表現と考えられます。 拡張正規表現と一致するシーケンスの発生により、フィールドが区切られます。

gsub 、 match 、 split 、および sub 組み込み関数の場合を除き、正規表現照合は、 入力レコードに基づきます。つまりレコード区切り文字 (変数 RS の値の最初の文字、デフォルトは復帰改行文字) は式に組み込めず、どの式もレコード区切り文字とは一致しません。 レコード区切り文字が復帰改行文字以外なら、式に組み込まれた復帰改行文字を照合させることができます。 この 4 つの組み込み関数では、正規表現照合はテキスト文字列に基づきます。 そのため、どの文字 (復帰改行文字やレコード区切り文字も含む) でも パターンに組み込め、適切なパターンを任意の文字と照合させることができます。 しかしすべての nawk 正規表現照合では、パターン、入力レコード、またはテキスト文字列に NULL 文字を使うと、結果が未定義になります。

パターン

パターンは、有効な式、コンマで区切られた 2 つの式で示された範囲、または 2 つの特殊パターン BEGIN と END のいずれかです。

特殊パターン

nawk ユーティリティは、2 つの特殊パターン BEGIN と END を認識します。各 BEGIN パターンは一度照合され、最初の入力レコードが読み込まれる前とコマンド行割り当てが行われる前に、対応するアクションが実行されます (以前の BEGIN アクションで getline 関数が使用されたときは例外) 。各 END パターンは一度照合され、最後の入力レコードが読み込まれた後に、対応するアクションが実行されます。この 2 つのパターンにはアクションが対応しています。

BEGIN と END は、その他のパターンとは組み合わせられません。複数の BEGIN および END パターンが使用できます。 BEGIN パターンに対応するアクションは、プログラム内の指定順序で実行されます。これは END パターンの場合も同じです。プログラム内では、END パターンが BEGIN パターンより前でもかまいません。

nawk プログラムが BEGIN パターンのアクションだけで構成されていて、BEGIN アクションが getline 関数を含んでいない場合、最後の BEGIN アクションの最後の文が実行されたとき、nawk は入力を読まずに終了します。 nawk プログラムが END パターンのアクションだけ、または BEGIN パターンと END パターンのアクションだけで構成されている場合、END アクションの文が実行される前に入力が読まれます。

式パターン　式パターンは、ブール型の文脈の式として評価されます。 結果が真 なら、パターンは一致すると考えられ、対応するアクションがあればそれが 実行されます。結果が偽なら、アクションは実行されません。

パターンの範囲　パターンの範囲は、コンマで区切られた 2 つの式から構成されます。この場合、最初の式の一致と 2 番目の式の一致までの間にあるすべてのレコードに対して アクションが実行されます。これにより、入力レコードの先頭から一致した範囲の最後まで、パターンの範囲を繰り返せます。

アクション　アクションは一連の文で構成されます。 文は以下のいずれかです。

```
if ( expression ) statement [ else statement ]
while ( expression ) statement
do statement while ( expression )
for ( expression ; expression ; expression ) statement
for ( var in array ) statement
delete array[subscript] # 配列の構成要素を削除
break
continue
{ [ statement ] . . . }
expression          # 通常変数 = 式
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next                # この入力行にある残りのパターンをスキップ
exit [expr] # 入力の残りをスキップ。終了ステータスは expr
return [expr]
```

1 つの文を、かっこで囲まれた文のリストで置き換えることができます。文は、復帰改行文字またはセミコロンで区切られ、指定された順に実行されます。

next 文は、現在の入力レコードの以降の処理をすべて破棄します。 BEGIN または END アクション内で next 文が現れたり呼び出されたりすると、動作は未定義です。

exit 文は、プログラムソース内の順序ですべての END アクションを呼び出した後、以降の入力は読まずにプログラムを終了させます。 END アクション内の exit 文は、以降の END アクションを実行せずにプログラムを終了させます。 exit 文内に式が指定されている場合は、その数値が nawk の終了ステータスです。ただしエラーが検出されたり、式を持った exit 文が実行される場合は除きます。

出力文　print 文も printf 文も、デフォルトでは標準出力に書き出します。 以下のような形式で *output_redirection* が指定されていれば、その宛先に書き出します。

```
> expression>> expression| expression
```

すべての場合、*expression* が評価され、書き出し先の完全パス名として (> または >> のとき) または実行されるコマンド (| のとき) として使用される文字列が生成されます。最初の 2 つの形式を使うと、その名前のファイルが現在オープンされていなければ オープンされ、必要ならファイルが作成されます。 最初の形式を使うと、ファイルは切り捨てられます。 次に出力データがファイルに付加されます。 ファイルが

オープンされている限り、同じ文字列値として式が評価される以降の呼び出しは、単に出力をファイルに付加していきます。 同じ文字列として評価される式で `close` 関数が呼び出されるまで、ファイルはオープンされたままです。

3 番目の形式は、コマンドの入力にパイプされたストリームへ 出力を書き出します。コマンド名として *expression* の値を持つストリームが現在オープンされていない場合は、ストリームが作成されます。作成されたストリームは、`command` 引数として *expression* の値を持ち、*mode* 引数として `w` の値を持つ `popen`(3C) 関数呼び出しで作成されるものと等価です。 ストリームがオープンされている限り、同じ文字列値として式が評価される 以降の呼び出しは、出力を既存のストリームに書き出します。 同じ文字列値として評価される式で `close` 関数が呼び出されるまで、ストリームはオープンされたままです。 `close` 関数が呼び出されると、`pclose` 関数が呼び出されたかのように、ストリームはクローズされます。

これらの出力文は、コンマで区切られた *expression* のリストをとります。このリストは、文法解釈において非終端記号 `expr_list`、`print_expr_list`、`print_expr_list_opt` によって参照されます。このリストはここでは *expression list* として参照され、各メンバーは *expression argument* として参照されます。

`print` 文は、現在の出力フィールド区切り文字 (上記の変数 `OSF` を参照) で区切られ、出力レコード区切り文字 (上記の変数 `ORS` を参照) で終わる、指定された出力ストリームに、各式の引数の値を書き出します。式の引数はすべて文字列とみなされ、必要なら変換されます。ただし、`COMVFMT` の値の代わりに `OFMT` の `printf` フォーマットが使用されます。 空の式リストは、入力レコード全体 (`$0`) を意味します。

`printf` 文は、本書でファイルフォーマットの説明に使用している ファイルフォーマット表記と似た表記に基づいて出力します。文字列 `format` として最初の式引数、`arg1` から `argn` の文字列として後続の式引数で指定されたとおりに、出力されます。 ただし、以下のような例外があります。

1. *format* はグラフィック表現ではなく、実際の文字列です。そのため空の文字位置は含めません。*format* 文字列内の空白文字は、変換の *flag* 以外の文脈では、通常の文字として扱われ出力されます。
2. 文字セットに 文字が含まれていて、文字が *format* 文字列に出現した場合は、通常の文字として扱われ出力されます。
3. バックスラッシュ文字で始まる *escape sequences* は、通常文字のシーケンスとして扱われ、出力されます。 なおこのようなシーケンスは、リテラル文字列に出現したとき、`nawk` によって字句が解析されますが、`printf` 文によっては特に処理されません。
4. *field width* または *precision* は、数字列の代わりに `*` 文字として指定できます。この場合、式リストの次の引数が取り出され、その数値がフィールド幅または精度とみなされます。
5. システムは、*format* 文字列で指定されていない空白文字を `d` または `u` 変換からの出力の先頭や末尾には付けません。
6. システムは、*format* 文字列で指定されていない先行ゼロを `o` 変換からの出力には付けません。

7. c 変換の場合、引数が数値なら、エンコーディングとして その値を持つ文字が出力されます。 値がゼロのとき、または文字セットのどの文字にもエンコーディングされていないとき、動作は未定義です。引数が数値を持たない場合、文字列値の最初の文字が出力されます。 文字列に文字が含まれていない場合、動作は未定義です。
8. 引数が必要な変換のそれぞれについて、次の式引数が評価されます。 c 変換以外は、変換仕様に応じて値が適切な型に変換されます。
9. *format* 文字列内のすべての変換仕様を満たすだけの十分な式引数がない場合、動作は未定義です。
10. *format* 文字列内の文字シーケンスが % 文字で始まるが、有効な変換仕様を形成していない場合、動作は未定義です。

print と printf はどちらも、少なくとも {LINE_MAX} バイトを出力できます。

関数

nawk 言語は、算術関数、文字列関数、入出力関数、および一般関数など、豊富な組み込み関数を備えています。

算術関数

int の場合を除き、算術関数は ISO C 規格に基づいています。エラーを戻すまたは動作が未定義だと ISO C 規格が指定している場合、動作は未定義です。 文法では引数なしまたは括弧なしの組み込み関数を認めていますが、以下のリストで引数や括弧が省略可能 (大括弧 [ ] で表示) と示されていない限り、省略した場合の結果は未定義です。

atan2(*y*,*x*)	*y*/*x* の逆正接値を返します。
cos(*x*)	*x* の余弦値を返します。 *x* の余弦をラジアン単位で 返します。
sin(*x*)	*x* の正弦値を返します。 *x* の正弦をラジアン単位で 返します。
exp(*x*)	*x* の指数関数を返します。
log(*x*)	*x* の自然対数を返します。
sqrt(*x*)	*x* の平方根を返します。
int(*x*)	引数を整数に切り捨てます。 $x > 0$ のときは、0 に向かう方向で整数化します。
rand()	$0 \le n < 1$ の範囲の乱数 n を返します。
srand([expr])	rand のシード値を expr に設定します。 expr が省略された場合は、時刻を使用します。 以前のシード値が返されます。

文字列関数

以下のリストの文字列関数がサポートされています。 文法では引数なしまたは括弧なしの組み込み関数を認めていますが、以下のリストで引数や括弧が省略可能 (大括弧 [ ] で表示) と示されていない限り、省略した場合の結果は未定義です。

gsub(*ere*,*repl*[, *in*])	後述の sub 関数と同様の機能ですが、相違点は (ed ユーティリティにおける一括置換のように) 指定されたときに、$0 または *in* 引数内の正規表現のすべてを置換することです。
index(*s*,*t*)	文字列 *s* の中で文字列 *t* が最初に出現する位置 (最初の文字を 1 として文字数を数えたもの) を返します。 出現しない場合はゼロを返します。
length[([*s*])]	引数を文字列と解釈し、その長さ (文字数) を返します。 引数が指定されていなければレコード $0 全体の長さを返します。
match(*s*,*ere*)	文字列 *s* の中で正規表現 *ere* が最初に出現する位置 (最初の文字を 1 として文字数を数えたもの) を返します。出現しなければ 0 を返します。 RSTART は開始位置 (返される値と同じ) に設定され、一致しない場合はゼロに設定されます。 RLENGTH は一致した文字列の長さに設定され、一致しない場合は –1 に設定されます。
split(*s*,*a*[, *fs*])	文字列 *s* を *a*[1] 、 *a*[2] 、 . . . 、 *a*[*n*] の配列要素に分割し、値 *n* を返します。この分割は、正規表現 *fs* によって行われ、*fs* が指定されていない場合は、フィールド区切り文字 FS によって行われます。各配列要素は、作成されたときに文字列値を持ちます。 配列要素に割り当てられた文字列が、現在のロケールの小数点文字がピリオドに変更されていて、数値列とみなされる場合は、配列要素もその数値列 の数値を持つことになります。 *fs* の値が NULL 文字列のとき、結果は未定義です。
sprintf(*fmt*,*expr*,*expr*,. . .)	*fmt* で指定した printf フォーマットに従って式をフォーマットし、その結果得られた文字列を返します。
sub(*ere*,*repl*[, *in*])	文字列 *in* の中で正規表現 *ERE* が最初に出現する位置を探し、その正規表現を文字列 *repl* に置き換え、置換した総数を返します。 *repl* 内に出現するアンパサンド (&) は、正規表現に一致する *in* 内の文字列で置換されます。文字列 *repl* の最初から最後までに出現するバックスラッシュ (\) が存在する場合、\ の次の文字は文字そのものとみなされ、特別な意味は失われます (たとえば \& は文字どおり & と解釈されます) 。 & と \ 以外は、各文字がどのような特別な意味を持つかは不定です。 *in* が指定され、*lvalue* 以外のとき、動作は未定義です。 *in* を省略すると、nawk は現在のレコード ($0) 内で置換を行います。

`substr`(*s*,*m*[, *n*]) 文字列 *s* 内の *m* 文字目から始まる *n* 文字の長さの部分文字列を返します。 *n* がない場合、部分文字列の長さは、文字列 *s* の長さによって限定されます。

`tolower`(*s*) 文字列 *s* に基づいて文字列を返します。現在のロケールの `LC_TYPE` カテゴリによって `tolower` マッピングを持つように指定されている *s* 内の各大文字が、マッピングで指定されている小文字で置換されます。 *s* 内のその他の文字は変更されません。

`toupper`(*s*) 文字列 *s* に基づいて文字列を返します。現在のロケールの `LC_TYPE` カテゴリによって `toupper` マッピングを持つように指定されている *s* 内の各小文字が、マッピングで指定されている大文字で置換されます。 *s* 内のその他の文字は変更されません。

以上の関数はすべて、パラメータとして正規表現 *ERE* をとります。ただし、以下に定義されている正規表現であるパターン または文字列値を持つ式は除きます。

入出力関数と一般関数

入出力関数と一般関数は次のとおりです。

`close`(*expression*) `print` または `printf` 文でオープンされた、または同じ文字列値を持つ *expression* での `getline` 呼び出しでオープンされたファイルやパイプをクローズします。クローズが成功すると、この関数はゼロを返します。それ以外の場合は、ゼロ以外の値を返します。

expression `|getline`[*var*] コマンドの出力からパイプされたストリームから、入力レコードを読み込みます。 コマンド名として *expression* の値でストリームが現在オープンされていない場合は、ストリームが作成されます。このストリームは、`command` 引数として *expression* の値と、*mode* 引数として `r` の値を指定して `popen` 関数呼び出しで作成されるものと同じです。 ストリームがオープンされている限り、*expression* が同じ文字列値に評価される以降の呼び出しは、ファイルから後続のレコードを読み込みます。 同じ文字列値として評価される式で `close` 関数が呼び出されるまで、ストリームはオープンされたままです。 `close` 関数が呼び出されると、`pclose` 関数が呼び出されたかのように、ストリームがクローズされます。 *var* がない場合は、`$0` と `NF` が設定されます。それ以外の場合は *var* が設定されます。

| の左側に (`getline` を含む式の先頭までに) 括弧で囲まれていない演算子 (連結演算も含む) があるとき、`getline` 演算子による式の評価は、あいまいになる可能性があります。 `$` 演算子の文脈では、| は `$` よりも優先度が低いかのようにふるまいます。 その他の演算子を評価した結果は不定であり、移植性のあるアプリケーションは、このような表記をすべて正しく括弧で囲まなければなりません。

`getline`	`$0` に、現在の入力ファイルの次の入力レコードを設定します。 `getline` のこの形式は、`NF`、 `NR`、および `FNR` 変数を設定します。
`getline` *var*	変数 *var* に、現在の入力ファイルの次の入力レコードを設定します。`getline` のこの形式は、`FNR` 変数と `NRR` 変数を設定します。
`getline`[*var*]<*expression*	指定されたファイルから次の入力レコードを読みます。 *expression* が評価され、完全パス名として使用される文字列を生成します。 その名前のファイルが現在オープンされていない場合は、オープンします。ストリームがオープンされている限り、*expression* が同じ文字列値に評価される以降の呼び出しは、ファイルから後続のレコードを読み込みます。 同じ文字列値として評価される式で `close` 関数が呼び出されるまで、ファイルはオープンされたままです。 *var* がない場合は、`$0` と `NF` が設定されます。それ以外の場合は、*var* が設定されます。 < の右側に (`getline` を含む式の最後までに) 括弧で囲まれていない 2 項演算子 (連結演算も含む) があるとき、`getline` 演算子による式の評価はあいまいになる可能性があります。 このような構文を評価した結果は不定であり、移植性のあるアプリケーションはこのような表記を すべて正しく括弧で囲まなければなりません。
`system`(*expression*)	*expression* で指定されているコマンドを、`system`(3C) 関数と等価な方法で実行し、コマンドの終了ステータスを返します。

`getline` 関数は、どの形式で実行した場合でも、正常終了時には `1` を、ファイルの終わりに達すると `0` を、エラー発生時には `-1` を返します。

ファイルやパイプラインの名前として文字列が使用されている場所では、文字列のテキストはまったく同じでなければなりません。 「同じ文字列値」とは、「等価文字列」を意味し、空白文字が違うだけでも異なるファイルを表現することになります。

ユーザー定義関数

`nawk` 言語ではユーザー定義関数も使用できます。 ユーザー定義関数は以下の形式で定義できます。

```
function name(args, . . .) { statements }
```

関数は、`nawk` プログラム内のどこからでも参照できます。 さらに、定義より前に使用できます。関数の有効範囲はグローバルです。

関数の引数は、スカラーまたは配列です。 スカラーを使用する関数に引数として配列名が渡されたとき、または配列を使用する関数に引数としてスカラー式が渡されたとき、その動作は未定義です。 関数の引数は、スカラーの場合は値によって、配列名の場合は参照によって引き渡されます。 引数名は関数に対してローカル、その他の変数名はすべてグローバルです。 引数名と関数名または特殊 nawk 変数名とに、同じ名前は使用されません。 グローバルな有効範囲を持つ変数名と関数名に、同じ名前を使用してはなりません。 同じ有効範囲内では、スカラー値と配列に同じ名前を使用してはなりません。

関数定義内のパラメータ数は、関数呼び出しのときのパラメータ数と一致する必要はありません。 ローカル変数としては、余分な仮パラメータを使用できます。 関数定義内の引数より少数の引数が関数呼び出しに指定された場合、関数本体でスカラーとして使用されている余分なパラメータは、NULL 文字列の文字列値とゼロの数値で初期設定されます。また、関数本体で配列として使用されている余分なパラメータは、空の配列として初期設定されます。 関数定義内の引数より多数の引数が関数呼び出しに指定された場合、動作は未定義です。

関数を呼び出すとき、関数名と左括弧の間に空白は置くことはできません。 関数呼び出しは、ネストすることができます。 再帰呼び出しも可能です。 ネストまたは再帰関数呼び出しから戻っても、参照によって引き渡された配列パラメータ以外、呼び出し側関数のパラメータのどの値も変わりません。 return 文は値を戻すために使用できます。関数定義の外に return 文が出現した場合、動作は未定義です。

関数定義において、左括弧より前および右括弧より後の復帰改行文字は 任意指定です。関数定義は、*pattern-action* の対が許可されている場所ならプログラム内のどこに置いてもかまいません。

使用法

index 、 length 、 match 、および substr 関数と、 ISO C 規格の同様の関数とを混同しないでください。 nawk バージョンは文字を扱い、 ISO C 規格はバイトを扱います。

連結は明示的な演算子ではなく隣接する式によって表現されるので、評価の正しい優先順位を示すため、括弧を使用しなければならないことがよくあります。

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の nawk の動作については、largefile(5) を参照してください。

使用例

sh を使うアプリケーションの場合、コマンド行に指定する nawk プログラムは、単一引用符内に指定する (たとえば *'program'*) がもっとも簡単な方法です。 nawk プログラムは一般に、二重引用符をはじめ、シェルに対して特殊な意味を持つ文字を含んでいることが多いためです。 nawk プログラムが単一引用符文字を含んでいる場合、プログラムのほとんどを単一引用符内の文字列として指定し、引用符で囲んだ単一引用符文字とシェルを連結するのが、一般にもっとも簡単な方法です。次の例を見てください。

```
awk '/'\''/ { print "quote:", $0 }'
```

これは標準入力からの単一引用符文字を含むすべての行に quote: という接頭語を付けて印字します。

以下に、簡単な nawk プログラムの例を示します。

例 **1** 第 3 フィールドの値が 5 より大きい入力行を、すべて標準出力に書き出します。

```
$3 > 5
```

例 **2** 10 行ごとに書き出します。

```
(NR % 10) == 0
```

例 **3** 部分文字列が正規表現と一致する行を書き出します。

```
/(G|D)(2[0-9][[:alpha:]]*)/
```

例 **4** 部分文字列として、文字 G または D の後にいくつかの数字と文字を伴っている行を出力します。

この例では、文字クラス digit と alpha を使って、言語独自の数字および文字を照合させています。

```
/(G|D)([[:digit:][:alpha:]]*)/
```

例 **5** 第 2 フィールドは正規表現と一致するが第 4 フィールドは一致しない、という行を出力します。

```
$2 ~ /xyz/ && $4 !~ /xyz/
```

例 **6** 第 2 フィールドにバックスラッシュを含んでいる行を出力します。

```
$2 ~ /\\/
```

例 **7** 第 2 フィールドにバックスラッシュを含んでいる行を出力します。

ここではバックスラッシュによるエスケープを 2 度解釈しています。 1 度は文字列の字句処理時、もう 1 度は正規表現の処理時です。

```
$2 ~ "\\\\"
```

例 **8** 各行の最後から 2 番目のフィールドと最後のフィールドを出力します。 フィールドはコロンで区切ります。

```
{OFS=":";print $(NF-1), $NF}
```

例 **9** 各行の行番号とフィールドの数を出力します。

行番号、コロン、フィールド数の 3 つの文字列は連結され、1 つの文字列として標準出力に書き出されます。

```
{print NR ":" NF}
```

例 **9** 各行の行番号とフィールドの数を出力します。　　*(続き)*

例 **10** 長さが 72 文字を超えている行を出力します。

```
{length($0) > 72}
```

例 **11** 先頭の 2 つのフィールドを、逆の順序で、OFS で区切って出力します。

```
{ print $2, $1 }
```

例 **12** 上記と同様ですが、入力フィールドはコンマ、または空白文字とタブ、あるいはその両方で区切られています。

```
BEGIN { FS = ",[\t]*|[\t]+" }       { print $2, $1 }
```

例 **13** 先頭のフィールドの値を合計し、その合計値と平均値を出力します。

```
{s += $1 } END {print "sum is ", s, " average is", s/NR}
```

例 **14** フィールドの内容を逆の順序で、1 フィールドにつき 1 行の形式で出力します。 1 入力行につき多数の出力行が発生します。

```
{ for (i = NF; i > 0; --i) print $i }
```

例 **15** “start” と “stop”という 2 つの文字列の間にあるすべての行を出力します。

```
/start/, /stop/
```

例 **16** 先頭フィールドの値が直前の行の先頭フィールドと異なっている行を、すべて出力します。

```
$1 != prev { print; prev = $1 }
```

例 **17** `echo` の動作をシミュレートします。

```
BEGIN { for (i = 1; i < ARGC; ++i) printf "%s%s", ARGV[i], i==ARGC-1?"\n":"" }
```

例 **18** PATH 環境変数に含まれているパス接頭辞を、1 行に 1 つの形式で出力します。

```
BEGIN  {
       n = split (ENVIRON["PATH"], path, ":")
       for (i = 1; i <= n; ++i)
              print path[i]
       }
```

例 **19** `input` ファイルの内容が、ページ番号 5 を開始ページとして出力されます。

いま `input` という名前のファイルがあり、以下のような形式のページヘッダーを含んでいるとします。

```
Page#
```

例 **19** input ファイルの内容が、ページ番号 5 を開始ページとして出力されます。 (続き)

また、program という名前のファイルの内容は次のとおりとします。

```
/Page/{ $2 = n++; }
{ print }
```

ここで次に示すコマンドを実行します。

```
nawk -f program n=5 input
```

これにより input ファイルの内容が、ページ番号 5 を開始ページとして出力されます。

環境

nawk の実行に影響を与える環境変数 LC_COLLATE 、 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

LC_NUMERIC　数値入力の解釈、数値と文字列との変換、数値出力のフォーマットに用いる、小数点文字を決定します。awk プログラム (コマンド行引数で指定される代入も含む) の処理で認識される小数点文字は、ロケールに関係なくピリオド (POSIX ロケールの小数点文字) です。

終了ステータス

以下の終了ステータスが返されます。

0　入力ファイルはすべて正しく処理された

>0　エラーが発生した

プログラム中で exit 式を使って終了ステータスを変更することができます。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/nawk

属性タイプ	属性値
使用条件	SUNWcsu

/usr/xpg4/bin/awk

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目

awk(1), ed(1), egrep(1), grep(1), lex(1), sed(1), popen(3C), printf(3C), system(3C), attributes(5), environ(5), largefile(5), regex(5), XPG4(5)

Aho, A. V., B. W. Kernighan, and P. J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988.

診断

file オペランドが指定され、指定されたファイルがアクセスできない場合、nawk は標準エラーに診断メッセージを書き出した後、動作を停止します。

program オペランドまたは *progfile* オペランドのどちらかで指定されているプログラムが、有効な nawk プログラム (機能拡張説明で指定されている) でないとき、動作は未定義です。

注意事項

nawk は awk の新しいバージョンで、以前のバージョンにはない機能を備えています。次の大きなリリースにおいて、nawk が awk のデフォルトのバージョンとなる予定です。

フィールドを使用した場合、入力中の空白は 出力上には含まれなくなります。

数値と文字列との間で明示的な変換を行う機能はありません。式を数値として扱うには 0 を加算してください。 また文字列として扱うには、NULL 文字列 ("") を連結してください。

名前 | eqn, neqn, checkeq – 数学的記述のタイプセット

形式 | **eqn** [-d *xy*] [-f *n*] [-p *n*] [-s *n*] [*file*...]

neqn [*file*...]

checkeq [*file*...]

機能説明 | eqn および neqn は、数式を記述するのに便利な言語プロセッサです。eqn は troff(1) 用のプリプロセッサで、 troff の出力を印刷できる装置用に提供されています。一方、neqn は nroff(1) 用のプリプロセッサで、端末での出力用に提供されています。通常この 2 つのコマンドは、以下の形式で指定します。

```
example% eqn file . . . | troff
example% neqn file . . . | nroff
```

ファイル名を表す *file*引数を省略すると、eqn または neqn は標準入力から読み込みます。数式の開始を示すには、行の先頭に .EQ を記述します。同様に数式の終了は、行の先頭に .EN を記述して表します。この 2 つの行は変換されないので、センタリングや番号付けなどを行うマクロパッケージ中に定義しておくこともできます。また、一対の文字を「区切り記号」として設定し、区切り記号に囲まれたテキストを eqn 入力として処理させることもできます。

区切り記号や .EQ/.EN が存在しない、または対で指定されていない場合、checkeq はメッセージを出力します。

オプション | 以下のオプションが指定できます。

-d*xy* コマンド行引数で指定される数式の区切り記号として、文字 *x* と *y* を指定します。ただしこの方法よりも、.EQ と .EN の間で delim *xy* を使って区切り記号を指定する方法がより一般的です。*x* と *y* には同じ文字を指定することも可能です。テキスト中に delim off と記述すると、区切り記号は有効でなくなります。区切り記号にも .EQ と .EN にも囲まれていないテキストは、すべてそのまま渡されます。

-f*n* ドキュメント全体を通じて使用するフォントとして *n* を指定します。このグローバルフォントの設定は、ドキュメントの本文中に gfont *n* 命令を指定して変更することもできます。*n* はフォント指定です。

-p*n* 下付きおよび上付きの添字のサイズを、直前の文字サイズより *n* ポイントだけ小さくします。 -p オプションを省略すると、添字のサイズは 3 ポイント小さくなります。

-s*n* ドキュメント全体を通じて使用する文字サイズとして *n* を指定します。このグローバルサイズの設定は、ドキュメントの本文中に gsize *n* 命令を指定して変更することもできます。*n* はポイントサイズです。

オペランド | 以下のオペランドが指定できます。

file eqn または neqn によって処理される nroff のファイルまたは troff のファイル。

EQN 言語

この説明を nroff を使って端末画面に表示した場合、端末画面の制限から neqn による出力箇所は正確には表示できません。出力の正確な表示を確認するために、このページを印刷してご覧ください。

eqn 中のトークンは、中括弧、二重引用符、チルド、山型記号、空白文字、タブ、または復帰改行文字で区切られます。中括弧 { } は、グループ分けに用いられます。一般的には、たとえば *x* のような 1 つの文字が記述できる箇所であれば、中括弧で囲んだ複雑な記述を代わりに指定できます。チルド (~) は出力中における 1 文字分の空白を、山型記号 (^) は半文字分の空白を表します。

下付きおよび上付きの添字:
これらは、キーワード sub と sup を使って生成できます。

`x sub i` という記述の出力結果は x_i になります。

`a sub i sup 2` の出力は $a_i^{\ 2}$ になります。

`e sup {x sup 2 + y sup 2}` の出力は $e^{x^2+y^2}$ になります。

分数:
分数は、キーワード over で指定します。

```
a over b
```

の出力結果は、

$$\frac{a}{b}$$

となります。

平方根の式:
平方根の式は、キーワード sqrt で指定します。

```
1 over sqrt {ax sup 2 +bx+c}
```

の出力結果は、

$$\frac{1}{\sqrt{ax^2+bx+c}}$$

となります。

制限値:
キーワード from と to は、種々の指定における最小値と最大値を表します。

```
lim from {n→ inf } sum from 0 to n x sub i
```

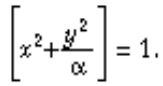
の出力結果は、

となります。

括弧:
大括弧、中括弧などを適切な高さで出力するには、左括弧に `left` を、右括弧には `right` をそれぞれ使用します。

```
left [ x sup 2 + y sup 2 over alpha right ] ~=~1
```

と記述すれば



という出力が得られます。なお、`right` 文節は省略することができます。キーワード `left` と `right` の直後に指定できる文字は、大括弧、中括弧、縦棒、上端と下端を表す `c` と `f`、何もない旨を示す `""` (対になるべき括弧のうち右括弧だけを使う場合に便利) です。

分数を縦に重ねる:
分数を縦に重ねるには、`pile`、`lpile`、`cpile`、または `rpile` を使用します。

```
pile {a above b above c}
```

という記述により

が出力されます。何重に積み重ねてもかまいません。文字を合わせる位置は、`lpile` は左詰め、`pile` と `cpile` はともにセンタリング (ただし縦方向の間隔が異なる)、そして `rpile` は右詰めとなります。

行列:
行列は `matrix`というキーワードで生成されます。

```
matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }
```

と記述すると、

x_i 1
y_2 2

が出力されます。カラムを右詰めにするには rcol を使用します。

文字の上下に付加する記号:
発音符記号のように文字の上下に付加する記号は、dot、dotdot、hat、tilde、bar、vec、dyad、underを使って指定できます。

```
x dot = f(t) bar
```
という記述の出力結果は、

$$\dot{x} = \overline{f(t)}$$

```
y dotdot bar ~=~ n under
```
の結果は、

$$\ddot{\bar{y}} = \underline{n},$$

```
x vec ~=~ y dyad
```
の結果は、

$$\vec{x} = \vec{\vec{y}}.$$

文字のサイズとフォント:
文字のサイズやフォントの変更は、size *n* または size ±*n*、roman、italic、bold、font *n* で指定します。ドキュメント全体を通じてグローバルに使用する文字サイズとフォントは、gsize *n* と gfont *n* をドキュメント中に指定するか、またはコマンド行引数の -s*n* と -f*n* を使って変更できます。

表示引数の位置:
一連の表示引数の位置をそろえることもできます。先頭の数式において、そろえたい表示引数の直前に mark と記述します。さらに後続の数式において、それと合わせたい表示引数の直前に lineup と記述します。

短縮形:
入力の短縮形を定義したり既存のキーワードを再定義するには、define を使用します。次に例を示します。

`define` *thing* `%` *replacement* `%`
これにより *thing* というトークンが新たに定義され、その後このトークンが現れるたびに *replacement* に置き換えられます。なお `%` の位置には、任意の文字 (ただし *replacement* に含まれていないもの) を指定できます。

キーワードと短縮形:
`sum int inf` のようなキーワード、および `>=` → や `!=` のような短縮形も処理されます。

ギリシャ文字:
ギリシャ文字は `alpha` または `GAMMA` のように、大文字・小文字のうち希望する方のつづりで出力できます。

数学用語:
`sin`、`cos`、`log` のような数学用語は自動的にローマン字体で出力されます。

\(bu (•) のような 4 文字からなる `troff`(1) のエスケープコードは、どこでも記述できます。二重引用符に囲まれた文字列 `"..."` は、そのまま渡されます。これによりキーワードをテキストとして入力でき、また (他の方法が使えないとき) `troff` との通信用に使うことができます。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目

`nroff`(1), `tbl`(1), `troff`(1), `attributes`(5), `ms`(5)

使用上の留意点

数字や括弧をボールドで出力したい場合、`bold "12.3"` のように引用符で囲んでください。

名前 | netscape – Solaris 版 Netscape Communicator の起動

形式 | **/usr/dt/bin/netscape** [options] [arguments]

機能説明 | Solaris 版 Netscape Communicator は、ブラウズ、電子メール、Web ベースのワープロ、チャット、グループスケジュール管理などの機能が統合されているコンポーネントで構成されています。これらを使用して、ユーザー同士がコミュニケーションをとったり、情報へアクセスしたり、情報を共有することができます。

オプション | オプション以外の引数は、ファイルまたは URL として解釈されます。

-component-bar	コンポーネントバーだけを表示します。
-composer	コマンド行で指定したすべての URL を Netscape Composer で開きます。
-discussions	Netscape メッセージセンターを表示します。
-display *dpy*	ディスプレイとして使用する X サーバーを指定します。
-dont-force-window-stacking	JavaScript の window.open() 属性の alwaysraised、alwayslowered、z-lock を無視します。
-dont-save-geometry-prefs	現在のセッションにおけるウィンドウのジオメトリ (位置とサイズ) 設定を保存しません。
-edit	「-composer」の説明を参照してください。
-geometry =*W*x*H*+*X*+*Y*	Netscape ウィンドウの位置とサイズを指定します。
-help	Netscape のコマンド行オプションを表示します。
-iconic	起動後に Netscape をアイコン化します。
-id *window-id*	-remote で指定したリモートコマンドを受け取る X ウィンドウを指定します。X ウィンドウを指定していない場合は、最初に見つかったウィンドウが使用されます。
-ignore-geometry-prefs	保存されているウィンドウジオメトリの設定を、現在のセッションで無視します。
-install	独自のカラーマップをインストールします。
-irix-session-management	IRIX セッションマネージャを有効にします。SGI システム上では、デフォルトで IRIX セッションマネージャが有効になって

	います。IRIX セッションマネージャは、他のプラットフォーム上でも使用することができ、IRIX システム以外のセッションマネージャと共に動作させることが可能です。詳細は、`-no-irix-session-management` を参照してください。
`-mail`	`-messenger` と同じです。
`-messenger`	Messenger Mailbox (Inbox) を表示します。
`-mono`	イメージを白黒で表示します。
`-ncols` *N*	`-install` を使用していない場合に、イメージに割り当てる色の最大数を設定します。
`-nethelp`	`NetHelp` (Netscape のオンラインヘルプシステム) を起動します。
`-news`	`-discussions` と同じです。
`-no-about-splash`	起動時のライセンスページを表示しません。
`-no-install`	デフォルトのカラーマップを使用します。
`-no-irix-session-management`	IRIX セッションマネージャを無効にします。`-irix-session-management` の説明を参照してください。
`-no-session-management`	セッションマネージャを無効にします。デフォルトでは、セッションマネージャは有効になります。`-session-management` の説明を参照してください。
`-noraise`	`-remote` の使用時に、リモートウィンドウを一番手前に表示しません。`-raise` および `-remote` の説明を参照してください。
`-raise`	`-remote` の使用時に、リモートウィンドウを一番手前に表示します。`-noraise` および `-remote` の説明を参照してください。
`-remote` *remote-command*	既存のプロセスに接続し、既存のプロセスを制御します。同一のコマンド行で複数回 `-remote` オプションを使用できます。コマンドの処理が失敗しない限り、コマンドが順次実行されます。現在実行中の Netscape プロセスがない場合、コマンドの処理は失敗します。その場合、エラーメッセージが標準エラー出力に報告され、コマンドはゼ

ロ以外のステータスで終了します。後述の「リモートアクション」の項と「使用例」を参照してください。

次のオプションは、-remote の動作をより細かく制御するためのオプションです。

-id *X_window_ID*	複数の Netscape Navigator ウィンドウが開いている場合に、制御するウィンドウをこのオプションによって指定します。このオプションを使用しない場合、最初に見つかったウィンドウが制御されます。「使用例」の項を参照してください。
-raise -noraise	-remote コマンド使用時に、Netscape ウィンドウを一番手前に表示するか表示しないかを指定します。デフォルトでは一番手前に表示します。 -raise および -noraise オプションは、引数 addBookmark および openURL と共に使用できます。「使用例」の項を参照してください。
-session-management	セッションマネージャを有効にします。デフォルトでは、セッションマネージャは有効になります。-no-session-management の説明を参照してください。
-version	バージョン番号とビルド日時を表示します。
-visual *id-or-number*	指定されたサーバー画像表示形式を使用します。
-xrm *resource-spec*	指定された X リソースを設定します。

リモートアクション

-remote 引数を付けて呼び出すと、Netscape Navigator はウィンドウを開かずに、既存のプロセスに接続し、そのプロセスを制御します。-remote スイッチに対する引数は、呼び出す Xt アクションと引数です。リモート制御は X の属性を使用して実装されているので、2 つのプロセスが同じマシン上で動作する必要はなく、またファイルシステムを共有する必要もありません。詳細については、http://home.netscape.com/newsref/std/x-remote.html を参照してください。

Netscape のアクション名はすべて、リソース名と同じです。たとえば、「Add Bookmark」メニュー項目に対応するアクション名を知るには、Netscape の「Add Bookmark」を調べます。すると、この文字列に設定されているリソースが

addBookmark であることがわかります。つまり、これがアクション名です。注: `Netscape` ファイルは、標準では、次のフルパス名です。
`/usr/dt/appconfig/netscape/lib/locale/C/app-defaults/Netscape`

通常の Xt と同様に「Actions in Translation」テーブルも使用できます。しかし、`-remote` オプションを使用すれば、次のように直接呼び出すことができます。

`netscape -remote 'addBookmark()'` このコマンドを実行すると、まるでユーザーが当該メニュー項目を選択したかのように、既存の Netscape Navigator プロセスが現在の URL をブックマークに追加します。

文書を開くには、次のように入力します。

```
netscape -remote 'openURL(http://home.netscape.com)'
```

引数なしでアクションを起動する場合と、対応するメニュー項目を選択した場合とで、実行結果は同じです。ただし、一部のアクションでは、以下の引数を渡すことができます。

addBookmark( )	現在表示している文書をブックマークリストに追加します。
addBookmark(*URL*)	指定した文書をブックマークリストに追加します。「使用例」の項を参照してください。
addBookmark(*URL*, *title*)	指定した文書とタイトルをブックマークリストに追加します。
mailto( )	`To:` フィールドが空のメール作成ウィンドウを開きます。
mailto(*a*, *b*. *c*)	指定したアドレスをデフォルトで `To:` フィールドに挿入します。
openFile( )	ファイルを指定するためのダイアログボックスを開きます。
openFile(*filename*)	指定したファイルを開きます。
openURL( )	URL を指定するためのダイアログボックスを開きます。
openURL(*URL*)	指定した文書を開きます。「使用例」の項を参照してください。
openURL(*URL*, new window)	新規ウィンドウを開き、指定した文書を表示します。
saveAs( )	URL を指定するためのダイアログボックを開きます。

saveAs(*output_file*)	指定したファイルへ HTML として保存します。
saveAs(*output_file*, *type*)	指定したファイルへ *type* 形式で (HTML、テキスト、または PostScript) 保存します。

終了ステータス

コマンド処理が失敗した場合、標準エラー出力にエラーメッセージが表示され、ゼロ以外の終了ステータスでコマンドが終了します。

使用例

次に、`-remote` コマンドオプションを使用する例を示します。 より詳しい説明および例については、 `http://home.netscape.com/newsref/std/x-remote.html` を参照してください。

例 **1** 開いている複数の Netscape ウィンドウのうち制御対象を選択する

```
example% netscape -id 0x3c00124 -remote 'openURL(http://www.sun.com)'
```

例 **2** ウィンドを一番手前に表示せずにブックマークを追加する

ウィンドウを一番上に表示せずにブックマークを追加し、URL を開いてからウィンドウを一番上に表示します。

```
example% netscape -noraise -remote 'addBookmark(http://www.sun.com)' \
   -raise -remote 'openURL(http://home.netscape.com)'
```

例 **3** 指定した文書をブックマークリストに追加する

```
example% netscape -remote 'addBookmark(http://www.sun.com)'
```

例 **4** 指定した文書を開く

```
example% netscape -remote 'openURL(http://www.sun.com)'
```

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	NSCPcom

関連項目

`attributes`(5)

詳細は、Netscape Communicator のオンラインヘルプを参照してください。

名前 | newform – テキストファイル形式の変更

形式 | **newform** [-s] [-i*tabspec*] [-o*tabspec*] [-b*n*] [-e*n*] [-p*n*] [-a*n*] [-f] [-c*char*] [-l*n*] [*filename...*]

機能説明 | newform は *filename* 引数で指定されたファイルから行を読み取り、 形式を変更して標準出力に書き込みます。 ファイル名の指定が省略された場合には、 標準入力から読み取ります。 形式の変更は、コマンド行オプションの指定に従って 行います。

-s オプションを除き、コマンド行オプションは 任意の順序で指定でき、また繰り返して指定することもできます。 オプションとオプションの間の任意の位置にファイル名を 記述することも可能です。 オプションは記述された順序で処理されます。つまり、 -e15 -l60 という指定と -l60 -e15 という指定とでは、得られる結果は異なります。 複数のファイル名が指定された場合、 オプションはそのすべてに適用されます。

オプション | 以下のオプションを指定できます。

-s
: 各行において、先頭から最初のタブまでの文字を切り捨て、 切り捨てた部分の先頭から最大 8 文字を各行の終わりに付加します。この切り捨てた部分 (最初のタブ文字は含まない) の長さが 8 文字を超える場合には、 8 文字目は * で表され、以降の文字はすべて捨てられます。 最初のタブは常に捨てられます。

  タブを含んでいない行を持つファイルに対して このオプションを指定すると、 エラーメッセージが出力されプログラムは終了します。 切り捨てられた部分の文字は、 その行に対して有効な他のオプションがすべて処理されるまで 内部的に保持されます。 その後、行の最後尾にその部分が付加されます。

  例として、先頭に数値、次に複数のタブ、さらに その後にテキストが書かれている行からなる ファイルが存在していると仮定します。 このファイルを変換して、 2 番目以降のタブを空白文字に置き換えたものから始まり、 その後にテキスト部分を置き、短ければ 72 カラムまで空白を詰め (または長い場合には 72 カラムまでで切り捨て)、先頭にあった数値を 73 カラムから置く、という形式にしたい場合には、 次のように指定します。

  ```
  newform -s -i -l -a -e filename
  ```

-i*tabspec*
: 入力タブ指定。入力行に含まれるタブの各々を、 *tabspec* 引数の指定に従って展開します。 *tabspec* の値としては、 tabs(1) で説明されているすべての形式が指定できます。また – という値も指定可能で、この場合 newform は、標準入力から読み込むデータの先頭行に タブの処理方法が記述されていると見なします (fspec(4) を参照)。 *tabspec* 引数を省略すると、デフォルト値の –8 が用いられます。 *tabspec* の値が –0 のとき、タブは存在しないことを意味します。 このときタブが検出されると、 –1 が指定されていたものとして処理されます。

-o*tabspec*	出力タブ指定。 *tabspec* 引数の指定に従って、空白文字をタブに置き換えます。 *tabspec* に指定できる値は、上記の –i*tabspec* オプションと同じです。 *tabspec* 引数を省略すると、デフォルト値の –8 と見なされます。 *tabspec* の値が –0 のとき、空白文字はタブに変換されずにそのまま出力されます。
-b*n*	行の長さが有効な長さ (–l*n* オプションの説明を参照) を超えている場合、先頭から *n* 文字分を破棄します。 *n* なしで -b オプションを指定すると、デフォルトとして 行の長さが有効な長さと等しくなるように、 必要な数の文字を切り捨てます。 このオプションの便利な使い方として、 COBOL プログラムの各行の行番号を削除する例を以下に示します。 `newform -l1 -b7 filename`
-e*n*	-b*n* が先頭部分を破棄するのに対し、この -e*n* は終端部分を破棄します。それ以外は -b*n* と同じです。
-p*n*	行の長さが有効な長さに満たないとき、行の先頭に付加する 文字の数を指定します (文字の種類は -c*char* オプションの説明を参照)。 *n* なしで -p オプションを指定すると、デフォルトとして 行の長さが有効な長さと等しくなるように 必要な数の文字を付加します。
-a*n*	-p*n* が先頭部分に文字を付加するのに対し、この -a*n* は終端部分に文字を付加します。それ以外は -p*n* と同じです。
-f	最初の出力行として、タブ指定形式を表す行を出力します。 その内容は -o オプションの指定に対応しており、 -o が複数個指定されていれば最後のものに対応しています。 -o が 1 つも指定されていなければ、デフォルトのタブ形式の –8 が出力されます。
-c*char*	-p*n* や -a*n* オプションで付加する文字として、 *char* が示す文字を使用します。 *char* のデフォルト値は空白です。
-l*n*	行の有効な長さを *n* 文字とします。 *n* なしで -l オプションを指定すると、デフォルトとして 72 が用いられます。 -l そのものを省略すると、デフォルトとして 80 文字が有効な長さとなります。なお、個々の タブ文字およびバックスペース文字は、それぞれ 1 文字と数えられます。ただし -i オプションを使って、タブをいくつかの空白文字に 展開することも可能です。 -b オプションを有効に使用する目的で、 行の有効な長さをファイル中のどの行の長さよりも短い値に 設定したい場合、 –l1 と指定してください。

オペランド　以下にオペランドを示します。

filename	入力ファイル名

終了ステータス　以下の終了ステータスが返されます。

0	正常終了
1	エラーが発生した

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目　csplit(1), tabs(1), fspec(4), attributes(5)

診断　いずれの診断メッセージも致命的なエラーを表します。

usage: . . .
  newform コマンド行に不正なオプションが指定されている

"not -s format"
  タブが含まれていない行を検出した

"can't open file"
  ファイルがオープンできない

"internal line too long"
  内部作業バッファ中で行を展開したら、長さが 512 文字を超えた

"tabspec in error"
  タブ指定の形式が誤り。またはタブ位置の記述が 昇順になっていない

"tabspec indirection illegal"
  ファイル(または標準入力) から読み込まれた *tabspec* に、他のファイル (または標準入力) を参照する *tabspec* が含まれていた

注意事項　通常 newform は、物理的な文字だけを認識します。ただし -i および -o に関しては、各行を適切な論理カラム位置に合わせて出力するために、 バックスペース文字も認識します。

-i- または -o- により *tabspec* を標準入力から読み込むように指定した場合でも、newform はユーザーに対してプロンプトを表示しません。

-f オプションが指定され、最後の -o オプションが -o- と記述されていて、その前に -o- または -i- という指定が記述されている場合、 タブ指定形式を示す出力行の内容は不正確になります。

名前 | newgrp – 新たなグループへのログイン

コマンド | **/usr/bin/newgrp** [-| -l] [*group*]

sh 組み込み | **newgrp** [*argument*]

ksh 組み込み | ***newgrp** [*argument*]

コマンド | newgrp コマンドは、ユーザーの実および実効グループ ID を変更してユーザーを新たなグループへログインさせます。 ユーザーはログイン状態を保ち、またカレントディレクトリも変わりません。 newgrp を実行すると、常にシェルは新たなものに入れ替えられます。 これはコマンドが、グループ名不明のためにエラーで終了した場合でも同様です。

エクスポートされない変数は、 NULL またはデフォルト値にリセットされます。 エクスポートされた変数は、その値を保持します。 PS1、 PS2、 PATH、 MAIL、 HOME のようなシステム変数は、システムやユーザーにより エクスポートされない限り、デフォルト値にリセットされます。 たとえば、ユーザーが主プロンプト文字列として、デフォルトの $ の代わりに PS1 を使っていて、 PS1 をエクスポートしなかった場合、ユーザーの PS1 はデフォルト文字列の $ に 設定されます。これは newgrp がエラーで終了したとしても同様です。 なおシェルコマンドの export (sh(1) と set(1) を参照) は、変数をエクスポートするための方法で、これを使うと、 新たなシェルを呼び出す際に割り当て済みの変数値を保持できます。

オペランドとオプションをすべて省略した場合、 newgrp は、ユーザーのグループ ID (実 ID と実効 ID) を、ユーザーのパスワードファイルエントリに指定されているグループに 戻します。この方法を使うと、いったん newgrp コマンド実行により変更した状態を、元に戻すことができます。

グループがパスワードを持っていて、 ユーザーがそのグループのメンバーである旨が /etc/group ファイルに定義されていない場合、パスワードの入力が要求されます。グループ用にパスワードを生成する唯一の方法は、まず passwd(1) を実行し、さらに /etc/shadow から /etc/group へパスワードを「カット & ペースト」することです。 グループ用パスワードは、今日ではあまり使われません。

sh 組み込み | exec newgrp *argument* を実行した場合と同じです。 *argument* は newgrp コマンドのオプションまたはオペランド、もしくはその両方を表します。

ksh 組み込み | exec /bin/newgrp *argument* を実行した場合と同じです。 *argument* は newgrp コマンドのオプションまたはオペランド、もしくはその両方を表します。

1 つまたは 2 つの * (アスタリスク) が先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。

4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

オプション　以下のオプションを指定できます。

`-l | –`　環境を、ユーザーが実際に新グループのメンバーとしてログインした 場合に得られるものに変更します。

オペランド　以下のオペランドを指定できます。

group　グループデータベースから得られるグループ名、 または負でない数値のグループ ID。実グループ ID と実効グループ ID として新たに用いる ID を指定します。 *group* が負でない数値で、グループ名としてグループデータベースに 存在している (`getgrnam`(3C) を参照) 場合、そのグループ名に対応した数値グループ ID が新たなグループ ID として用いられます。

argument　`sh` と `ksh` でのみ使用。 `newgrp` コマンドのオプションまたはオペランド、もしくはその両方。

環境　`newgrp` の実行に影響を与える環境変数 `LC_CTYPE` 、 `LC_MESSAGES` 、 `NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス　`newgrp` が新たなシェル実行環境を生成できた場合、グループ ID の変更が正常終了したか否かに関わらず、終了ステータスは シェルの終了ステータスと同じになります。環境が生成できなかったとき、以下の終了ステータスが返されます。

`>0`　エラーが発生した

ファイル　`/etc/group`　システムのグループファイル

`/etc/passwd`　システムのパスワードファイル

属性　次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目　`login`(1), `ksh`(1), `set`(1), `sh`(1), `intro`(2), `getgrnam`(3C), `group`(4), `passwd`(4), `attributes`(5), `environ`(5)

名前 | news – 新規項目の表示

形式 | **news** [-a] [-n] [-s] [*items*]

機能説明 | news は、絶えずユーザーに現在のイベントを通知するときに使用します。 規則上、これらのイベントは /var/news ディレクトリにファイルで記述されます。

引数なしで起動すると、 news は /var/news 内にある現在のファイルすべての内容を、 最も新しいものから順に、各々の前に適切なヘッダを付けて表示します。 news は、「基準時刻」を、 .news_time という名前のファイルの変更日付として管理します。 このファイルはユーザーのホームディレクトリに格納されます (ディレクトリの ID は $HOME という環境変数により決定されます) 。 この「基準時刻」よりも新しいファイルだけが 「新規」と見なされます。

オプション | -a　news は、「基準時刻」とは無関係にすべての項目を表示します。 この場合、格納した時間は変更しません。

-n　news は、「新規」の内容を表示せず、かつ格納した時間を変更せずに、「新規」の項目の名前を報告します。

-s　news は、「新規」の項目の名前も内容も表示せず、 かつ格納した時間を変更せずに、 存在する「新規」の項目の数を報告します。 そのような news の起動を .profile ファイルまたはシステムの /etc/profile に挿入すると役立ちます。

他の引数はすべて、表示すべき特定の新規項目と見なされます。

新規項目の表示中に *delete* を入力すると、表示が停止し、次の項目を起動します。 *delete* を入力してから 1 秒以内に別の *delete* を入力すると、プログラムが終了します。

環境 | news の実行に影響を与える環境変数 LC_CTYPEについての詳細は、 environ(5) を参照してください。

ファイル | /etc/profile

/var/news/*

$HOME/.news_time

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 | profile(4), attributes(5), environ(5)

名前 | mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式 | **mailp** [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明 | `mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション | 次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D` 出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。

`-F` メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。

`-h` バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。

`-l` 横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。

`-P` *printer* `-d` オプションを指定した場合と同じです。

`-s` *subject* *subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename 読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`1` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

名前 | nice – コマンドを変更されたスケジューリング優先順位で実行

形式 | **/usr/bin/nice** [-*increment* | -n *increment*] *command* [*argument*...]

/usr/xpg4/bin/nice [-*increment* | -n *increment*] *command* [*argument*...]

csh 組み込みコマンド | **nice** [-*increment* | +*increment*] [*command*]

機能説明 | nice ユーティリティは、 異なるシステムのスケジューリング優先順位で 動作するように *command* を実行します。 priocntl(1) コマンドは、 スケジューラ機能とのより汎用的なインタフェースです。

起動するプロセス (通常は、ユーザーのシェル) は、 nice コマンドをサポートする スケジューリングクラスで 実行されていなければなりません。

C シェル (csh(1) 参照) を使用している場合、 コマンドのフルパス名を指定する必要があります。 指定しないと、 csh 組み込みコマンドの nice が実行されます。 以下の「 csh 組み込みコマンド 」の項を参照。

/usr/bin/nice | nice が引数のあるコマンドを実行する場合、 デフォルトのシェル /usr/bin/sh が使われます (sh(1) 参照) 。

/usr/xpg4/bin/nice | nice が引数のあるコマンドを実行する場合、/usr/xpg4/bin/sh が使われます (ksh(1) 参照) 。

csh 組み込みコマンド | nice という csh の組み込みコマンドもあります。こちらの動作は、ここで説明した nice とは異なります。詳しくは csh(1) を参照してください。

オプション | 以下のオプションを指定できます。

–increment | –n *increment* | *increment* (増分値) は 1 から 19 の範囲で指定してください。 省略されている場合は、 増分値は 10 になります。 19 を超える増分値を *increment* に指定すると、 19 とみなされます。 スーパーユーザーは、たとえば – –10 のように、 負数の *increment* を使用することにより、 通常よりも高い優先順位でコマンドを実行することができます。 特権を持たないユーザーが負数の増分値を指定した場合は無視されます。

オペランド | 以下のオペランドを指定できます。

command | 呼び出すコマンドの名前。 *command* に組み込みコマンド (shell_builtins(1) を参照) を指定した場合、処理の結果は保証されません。

argument | *command* を呼び出す際に引数として与える文字列。

環境 | nice の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | *command* で指定したコマンドが呼び出された場合、 そのコマンドの終了ステータスが nice の終了ステータスとなります。呼び出されなかった場合には、 nice は以下の終了ステータスを返します。

1-125	エラーが発生した
126	コマンドは見つかったが呼び出すことができなかった
127	コマンドが見つからなかった。

属性　次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/nice

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/nice

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目　csh(1), ksh(1), nohup(1), priocntl(1), sh(1), shell_builtins(1), nice(2), attributes(5), environ(5), XPG4(5)

名前 | nm – オブジェクトファイルのネームリストを表示

形式 | **/usr/ccs/bin/nm** [-ACDhlnPprRsTuVv] [-efox] [-g | -u] [-t *format*] *file...*

/usr/xpg4/bin/nm [-ACDhlnPprRsTuVv] [-efox] [-g | -u] [-t *format*] *file...*

機能説明 | nm ユーティリティは、*file* オペランドで指定された ELF オブジェクトファイルの各々に対して、 シンボルテーブルを表示します。

正しいファイルが指定されたにもかかわらず、そのファイルの シンボリック情報が存在しないとき、 nm はそのことを報告します。これはエラーとは見なされません。

オプション | 以下のオプションを使って、 nm の出力内容を制御できます。

`-A` 各行に、オブジェクトの完全パス名とライブラリ名を出力します。

`-C` C++ シンボル名を、表示する前に複合化します。

`-D` SHT_DYNSYM シンボル情報を表示します。 これは `ld.so.1` によって使用されるシンボルテーブルで、 ストリップされた動的実行可能ファイルにも 存在します。 デフォルトでは SHT_SYMTAB シンボルテーブルが表示されます。

`-e` 後述の「注意事項」を参照してください。

`-f` 後述の「注意事項」を参照してください。

`-g` 外部 (グローバル) シンボル情報だけを出力します。

`-h` 見出しのデータを表示しません。

`-l` WEAK と GLOBAL のシンボルを、 WEAK シンボルのキー文字の後に * を付加することにより区別します。

`-n` 外部シンボルを、名前順にソートして出力します。

`-o` シンボルの値とサイズを、10 進数ではなく 8 進数で出力します (`-t o` と同じ)。

`-p` 簡単に解析できる、簡易形式の出力を生成します。 各シンボル名の前には、シンボルの値 (未定義なら空白) と、シンボルの状態を示す文字が表示されます。 その文字とは以下のいずれかです。

- `A` 絶対シンボル
- `B` bss (初期化されていないデータスペース) シンボル
- `C` COMMON シンボル
- `D` データオブジェクトシンボル
- `F` ファイルシンボル
- `N` タイプのないシンボル

L　スレッドローカル記憶領域シンボル

S　セクションシンボル

T　テキストシンボル

U　未定義

シンボルのバインディング属性の意味は次のとおりです。

LOCAL　このキー文字は小文字

WEAK　このキー文字は大文字。また -l 修飾子が指定されていると、大文字のキー文字の後に * が付加される

GLOBAL　このキー文字は大文字

-P　後述の「標準出力」の項で述べるような、 移植性のある出力形式で情報を出力します。

-r　各出力行に、オブジェクトファイルまたは アーカイブ名を付加します。

-R　オブジェクトファイルとシンボル名の前に、 アーカイブ名 (もし存在していれば) を出力します。 -r オプションが同時に指定されている場合には、この -R オプションは無視されます。

-s　セクションインデックスの代わりに セクション名を出力します。

-t*format*　個々の数値を指定された形式で出力します。 出力形式は、オプション引数の *format* により 1 文字で指定します。

d　オフセットを 10 進数で出力 (デフォルト)

o　オフセットを 8 進数で出力

x　オフセットを 16 進数で出力

-T　後述の「注意事項」を参照してください。

/usr/ccs/bin/nm　-u　未定義のシンボルだけを出力します。

/usr/xpg4/bin/nm　-u　未定義の各シンボルについての一覧を出力します。 後述の「出力」を参照してください。

-v　外部シンボルを、値の順にソートして出力します。

-V　実行した nm コマンドのバージョンを標準エラー出力に書き出します。

-x　シンボルの値とサイズを、10 進数ではなく 16 進数で出力します (-t x と同じ)。

オプションはいくつでも指定できます。 複数個指定する場合その順序は問いません。また、 コマンド行上のどこに記述しても構いません。 矛盾する組み合せでオプションを指定すると (たとえば -v と -n または、 -o と -x)、最初のオプションが有効となり、 2 番目は無視されて警告メッセージが出力されます (例外は -R オプションを参照)。

オペランド

以下のオペランドを指定できます。

file オブジェクトファイル、実行可能ファイル、または オブジェクトファイルライブラリのパス名

標準出力

各シンボルに関して次の情報を出力します。

Index シンボルのインデックスです (インデックスは大括弧[ ] で囲んで表示されます)。

Value シンボルの値で、以下のいずれかです。

- 再配置可能ファイル中で定義されている シンボルのセクションオフセット
- セクションインデックスが SHN_COMMON であるシンボルの境界合わせ上の制約
- 実行可能および動的ライブラリファイル中の仮想アドレス

Size 対応するオブジェクトのサイズで、単位はバイトです。

Type シンボルのタイプで、以下のいずれかです。

NOTYPE	タイプは指定されていない
OBJECT	配列や変数のようなデータオブジェクト
FUNC	関数あるいは他の実行可能コード
REGI	レジスタシンボル (SPARC の場合のみ)
SECTION	セクションシンボル
FILE	ソースファイル名
COMMON	初期化されていない共通ブロック
TLS	スレッドローカル記憶領域に関連する変数

Bind シンボルのバインディング属性で、以下のいずれかです。

LOCAL	このシンボルの有効範囲は、 その定義を含んでいるオブジェクトファイルに限定されている
GLOBAL	このシンボルは、結合されているすべてのオブジェクトファイルから見ることができる
WEAK	このシンボルは、実質的にはグローバルシンボルだが、 GLOBAL より

も優先順位は低い

Other	将来の拡張用に予約されているフィールドで、 現在は 0 を含んでいます。
Shndx	シンボル定義のもととなる、 セクションヘッダテーブルインデックスです。 ただし例外として、次の 3 つの値のいずれかが 表示されることがあります。
ABS	シンボルの値は再配置しても変わらないことを表す
COMMON	未割付ブロックを表し、値は境界合わせ用の制約を表す
UNDEF	未定義のシンボルを表す
Name	シンボル名です。
Object Name	-A オプションが指定されているとき、オブジェクトまたはライブラリの 名前を表します。

-P オプションが指定された場合には、上記の情報が以下に説明する 移植性のある形式で出力されます。この形式には 3 種類あり、 -t d 、 -t o 、 -t x のどれが指定されたかによって使い分けられます。

"%s%s %s %d %d\n", <*library/object name*>, *name*, type, *value*, size "%s%s %s %o %o\n", <*library/object name*>, *name*, type, *value*, size "%s%s %s %x %x\n", <*library/object name*>, *name*, type, *value*, size

このうち <*library/object name*> は次のような形式です。

- -A が指定されていなければ <*library/object name*> は空の文字列です。
- -A が指定され、対応する *file* オペランドがライブラリ名を示していない場合は、次のようになります。

  "%s: ", *file*

- -A が指定され、対応する *file* オペランドがライブラリ名を示している場合には、次のようになります。 なお <*object file*> は、当該シンボルを含んでいるライブラリのオブジェクトファイル名です。

  "%s[%s]: ", *file*, <*object file*>

-A が省略されたとき、 *file* オペランドが複数指定されているか、あるいは 1 つだけ指定された *file* がライブラリを表していれば、 nm は、後続のシンボルを含むオブジェクトを識別する行を、 そのシンボルを含む行の前に出力します。出力形式は次のとおりです。

- 対応する *file* オペランドがライブラリ名を示していないとき

"%s:\n", *file*

- 対応する *file* オペランドがライブラリ名を示しているとき (なお <*object file*> は、後続のシンボルを含むライブラリ中のファイル名)

  "%s[%s]:\n", *file*, <*object file*>

-P が指定され、 -t が省略された場合には、出力形式は -t x が指定された場合と同一になります。

環境

nm の実行に影響を与える環境変数 LC_COLLATE 、 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/ccs/bin/nm

属性タイプ	属性値
使用条件	SUNWbtool

/usr/xpg4/bin/nm

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目

ar(1), as(1), dump(1), ld(1), ld.so.1(1), a.out(4), ar(3HEAD), attributes(5), environ(5), XPG4(5)

注意事項

以下のオプションは、オブジェクトファイルの形式が 変わってしまったため現在ではあまり使用されていません。 将来のリリースで削除されることになります。

-e　外部シンボルと静的シンボルだけを出力します。 現在、シンボルテーブルには、この 2 つしか含まれていません。 自動シンボルは、シンボルテーブル内には存在しなくなり、 cc -g で生成されるデバッグ情報に出力されます。 この情報は、 dump(1) を使って検査することができます。

-f　完全な出力を生成します。なお .text や .data などの冗長シンボルは、現在はもう存在していません。 したがってこの完全出力の生成は、 デフォルト出力の生成と同じ意味になります。

-T　デフォルトでは、 nm はシンボルの名前全体を出力します。 シンボル名の出力カラムの位置が最終カラムに移動したので、 オーバフローの心配はなく、 シンボル名を途中で切り捨てる必要はなくなりました。

名前 | nohup – ハングアップおよび停止の影響を受けないコマンドの実行

形式 | **/usr/bin/nohup** *command* [*argument*...]

/usr/bin/nohup -p [-Fa] *pid* [*pid*...]

/usr/bin/nohup -g [-Fa] *gpid* [*gpid*...]

/usr/xpg4/bin/nohup *command* [*argument*...]

機能説明 | nohup ユーティリティは、 *command* で示すコマンドを、指定された引数とともに呼び出します。 *command* が呼び出されたとき、 nohup は、そのプロセスが SIGHUP シグナルを無視するように設定します。

-p または -g フラグを指定して実行すると、 nohup は、プロセス ID またはグループ ID のリストで指定された実行中のプロセスが、ハングアップおよび停止の影響を受けないように設定します。

指定したコマンドが実行に時間を要する場合、 ユーザーがコマンド終了前にログアウトしたい場合は、 nohup ユーティリティを使うと便利です。シェルが 終了するときに、システムはその子プロセスに SIGHUP シグナルを送信します。それにより、デフォルト では子プロセスが終了します。 停止、実行中、バックグラウンドにある すべてのジョブは、その呼び出しが nohup コマンドによるものだった場合、 もしくはプロセスがプログラム上 SIGHUP を無視するように選択されていた場合、 SIGHUP を無視して続行します。

/usr/bin/nohup
: /usr/bin/nohup によって実行されるプロセスは、 SIGHUP (ハングアップ) および SIGQUIT (中止) シグナルを受け付けません。

/usr/bin/nohup -p [-Fa]
: ID に指定されたプロセスは、 SIGHUP (ハングアップ) および SIGQUIT (中止) シグナルを受け付けません。また、制御端末への出力は、すべて nohup.out ファイルにリダイレクトされます。-F を指定すると、nohup は、各プロセスに制御を強制します。-a を指定すると、プロセスが SIGHUP (ハングアップ) または SIGQUIT (中止) のシグナルハンドラをインストールしている場合でも、これらのシグナル設定を変更します。

/usr/bin/nohup -g [-Fa]
: ID によって指定たプロセスと同じグループに属するすべてのプロセスは、SIGHUP (ハングアップ) および SIGQUIT (中止) シグナルを受け付けません。また、制御端末への出力は、すべて nohup.out ファイルにリダイレクトされます。-F を指定すると、nohup は、各プロセスに制御を強制します。-a を指定すると、プロセスが SIGHUP (ハングアップ) および SIGQUIT (中止) シグナルのどちらかに対してハンドラをインストールした場合でも、これらのシグナル設定を変更します。

/usr/xpg4/bin/nohup
: /usr/xpg4/bin/nohup によって実行されるプロセスは、 SIGHUP シグナルを受け付けません。

nohup ユーティリティは、プロセスに対し SIGTERM (終了) シグナルを受け付けないようには設定しないので、 プロセス自身があるいはシェルがそのように設定しないかぎり、プロセスは SIGTERM シグナルを受信します。

nohup.out が現在のディレクトリに書き込み可能でない場合、$HOME/nohup.out に出力先が変更 (リダイレクト) されます。 ファイルが生成されると、読み取りと書き込みのアクセス権 (600 、詳しくは chmod(1) を参照) が与えられます。 標準エラーは、端末の場合出力先を標準出力に変更され、 その他の場合は変更されません。 nohup が実行するプロセスの優先度は変更されません。

オプション

以下のオプションを指定できます。

-a　どのような場合でも、対象プロセスの設定を変更します。このオプションは、-p または -g と一緒に使用した場合にのみ有効です。

-F　制御を強制します。別のプロセスが制御していても、対象のプロセスを捕捉します。このオプションは、-p または -g と一緒に使用した場合にのみ有効です。

-g　リストしたプロセスグループに対して処理を実行します。-p オプションと同時には使用できません。

-p　リストしたプロセスに対して処理を実行します。-g オプションと同時には使用できません。

オペランド

以下のオペランドを指定できます。

pid　nohup -p が使用する、10 進数のプロセス ID。

pgid　nohup -g が使用する、10 進数のプロセスグループ ID。

command
command　呼び出すコマンドの名前。このオペランドに shell_builtins(1) ユーティリティを指定した場合の処理結果は定義されていません。

argument　*command* オペランドのコマンドを呼び出す際に引数として与える文字列。

使用例

例 **1** nohup をパイプラインまたはコマンドのリストに適用する

nohup をパイプラインまたはコマンドのリストに 適用した方が望ましい場合がよくあります。 これは、シェルスクリプトと呼ばれる、 パイプラインおよびコマンドリストを 単一のファイルに格納する方法でしか実現されません。 そうすれば、以下を実行できます。

```
example$ nohup sh file
```

これで、 nohup は *file* 内のすべてに適用されます。 *file* というシェルスクリプトを頻繁に実行する予定の場合、 *file* 実行権を指定すれば、 sh の入力が少なくて済みます。

アンパサンド '&' を追加すると、 file の内容がバックグラウンドで実行され、割り込みも無視されます (sh(1) 参照) 。

例 **1** nohup をパイプラインまたはコマンドのリストに適用する　*(続き)*

```
example$ nohup file &
```

例 **2** nohup -p をプロセスに適用する

```
example$ long_running_command &
example$ nohup -p `pgrep long_running_command`
```

例 **3** nohup -g をプロセスグループに適用する

```
example$ make &
example$ ps -o sid -p $$
   SID
81079
example$ nohup -g `pgrep -s 81079 make`
```

環境

nohup の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 PATH 、 NLSPATH についての詳細は、 environ(5) を参照してください。

HOME　ユーザーのホームディレクトリのパス名を定義します。 出力ファイル nohup.out が現在のディレクトリ内に生成できない場合、 nohup コマンドは HOME が示すディレクトリを使ってファイルを作成します。

終了ステータス

以下の終了ステータスが返されます。

126　*command* で示すコマンドユーティリティは 見つかったが呼び出すことができなかった。

127　nohup 中でエラーが発生した、または *command* で示すコマンドが見つからなかった。

上記以外の場合、 *command* オペランドの終了ステータスが nohup の終了ステータスになります。

ファイル

nohup.out　標準出力が端末で現在のディレクトリが書き込み可能な場合に、 nohup 実行 (結果) の出力用に使われるファイル

$HOME/nohup.out　標準出力が端末で現在のディレクトリが書き込み可能でない場合に、 nohup 実行 (結果) の出力用に使われるファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/nohup

属性タイプ	属性値
使用条件	SUNWcsu

CSI	対応済み

/usr/xpg4/bin/nohup

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目 batch(1), chmod(1), csh(1), ksh(1), nice(1), pgrep(1), proc(1), ps(1), sh(1), shell_builtins(1), signal(3C), proc(4), attributes(5), environ(5), XPG4(5)

警告 ログアウトをしようといた時に ログインシェルとして Korn シェル (ksh(1)) を実行し、 nohup されたジョブを実行していた場合、 次のような警告メッセージが表示されます。

```
You have jobs running.
```

ログアウトを有効にするために ログアウトを 2 回行う必要があります。 バックグラウンドジョブは実行し続けます。

注意事項 nohup という C シェル (csh(1)) 組み込みコマンドは、終了シグナル SIGHUP の影響が及ばないようにします。 しかし、出力先を nohup.out に変更しません。 csh (1) を使用すると、'&' で実行するコマンドは、バックグラウンドにいる間は、自動的に HUP シグナルの影響を受けなくなります。

nohup は、コマンドシーケンスを認識しません。 以下のコマンドの例では、 nohup ユーティリティが適用するのは command1 だけです。

```
example$ nohup command1; command2
```

以下のコマンドは構文的に正しくありません。

```
example$ nohup (command1; command2)
```

名前 | jobs, fg, bg, stop, notify – プロセスの実行の制御

sh **jobs** [-p | -l] [% *job_id*...]

jobs -x *command* [*arguments*]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

csh **jobs** [-l]

fg [% *job_id*]

bg [% *job_id*...]

notify [% *job_id*...]

stop % *job_id*...

stop *pid*...

ksh **jobs** [-lnp] [% *job_id*...]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

sh ジョブ制御が有効なとき、Bourne シェルに組み込まれた jobs は、停止中またはバックグラウンドで実行中の すべてのジョブを表示します。 %*job_id* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが表示されます。次のオプションを使って、ジョブに関する表示を変更できます。

-l ジョブのプロセスグループ ID および作業ディレクトリを表示します。

-p ジョブのプロセスグループ ID のみを表示します。

-x *command* または *argument* 中に見つかった *job_id* を、対応するプロセスグループ ID に置き換え、*command* に *argument* を渡して実行します。

シェルを jsh として呼び出すと、sh の説明で述べたすべての機能に加えて、ジョブ制御が可能になります。通常、ジョブ制御は対話型シェルに対してだけ可能です。通常、非対話型シェルは、ジョブ制御の機能 を使用しません。

ジョブ制御が可能なとき、ユーザーが端末から入力したコマンドまたはパイプラインは、すべて *job_id* と呼ばれます。どのジョブも、必ずフォアグラウンド、バックグラウンド、または停止のいずれかの状態にあります。 これらの用語の定義を次に示します。

1. フォアグラウンド状態にあるジョブは、 制御している端末への読み取りおよび 書き込みアクセス権を持っています。
2. バックグラウンド状態にあるジョブは、 制御している端末への読み取りアクセスを拒否されていますが、 条件付き書き込みアクセス権は持っています (`stty`(1) を参照)。
3. 停止ジョブは保留状態に置かれたジョブであり、通常は `SIGTSTP` シグナルにより、この状態になります (`signal`(3HEAD) を参照)。

シェルが起動するすべてのジョブには、 ジョブ番号 (*job-id number*) と呼ばれる正の整数が割り当てられます。シェルはこの番号を把握し、 特定のジョブを示す識別子として使用します。また、シェルは現在 (*current*) および前回 (*previous*) のジョブも把握しています。現在のジョブとは、最後に起動または再起動されたジョブです。 前回のジョブとは、その直前のジョブです。

ジョブ識別子の正しい構文は次のような形式です。

`%`*job_id*

job_id は、次のいずれかの形式で指定できます。

`%` または `+`	現在のジョブ
–	前回のジョブ
`?`<*string*>	*string* を含むコマンド行 (一意に表す) に対応したジョブ
n	ジョブ番号が *n* のジョブ
pref	コマンド名の先頭が *pref* のコマンド。 たとえば `ls -l` *name* がバックグラウンドで実行中だった場合、`% ls` と指定すればこのコマンドを示すことができます。*pref* は、引用符で囲まない限り、空白文字を含めることができません。

ジョブ制御が有効なとき、`fg` は中断しているジョブの実行を フォアグラウンドで再開します。 またバックグラウンドで実行中のジョブを フォアグラウンドに移動します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

ジョブ制御が有効なとき、`bg` は中断されているジョブの実行を バックグラウンドで再開します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

`stop` は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (`ps`(1) を参照) 。

csh C シェルに組み込まれた `jobs` は、引数なしでジョブ制御下で活動中のジョブを一覧表示します。

`-l` 通常の情報の他に、プロセス ID を表示します。

シェルは、番号の付いた *job_id* を各コマンドシーケンスと対応付けて、バックグラウンドで動作中のコマンド、または TSTP シグナル (通常は Control-Z) によって停止したコマンドの動作を追跡します。コマンドまたはコマンドシーケンス (セミコロンで区切られたリスト) をメタキャラクタ & を使用してバックグラウンドで起動した場合、 シェルは角括弧で囲まれたジョブ番号と 関連するプロセス番号のリストを表示します。以下に例を示します。

[1] 1234現在のジョブリストを見るには、組み込みコマンド jobs を使用します。最後に停止したジョブ (停止したジョブがない場合は、 最後にバックグラウンドに投入されたジョブ) を「現在のジョブ」といい、 '+' で示します。前のジョブは '-' で示します。 現在のジョブが終了したりフォアグラウンドに移された場合、 前のジョブが新しく現在のジョブになります。

ジョブの操作方法については、組み込みコマンド bg、 fg、 kill、 stop、 % の説明を参照してください。

ジョブの参照は '%' で始まります。 パーセント記号だけの指定は、現在のジョブを示します。

% %+ %%	現在のジョブ
%-	前のジョブ
%*j*	'kill -9 %*j*' のようにジョブ *j* を参照します。 *j* はジョブ番号、またはジョブを起動した コマンド行を一意に表す文字列です。 たとえば 'fg %vi' は、停止した vi ジョブをフォアグラウンドに移します。
%?*string*	*string* を含むコマンド行 (一意に表す) に対応したジョブを指定します。

バックグラウンドで動作中のジョブは、端末からの読み取り時に停止します。 バックグラウンドジョブは、通常出力を生成しますが、'stty tostop' コマンドを使用して抑止することも可能です。

fg は現在のジョブまたは指定された *job_id* をフォアグラウンドへ移します。

bg はバックグラウンドで、現在のジョブ または指定されたジョブを実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

notify は現在のジョブまたは指定されたジョブの状態が変わったとき、 その旨非同期にユーザーに知らせます。

ksh jobs は、現在のシェル環境で開始されたジョブの状況を表示します。 jobs がジョブの終了を報告したとき、シェルはそのジョブのプロセス ID を、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除します。

特定のジョブの報告だけが必要なら、*job_id* を使ってジョブを指定します。*job_id* を 1 つも指定しないと、全ジョブに関する情報が出力されます。

以下のオプションは、 `jobs` の出力を変更または拡張するために使用します。

`-l` (文字のエル) 個々のジョブに関して詳細な情報を出力します。具体的には、 ジョブ番号、現在のジョブ、プロセスグループ ID 、状態、ジョブを生成したコマンドを出力します。

`-n` 前回通知を受けた後に停止または終了した ジョブだけを表示します。

`-p` 選択されたジョブのプロセスグループリーダのプロセスグループ ID だけを出力します。

デフォルトでは、`jobs` は、停止しているすべてのジョブの状態、実行中のバックグラウンドジョブの 状態、そして状態が変わったのにシェルによりまだ報告されていない すべてのジョブの状態を表示します。

`set` コマンドの `monitor` オプションを有効にすると、対話型シェルが `job` を各パイプラインと関連付けます。 このオプションは、`jobs` コマンドが表示する現在のジョブのテーブルを維持し、これらのジョブに整数番号を割り当てます。ジョブを `&` で非同期に起動すると、シェルは、`[1] 1234` という形式の行を表示します。 非同期に起動されたジョブはジョブ番号 1 であり、プロセス ID が 1234 であるプロセスが 1 つ (トップレベル) あることを示します。実行中のジョブがあるが、 別に実行したいジョブがある場合、`^Z` (Control-Z) キーを押せば、 現在のジョブに `STOP` シグナルが送信されます。 そうするとシェルは通常、 ジョブが「停止」されたことを示し (後述の「出力」の項を参照)、 プロンプトを表示します。これで、このジョブの状態を `bg` コマンドでバックグラウンドで処理するか、または他のコマンドを実行してから、 `fg` というコマンドで ジョブをフォアグラウンドに移すことができます。`^Z` は直ちに有効になります。つまり `^Z` は、 保留中の出力や読み取られていない入力が 直ちに中止されるという点で、割り込みに似ています。

シェル内のジョブを参照する方法はいくつかあります。そのジョブのいずれかのプロセスの ID を使っても、また以下のいずれかを使っても参照できます。

`%`*number* *number* が示す番号のジョブ

`%`*string* コマンド行が *string* で始まっていたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%?`*string* コマンド行が *string* を含んでいたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%%` 現在のジョブ

`%+` `%%` と同じ

`%-` 直前のジョブ

シェルは、 プロセスの状態が変更すると、 直ちにそれを検出します。 ジョブがブロックされてそれ以上進めない状態になると、 シェルはその旨をユーザーに通知します。 ただし、ユーザーの作業の妨げにならないよう、 この通知はプロンプトを発行

する直前にだけ行います。 モニタモードが有効なとき、 完了した各バックグラウンドジョブは、 CHLD に設定されているトラップを起こします。 ジョブの実行中または停止中にシェルを終了しようとすると、 「停止中 (実行中) のジョブがある ('You have stopped (running) jobs.') 」旨の警告を受けます。 jobs コマンドを使用すれば、 どのジョブが該当するのかを確認できます。これを実行するか、 または直ちにシェルを再終了しようとすると、 シェルは 2 度目の警告は出さず、 停止中のジョブは終了します。

fg は、バックグラウンドジョブを、現在の環境からフォアグラウンドへ移します。 fg を使ってジョブをフォアグラウンドへ移した場合、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除されます。 fg コマンドが使えるのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をフォアグラウンドで実行します。*job_id* が指定されないと、 現在のジョブをフォアグラウンドで実行します。

bg は、現在の環境で中断されたジョブを、バックグラウンドジョブとして 実行することにより再開します。 *job_id* が示すジョブがすでにバックグラウンドジョブを実行している場合、bg は何も行わず正常に終了します。bg を使ってジョブをバックグラウンドへ移した場合、あたかも非同期リストから起動されたかのように、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID 」の 1 つとなります。 bg コマンドが使えるのは、 ジョブ制御をサポートするシステム上だけです。*job_id* が省略された場合は、 現在のジョブをバックグラウンドで実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

出力

-p オプションを指定すると、各プロセス ID に対して次に示す 1 行の情報が出力されます。

"%d\n", *"process ID"*

-p を省略すると、 -l オプションも省略されていれば、以下の形式の一連の行が出力されます。

"[%d] %c %s %s\n", *job-number*, *current*, *state*, *command*

各フィールドの意味を以下に説明します。

current

文字 + は、 fg および bg コマンド用のデフォルトとして使用するジョブを表します。 このデフォルトジョブは、 *job_id* %+ または %% を使って指定することもできます。文字 – は、現在のデフォルトジョブが終了してしまった場合に デフォルトとなるジョブを表します。このジョブは、 *job_id* %– を使って指定することもできます。 その他のジョブに関しては、このフィールドは空白文字として出力されます。 + や – を使って表せるジョブの数は、どちらも最大 1 つです。停止中の

ジョブがあれば、現在のジョブも停止ジョブとなります。 停止中のジョブが 2 つ以上あれば、以前のジョブも停止ジョブとなります。

job-number

`wait` 、 `fg` 、 `bg` 、 `kill` の各ユーティリティ用にプロセスグループを識別するのに使用する番号。 これらのユーティリティを使うと、ジョブはジョブ番号の後に `%` を付加することにより識別できます。

state

以下の文字列 (POSIX ロケールにある) のいずれかです。

`Running`	ジョブはシグナルによって中断されておらず、 終了もしていないことを表す。
`Done`	ジョブは終了して、ゼロの終了ステータスを返したことを表す。
`Done(`*code*`)`	ジョブは正常に終了し、指定されたゼロ以外の終了ステータス (*code* が示す 10 進数) を返したことを表す。
`Stopped`	
`Stopped (SIGTSTP)`	`SIGTSTP` シグナルがジョブを停止したことを表す。
`Stopped (SIGSTOP)`	`SIGSTOP` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTIN)`	`SIGTTIN` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTOU)`	`SIGTTOU` シグナルがジョブを停止したことを表す。

利用者側で、文字列 `Stopped` の代わりに `Suspended` を使うよう定義することができます。 ジョブをシグナルが終了した場合、 `state` の形式は不定ですが、ここに示した他の `state` 形式とは明確に区別できるものです。その出力上で、 ジョブを終了させたシグナルの名前または説明が与えられます。

command

シェルに与えられた関連コマンド。

`-l` オプションを指定すると、プロセスグループ ID を示すフィールドが `state` フィールドの前に挿入されます。さらに、プロセスグループ内の より多くのプロセスが別の行に出力されることがあります。 その内容は、プロセス ID と `command` フィールドだけです。

環境

jobs 、 fg 、 bg の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

jobs 、 fg 、 bg は、以下の終了ステータスを返します。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), kill(1), ksh(1), ps(1), sh(1), stop(1), shell_builtins(1), stty(1), wait(1), signal(3HEAD), attributes(5), environ(5)

名前 | nroff – ディスプレイまたはラインプリンタ用に文書をフォーマット

形式 | **nroff** [-ehiq] [-m*name*] [-n*N*] [-o*pagelist*] [-r*aN*] [-s*N*] [-T*name*] [-u*N*]

機能説明 | nroff ユーティリティは、指定されたファイル中のテキストを、 タイプライタ的なデバイスへ出力するためにフォーマットします。 troff(1) の説明も参照してください。

file 引数を省略すると、 nroff は標準入力からテキストを読み込みます。1 つの – からなる引数は、標準入力に対応したファイル名であるとみなされます。

オプション | 以下のオプションが指定できます。 オプション群は、*file* 引数指定の前であれば どのような順序で記述してもかまいません。

-e — 端末の解像度に従い、 語と語の間の空白を均等にして左右両端をそろえた形式でテキストを出力します。

-h — 横方向に空白を出力する際、空白文字と タブ文字を併用します。これにより出力の速度を上げ、出力文字数を減らすことができます。 タブ文字の位置は、文字 8 個分の幅ごとに設定されています。

-i — 指定した入力ファイルをすべて読み込んだあとで標準入力を読み取ります。

-q — .rd 要求から読み込まれた出力を出力しません。

-m*name* — 指定した入力ファイルの前に、マクロファイル /usr/share/lib/tmac/tmac.*name* を付加します。

-n*N* — 生成した最初のページのページ番号を *N* とします。

-o*pagelist* — ページ指定。 *pagelist* で示したページだけを印刷します。 このリストには、複数のページ番号またはページ番号の 範囲 (またはその両方) をコンマで区切って記述します。範囲として *N*–*M* と記述すればページ番号 *N* から *M* までが出力され、リストの先頭に -N と記述すれば先頭ページからページ番号 *N* までが出力され、最後に *N*– と記述すればページ番号 *N* から最終ページまでが出力されます。

-r*aN* — レジスタ設定。 1 文字の引数 *a* で示すレジスタに、値 *N* を設定します。

-s*N* — *N* ページごとの停止。 nroff は、 *N* ページ目を印刷する前にいったん停止し、 復帰改行（NEWLINE）を受け取ると印刷を再開します。これを最後のページまで繰り返します。 *N* のデフォルト値は 1 です。 本機能は用紙のローディングや取り替えを行う目的で使用できます。

-T *name* — *name* で示したデバイス用に出力を準備します。 *name* としては以下の値が使用できます。

37	Teletype Corporation Model 37 型端末。この値がデフォルトです。
lp | tn300	GE — 「半改行」機能を持たないラインプリンタまたは端末
300	DASI-300
300-12	DASI-300 — 12 ピッチ
300S	DASI-300S
300S-12	DASI-300S
382	DASI-382 (fancy DTC 382)
450	DASI-450 (Diablo Hyterm)
450-12	DASI-450 (Diablo Hyterm) —12 ピッチ
832	AJ 832

-u*N* 所定の位置にマウントされているフォントのボールド係数を 3 対 *N* に設定します。*N* を指定しない場合、ボールド係数は 0 に設定されます。

使用例

例 1 マクロパッケージを使ってフォーマット

本コマンドはマクロパッケージ -me を使って users.guide をフォーマットし、出力時には 4 ページごとに印刷をいったん停止します。

```
example% nroff –s4 –me users.guide
```

環境

nroff の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

ファイル

/usr/tmp/trtmp*	一時ファイル (後述の「注意事項」を参照)
/usr/share/lib/tmac/tmac.*	標準マクロファイル
/usr/share/lib/nterm/*	nroff 用端末駆動テーブル
/usr/share/lib/nterm/README	端末記述ファイルのインデックス

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc
CSI	対応済み

関連項目 | checknr(1), col(1), eqn(1), man(1), tbl(1), troff(1), attributes(5), environ(5), me(5), ms(5), term(5), rd(9F)

注意事項 | 現在、 /usr/tmp は /var/tmp へのシンボリックリンクです。

以前のマニュアルに書かれていた、数値レジスタ yr が「現在の年号の末尾の 2 桁」を示す、という記述は正しくありません。yr は、実際には 1900 年以降の数値を対象にします。2099 年までの、現在の年号の末尾の 2 桁を正しく獲得するには、以下の文字列レジスタ yr の定義をドキュメントに含めると、 2 桁の年号を表示するために使用できます。yr の代わりに、別の 1 つまたは 2 つの文字をレジスタ名として使用できます。

```
.\" definition of new string register yy--last two digits of year
.\" use yr (# of years since 1900) if it is < 100
.ie \n(yr<100 .ds yy \n(yr
.el \{              .\" else, subtract 100 from yr, store in ny
.nr ny \n(yr-100
.ie \n(ny>9 \{      .\" use ny if it is two digits
.ds yy \n(ny
.\" remove temporary number register ny
.rr ny \}
.el \{.ds yy 0
.\" if ny is one digit, append it to 0
.as yy \n(ny
.rr ny \} \}
```

名前 | od – 8 進ダンプ

形式 | **/usr/bin/od** [-bcCDdFfOoSsvXx] [-] [*file*] [*offset_string*]

/usr/bin/od [-bcCDdFfOoSsvXx] [-A *address_base*] [-j *skip*] [-N *count*] [-t *type_string*...] [-] [*file*...]

/usr/xpg4/bin/od [-bcCDdFfOoSsvXx] [-] [*file*] [*offset_string*]

/usr/xpg4/bin/od [-bcCDdFfOoSsvXx] [-A *address_base*] [-j *skip*] [-N *count*] [-t *type_string*...] [-] [*file*...]

機能説明 | od コマンドは、指定された入力ファイルを順番に標準出力にコピーします。 このとき、 -t または -bcCDdFfOoSsvXx オプションで指定された出力タイプに従って入力データを変換します。 出力タイプ指定が省略された場合には、 -t o2 が指定されたものと見なします。 -bcCDdFfOoSstvXx オプションを複数回記述して、複数のタイプを指定することも可能です。 その場合、記述した順序に従って出力行がタイプごとに書き出されます。 入力ファイルを示す *file* を省略した場合には、標準入力が使用されます。 [*offset_string*] オペランドは、 -A 、 -j 、 -N 、 -t の各オプションと排他的です。 このマニュアルページでは、以下の用語を使用しています。

ワード	16 ビットのユニットを表します。マシンのワードサイズとは無関係です。
ロングワード	32 ビットのユニットを表します。
ダブルロングワード	64 ビットのユニットを表します。

オプション | 以下のオプションを指定できます。

-A*address_base* 入力オフセット値の基数を指定します。 *address_base* 引数は以下に述べる文字のいずれかでなければなりません。 文字 d、 o、 x は、それぞれ 10 進、 8 進、 16 進を表します。文字 n は、オフセットは出力しない旨を表します。 -A n を指定しない限り、出力行の先頭には、 次に書き出されるバイトの入力オフセット値 (入力ファイルでの累積値) が付加されます。また、入力データがすべて処理された後は、 最後に出力されたバイトの次のバイトのオフセット値が表示されます。 -A*address_base* オプションも [*offset_string*] オペランドも指定されない場合、入力オフセット値は 8 進数で表示されます。

-b バイトを 8 進数で解釈します。この指定は -t o1 と同じ意味です。

/usr/bin/od

-c シングルバイト文字を表示します。非図形文字のなかには、 C 言語のエスケープで表示される ものもあります。

NULL	**\0**
バックスペース	**\b**
用紙送り	**\f**
復帰改行	**\n**
リターン	**\r**
タブ	**\t**

その他の文字は、 3 桁の 8 進数で表示されます。 たとえば、次のようにします。

```
echo "hello world" | od –c
0000000   h   e   l   l   o       w   o   r   l   d  \n
0000014
```

/usr/xpg4/bin/od

-c
: LC_CTYPE ロケール カテゴリの現在の設定に基づき、バイトを シングルバイト文字または複数バイト文字で解釈します。 印刷可能な複数バイト文字は、 文字の最初のバイトに対応する領域に書き込まれます。 残りのバイトに対応する領域には、文字が継続することを示す ** が書き込まれます。 非図形文字は、 -C オプションを使用したときと同じように解釈されます。

-C
: LC_CTYPE ロケール カテゴリの現在の設定に基づき、バイトを シングルバイト文字または複数バイト文字で解釈します。 印刷可能な複数バイト文字は、 文字の最初のバイトに対応する領域に書き込まれます。 残りのバイトに対応する領域には、文字が継続することを示す ** が書き込まれます。非図形文字のなかには、 C 言語のエスケープで表示されるものもあります。

```
NULL            \0
バックスペース      \b
用紙送り        \f
復帰改行        \n
リターン          \r
タブ             \t
```

その他の非図形文字に関しては、各バイトにつき 3 桁の 8 進数が表示されます。

-d
: ワードを符号なし 10 進数で解釈します。この指定は -t u2 と同じ意味です。

-D
: ロングワードを符号なし 10 進数で解釈します。 この指定は -t u4 と同じ意味です。

-f
: ロングワードを浮動小数点で解釈します。 この指定は -t f4 と同じ意味です。

-F
: ダブルロングワードを拡張精度で解釈します。 この指定は -t f8 と同じ意味です。

-j*skip*
: 入力データの先頭の *skip* 個のバイトをスキップします。 od コマンドによる読み取りまたはシークは、連結された入力ファイルにおいて *skip* バイトを超えた地点から行われます。入力データの合計長が *skip* バイトに満たない場合、 od コマンドは標準エラー出力に診断メッセージを書き出し、 ゼロ以外の終了ステータスコードで処理を終了します。

  デフォルトでは *skip* 引数は 10 進数と解釈されます。 先頭に 0x または 0X が付加されていれば、オフセット値は 16 進数と解釈されます。また先頭が 0 のときは、 8 進数と解釈されます。また数値の後に文字 b、 k、または m を付加すると、それぞれ 512、

1024、または 1 048 576 バイトの倍数と見なされます。 *skip* 値が 16 進数のとき、最後に b が付加されていてもそれは 16 進の数字と見なされます。 アドレス表示は 0000000 から始まります。その基数が *skip* 引数の基数によって示されることはありません。

-N*count*
count が示すバイト数を超えない範囲で入力データをフォーマットします。 デフォルトでは *count* は 10 進数と解釈されます。 先頭に 0x または 0X が付加されていれば、 *count* は 16 進数と解釈されます。また先頭が 0 のときは、 8 進数と解釈されます。入力データが (-j*skip* 指定時はスキップ完了後) *count* バイト分存在しない場合でも、エラーとは見なされません。 od コマンドは、存在している分の入力データをフォーマットします。 表示されるアドレスの基数が *count* 引数の基数によって表されることはありません。

-o
ワードを 8 進数で解釈します。 この指定は -t o2 と同じ意味です。

-O
ロングワードを符号なし 8 進数で解釈します。 この指定は -t o4 と同じ意味です。

-s
ワードを符号付き 10 進数で解釈します。 この指定は -t d2 と同じ意味です。

-S
ロングワードを符号付き 10 進数で解釈します。 この指定は -t d4 と同じ意味です。

-t*type_string*
出力タイプを指定します。 *type_string* は文字列で、入力データを書き出す際に用いる出力タイプを表します。 複数の文字を使って複数のタイプを指定できます。 *type_string* は、出力タイプを表す以下の文字で構成されていなければなりません。

a
「名前を与えられた文字」を表します。 このタイプ指定では、各バイトのうち最下位の 7 ビットだけが用いられます。 以下の表に示す値を持つバイトが、文字に対応した名前を使って 出力されます。

od における名前を与えられた文字

値	名前	値	名前	値	名前	値	名前
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb

値	名前	値	名前	値	名前	値	名前
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

c 「文字」を表します。 LC_CTYPE ロケール カテゴリの現在の設定に基づき、バイトを シングルバイト文字または複数バイト文字で解釈します。 印刷可能な複数バイト文字は、 文字の最初のバイトに対応する領域に書き込まれます。 残りのバイトに対応する領域には、文字が継続することを示す ** が書き込まれます。次の非図形文字は、 C 言語のエスケープで表示されます。

\0、 \a、 \b、 \f、 \n、 \r、 \t、 \v

その他の非図形文字に関しては、各バイトにつき 3 桁の 8 進数が表示されます。

タイプ指定文字である d、 f、 o、 u、 x の後には、出力タイプが発生するたびに変換するバイト数を示す 符号なし 10 進数を記述することもできます。

f 「浮動小数点数」を表します。後に文字 F、 D、または L を付加して、タイプが float、 double、 long double の項目に対して変換を行う旨を表すこともできます。

d, o, u, x それぞれ符号付き 10 進数、 8 進数、 符号なし 10 進数、 16 進数を表します。後に文字 C、 S、 I、または L を付加して、タイプが char、 short、 int、または long の項目に対して変更を行う旨を表すこともできます。

1 つの *type_string* 中に複数のタイプを続けて指定できます。 また -t オプションを複数個指定する可能です。 その場合、タイプ指定文字を記述した順序に従って、 出力行がタイプごとに書き出されます。

-v すべてのデータを表示します (冗長) 。このオプションを省略すると、直前の出力行に等しい (バイトオフセットを除く) 行は、すべてアスタリスク (*) だけからなる行に置き換えられます。

-x ワードを 16 進数で解釈します。 この指定は -t x2 と同じ意味です。

-X ロングワードを 16 進数で解釈します。 この指定は -t x4 と同じ意味です。

オペランド　以下のオペランドは、 /usr/bin/od と /usr/xpg4/bin/od で指定できます。

–　指定されたすべてのファイルに加えて、 標準入力を使用します。 このオペランドを省略すると、 *file* オペランドを指定しなかった場合だけ、 標準入力が使用されます。

/usr/bin/od　以下のオペランドは、 /usr/bin/od でのみ指定されます。

file　読み出すファイルのパス名。 このオペランドを省略すると、標準入力が用いられます。 オペランドの数が 2 以内であり、 -A、 -j、 -N、 -t オプションがすべて省略され、 さらに以下の項目のいずれかが真である場合には、 対応するオペランドはファイルのオペランドではなく、 オフセット値指定と見なされます。

1. 最後のオペランドの最初の文字が正の記号 (+) である。
2. 2 番目のオペランドの最初の文字が数値である。
3. 2 番目のオペランドの最初の文字が x であり、2 番目のオペランドの 2 番目の文字が小文字の 16 進数文字または数字である。
4. 2 番目のオペランドが "x" である。
5. 2 番目のオペランドが "." である。

-N *count* オプションを省略した場合、表示はファイルの終わりに 到達するまで続けられます。

[+] [0] *offset* **[.] [b | B] [+] [0] [***offset***] [.] [+] [0x | x] [***offset***] [+] [0x | x]** *offset* **[B]**　*offset_string* オペランドは、ファイル中のダンプを始める位置のバイトオフセット値を 指定します。デフォルトでは、オフセット値は 8 進数のバイトで解釈され表示されます。 *offset* が 0 で始まる場合には、 オフセット値は 8 進数で解釈されます。 *offset* が x または 0x で始まる場合には、 オフセット値は 16 進数で解釈され、 b を付加した場合には、 16 進数の数字であるとみなされます。"." を付加すると、オフセット値は 10 進数で解釈されます。 b または B を付加すると、 オフセット値は 512 バイト単位で解釈されます。 *file* 引数を省略する場合には、オフセット値の先頭に (+) を付加しなければなりません。 アドレスの表示は、指定されたオフセットから始まります。 アドレスの基数は、指定してあれば オフセットの基数と同じになります。指定していない場合は 8 進数になります。 10 進数は 8 進数を上書きし、 同じオフセットオペランドに 16 進数と 10 進数の両方の変換を指定するとエラーになります。

/usr/xpg4/bin/od　以下のオペランドは、 /usr/xpg4/bin/od でのみ指定できます。

file　始めの 2 つの項目のどちらかが真でなければならないことを 除いては、 /usr/bin/od と同じ。

+x*offset* [B]
[+] [0] *offset* [.] [b|B]
+ [*offset*] [.]
[+][0x][*offset*]
[+][0x] *offset* [B]
+x [*offset*]
+x*offset* [B]　オフセット値の記述は /usr/bin/od と同じです。

環境

od の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 LC_NUMERIC 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/od

属性タイプ	属性値
使用条件	SUNWtoo
CSI	対応済み

/usr/xpg4/bin/od

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

sed(1), attributes(5), environ(5), XPG4(5)

名前 | trap, onintr – (ハードウェア) シグナルに応答するためのシェル組み込み関数

sh | **trap** [*argument n* [*n2*...]]

csh | **onintr** [-| *label*]

ksh | ***trap** [*arg sig* [*sig2*...]]

sh | *argument* が示す trap コマンドを、シェルが数値形式または 名前形式で指定されているシグナル（*n*） を受信した時に読み取り、実行します。なお *argument* は、トラップ設定時とトラップ取り出し時に 1 度ずつ検索されます。 トラップコマンドは、シグナル番号または 対応するシグナル名の順序で実行されます。 現在のシェルへの入力時に無視されたシグナルにトラップを 設定しようとしても無効となります。 シグナル 11（メモリフォルト） にトラップを指定しようとすると、 エラーになります。 *argument* を省略すると、 *n* のトラップはすべてその元の値に再設定されます。 *argument* が NULL 文字列の場合、 シェルおよびシェルが呼び出すコマンドは、 このシグナルを無視します。 *n* が 0 の場合、 *argument* が示すコマンドはシェル終了時に実行されます。 引数なしの trap コマンドは、コマンドの一覧を 各々が対応しているシグナル番号とともに表示します。

csh | onintr は割り込み時のシェルの動作を制御します。 引数を指定しないと、onintr は割り込み時にはデフォルトの動作を実行します （シェルはシェルスクリプトを終了して、 端末のコマンド入力レベルに戻ります）。- 引数を指定すると、シェルはすべての割り込みを無視します。*label* 引数を指定すると、割り込みを受信するか 割り込みのために子プロセスが終了したときに、シェルは goto *label* を実行します。

ksh | trap は *arg* を *sig* が示すシグナルをシェルが受信したときに読み取られ、 実行されるコマンドとして使用します。 なお *arg* は、 トラップ設定時と トラップ取り出し時に 1 度ずつ検索されます。各 *sig* は、 数値またはシグナルの名前です。trap コマンドは、 シグナル番号の順序で実行されます。 現在のシェルで無視されているシグナルにトラップを 設定しようとしても無効となります。*arg* を省略するか、 または - と指定する場合、各 *sig* 用のトラップはすべてその元の値に再設定されます。*arg* が NULL 文字列（"" などの空の文字列）の場合、 シェルおよびシェルが呼び出すコマンドは、このシグナルを無視します。 *sig* が ERR の場合、 コマンドが 0 以外の終了状態で終わると必ず *arg* が実行されます。*sig* が DEBUG の場合、 各コマンドの後に *arg* が実行されます。 *sig* が 0 または EXIT で、 トラップが関数の外側で設定されている場合、シェルの終了時に *arg* が示すコマンドが実行されます。 引数なしの trap コマンドは、 コマンドの一覧を 各々が対応しているシグナル番号とともに表示します。

1 つまたは 2 つのアスタリスク（*）が先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。つまり、 チルド置換は = 符号の後に実行され、ワード分割とファイル名生成は実行されません。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | csh(1), exit(1), ksh(1), sh(1), attributes(5)

名前 | pack, pcat, unpack – ファイルの圧縮および復元

形式 | **pack** [-f] [-] *file*...

pcat *file*...

unpack *file*...

pack | pack コマンドは、指定されたファイルを圧縮した形で保存します。 可能であれば (そして便利ならば)、 *file* という入力ファイルは、 *file* と同じアクセスモード、 同じアクセスの日付や変更した日付、同じ所有者を持つ *file*.z という圧縮されたファイルに置き換えられます。 pack の実行が正常に終了すると、 *file* は削除されます。

圧縮の総量は、入力ファイルのサイズおよび文字度数分布によります。 デコード用ツリーがそれぞれの .z ファイルの最初の部分を形成しているので、 3 ブロック未満のファイルを圧縮することは、あまり意味がありません。 ただし、プリンタプロットや図形の場合に起こるように、 文字度数分布が非常に偏っているものは例外です。

一般的にテキストファイルを元のサイズの 60% から 75% に圧縮します。大きい文字セットを使用し、 文字分布が一様なロードモジュールは、 元のサイズのおよそ 90% にしか圧縮されません。

pack は、圧縮できなかったファイルの数を示す値を返します。

次の場合には、圧縮は行われません。

- ファイルがすでに圧縮されている場合
- ファイル名が 14 – 2 バイトより長い場合
- ファイルにリンクがある場合
- ファイルがディレクトリの場合
- ファイルをオープンできない場合
- ファイルが空の場合
- 圧縮によってディスクブロックを減らせない場合
- *file*.z というファイルが既に存在している場合
- .z ファイルを作成することができない場合
- 処理中に、入出力エラーが発生した場合

ファイル名の最後のセグメントは、 {NAME_MAX} – 2 文字以下でなければなりません。 この 2 文字は、追加される拡張子 .z 用です。ディレクトリは圧縮できません。

pcat | pcat コマンドは、 cat(1) が通常ファイルに対して行うことを、 圧縮したファイルに対して行います。 ただし pcat をフィルタとして使うことはできません。 指定したファイルは、復元され、標準出力に書き込まれます。

pcat は、復元できなかったファイルの数を返します。 以下の場合にはエラーとなります。

- ファイルがオープンできなかった
- ファイルが、 pack の出力ファイルと認識できなかった

unpack unpack コマンドは、 pack で作成したファイルを復元します。 コマンドで指定したファイル *file* に対して、 *file*.z というファイル (あるいは *file* が .z で終わる場合は単に *file*)を検索します。 このファイルが圧縮されたファイルである場合は、 復元したファイルに置き換えます。新たなファイル名は .z 接頭辞が取り除かれ、アクセスモード、アクセス日付や変更日 付および所有者名は圧縮されたファイルと同じです。

unpack は、復元できなかったファイルの数を示す値を返します。 pcat で述べた理由のほかに、次のような場合には復元できません。

- "unpackされた"ときのファイル名が既に存在する場合
- 復元したファイルを作成できない場合
- ファイル名の長さ (拡張子 .z を除く) が 14 バイトを超えている場合

オプション 以下のオプションを指定できます。

-f *file* の強制圧縮。ディレクトリ全体を圧縮するのに有効です。 ただし、中には 圧縮しても小さくならないファイルがあります。 unpack または pcat は、圧縮したファイルを元の形式に復元できます。

オペランド 以下のオペランドを指定できます。

file pack、 unpack、または pcat するファイルのパス名。 *file* には拡張子 .z を指定しても省略してもかまいません。

– pack は、 1 バイトごとに Huffman (最小冗長度) コードを使用します。 – 引数を使用すると、それぞれのバイトの使用回数、 相対頻度およびバイトのコードを標準出力に出力するよ うに内部フラグが設定されます。 *file* の代わりに – を追加すると、内部フラグを設定し、リ セットします。

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の pack 、 pcat 、 unpack の動作については、 largefile(5) を参照してください。

使用例 例 **1** 圧縮ファイルを表示する

file.z という名前の圧縮ファイルを見るには、次のようにします。

```
example% pcat file.z
```

あるいは、単に次のようにします。

```
example% pcat file
```

例 **2** 圧縮したファイルの復元コピーを作成する

file.z という名前の圧縮したファイルの復元コピー nnn を file.z を破壊せずに作成するには 、次のコマンドを使用します。

```
example% pcat file >nnn
```

環境

pack 、 pcat 、 unpack の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した。圧縮または復元できなかった ファイルの数が返される

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

cat(1), compress(1), zcat(1), attributes(5), environ(5), largefile(5)

名前 | more, page – テキストファイルの表示またはページング

形式 | **/usr/bin/more** [-cdflrsuw] [-lines] [+ *linenumber*] [+/ *pattern*] [*file...*]

/usr/bin/page [-cdflrsuw] [-lines] [+ *linenumber*] [+/ *pattern*] [*file...*]

/usr/xpg4/bin/more [-cdeisu] [-n *number*] [-p *command*] [-t *tagstring*] [*file...*]

/usr/xpg4/bin/more [-cdeisu] [-n *number*] [+ *command*] [-t *tagstring*] [*file...*]

機能説明 | more ユーティリティは、端末の画面上にテキストファイルの内容を 1 画面ずつ表示するフィルタです。 通常、画面が一杯になると休止します。そのとき /usr/bin/more は --More-- を、 /usr/xpg4/bin/more は *file* を、いずれも画面の最下段に表示します。 more がパイプではなくファイルから読み取る場合、 それまでに表示された文字のパーセンテージも併せて表示します。

more ユーティリティは、 RETURN 文字が入力されると、もう 1 行スクロールアップします。 SPACE 文字が入力されると、次の画面を表示します。 その他のコマンドは、以下で説明します。

page ユーティリティは、テキストの次の画面を 表示する前に画面をクリアします。 page は、 more とは異なり、 前画面の最後の行を現画面の最初に重複して表示することはしません。

more ユーティリティは、端末を NOECHO モードに設定し、連続して出力できるようにします。 通常、入力するコマンドは端末上に表示されません。 ただし、 / コマンドと ! コマンドの場合を除きます。

/usr/bin/more ユーティリティは、指定された最後のファイルを 表示して終了します。 /usr/xpg4/bin/more は、指定された最後のファイルの最終行で、コマンド入力を要求する プロンプトを発行します。

標準出力が端末の場合、 more は cat(1) とまったく同じように動作しますが、 複数のファイルが指定された場合、ファイルの前にヘッダが表示されます。

オプション | /usr/bin/more と /usr/xpg4/bin/more の両方で以下のオプションが使用できます。

-c　表示前に画面をクリアします。 より高速に表示するため、画面をスクロールせずに描画し直します。 端末が行の終端までクリアする機能を持たない場合、 このオプションは無視されます。

-d　認識できないコマンドが使用された場合、 端末ベルを鳴らさずにエラーメッセージを表示します。 これは経験の少ないユーザーに役立ちます。

-s　圧縮。 複数の空行を単一の空行と置換します。 これは、 nroff(1) を画面表示させる場合に役立ちます。

/usr/bin/more | 以下のオプションは /usr/bin/more でのみ指定できます。

`-f` 長い行を折り返しません。 `nroff`(1) 出力を `ul`(1) によりパイプ連結したとき生成されるような非表示文字、またはエスケープシーケンスが行にあるときに役立ちます。

`-l` FORMFEED 文字 (CTRL-L) をページの中断とみなしません。 `-l` を使用しない場合、 `more` は休止して、 `^L` 文字 (CTRL-L) の入った行の後のコマンドを受け入れます。 また、ファイルが FORMFEED で始まっている場合、ファイルを表示する前に画面をクリアします。

`-r` 通常、 `more` は解釈できない制御文字を無視します。 `-r` オプションにより、これらの文字は `^C` として表示されます。

`-u` 下線引きエスケープシーケンスを生成しません。通常、 `more` は、 `nroff`(1) が生成するような下線引きを、端末に適切な方法で処理します。 端末が下線引きを実行できるか、 あるいは強調モードを備えている場合、 `more` はテキストファイルで呼び出されたものに対応するエスケープシーケンスを提供します。

`-w` 通常、 `more` はその出力の終端まで来ると終了します。 しかし、 `-w` を用いると、 `more` は終了する前にプロンプトを出し、キーが押されるのを待ちます。

-lines デフォルト (端末画面内の行の数から 2 を引いた数) の代わりに、指定された数の *lines* を行数分画面に表示します。

+linenumber *linenumber* から表示します。

+/pattern *pattern* という正規表現の入った行より上の 2 行から表示します。注意 : エディタと異なり、この形式は '/' で終了してはいけません。 スラッシュで終了すると、その後のスラッシュは検索パターンの文字と見なされます。

/usr/xpg4/bin/more 以下のオプションは `/usr/xpg4/bin/more` でのみ指定できます。

`-e` 引数リストの最後のファイルの最終行を出力すると、 ただちに終了します。

`-i` 検索時に、大文字と小文字を区別せずにパターンマッチングを行います。

`-n` *number* 1 画面当たりの行数を指定します。 *number* 引数の値は正の整数です。 `-n` オプションの値は、環境から得られる値に優先して用いられます。

`-p` *command*

+command 検査対象の各ファイルに対し、最初に *command* 引数中の `more` コマンドを実行します。それが行番号や正規表現検索など 位置決め用のコマンドであれば、コマンドの最終結果を表すように現在の位置を設定します。ファイルの中間行は書き出しません。例として次の 2 つのコマンドを見てください。

more -p 1000j file
more -p 1000G fileこれらは、現在の位置を行番号 1000 として表示を開始する、という同じ動作をします。しかも、 j がファイルの検査中に呼び出されていれば書き出して画面から 消したであろうと思われる行は飛ばします。 位置決めコマンドが正常終了でなければ、ファイルの先頭行が 現在の位置となります。

`-t` *tagstring*
: *tagstring* 引数で指定したタグを持つファイルの内容を 1 画面分表示します。詳細については ctags(1) ユーティリティの説明を参照してください。

`-u`
: バックスペース文字を印刷可能文字として扱い、 ^H (CTRL-H) と表示します。このとき、ある種の端末では下線付きまたは強調モードテキスト で出力するような特殊な処理や、下線を付ける処理は行いません。 さらにこのオプションが指定されると、 行の終わりのキャリッジリターンを無視しません。

`-t` *tagstring* と `-p` *command* (または旧式の *+command*) オプションの両方が指定された場合、 `-t` *tagstring* が先に処理されます。

環境
: more は、端末の terminfo(4) エントリを使用して、そのディスプレイ特性を判別します。 more は MORE という環境変数に設定済みオプションがないか調べます。 たとえば、デフォルトで -c モードを使用してファイルをページングするときは、 この環境変数の値を -c に設定します (通常、この環境変数を設定するコマンドシーケンスは .login ファイルまたは .profile ファイルに格納します) 。

コマンド
: コマンドはただちに有効になります。 コマンドが *file* 、 *command*、 *tagstring* 、 *pattern* のいずれかを必要としない限り、 キャリッジリターンを入力する必要はありません。 コマンド文字自体を指定しないかぎり、行抹消文字を入力すれば、 入力中の数値引数を取り消すことができます。 さらに、消去文字を入力すれば、 '--More--(*xx*%)' または *file* メッセージを再表示できます。

以下のコマンドにおいて、 *i* は数値引数です (デフォルトでは 1) 。

*i*SPACE
: 次の画面を表示します。 *i* を指定すると、 *i* 行追加表示します。

*i*RETURN
: 次の行を表示します。 *i* を指定した場合は、 *i* 行追加表示します。

*i*b

i^B
: (Control-b) 画面を *i* 個、逆にスキップして次の画面を表示します。

*i*d

i^D
: (Control-d) 画面を半画面分または *i* 行分、順方向にスクロールします。 *i* が指定されていれば、その値が以降の d および u コマンド用のデフォルトとなります。

*i*f
: 画面を *i* 個スキップして次の画面を表示します。

h
: ヘルプ。 more のすべてのコマンドの説明を表示します。

^L	(Control-l) 画面の再表示。
*i*n	直前に入力した *pattern* の *i* 番目の一致を検索します。
q	
Q	more を終了します。
*i*s	*i* 行スキップしてから 1 画面分を表示します。
v	現在のファイルの現在行で、 vi エディタに入ります。
*i*z	SPACE と同じ。 ただし *i* を指定した場合、 その値が画面当たりの行数の新しいデフォルト値になります。
=	現在の行番号を表示します。
i/*pattern*	*pattern* という正規表現の *i* 番目の一致を順方向に検索します。 *pattern* という正規表現の *i* 番目の一致またはパイプの終端、どちらか先に見つかった方を含む行の前 2 行から画面を表示します。 more がファイルを表示中であり、一致がない場合、 そのファイルにおける位置は変更されません。 正規表現は、消去文字と抹消文字を使用して編集することができます。 第 1 カラムを越えて消去すると、検索コマンドが取り消されます。
!*command*	シェルを起動し、 *command* を実行します。 % および ! という文字を command 内で使用すると、それぞれ、 現在のファイル名および直前のシェルコマンドに置換されます。 現在のファイル名がない場合、 % は展開されません。 これらの文字の前にバックスラッシュを追加して、 展開をエスケープしてください。
:f	現在のファイル名と行番号を表示します。
i:n	コマンド行に指定された *i* 番目後のファイル名、または *i* が範囲外の場合はリスト内の最後のファイル名までスキップします。
i:p	コマンド行に指定された *i* 番目前のファイル名、または *i* が範囲外の場合はリスト内の最初のファイル名までスキップします。 ファイル内に more を位置決めしている間に指定すると、ファイルの最初に移動します。 more がパイプから読み取り中の場合、 more は単に端末ベルを鳴らすだけです。
:q	
:Q	more を終了します (q または Q と同じ) 。

/usr/bin/more

以下のコマンドは /usr/bin/more でのみ使用できます。

'	単一引用符。 直前の検索が開始された点に移動します。 現在のファイルで検索を実行していない場合、 ファイルの最初に移動します。
.	ドット。 直前のコマンドを繰り返します。
^ \	テキストの一部表示を停止します。 more は出力の送信を停止し、通常の --More-- プロンプトを表示します。 一部の出力は結果的に失われること

があります。

/usr/xpg4/bin/more

以下のコマンドは /usr/xpg4/bin/more でのみ使用できます。

i^F	(Control-f) 画面を *i* 個スキップして次の 1 画面分を表示します (*i*f と同じ) 。
^G	(Control-g) 現在の行番号を表示します (= と同じ) 。
*i*g	ファイル中の行番号 *i* に進みます。デフォルトは先頭行です。
*i*G	ファイル中の行番号 *i* に進みます。デフォルトは最終行です。
*i*j	次の行を表示します。 *i* が指定されていればその行数分を表示します (*i*RETURN と同じ) 。
*i*k	画面を逆方向にスクロールします。 *i* 指定時はその行数、省略時は 1 行です。
m*letter*	*letter* で示す名前で現在の位置をマークします。
N	逆方向に検索します。
r	画面を再表示します。
R	画面を再表示し、バッファ内に入力があればそれを破棄します。
*i*u	
i^U	(Control-u) 画面を逆方向にスクロールします。 *i* 指定時はその行数、省略時は半画面分です。 *i* が指定されていれば、その値が以降の d および u コマンド用のデフォルトとなります。
ZZ	more を終了します (q と同じ) 。
:e *file*	新たなファイルを検査 (表示) します。 file を省略すると、現在のファイルが再表示されます。
:t *tagstring*	*tagstring* 引数が示すタグの位置へ進み、タグを含む行が現在の位置となるように 画面をスクロールします。詳しくは ctags ユーティリティの説明を参照してください。
' *letter*	以前に *letter* という名を付けてマークした位置に戻ります。
''	最新の 1 画面分を超える移動を行なったときの元の位置に戻ります。 デフォルトはファイルの先頭です。
i?[*!*]*pattern*	ファイルを逆方向に検索し、 *pattern* を含んでいる *i* 番目の行を見つけます。 *!* は、 *pattern* を含んでいない *i* 番目の行の検索です。
i/ !*pattern*	ファイルを順方向に検索し、 *pattern* を含んでいない *i* 番目の行を見つけます。
![*command*]	シェルまたは指定したコマンドを呼び出します。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の more と page の動作については、 largefile(5) を参照してください。

環境

more の実行に影響を与える環境変数 LC_COLLATE(/usr/xpg4/bin/more のみ)、LC_CTYPE、 LC_MESSAGES、 NLSPATH 、 TERM についての詳細は、 environ(5) を参照してください。

/usr/xpg4/bin/more

以下の環境変数も /usr/xpg4/bin/more の実行に影響を与えます。

COLUMNS　画面の水平方向のサイズとして、 システムが選択した値の代わりに用いる値を指定します。

EDITOR　エディタを選択する際に v コマンドを使用します。

LINES　画面の垂直方向のサイズとして、 システムが選択した値の代わりに用いる値を指定します。 画面当たりの行数を決める際、 -n オプションの値が LINES の値に優先します。

MORE　前述の「オプション」の項で説明した、オプションを指定する文字列です。 コマンド行に記述する場合と同様に、 オプションとオプションの間は空白文字で区切り、個々のオプションは – で始まらなければなりません。 MORE で指定したオプションの後で、コマンド行のオプションが処理されます。 つまり、コマンド行が次に示すように記述されていると見なされます。 more $MORE *options operands*

終了ステータス

次の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

ファイル

/usr/lib/more.help　/usr/bin/more と /usr/bin/page のためだけのヘルプファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/more
/usr/bin/page

属性タイプ	属性値
使用条件	SUNWcsu
CSI	未対応

/usr/xpg4/bin/more

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

cat(1), csh(1), ctags(1), man(1), nroff(1), script(1), sh(1), ul(1), environ(4), terminfo(4), attributes(5), environ(5), largefile(5)

/usr/bin/more
/usr/bin/page

regcomp(3C)

/usr/xpg4/bin/more | `regex`(5), `XPG4`(5)

/usr/bin/more | 逆方向のスキップを大規模なファイルに対して行うと遅くなります。

/usr/xpg4/bin/more | 端末が正しく設定されていないと、このユーティリティも正しく動作しません。

名前 | passwd – ログインパスワードおよびパスワード属性の変更

形式 | **passwd** [-r files | -r ldap | -r nis | -r nisplus] [*name*]

passwd [-r files] [-egh] [*name*]

passwd [-r files] -s [-a]

passwd [-r files] -s [*name*]

passwd [-r files] [-d | -l] [-f] [-n *min*] [-w *warn*] [-x *max*] *name*

passwd -r ldap [-egh] [*name*]

passwd -r nis [-egh] [*name*]

passwd -r nisplus [-egh] [-D *domainname*] [*name*]

passwd -r nisplus -s [-a]

passwd -r nisplus [-D *domainname*] -s [*name*]

passwd -r nisplus [-l] [-f] [-n *min*] [-w *warn*] [-x *max*]
[-D *domainname*] *name*

機能説明 | passwd コマンドは、パスワードを変更したり、あるいはユーザーのログイン *name* に関連するパスワードの属性を表示したりします。さらに特権ユーザーは passwd を使用して、すべてのログイン *name* に関連するパスワードおよび属性を登録もしくは変更できます。

パスワードを変更する場合、passwd は、古いパスワードがあれば、それを入力するよう要求します。次に、新しいパスワードを 2 回入力するように 要求します。古いパスワードを入力すると、passwd は、そのパスワードが十分に 「時間経過」しているかをチェックします。「時間経過」が不十分である場合は、passwd は終了します。詳細は pwconv(1M) 、nistbladm(1) 、および shadow(4) を参照してください。

LDAP、NIS または NIS+ がシステム上で有効である場合、passwd は NIS または NIS+ のデータベースを変更します。NIS または NIS+ のパスワードは、ローカルマシン上のパスワードとは異なる場合があります。NIS または NIS+ が動作している場合には、passwd -r を使用してローカルマシン上のパスワードを変更してください。

pwconv コマンドは、/etc/passwd からの情報をもとに /etc/shadow を作成し、更新します。pwconv は、/etc/passwd の パスワードのフィールドで 'x' という特殊な値を探します。この 'x' という値は、ユーザー用のパスワードがすでに /etc/shadow にあり、修正すべきではないということを表します。

時間経過が十分である場合、新しいパスワードの構造が要求に合うかどうか チェックが行われます。新しいパスワードを 2 回入力した時点で、2 つの新しいパスワードが比較されます。2 つのパスワードが同じでない場合は、新しいパスワードに対して、最大 2 回までプロンプトが繰り返されます。

パスワードは、以下の要求に合うように作らなければなりません。

- 各パスワードは、PASSLENGTH 文字でなければなりません。PASSLENGTH は、/etc/default/passwd に定義され、6 に設定されています。最初の 8 文字のみが意味を持ちます。
- 各パスワードには、2 つ以上の英字および 1 つの以上の数字もしくは特殊文字がなければなりません。この場合の「英字」は、すべての大文字または小文字を意味します。
- 各パスワードは、ユーザーのログイン *name* と異なっていなければならず、そのログイン *name* を反転したりずらしたものでもいけません。比較においては、大文字およびそれに対応する 小文字は同じものとして扱われます。
- 新しいパスワードは、古いパスワードと 3 文字以上違わなくてはなりません。比較においては、大文字およびそれに対応する小文字は同じものとして扱われます。

上記の条件がすべて満たされた場合、デフォルトでは passwd コマンドは /etc/nsswitch.conf を参照してパスワード更新を実行すべきレポジトリ (記録場所) を決めます。具体的には passwd と passwd_compat の両エントリを検索します。これらのエントリに対応したリソースつまりレポジトリが更新されます。なお、使用可能なパスワード更新定義形式は、以下に示す形式に限定されています。いずれの形式にも合っていない場合、passwd(1) が異常終了するのでシステムにログインできません。

- passwd: files
- passwd: files ldap
- passwd: files nis
- passwd: files nisplus
- passwd: compat (==> files nis)
- passwd: compat (==> files ldap)

  passwd_compat: ldap
- passwd: compat (==> files nisplus)

  passwd_compat: nisplus

NIS+ パスワードテーブルを所有するネットワーク管理者は、パスワードのどの属性をも変更できます。

files の場合、スーパーユーザー (たとえば、実ユーザー ID や実効ユーザー ID が 0 であるユーザー。id(1M) および su(1M) を参照) は、どのパスワードも変更することができます。したがって、passwd は、特権ユーザーに古いパスワードを入力するよう要求しません。特権ユーザーは、パスワードの有効期限の設定やパスワード構造の要求などの制限も受けません。特権ユーザーは、新しいパスワードの要求に答えてキャリッジリターンを入力するだけで NULL パスワードを作成することができます (これは、passwd -d とは異なります。「password」プロンプトが表示され続けるからです)。NIS が有効である場合、ルートマスター上のスーパーユーザーは、古い NIS パスワードの入力を要求されずにどのパスワードも変更でき、パスワードの構成要件にはなりません。

通常、引数なしで入力された passwd は現在のユーザーのパスワードを変更します。ユーザーがログインし、su(1M) を呼び出してスーパーユーザーまたは別のユーザーになるとき、passwd はスーパーユーザーや新しいユーザーのパスワードではなく、元のユーザーのパスワードを変更します。

-s オプションを使用すれば、すべてのユーザーは、自分自身のログイン *name* のパスワードの属性を表示することができます。ただしこれは -r nisplus 引数を使っている場合に限ります。そうでない場合には、-s オプションはスーパーユーザーだけしか使用できません。

表示の書式は、次のようになります。

name status mm/dd/yy min max warn

パスワードの有効期限の情報が存在しない場合は、次のようになります。

name status

各情報の意味は以下のとおりです。

name ユーザーのログイン ID

status *name* のパスワードステータス。PS は、パスワードされた状態またはロックされた状態を表し、LK は、ロックされ た状態を表し、NP は、パスワードがないことを表します。

mm/dd/yy *name* のパスワードが最近変更された日付 (すべてのパスワードの日付は、グリニッジ標準時 (ユニバーサル時間) を使用して決定されます。したがって、時差があるところでは最大 1 日ずれることがあるので注意してください) 。

min *name* に対するパスワード変更に最低限必要な日数。MINWEEKS は、/etc/default/passwd にあり、NULL に設定されています。

max パスワードが *name* に対して有効である最大日数。MAXWEEKS は、/etc/default/passwd にあり、NULL に設定されています。

warn パスワードの有効期限切れが近いことを示す警告を *name* が受ける日を、*max* に相対的な日数で表す。

セキュリティ

passwd は pam(3PAM) を使って、パスワード管理を行います。PAM 構成ポリシーは passwd で使用するモジュールを明記しています。このポリシーは /etc/pam.conf で見ることができます。以下に UNIX パスワード管理モジュールを使用する passwd コマンドのエントリの入った pam.conf ファイルの抜粋を示します。

```
passwd  auth required     pam_passwd_auth.so.1
```

passwd サービスのエントリがない場合には other のサービスのエントリを 使用します。複数の認証モジュールが記述されている場合、複数のパスワードが必要となる可能性があります。

オプション

以下のオプションを指定できます。

`-a`	すべてのエントリのパスワードの属性を表示します。`-s` オプションとともにのみ使用し、*name* を指定してはいけません。レポジトリが `nisplus` の場合、本コマンドを発行したユーザーが 読み取りを許されているローカルドメイン中の NIS+ パスワードテーブルにあるエントリだけが表示されます。`files` の場合には、この指定はスーパーユーザーだけが使用できます。
`-D` *domainname*	*domainname* が示すドメイン中にある `passwd.org_dir` を参照することを指定します。このオプションを省略すると、`nis_local_directory`(3NSL) が返すドメイン名が使用されます。このドメイン名は `domainname`(1M) が返すものと同一です。
`-e`	ログインシェルを変更します。レポジトリが `files` の場合、スーパーユーザーだけが使用できます。通常のユーザーはレポジトリが `ldap`、 `nis` または、`nispluse` の場合に変更できます。シェルの選択は、`getusershell`(3C) によって制限されています。ユーザーが現在 `getusershell` によって許可されていないシェルを保持している場合、スーパーユーザーだけがそれを変更できます。
`-g`	gecos (finger) 情報を変更します。レポジトリが `files` の場合、スーパーユーザーだけが使用できます。通常のユーザーはレポジトリが `ldap`、 `nis` または、 `nispluse` の場合に変更できます。
`-h`	ホームディレクトリを変更します。
`-r`	処理の対象とするレポジトリ (記録場所) を指定します。`files` 、`ldap`、`nis` 、`nisplus` のいずれかを指定できます。
`-s` *name*	*name* が示すログイン名用のパスワード属性を表示します。レポジトリが `nisplus` の場合、この指定はすべてのユーザーが使用できます。`files` の場合には、スーパーユーザーだけが使用できます。また `nis` の場合には、パスワードの有効期限を設定する (エージング) 機能がサポートされていないので、この指定はどのユーザーも使用できません。

特権ユーザーのオプション

特権ユーザーのみが、次のオプションを使用することができます。

`-d`	*name* に対するパスワードを削除します。ログイン *name* に対するパスワードを入力するように要求することはありません。この指定は、レポジトリが `files` の場合にのみ有効です。
`-f`	*name* のパスワードの期限を切ることによって、次のログインセッションで パスワードを強制的にユーザーに変更させます。
`-l`	*name* に対するパスワードエントリをロックします。
`-n` *min*	*name* の最小のフィールドを設定します。*min* フィールドには、*name* に対するパスワードを変更するために最低限必要な日数が設定されます。*min* が *max* より大きい場合は、ユーザーはパスワードを変更できません。このオプションは、必ず `-x` オプションとと

もに使用してください。ただし、*max* が -1 に設定されている場合は例外です (最低経過日数を設定する機能がオフ)。この場合は、*min* を設定する必要はありません。

-w *warn*　*name* の警告フィールドを設定します。*warn* フィールドには、パスワードの有効期限が切れる何日前に、期限切れ近いことを示す警告をユーザーが受けるか、その日数が設定されます。このオプションは、パスワードの有効期限が切れている場合は 無効です。

-x *max*　*name* の最大のフィールドを設定します。*max* フィールドには、パスワードが *name* に対して有効である最大の日数が設定されます。*max* が -1 に設定された場合は、*name* の有効期限を設定する機能はただちにオフになります。*max* が 0 に設定された場合は、ユーザーは次のログインセッションで パスワードを強制的に変更させられ、有効期限を設定する機能はオフになります。

オペランド

以下のオペランドを指定できます。

name　ユーザーのログイン名

環境

LC_* 変数 (LC_CTYPE、 LC_MESSAGES、 LC_TIME、 LC_COLLATE、 LC_NUMERIC、 LC_MONETARY) (environ(5) 参照) のいずれも環境に設定されていなければ、それぞれ対応する ロケール のカテゴリにおける passwd の動作は、環境変数 LANG によって決定されます。もし、LC_ALL が設定されていれば、その内容が LANG 変数やその他の LC_* 変数より優先されます。上記の変数が環境にまったく設定されていなければ、C ロケール (米国スタイル)が passwd の動作を決定します。

LC_CTYPE　passwd の文字の処理方法を決定します。LC_CTYPE に有効な値が設定されていると、passwd は、その ロケール にあった文字を含むテキストやファイル名を表示および処理できます。passwd は拡張 Unix コード(EUC)も表示および処理できます。この場合、文字は 1 バイト幅、2 バイト幅、3 バイト幅のいずれも使用できます。また、passwd は 1、2、またはそれ以上のカラム幅の EUC 文字も処理することができます。C ロケール においては、ISO 8859-1 の文字だけが有効です。

LC_MESSAGES　診断メッセージや情報メッセージの表示方法を決定します。また、メッセージの言語とスタイル、そして肯定応答および否定応答の正しい形も決定します。C ロケールにおいては、メッセージはプログラム自身が使用しているデフォルトの形で表示されます (通常、米語)。

終了ステータス

passwd は、処理終了時に以下のいずれかの値を返します。

0　正常終了

1　アクセス権が与えられていません

2　オプションの組み合わせが無効です

3　予想できない失敗。パスワードファイルは変更されません

4 予想できない失敗。パスワードファイルがありません

5 パスワードファイルは使用中です。後で行なってください

6 オプションに対する引数が無効です

7 有効期限のオプションが無効です

8 メモリーがありません

9 システムエラー

10 アカウントが期限切れです

ファイル

/etc/oshadow

/etc/shells

/etc/passwd パスワードファイル

/etc/shadow シャドウパスワードファイル

/etc/default/passwd デフォルト値は /etc/default/passwd 中の次のフラグで設定することができます。例: MAXWEEKS=26

- MAXWEEKS パスワードが有効な最大期間
- MINWEEKS パスワード変更に最低限必要な期間
- PASSLENGTH パスワードに最低限必要な文字数
- WARNWEEKS パスワードの有効期間が切れる前の警告期間

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

finger(1), login(1), nistbladm(1), domainname(1M), eeprom(1M), id(1M), passmgmt(1M), pwconv(1M), su(1M), useradd(1M), userdel(1M), usermod(1M), crypt(3C), getpwnam(3C), getspnam(3C), getusershell(3C), nis_local_directory(3NSL), pam(3PAM), loginlog(4), nsswitch.conf(4), pam.conf(4), passwd(4), shadow(4), attributes(5), environ(5), pam_authtok_check(5), pam_authtok_get(5), pam_authtok_store(5), pam_dhkeys(5), pam_ldap(5), pam_unix(5), pam_unix_account(5), pam_unix_auth(5), pam_unix_session(5)

注意事項

pam_unix(5) モジュールは、将来のリリースではサポートされなくなる可能性があります。同様の機能は、pam_unix_account(5)。pam_unix_auth(5)、pam_unix_session(5)、pam_authtok_check(5)、pam_authtok_get(5)、pam_authtok_store(5)、pam_dhkeys(5)、および pam_passwd_auth(5) で提供されています。

nispasswd および ypasswd コマンドは、passwd のラッパーです。nispasswd および ypasswd の代わりに、passwd -r *repository_name* を使用することをお勧めします。

NIS+ は、Solaris™ オペレーティング環境の将来のリリースではサポートされなくなる可能性があります。Solaris 9 オペレーティング環境には、NIS+ から LDAP への移行を支援するツールが含まれています。詳細については、http://www.sun.com/directory/nisplus/transition.html を参照してください。

名前 | paste – 複数のファイルの対応する行および以降の行のマージ

形式 | **paste** [-s] [-d *list*] *file...*

機能説明 | paste ユーティリティは、複数の入力ファイルから得られる対応する行を連結し、その結果を標準出力に書き出します。

paste のデフォルト動作は、複数の入力ファイル中の対応する行の連結です。 最終ファイル以外の各入力ファイルから得た行において、 復帰改行文字をタブ文字に置き換えます。

入力ファイルの 1 つまたは複数 (すべてではない) でファイルの終わりが検出された場合、 -s オプションが指定されていなければ、 paste はそれらのファイルから空行を読み込んだかのように動作します。

オプション | 以下のオプションを指定できます。

-d *list*
: *list* 引数中にバックスラッシュ (\ ) が指定された場合を除き、 *list* 中の各文字は区切り文字を表す要素となります。 *list* 中にバックスラッシュが指定されていれば、後述するように、 そのバックスラッシュとそれに続くいくつかの文字が 区切り文字を表す要素となります。 これらの要素により、入力行の復帰改行文字を置き換える文字として、 デフォルトのタブの代わりに用いるべき区切り文字を指定します。 *list* 中の要素は循環して使われます。つまりリストの最終要素に到達したら、 次は先頭の要素に戻ります。

-s オプションが指定されると、次のような動作が行われます。

- ファイル中の最後の復帰改行文字は変更されない。
- 各 file オペランドを処理した後、区切り文字はリストの先頭要素に戻る。

-s オプションが省略されたときは、次のような動作となります。

- 最後の file が示すファイル中の復帰改行文字は変更されない。
- 1 つのファイルから得た 1 つの行が処理されるたびに、区切り文字はリストの先頭要素に戻る。

list 中にバックスラッシュが指定された場合、 そのバックスラッシュとそれに続く 1 文字の組み合わせで以下のように区切り文字を表します。

\n
: 復帰改行文字

\t
: タブ文字

\\
: バックスラッシュ文字

\0
: 空の文字列 (NULL 文字ではない) 。 \0 の直後の文字が x または X のとき、もしくは LC_CTYPE digit キーワードで定義された文字のとき、処理結果は予測できません。

バックスラッシュの直後の文字が上記以外の場合、 処理結果は予測できません。

`-s` コマンド行で指定された順序で、 各ファイルにおいてすべての行を連結します。 各ファイルにおいて、最終行にない復帰改行文字は、 `-d` オプションで指定されない限りタブ文字に置き換えられます。

オペランド 以下のオペランドを指定できます。

file 入力ファイルのパス名。 `-` を指定すると、標準入力と見なされます。 `-` を複数個指定すると、その各々に対して標準入力から 1 行ずつが循環して読み込まれます。 最高 12 個の *file* オペランドを指定できます。

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `paste` の動作については、`largefile`(5) を参照してください。

使用例 **例 1** ディレクトリを 1 カラムで表示

```
ls | paste -d" " -
```

例 2 ディレクトリを 4 カラムで表示する

```
ls | paste - - - -
```

例 3 2 行ずつ連結する

```
paste -s -d"\ t\ n" file
```

環境 `paste` の実行に影響を与える環境変数 `LC_CTYPE` 、 `LC_MESSAGES` についての詳細は、 `environ`(5) を参照してください。

終了ステータス 以下の終了ステータスが返されます。

`0` 正常終了

`>0` エラーが発生した

属性 次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 `cut`(1), `grep`(1), `pr`(1), `attributes`(5), `environ`(5), `largefile`(5)

診断

`line too long` 出力行は 511 文字に制限されます。

`too many files` `-s` オプションを除いては、入力ファイルを 12 以上指定することはできません。

`no delimiters`	`-d` オプションに指定されたリストが空でした。
`cannot open` *file*	指定されたファイルがオープンできませんでした。

名前 | pack, pcat, unpack – ファイルの圧縮および復元

形式 | **pack** [-f] [-] *file*...

pcat *file*...

unpack *file*...

pack | pack コマンドは、指定されたファイルを圧縮した形で保存します。 可能であれば (そして便利ならば)、 *file* という入力ファイルは、 *file* と同じアクセスモード、 同じアクセスの日付や変更した日付、同じ所有者を持つ *file*.z という圧縮されたファイルに置き換えられます。 pack の実行が正常に終了すると、 *file* は削除されます。

圧縮の総量は、入力ファイルのサイズおよび文字度数分布によります。 デコード用ツリーがそれぞれの .z ファイルの最初の部分を形成しているので、 3 ブロック未満のファイルを圧縮することは、あまり意味がありません。 ただし、プリンタプロットや図形の場合に起こるように、 文字度数分布が非常に偏っているものは例外です。

一般的にテキストファイルを元のサイズの 60% から 75% に圧縮します。大きい文字セットを使用し、 文字分布が一様なロードモジュールは、 元のサイズのおよそ 90% にしか圧縮されません。

pack は、圧縮できなかったファイルの数を示す値を返します。

次の場合には、圧縮は行われません。

- ファイルがすでに圧縮されている場合
- ファイル名が 14 – 2 バイトより長い場合
- ファイルにリンクがある場合
- ファイルがディレクトリの場合
- ファイルをオープンできない場合
- ファイルが空の場合
- 圧縮によってディスクブロックを減らせない場合
- *file*.z というファイルが既に存在している場合
- .z ファイルを作成することができない場合
- 処理中に、入出力エラーが発生した場合

ファイル名の最後のセグメントは、 {NAME_MAX} – 2 文字以下でなければなりません。 この 2 文字は、追加される拡張子 .z 用です。ディレクトリは圧縮できません。

pcat | pcat コマンドは、 cat(1) が通常ファイルに対して行うことを、 圧縮したファイルに対して行います。 ただし pcat をフィルタとして使うことはできません。 指定したファイルは、復元され、標準出力に書き込まれます。

pcat は、復元できなかったファイルの数を返します。 以下の場合にはエラーとなります。

- ファイルがオープンできなかった
- ファイルが、 pack の出力ファイルと認識できなかった

unpack unpack コマンドは、 pack で作成したファイルを復元します。 コマンドで指定したファイル *file* に対して、 *file*.z というファイル (あるいは *file* が .z で終わる場合は単に *file*)を検索します。 このファイルが圧縮されたファイルである場合は、 復元したファイルに置き換えます。新たなファイル名は .z 接頭辞が取り除かれ、アクセスモード、アクセス日付や変更日 付および所有者名は圧縮されたファイルと同じです。

unpack は、復元できなかったファイルの数を示す値を返します。 pcat で述べた理由のほかに、次のような場合には復元できません。

- "unpackされた"ときのファイル名が既に存在する場合
- 復元したファイルを作成できない場合
- ファイル名の長さ (拡張子 .z を除く) が 14 バイトを超えている場合

オプション 以下のオプションを指定できます。

-f *file* の強制圧縮。ディレクトリ全体を圧縮するのに有効です。 ただし、中には 圧縮しても小さくならないファイルがあります。 unpack または pcat は、圧縮したファイルを元の形式に復元できます。

オペランド 以下のオペランドを指定できます。

file pack、 unpack、または pcat するファイルのパス名。 *file* には拡張子 .z を指定しても省略してもかまいません。

– pack は、 1 バイトごとに Huffman (最小冗長度) コードを使用します。 – 引数を使用すると、それぞれのバイトの使用回数、 相対頻度およびバイトのコードを標準出力に出力するよ うに内部フラグが設定されます。 *file* の代わりに – を追加すると、内部フラグを設定し、リ セットします。

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の pack 、 pcat 、 unpack の動作については、 largefile(5) を参照してください。

使用例

例 1 圧縮ファイルを表示する

file.z という名前の圧縮ファイルを見るには、次のようにします。

```
example% pcat file.z
```

あるいは、単に次のようにします。

```
example% pcat file
```

例 2 圧縮したファイルの復元コピーを作成する

file.z という名前の圧縮したファイルの復元コピー nnn を file.z を破壊せずに作成するには 、次のコマンドを使用します。

```
example% pcat file >nnn
```

環境

pack 、 pcat 、 unpack の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した。圧縮または復元できなかった ファイルの数が返される

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

cat(1), compress(1), zcat(1), attributes(5), environ(5), largefile(5)

名前 | perl – 抽出および出力を行う言語 (Practical Extraction and Report Language)

形式 | **perl** [-sTuU] [-hv] [-V[:*configvar*]] [-cw] [-d[:*debugger*]] [-D[*number/list*]] [-pna] [-F*pattern*] [-l[*octal*]] [-0[*octal*]] [-I*dir*] [-m[-]*module*] [-M[-]'*module*...'] [-P] [-S] [-x[*dir*]] [-i[*extension*]] [-e '*command*'] [--] [*programfile*] [*argument*]...

機能説明 | perl 関連のマニュアルページは、以下のように複数のマニュアルページに分割されています。

perl	Perl の概要 (このマニュアルページ)
perlfaq	Perl に関する FAQ
perltoc	Perl に関するマニュアルの目次
perlbook	Perl 関連書籍
perlsyn	Perl の構文
perldata	Perl のデータ構造体
perlop	Perl の演算子と優先順位
perlsub	Perl サブルーチン
perlfunc	Perl の組み込み関数
perlreftut	Perl リファレンスの概要
perldsc	Perl のデータ構造体の概要
perlrequick	Perl の正規表現についてのクイックスタート
perlpod	Perl に関する過去のマニュアル
perlstyle	Perl のスタイルガイド
perltrap	Perl の注意点
perlrun	Perl の実行とオプション
perldiag	Perl の診断メッセージ
perllexwarn	Perl の警告とその制御
perldebtut	Perl のデバッグについてのチュートリアル
perldebug	Perl のデバッグ
perlvar	Perl の定義済み変数
perllol	Perl のデータ構造体の配列
perlopentut	Perl open() についてのチュートリアル
perlretut	Perl の正規表現についてのチュートリアル
perlre	Perl の正規表現についての詳細
perlref	Perl についてのリファレンスの詳細
perlform	Perl の書式
perlboot	Perl OO についての初心者用チュートリアル
perltoot	Perl OO についてのチュートリアル (パート 1)
perltootc	Perl OO についてのチュートリアル (パート 2)
perlobj	Perl オブジェクト
perlbot	Perl OO を使用するコツと使用例
perltie	単純な変数によって無効になる Perl オブジェクト
perlipc	Perl プロセス間の通信
perlfork	Perl fork() の情報
perlnumber	Perl の数に関する意味論
perlthrtut	Perl スレッドについてのチュートリアル
perlport	Perl の移植
perllocale	Perl でサポートされるロケール
perlunicode	Perl でサポートされる Unicode

```
perlsec           Perl のセキュリティ


perlmod           Perl モジュール: 動作のしくみ
perlmodlib        Perl モジュール: 記述方法と使用方法
perlmodinstall    Perl モジュール: CPAN からのインストール方法
perlnewmod        Perl モジュール: 配布用の新規モジュールの作成

perlfaq1          Perl に関する一般的な質問
perlfaq2          Perl の入手と学習
perlfaq3          プログラミングツール
perlfaq4          データ処理
perlfaq5          ファイルとフォーマット
perlfaq6          regexes (正規表現)
perlfaq7          Perl 言語について
perlfaq8          システムとの相互作用
perlfaq9          ネットワーク

perlcompile       Perl コンパイラ群の概要

perlembed         C または C++ のアプリケーションに Perl を埋め込む方法
perldebguts       Perl デバッグのコツとヒント
perlxstut         Perl XS (チュートリアル)
perlxs            Perl XS アプリケーションプログラミングインタフェース
perlclib          標準 C ライブラリ関数の内部代替関数
perlguts          Perl を拡張するための内部関数
perlcall          C から Perl を呼び出すための規約
perlutil          配布される Perl に同梱のユーティリティ
perlfilter        Perl ソースフィルタ
perldbmfilter     Perl DBM フィルタ
perlapi           Perl API リスト (自動生成)
perlintern        Perl 内部関数 (自動生成)
perlapio          Perl の内部入出力 abstract (抽象) インタフェース
perltodo          Perl に必要な作業
perlhack          Perl の応用

perlhist          Perl の歴史
perldelta         前バージョンからの変更点
perl5005delta     Perl バージョン 5.005 に加えられた変更点
perl5004delta     Perl バージョン 5.004 に加えられた変更点

perlsolaris       Solaris 用の Perl についての注意事項
```

Perl のマニュアルページ全体を初めて読む場合は、上記の順序で読んでいくことをお勧めします。内容を理解しやすい順番になっています。

上記のマニュアルページは /usr/local/man/ ディレクトリにインストールされています。

上記のマニュアルページのほかにも、Perl モジュールに関するマニュアルページがあります。これらも /usr/local/lib/perl5/man ディレクトリにあります。マニュアルページの一部は標準で Perl に提供されています。ユーザーが任意でインストールするサードパーティのモジュールに関するマニュアルページも、同じディレクトリに置かれます。

Perl 関連のマニュアルページに対して catman(1M) を実行することはサポートされていません。その他の Solaris 固有の情報については、後述の「注意事項」の項を参照してください。

Perl に関する情報は、/usr/perl5/bin/perldoc スクリプトを使用して参照することもできます。

プログラム中に何か問題があるときにその場所を特定できない場合は、-w スイッチを使用してください。多くの場合に問題箇所を特定するできます。

Perl は、テキストファイルを読み取り、そこから情報を抽出して、その情報をもとにして報告を出力する言語です。また、システム管理作業を行うのにも適した言語です。Perl は、美しいこと (小規模、上品、最小限) よりも実用性 (使いやすい、効率、完全) を目指して作成されています。

Perl は (筆者の意見では) 、C、sed、awk、sh の長所を組み合わせて作成されているので、これらの言語に慣れている方には、Perl を使用することはそれほど難しくないでしょう。(csh、Pascal、BASIC-PLUS の特徴も入っているという意見もあります。) Perl の式の構文はかなりの部分で、C の式の構文に対応しています。多くの UNIX ユーティリティとは異なり、Perl にはデータのサイズに制限がありません。つまり、メモリーが十分にあれば、Perl ではファイル全体を 1 つの文字列中に入れてしまうこともできます。再帰の深さにも制限がありません。パフォーマンスの低下を防ぐために、ハッシュテーブル (連想配列と呼ばれることもある) のサイズが必要に応じて自動的に大きくなります。Perl は、高度なパターンマッチング技術を使用して、大量のデータを高速に読み取ることができます。Perl はテキストファイルを扱うことを用途として最適化されていますが、バイナリデータを扱ったり、dbm ファイルをハッシュテーブルのようにして扱うこともできます。データフローのトレース機構によって、多くの不注意によるセキュリティホールが防止されているので、Perl の setuid スクリプトは C プログラムよりも安全です。

通常なら sed、awk、または sh を使用して記述するような処理をさせたい場合に、sed、awk、sh では不可能またはパフォーマンスをもっと速くしたい、けれども C で記述するほどではないというとき、Perl を使用するのが適しています。また、sed および awk のスクリプトを Perl スクリプトに変換するプログラムもあります。

1993 年に誕生して以来 (perlhist のマニュアルページを参照)、Perl はバージョン 5 で、ほぼ全体が改訂され、次のような特長が追加されています。

- 多数のモジュールを使用した、モジュール性と再利用性
  perlmod、perlmodlib、および perlmodinstall のマニュアルページを参照してください。
- 組み込みと拡張性
  perlembed、perlxstut、perlxs、perlcall、perlguts、および xsubpp のマニュアルページを参照してください。
- 独自のマジック変数の活用 (複数の DBM同時実装を含む)
  perltie および AnyDBM_File のマニュアルページを参照してください。
- サブルーチンのオーバーライド、自動ロード、プロトタイプ化が可能。perlsub のマニュアルページを参照してください。

- 任意の多重データ構造体および匿名の関数
  perlreftut、perlref、perldsc、および perllol のマニュアルページを参照してください。
- オブジェクト指向プログラミング
  perlobj、perltoot、および perlbot のマニュアルページを参照してください。
- C コードまたは Perl バイトコードへのコンパイル
  B および B::Bytecode のマニュアルページを参照してください。
- 軽量プロセス (スレッド) のサポート
  perlthrtut および Thread のマニュアルページを参照してください。 Solaris の一部として出荷されている Perl では、スレッドサポートが有効化されていません。スレッドサポートが必要な場合は、独自のバージョンの Perl を構築およびインストールしてください (後述の「注意事項」の項を参照)。
- 国際化、各国語化、および Unicode のサポート
  perllocale および utf8 のマニュアルページを参照してください。
- 字句のスコープ
  perlsub のマニュアルページを参照してください。
- 正規表現の拡張
  perlre および perlop のマニュアルページを参照してください。
- 統合エディタの採用による、デバッガの機能拡張および
  対話型の Perl 環境。perldebug のマニュアルページを参照してください。
- POSIX 1003.1 準拠
  POSIX のマニュアルページを参照してください。

使用条件

Perl は、ほとんどのオペレーティングシステム環境の、ほぼすべての UNIX 仕様のプラットフォーム上で利用できます。一覧については、perlport のマニュアルページの「Supported Platforms」の項を参照してください。

環境

Solaris に含まれている Perl は、/usr/local ではなく、/usr/perl5 にインストールされます。これは、ユーザーがデフォルトの /usr/local にインストールする別バージョンの Perl と衝突しないようにするためです

追加でインストールするモジュールは、/usr/perl5/site_perl/5.6.1 ディレクトリに置かれます。/usr/perl5/vendor_perl ディレクトリには、Sun が提供するモジュールが置かれます。

perldoc および perlbug などの Perl スクリプトは、/usr/perl5/bin ディレクトリにあります。これらのスクリプトを使用する場合は、PATH 環境変数に /usr/perl5/bin を追加してください。

perlrun のマニュアルページも参照してください。

著者

Larry Wall 他

ファイル

"@INC"　Perl ライブラリの場所

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWpl5u SUNWpl5m SUNWpl5p SUNWopl5u SUNWopl5m SUNWopl5p
インタフェースの安定性	
• スクリプトインタフェース	開発中
• XSUB インタフェース	開発中
• バイナリインタフェース	不安定
• ディレクトリ構成	開発中

関連項目

a2p	awk から perl への変換プログラム
s2p	sed から perl への変換プログラム
http://www.perl.com/	Perl のホームページ
http://www.perl.com/CPAN	総合的な Perl アーカイブ

診断 | use warnings プラグマ (および -w スイッチ) を使用すると、診断メッセージが表示されます。

Perl の診断メッセージについての説明は、perldiag のマニュアルページを参照してください。use diagnostics プラグマを使用すると、警告メッセージおよびエラーメッセージがより詳細に出力されます。

コンパイル時のエラーメッセージには、エラー箇所の行番号、確認が必要な次のトークンまたはトークンタイプが示されます。複数の -e スイッチによって Perl に渡されるスクリプトでは、各 -e スイッチが 1 行として数えられます。

setuid スクリプトには、"Insecure dependency" などのエラーメッセージとして示されるような制限事項があります。詳細は perlsec のマニュアルページを参照してください。

次に、-w について説明します。

注意事項 | Solaris 9 には、Perl 5.005_03 (Solaris 8 で提供) と Perl 5.6.1 の、 2 つのバージョンの Perl が含まれています。/bin/perl は 5.6.1 のインタープリタへのリンクで、/usr/perl5/bin は /usr/perl5/5.6.1/bin ディレクトリへのリンクです。バージョン 5.005_03 は、将来の Solaris リリースでは削除される可能性があります。

Perl 5.6.1 のインタープリタは 32 ビットアプリケーションですが、Perl 5.6.1 自体は大規模ファイルの処理と64 ビットの整数の使用を想定して設計されています。詳しい構成情報を参照するには、`perl -V` および `perlbug -dv` を使用してください。

5.6.1 は、5.005_03 バージョンとバイナリ互換ではありません。主な原因は、5.6.1 に大規模ファイルと 64 ビット整数のサポートが追加されているためです。ユーザーがインストールしたモジュールが存在する場合、それが XSUB ベースのモジュールであれば再コンパイル、XSUB ベースでないモジュールであれば再インストールする必要があります。

5.005_03 を必要とするアプリケーションでは、`/usr/perl5/5.005_03/bin/perl` を明示的に使用してください。お勧めはしませんが、デフォルトの Perl バージョンを 5.005_03 にすることも可能です。5.005_03 をデフォルトの Perl バージョンにするには、スーパーユーザーになって以下のコマンドを実行します。

```
# rm /usr/bin/perl
# ln -s ../perl5/5.00503/bin/perl /usr/bin/perl
# rm /usr/perl5/bin
# ln -s ./5.00503/bin /usr/perl5/bin
# rm /usr/perl5/man
# ln -s ./5.00503/man /usr/perl5/man
# rm /usr/perl5/pod
# ln -s ./5.00503/pod /usr/perl5/pod
```

独自のバージョンの Perl を作成してインストールする場合でも、`/usr/perl5` にある 5.6.1 バージョンの Perl は削除しないでください。このバージョンの Perl は、複数のシステムユーティリティで使用されています。5.005_03 バージョンを使用しない場合は、削除してもかまいません。Perl のパッケージ名は、以下のとおりです。

```
SUNWpl5u     Perl 5.6.1
SUNWpl5p     Perl 5.6.1 (POD ドキュメント)
SUNWpl5m     Perl 5.6.1 (マニュアルページ)
SUNWopl5u    Perl 5.005_03
SUNWopl5p    Perl 5.005_03 (POD ドキュメント)
SUNWopl5m    Perl 5.005_03 (マニュアルページ)
```

使用上の留意点

`-w` スイッチを使用することは、必須事項ではありません。

Perl は、型キャストを行う `atof()` や、浮動小数点を出力する `sprintf()` など、さまざまな処理のマシン上での定義に影響されます。

標準入出力が、特定のストリーム上の読み取りと書き込みの間にシークまたは EOF を必要とする場合、Perl がそれを実行します。ただしこれは `sysread()` および `syswrite()` には適用されません。

組み込みデータ型にサイズ制限 (メモリーサイズを除く) はありませんが、若干の制限事項があります。変数名は 251 文字未満にする必要があります。診断メッセージに示される行番号は内部で short 型整数として格納されるので、表示できる行番号は 65535 行目までです。これ以上の行番号は通常 wraparound による影響を受けます。

発見したバグについて、電子メールで <perlbug@perl.com> 宛に報告することができます。この時、使用している環境の構成情報 (Perl のソースツリーにある myconfig コマンドまたは perl -V の実行結果) も必ずお知らせください。

名前 | pg – CRT 用のファイル閲覧フィルタ

形式 | **pg** [–*number*] [-p *string*] [-cefnrs] [+ *linenumber*] [+/ *pattern* /] [*filename*...]

機能説明 | pg コマンドは、 *filename* を CRT 上で 1 度に 1 画面分調べることができるフィルタです。 ユーザーが復帰改行を入力すると、 他のページが表示されます。他の機能については次に示します。

このコマンドは、ユーザーがすでに通過したものを 元に戻って見直すことができるという点で、 今までのページングコマンドと異なります。 この方法については、以下で説明します。

端末の属性を判定するために、 pg は、 terminfo(4) データベースを操作し、環境変数 TERM によって指定された端末タイプを求めます。 TERM が定義されていない場合、端末タイプは dumb と見なされます。

オプション

-*number*	ウィンドウのサイズ (行数) を指定するための整数 (24 行を表示する端末では、デフォルトのウィンドウサイズは 23)
-p *string*	pg は、 *string* をプロンプトとして使用します。プロンプトの文字列に %d がある場合は、プロンプトの中の %d は、プロンプトが出されたときの現在のページ番号と置き換えられま す。デフォルトのプロンプト文字列は “:” です。
-c	各ページを表示する前にカーソルをホームポジションに戻し、 画面をクリアします。 terminfo(4) データベースに、使用する端末タイプ用の clear_screen が定義されてない場合は、このオプションは無視されます。
-e	pg は、各ファイルの終りで停止しません。
-f	通常、 pg は、スクリーン幅よりも長い行を分割します。 しかし、表示されているテキストの文字シーケンスのなかには、 好ましくない結果を生じるものもあります (たとえば、下線のためのエスケープシーケンス) 。 -f オプションは、 pg が行を分割しないようにします。
-n	通常、コマンドは <*newline*> 文字で終了しなければなりません。このオプションを指定すると、 コマンド文字を入力するとすぐにコマンドは自動的に終了します。
-r	制限モード。シェルエスケープは却下されます。 pg は、エラーメッセージを出力しますが、終了しません。
-s	pg は、 標準出力モード (通常は、反転映像) で すべてのメッセージおよびプロンプトを出力します。
+*linenumber*	*linenumber* から開始します。
+/*pattern*/	正規表現パターンを含む最初の行から開始します。

オペランド | 以下のオペランドを指定できます。

filename　表示するテキストファイルのパス名。 *filename* を省略するか、あるいは – を指定すると、標準入力と見なされます。

コマンド

pg 入力を待っているときに打ち込むことができる応答は、 閲覧の続行、検索、閲覧環境の変更という 3 つのカテゴリに分割できます。

閲覧を続けるためのコマンドには、前に *address* が来ます。 これは、次のテキストが表示される場所を示す 任意の符号付き数 です。この *address* は、コマンドによって、ページか行のいずれかに解釈されます。符号付き *address* は、現在のページあるいは行からの相対的な場所を指定し、符号なし *address* は、ファイルの初めからの絶対アドレスを指定します。 いずれのコマンドにも、 アドレスが指定されていない場合に使用される デフォルトのアドレスがあります。

閲覧のためのコマンドとそのデフォルトは、次のとおりです。

(+1)<*newline*> または <*blank*>　1 ページ表示します。アドレスは、ページ単位で指定されます。

(+1) `l`　相対アドレスと共に使うと、 pg は指定された行数だけ画面を順方向または逆方向にスク ロールをシミュレートします。絶対アドレスと共に使うと、 このコマンドは、指定された行から始まる 1 画面を出力します。

(+1) `d` または `^D`　順方向または逆方向に半画面のスクロールをシミュレートします。

i`f`　テキストの *i* 個のスクリーンをスキップします。

i`z`　<*newline*> と同じですが、 *i* がある場合、 *i* が デフォルトの新しい 1 画面の行数になります。

次の閲覧コマンドに *address* は必要ありません。

`.` または `^L`　ピリオドを単独で打ち込むと、 テキストの現在のページが再表示されます。

`$`　ファイルの中の最後の 1 ウィンドウを表示します。 入力がパイプである場合は、注意してください。

以下のコマンドは、 テキスト中のテキストパターンの検索に利用できます。 正規表現については `regex`(5) のマニュアルページを参照してください。 `-n` オプションを指定する場合でも、必ず <*newline*> で終了しなければなりません。

i/*pattern*/　*i* 回目の *pattern* の 出現を順方向に検索します (デフォルトは *i*1=*1*) 。検索は、現在のページの直後から、 現在のファイルが終了するまで続きます。循環はしません。

i^*pattern*^

i?*pattern*?　*i* 回目の *pattern* の 出現を逆方向に検索します (デフォルトは *i*=1) 。検索は、現在のページの直前から、 現在のファイルの初めまで続きます。循環はしません。 ^ 表記は、 ? の取り扱いが適切でない Adds 100 端末に有効です。

通常 pg は検索の後で、 発見された行を画面の最上段に表示します。 検索コマンドに m または b を追加することによって、それ以降ウィンドウの中央、 または最下段で見つかった行を残しておくことができます。 接尾辞 t を使用して、元の状態に戻すことができます。

pg のユーザーは、以下のコマンドを使用して、 閲覧の環境を変更することができます。

*i*n	コマンド行の中の *i* 個後のファイルの閲覧を始めます。 *i* は、符号なしの数で、デフォルトは 1 です。
*i*p	コマンド行の中の *i* 個前のファイルの閲覧を始めます。 *i* は、符号なしの数で、デフォルトは 1 です。
*i*w	テキストの他のウィンドウを表示します。 *i* が存在する場合は、ウィンドウサイズを *i* に設定します。
s *filename*	指定されたファイルに入力を保存します。 現在閲覧されているファイルのみが保存されます。 s と *filename* の間の空白は任意です。 -n オプションを指定する場合でも、このコマンドは、必ず <*newline*> で終了しなければなりません。
h	利用可能なコマンドの一覧を簡略化して表示します。
q または Q	pg を終了します。
!command	command を、 SHELL 環境変数に指定されているシェルに引き渡します。 SHELL 環境変数に値が指定されていない場合は 、デフォルトのシェルを使用します。 -n オプションを指定する場合でも、このコマンドは、必ず <*newline*> で終了しなければなりません。

出力が端末に送信されるときはいつでも、ユーザーは、中止キー (通常は CTRL-\ ) または、割り込み (ブレーク) キーを打つことができます。これによって、 pg は出力の送信を停止し、プロンプトを表示します。 またユーザーは、前述のコマンドの 1 つを通常の方法で入力することができます。 残念ながら、この場合、出力がなくなる場合があります。 なぜなら、終了シグナルが発生すると、 端末の出力待ちキューの文字がフラッシュされるからです。

標準出力が端末ではない場合は、 pg は cat(1) と同じような働きをします。 ただし、ファイルが 2 つ以上ある場合、 各ファイルの前にヘッダが出力されます。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の pg の動作については、 largefile(5) を参照してください。

使用例

例 1 pg を使用してシステム情報を読み取る

次のコマンド行は、 pg を使用してシステム情報を読み取ります。

```
example% news | pg -p "(Page %d):"
```

環境

pg の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

以下の環境変数も pg の実行に影響を与えます。

COLUMNS　水平方向の画面サイズを決定します。 この変数が設定されていない、あるいは NULL に設定されているときは、 TERM の値、ウィンドウサイズ、またはボーレート、もしくはこれらの 値の組み合わせにより、 画面サイズ計算用の端末タイプが表されます。

LINES　画面に表示する行数を決定します。 この変数が設定されていない、あるいは NULL に設定されているときは、 TERM の値、ウィンドウサイズ、またはボーレート、もしくはこれらの 値の組み合わせにより、 画面サイズ計算用の端末タイプが表されます。

SHELL　! コマンド用に実行するコマンドインタプリタの名前を決定します。

TERM　端末の属性を決定します。またオプションとして、 TERM の値に基づいた、システムに依存するデータベースの 検索を試みます。情報が何も存在しない場合には、 カーソルによるアドレス指定が不可能な端末と見なされます。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

ファイル

/tmp/pg*　入力がパイプからの場合の一時ファイル

/usr/share/lib/terminfo/?/*　端末情報データベース

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

cat(1), grep(1), more(1), terminfo(4), attributes(5), environ(5), largefile(5), regex(5)

注意事項

端末入力を待っている間に、 pg は BREAK 、 CTRL-C 、および CTRL–\ に応答して実行を終了します。 しかし、プロンプトの出ている間であれば、これらのシグナルは、 pg の現在のタスクに割り込み、プロンプトモードに戻ります。 割り込みによってパイプラインの中の他のコマンドが 終了してしまう可能性があるので、 パイプから入力を読み取るときには注意が必要です。

区切り記号 / 、 ^ 、または ? は、検索コマンドから省略することができます。

区切り記号タブが 8 カラムごとに設定されていないと、好ましくない結果が生じることがあります。

端末 I/O オプションを変更する他のコマンドとともに pg をフィルタとして使用すると、 端末設定が正しく復元されないことがあります。

名前 | pgrep, pkill – 名前または他の属性によるプロセスの検出またはシグナル送信

形式 | **pgrep** [-flvx] [-n | -o] [-d *delim*] [-P *ppidlist*] [-g *pgrplist*] [-s *sidlist*] [-u *euidlist*] [-U *uidlist*] [-G *gidlist*] [-J *projidlist*] [-t *termlist*] [-T *taskidlist*] [*pattern*]

pkill [-*signal*] [-fnvx] [-P *ppidlist*] [-g *pgrplist*] [-s *sidlist*] [-u *euidlist*] [-U *uidlist*] [-G *gidlist*] [-J *projidlist*] [-t *termlist*] [-T *taskidlist*] [*pattern*]

機能説明 | pgrep ユーティリティはシステム上のアクティブなプロセスを調べて、コマンド行に指定した条件を満たす属性を持つプロセスのプロセス ID を報告します。 各プロセス ID は 10 進数値として出力され、プロセス ID は区切り文字 (デフォルトは復帰改行) で区切られます。コマンド行で属性オプションに値を指定するときは、複数の値をコンマで区切って指定できます。

```
pgrep -G other,daemon
```

この例は、実グループ ID が other または daemon であるプロセスを検出します。条件となるオプションを複数指定すると、pgrep は、条件オプションの論理積に一致する属性を持つプロセスを検出します。

```
pgrep -G other,daemon -U root,daemon
```

この例は、次の属性を持つプロセスを検出します。

(実グループ ID が other または daemon であり) かつ
(実ユーザー ID が root または daemon である)

pkill の機能は pgrep の機能と同等ですが、検出したプロセスのプロセス ID を出力するのではなく、kill(1) と同様に、各プロセスにシグナルを送信します。シグナルの名前または番号は pkill への最初のコマンド行オプションとして指定できます。

オプション | 次のオプションを指定できます。

-d *delim* — 一致したプロセス ID 間に置かれる区切り文字を指定します。-d オプションを指定しない場合、デフォルトの区切り文字は復帰改行です。-d オプションは pgrep へのオプションとして指定した場合にのみ有効です。

-f — 正規表現 *pattern* の照合は、プロセスの引数を含む文字列全体 (/proc/*nnnnn*/psinfo ファイルの pr_psargs フィールドから取得) に対して行われます。-f オプションを指定しないと、正規表現 *pattern* の照合は、実行可能ファイルの名前 (/proc/*nnnnn*/psinfo ファイルの pr_fname フィールドから取得) に対してのみ行われます。

`-g` *pgrplist*
: 指定したリストのプロセスグループ ID があるプロセスだけに一致します。リストにグループ 0 がある場合、これは `pgrep` または `pkill` プロセスのプロセスグループ ID として解釈されます。

`-G` *gidlist*
: 指定したリストの実グループ ID があるプロセスだけに一致します。各グループ ID はグループ名または数値のグループ ID のどちらでも指定できます。

`-J` *projidlist*
: 指定したリストのプロジェクト ID があるプロセスだけに一致します。各プロジェクト ID はプロジェクト名または数値のプロジェクト ID のどちらでも指定できます。

`-l`
: 詳細出力形式。一致したプロセスの名前とID を出力します。`-f` オプションが指定されているかどうかによって (上記の説明を参照)、プロセス名は `pr_psargs` または `pr_fname` フィールドから取得されます。`-l` オプションは `pgrep` へのオプションとして指定した場合にのみ有効です。

`-n`
: 指定した条件すべてを満たすプロセスの中で、最新の (最後に作成された) プロセスだけに一致します。このオプションは、`-o` オプションと同時には使用できません。

`-o`
: 指定した条件すべてを満たすプロセスの中で、もっとも古い (最初に作成された) プロセスだけに一致します。このオプションは、`-n` オプションと同時には使用できません。

`-P` *ppidlist*
: 指定したリストの親プロセス ID があるプロセスだけに一致します。

`-s` *sidlist*
: 指定したリストのセッション ID があるプロセスだけに一致します。リストのセッション ID 0 がある場合、これは `pgrep` または `pkill` プロセスのセッション ID として解釈されます。

`-t` *termlist*
: 指定したリストの端末に関連するプロセスだけに一致します。各端末は、`/dev` 内にある端末へのパス名 (つまり、`/dev/` 以降のパス名) として指定します。たとえば、`term/a` や `pts/0` などです。

`-T` *taskidlist*
: 指定したリストのタスク ID があるプロセスだけに一致します。リストにタスク ID 0 がある場合、これは `pgrep` または `pkill` プロセスのタスク ID として解釈されます。

`-u` *euidlist*
: 指定したリストの実効ユーザー ID があるプロセスだけに一致します。各ユーザー ID はログイン名または数値のユーザー ID のどちらでも指定できます。

`-U` *uidlist*
: 指定したリストの実ユーザー ID があるプロセスだけに一致します。各ユーザー ID はログイン名または数値のユーザー ID のどちらでも指定できます。

`-v`
: 一致の意味を反転します。指定した条件を「満さない」すべてのプロセスに一致します。

`-x` 指定した *pattern* に引数の文字列または実行可能ファイル名が完全に一致するプロセスだけに一致します。「完全に一致する」というのは、プロセス引数の文字列または実行可能ファイルの名前のすべての文字が *pattern* に一致することを意味します。

-signal 一致した各プロセスに送信するシグナルを指定します。シグナルを指定しないと、デフォルトで `SIGTERM` が送信されます。*signal* は `signal`(3HEAD) で定義されているシンボル名 (`SIG` 接頭辞なし) または対応するシグナル番号 (10 進数) のどちらでも指定できます。*-signal* オプションは `pkill` への最初のオプションとして指定した場合にのみ有効です。

オペランド

次のオペランドを指定できます。

pattern 実行可能ファイルの名前またはプロセス引数の文字列全体と照合させる拡張正規表現 (ERE) パターンを指定します。ERE 構文の詳細については、`regex`(5) のマニュアルページを参照してください。

使用例

例 1 プロセス ID を取得する

`sendmail` のプロセス ID を取得します。

```
example% pgrep -x -u root sendmail
283
```

例 2 プロセスを強制終了する

最後に作成された `xterm` を強制終了します。

```
example% pkill -n xterm
```

終了ステータス

次の終了ステータスが返されます。

`0` 1 つまたは複数のプロセスが一致した

`1` どのプロセスも一致しなかった

`2` 無効なコマンド行オプションが指定された

`3` 致命的なエラーが発生した

ファイル

`/proc/`*nnnnn*`/psinfo` プロセス情報ファイル

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | kill(1), proc(1), ps(1), truss(1), kill(2), signal(3HEAD), proc(4), attributes(5), regex(5)

注意事項 | どちらのユーティリティも /proc/*nnnnn*/psinfo ファイルの pr_fname または pr_psargs フィールドに対して ERE で *pattern* 引数を照合することができます。これらの文字列の長さ制限は <sys/procfs.h> に定義されています。現在の制限よりも長い文字列に一致する可能性があるパターンを使用すると、予期したプロセスと一致しないことがあります。

pattern 引数に ERE のメタキャラクタが含まれており、そのメタキャラクタがシェルのメタキャラクタでもある場合、そのメタキャラクタを適切なシェル引用符で囲む必要があります。

終了したプロセスは、pgrep と pkill のいずれでも検出することはできません。

現在の pgrep または pkill プロセスは、自身を照合候補とすることはありません。

pkginfo(1)

名前 | pkginfo – ソフトウェアパッケージ情報の表示

形式 | **pkginfo** [-q | -x | -l] [-p | -i] [-r] [-a *arch*] [-v *version*] [-c *category*...] [*pkginst*...]

pkginfo [-d *device*] [-R *root_path*] [-q | -x | -l] [-a *arch*] [-v *version*] [-c *category*...] [*pkginst*...]

機能説明 | pkginfo は、システムにインストールされているソフトウェアパッケージの情報を表示します (最初の形式)。あるいは、特定のデバイスまたはディレクトリ上にあるソフトウェアパッケージの情報を表示します (2 番目の形式)。

オプションを指定しないと、pkginfo は、一次カテゴリ、パッケージのインスタンス、および完全または部分的にインストールされているすべてのパッケージの名前を一覧表示します。これらの情報はパッケージごとに 1 行に表示されます。

オプション | -d オプションと組み合わせて使用すると、-p オプションと -i オプションは無効になります。

-q オプション、-x オプション、および -l オプションを同時に指定することはできません。

-a *arch* — パッケージのアーキテクチャを arch として指定します。

-c *category* — *category* に一致するパッケージを表示します。カテゴリは pkginfo(4) ファイルの CATEGORY パラメータで定義されます。複数のカテゴリが設定されている場合、リスト内の 1 つのカテゴリとだけ一致する必要があります。照合に大文字と小文字の区別はありません。

-d *device* — ソフトウェアが存在するデバイス *device* を定義します。*device* はディレクトリの絶対パス名、または、テープ、フロッピーディスク、着脱式ディスクなどの識別子のどちらでもかまいません。spool という特殊なトークンを使用すると、デフォルトのインストールスプールディレクトリ (/var/spool/pkg) を指定することができます。

-i — 完全にインストールされているパッケージの情報だけを表示します。

-l — 詳細な出力形式を指定します。つまり、指定されたパッケージについて利用できる情報をすべて表示します。

-p — 部分的にインストールされているパッケージの情報だけを表示します。

-q — 情報を表示しません。このオプションは、パッケージがインストールされているかどうかをプログラム内で検査するときに使用します。

-r — 再配置可能なパッケージのインストールのベースディレクトリを一覧表示します。

`-R` *root_path*　*root_path* として使用するディレクトリのフルパス名を定義します。すべてのファイル (パッケージシステム情報ファイルを含む) は *root_path* から始まるディレクトリツリーに再配置されます。

`-v` *version*　パッケージのバージョンを *version* として指定します。バージョンは `pkginfo`(4) ファイルの `VERSION` パラメータで定義されます。互換性があるすべてのバージョンを要求するには、バージョン名の前にチルド (〜) を付けます。バージョンの比較では、複数の空白は 1 つの空白に置き換えられます。

`-x`　抽出されたパッケージ情報の一覧を示します。一覧には、パッケージの省略名、名前、アーキテクチャ (もしあれば)、およびバージョン (もしあれば) が含まれます。

オペランド

pkginst　インスタンスによるパッケージの指定。インスタンスはパッケージの省略形または特定のインスタンス (たとえば、`inst.1` や `inst.2`) のどちらでもかまいません。パッケージのすべてのインスタンスを要求するには、`inst.*` を使用します。シェルによっては、アスタリスク文字 (`*`) が特殊な意味を持つことがあり、`*` をエスケープしなければならない場合があります。C シェルでは、`*` は単一引用符 (') で囲むか、バックスラッシュ (\) を前につける必要があります。

終了ステータス

`0`　正常終了

`>0`　エラーが発生した

ファイル

`/var/spool/pkg`　デフォルトのインストールスプールディレクトリ

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`pkgtrans`(1), `pkgadd`(1M), `pkgask`(1M), `pkgchk`(1M), `pkgrm`(1M), `pkginfo`(4), `attributes`(5)

Application Packaging Developer's Guide

名前 | pgrep, pkill – 名前または他の属性によるプロセスの検出またはシグナル送信

形式 | **pgrep** [-flvx] [-n | -o] [-d *delim*] [-P *ppidlist*] [-g *pgrplist*] [-s *sidlist*] [-u *euidlist*] [-U *uidlist*] [-G *gidlist*] [-J *projidlist*] [-t *termlist*] [-T *taskidlist*] [*pattern*]

pkill [-*signal*] [-fnvx] [-P *ppidlist*] [-g *pgrplist*] [-s *sidlist*] [-u *euidlist*] [-U *uidlist*] [-G *gidlist*] [-J *projidlist*] [-t *termlist*] [-T *taskidlist*] [*pattern*]

機能説明 | pgrep ユーティリティはシステム上のアクティブなプロセスを調べて、コマンド行に指定した条件を満たす属性を持つプロセスのプロセス ID を報告します。 各プロセス ID は 10 進数値として出力され、プロセス ID は区切り文字 (デフォルトは復帰改行) で区切られます。コマンド行で属性オプションに値を指定するときは、複数の値をコンマで区切って指定できます。

```
pgrep -G other,daemon
```

この例は、実グループ ID が other または daemon であるプロセスを検出します。条件となるオプションを複数指定すると、pgrep は、条件オプションの論理積に一致する属性を持つプロセスを検出します。

```
pgrep -G other,daemon -U root,daemon
```

この例は、次の属性を持つプロセスを検出します。

(実グループ ID が other または daemon であり) かつ
(実ユーザー ID が root または daemon である)

pkill の機能は pgrep の機能と同等ですが、検出したプロセスのプロセス ID を出力するのではなく、kill(1) と同様に、各プロセスにシグナルを送信します。シグナルの名前または番号は pkill への最初のコマンド行オプションとして指定できます。

オプション | 次のオプションを指定できます。

-d *delim* — 一致したプロセス ID 間に置かれる区切り文字を指定します。-d オプションを指定しない場合、デフォルトの区切り文字は復帰改行です。-d オプションは pgrep へのオプションとして指定した場合にのみ有効です。

-f — 正規表現 *pattern* の照合は、プロセスの引数を含む文字列全体 (/proc/*nnnnn*/psinfo ファイルの pr_psargs フィールドから取得) に対して行われます。-f オプションを指定しないと、正規表現 *pattern* の照合は、実行可能ファイルの名前 (/proc/*nnnnn*/psinfo ファイルの pr_fname フィールドから取得) に対してのみ行われます。

`-g` *pgrplist*
: 指定したリストのプロセスグループ ID があるプロセスだけに一致します。リストにグループ 0 がある場合、これは `pgrep` または `pkill` プロセスのプロセスグループ ID として解釈されます。

`-G` *gidlist*
: 指定したリストの実グループ ID があるプロセスだけに一致します。各グループ ID はグループ名または数値のグループ ID のどちらでも指定できます。

`-J` *projidlist*
: 指定したリストのプロジェクト ID があるプロセスだけに一致します。各プロジェクト ID はプロジェクト名または数値のプロジェクト ID のどちらでも指定できます。

`-l`
: 詳細出力形式。一致したプロセスの名前とID を出力します。`-f` オプションが指定されているかどうかによって (上記の説明を参照)、プロセス名は `pr_psargs` または `pr_fname` フィールドから取得されます。`-l` オプションは `pgrep` へのオプションとして指定した場合にのみ有効です。

`-n`
: 指定した条件すべてを満たすプロセスの中で、最新の (最後に作成された) プロセスだけに一致します。このオプションは、`-o` オプションと同時には使用できません。

`-o`
: 指定した条件すべてを満たすプロセスの中で、もっとも古い (最初に作成された) プロセスだけに一致します。このオプションは、`-n` オプションと同時には使用できません。

`-P` *ppidlist*
: 指定したリストの親プロセス ID があるプロセスだけに一致します。

`-s` *sidlist*
: 指定したリストのセッション ID があるプロセスだけに一致します。リストのセッション ID 0 がある場合、これは `pgrep` または `pkill` プロセスのセッション ID として解釈されます。

`-t` *termlist*
: 指定したリストの端末に関連するプロセスだけに一致します。各端末は、`/dev` 内にある端末へのパス名 (つまり、`/dev/` 以降のパス名) として指定します。たとえば、`term/a` や `pts/0` などです。

`-T` *taskidlist*
: 指定したリストのタスク ID があるプロセスだけに一致します。リストにタスク ID 0 がある場合、これは `pgrep` または `pkill` プロセスのタスク ID として解釈されます。

`-u` *euidlist*
: 指定したリストの実効ユーザー ID があるプロセスだけに一致します。各ユーザー ID はログイン名または数値のユーザー ID のどちらでも指定できます。

`-U` *uidlist*
: 指定したリストの実ユーザー ID があるプロセスだけに一致します。各ユーザー ID はログイン名または数値のユーザー ID のどちらでも指定できます。

`-v`
: 一致の意味を反転します。指定した条件を「満さない」すべてのプロセスに一致します。

`-x` 指定した *pattern* に引数の文字列または実行可能ファイル名が完全に一致するプロセスだけに一致します。「完全に一致する」というのは、プロセス引数の文字列または実行可能ファイルの名前のすべての文字が *pattern* に一致することを意味します。

-signal 一致した各プロセスに送信するシグナルを指定します。シグナルを指定しないと、デフォルトで `SIGTERM` が送信されます。*signal* は `signal`(3HEAD) で定義されているシンボル名 (`SIG` 接頭辞なし) または対応するシグナル番号 (10 進数) のどちらでも指定できます。*-signal* オプションは `pkill` への最初のオプションとして指定した場合にのみ有効です。

オペランド

次のオペランドを指定できます。

pattern 実行可能ファイルの名前またはプロセス引数の文字列全体と照合させる拡張正規表現 (ERE) パターンを指定します。ERE 構文の詳細については、`regex`(5) のマニュアルページを参照してください。

使用例

例 1 プロセス ID を取得する

`sendmail` のプロセス ID を取得します。

```
example% pgrep -x -u root sendmail
283
```

例 2 プロセスを強制終了する

最後に作成された `xterm` を強制終了します。

```
example% pkill -n xterm
```

終了ステータス

次の終了ステータスが返されます。

`0` 1 つまたは複数のプロセスが一致した

`1` どのプロセスも一致しなかった

`2` 無効なコマンド行オプションが指定された

`3` 致命的なエラーが発生した

ファイル

`/proc/`*nnnnn*`/psinfo` プロセス情報ファイル

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | kill(1), proc(1), ps(1), truss(1), kill(2), signal(3HEAD), proc(4), attributes(5), regex(5)

注意事項 | どちらのユーティリティも /proc/*nnnnn*/psinfo ファイルの pr_fname または pr_psargs フィールドに対して ERE で *pattern* 引数を照合することができます。これらの文字列の長さ制限は <sys/procfs.h> に定義されています。現在の制限よりも長い文字列に一致する可能性があるパターンを使用すると、予期したプロセスと一致しないことがあります。

pattern 引数に ERE のメタキャラクタが含まれており、そのメタキャラクタがシェルのメタキャラクタでもある場合、そのメタキャラクタを適切なシェル引用符で囲む必要があります。

終了したプロセスは、pgrep と pkill のいずれでも検出することはできません。

現在の pgrep または pkill プロセスは、自身を照合候補とすることはありません。

名前 | cd, chdir, pushd, popd, dirs – 現在の作業用ディレクトリの変更

形式 | **/usr/bin/cd** [*directory*]

sh | **cd** [*argument*]

chdir [*argument*]

csh | **cd** [*dir*]

chdir [*dir*]

pushd [+ | *n* | *dir*]

popd [+ *n*]

dirs [-l]

ksh | **cd** [*arg*]

cd *old new*

/usr/bin/cd | /usr/bin/cd ユーティリティは、cd ユーティリティ自身だけの現在のディレクトリを変更します。これは、後述するシェル組み込みの cd とは対照的です。/usr/bin/cd はプロセスの呼び出しには影響しませんが、あるディレクトリを現在のディレクトリとして設定できるかどうかを決定するのに使用できます。

sh | Bourne シェルに組み込まれている cd は、現在のディレクトリを *argument* で指定されたディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *argument* になります。 シェル変数 CDPATH は、 *argument* を含むディレクトリの検索パスを定義します。代替ディレクトリ名は、 コロン (:) で区切ります。 デフォルトのパスは空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは空のパス名で指定します。 このパス名は、等号の直後か、 パスリスト内にある区切り文字のコロンの間に指定します。*argument* の先頭文字が / 、 . 、または .. の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *argument* を検索します。 cd は、 *argument* 中で実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は効率が悪くなります。 そのため、 cd コマンドは、シェルに組み込まれています。 (pwd(1) 、 sh(1) 、 chdir(2) を参照)

chdir は、cd を呼び出すもうひとつの方法です。

csh | *dir* 引数を省略すると、 C シェルに組み込まれている cd は、シェル変数 HOME の値を新たな作業用ディレクトリとして 使用します。 *dir* を指定した場合、それが / 、 . 、または .. で始まる完全なパス名であれば、その *dir* が新たな作業用ディレクトリとなります。 それ以外の場合は、シェル変数 CDPATH が指定するパスと相対関係を持つディレクトリの中から 該当するものを探し出します。 CDPATH の構文は PATH シェル変数と同一で、セマンティクスも似ています。 cd は *dir* に対する実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、C シェルに組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

chdir はシェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。 引数を指定しないと、 そのユーザーのホームディレクトリに変更します。 *dir* が現在のディレクトリからは見つからない相対パス名の場合、変数 cdpath 内のディレクトリリストを検索します。 *dir* が / で始まる値を持つシェル変数の名前である場合、 その変数の値が示すディレクトリに変更します。

pushd はディレクトリスタックにディレクトリをプッシュ (押し込む) します。 引数を指定しないと、スタックにある先頭の 2 つの構成要素を交換します。

+*n* *n* 番目のエントリがスタックの先頭になるよう回転し、 そのディレクトリに移ります。

dir 現在の作業用ディレクトリをスタックにプッシュし、 そのディレクトリに移ります。

popd はディレクトリスタックからポップして (取り出して)、新たに先頭となったディレクトリへ cd します。ディレクトリスタックの構成要素の先頭番号は、 0 となります。

+*n* スタック内の *n* 番目のエントリを破棄します。

dirs はディレクトリスタックを出力します。 現在のディレクトリが最も左に現れるように 時間順に出力されます。 -l 引数を指定すると、 ~ を使った省略形ではなく、完全な形式で出力されます。

ksh Korn シェルに組み込まれた cd コマンドは、上記 2 つの形式の いずれかで入力します。第 1 の形式は、現在のディレクトリを *arg* に変更します。 *arg* が – の場合、ディレクトリを直前のディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *arg* になります。PWD 変数は、現在のディレクトリに設定されます。シェル変数 CDPATH は、*arg* を含むディレクトリの検索パスを定義します。 ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは NULL のパス名で指定します。 このパス名は、 等号の直後か、パスリスト内にある区切り文字のコロンの間に指定します。 *arg* の先頭文字が / 、 . 、 または . . の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *arg* を検索します。 cd の第 2 の形式は、 PWD 中の現在のディレクトリ名における *old* という文字列を *new* という文字列に置換し、 この新規のディレクトリへ変更しようとします。 cd コマンドは rksh では実行できません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、 ksh に組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

オペランド 以下のオペランドを指定できます。

directory 新たな作業用ディレクトリとなるディレクトリの絶対または相対パス名。 cd が相対パス名をどのように解釈するかは、環境変数 CDPATH の設定により異なります。

出力 CDPATH に設定されている空でないディレクトリ名が用いられる場合、 新たな作業用ディレクトリの絶対パス名が 以下のような形式で標準出力に出力されます。

"%s\n", <*new directory*>

それ以外の場合には、何も出力されません。

環境

cd の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

CDPATH　コロンで区切られた、ディレクトリを示すパス名のリスト。 *directory* オペランドの先頭文字がスラッシュ (/) でなく、 先頭部分が . でも .. でもない場合には、 cd はこのリスト内のパス名を順番に検索し、環境変数 CDPATH に指定されている名前のディレクトリから *directory* を探します。その結果、最初に見つかったディレクトリ名が 新たな作業用ディレクトリとなります。 ディレクトリのパス名として空の文字列を指定すると、 それは現在のディレクトリと見なされます。 CDPATH は、設定されていないときには空の文字列として扱われます。

HOME　*directory* オペランドが省略されたときに用いるホームディレクトリの名前

PWD　現在の作業用ディレクトリのパス名。 この変数は、そのディレクトリに移った後に cd により設定されます。

終了ステータス

以下の終了ステータスが返されます。

0　ディレクトリが正常に変更された。

>0　エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), pwd(1), sh(1), chdir(2), attributes(5), environ(5)

名前 | pr – ファイルの出力

形式 | **/usr/bin/pr** [+ *page*] [-column] [-adFmrt] [-e [*char*] [*gap*]] [-h *header*] [-i [*char*] [*gap*]] [-l *lines*] [-n [*char*] [*width*]] [-o *offset*] [-s [*char*]] [-w *width*] [-fp] [*file...*]

/usr/xpg4/bin/pr [+ *page*] [-column | -c *column*] [-adFmrt] [-e [*char*] [*gap*]] [-h *header*] [-i [*char*] [*gap*]] [-l *lines*] [-n [*char*] [*width*]] [-o *offset*] [-s [*char*]] [-w *width*] [-fp] [*file...*]

機能説明 | pr ユーティリティは、印刷とページ設定用のフィルタです。 複数のファイルを指定すると、その各々を読み込み、 書式化して、標準出力に書き出します。 デフォルトでは、入力データを 66 行からなるページに分割します。各ページには以下のものが含まれます。

- 5 行からなるヘッダー。ページ番号、日付、時刻、ファイルのパス名を含む
- 5 行の空行からなるトレーラ

標準出力が端末に割り当てられているとき、診断メッセージは pr ユーティリティの処理が完了してから出力されます。

複数カラム出力を示すオプションを指定すると、 テキストカラムは等しい幅で出力されます。 この幅に収まらない入力行は切り捨てられます。 デフォルトでは、出力テキストカラムは 1 つ以上の空白文字で区切られます。

オプション | 以下のオプションを指定できます。 なおオプションの説明において、 *column* 、 *lines* 、 *offset* 、 page 、 *width* は正の 10 進整数を表し、 *gap* は負でない 10 進整数を表します。オプション引数の中には、省略可能なものがあります。 また直前のオプション文字と分離して独立した引数として 指定できないものがあります。具体例としては、 -s オプションと引数とを離して記述することはできません。 また -e、 -i、 -n の各オプションは、 2 つの引数を記述するのであれば、 それらをオプション文字と離すことはできません。

以下のオプションは、 /usr/bin/pr と /usr/xpg4/bin/pr の両方で使用できます。

+page
: 書式化された入力の、 page で示すページ番号から出力を開始します。

-column
: *column* で示す数 (デフォルトは 1) のカラムからなる、複数カラム出力を生成します。 これらのカラムには、入力ファイルからテキストを受け取った順序で データが書き出されます。 このオプションは -m オプションと同時に使うことはできません。 複数カラム出力を指定すると、 -e と -i の両オプションも指定されたと見なされます。 同一行数となるようなテキストカラムの生成を指定したか否か に関わらず、テキストカラムがページの長さを超えることはありません (-l オプションの説明を参照) 。このオプションを -t とともに指定した場合、最少の行数で出力データを書き出します。

`-a`	出力ページ内において、連鎖的にカラムが印刷されるよう、 *-column* オプション指定による出力を加工します。 たとえば *column* の値が 2 のとき、最初の入力行が 1 行目のカラム 1 に、 2 番目の入力行が 1 行目のカラム 2 に出力された後、続く 3 番目の行は 2 行目のカラム 1 に置かれます。
`-d`	入力中に復帰改行文字が現れるたびに、その後にもう一つ 復帰改行文字を挿入して、ダブルスペース (2 倍の改行幅) で出力を生成します。
`-e [` *char* `][` *gap* `]`	入力中の各タブ文字を、式 *n *gap+1* を満たす次のカラム位置に展開します。 *n* は正の整数です。 *gap* に `0` を指定するか *gap* を省略すると、デフォルトとして `8` が用いられます。入力中のタブ文字は、 すべて適切な数の空白文字に展開されます。 数字でない文字 *char* を指定すると、それは入力タブ文字と見なされます。
`-f`	改ページの際に、デフォルトである一連の復帰改行文字の代わりに、 用紙送り文字を使用します。 標準出力が端末に割り当てられているとき、 最初のページの出力を始める前に一時停止します。
`-h` *header*	ページヘッダーの見出しとして、 `file` オペランドの値の代わりに *header* で示す文字列を使用します。
`-l` *lines*	ページの長さを、デフォルトの 66 の代わりに *lines* で示す行数に設定します。 *lines* の値がヘッダーとトレーラの長さの合計行数以下の場合、 `pr` は `-t` オプションが有効な場合と同じように、 ヘッダーとトレーラの出力を抑止します。
`-m`	ファイルをマージします。 `pr` は `file` で指定された複数のファイルから 1 行ずつを読み込み、同じ幅のテキストカラムにそれらの行の内容を印刷して 標準出力を生成します。 最大 9 個のファイルのマージがサポートされています。
`-n [` *char* `][` *width* `]`	行番号の桁数を *width* で示す値とします。デフォルトは `5` です。出力上において行番号は、 デフォルト出力の場合には各テキストカラムの先頭の *width* 個のカラムを、 `-m` 出力の場合には各行の先頭の *width* 個のカラムを占めます。数字でない文字 *char* を指定すると、行番号と後続の出力を隔てるために行番号に付加されます。 *char* のデフォルトはタブ文字です。
`-o` *offset*	各出力行の先頭に *offset* 個の空白文字を付加します。 `-o` オプションを省略すると、デフォルトのオフセット値は `0` となります。ここで指定するオフセット値は、出力行の幅に追加されます (後述する `-w` オプションの説明を参照) 。

`-p`
: 標準出力が端末に割り当てられているとき、 ページの出力を始める前に一時停止します。 `pr` は警告 (ALERT) 文字を標準エラー出力に送り、 `/dev/tty` 上で改行文字が読み込まれるのを待ちます。

`-r`
: ファイルをオープンできなくても、それに関する診断レポートを出力しません。

`-s [` *char* `]`
: テキストカラムを区切る文字として、一連の空白文字の代わりに、 *char* で示す 1 文字を使用します。 *char* のデフォルトはタブ文字です。

`-t`
: 各ページに通常出力される 5 行のヘッダーと 5 行のトレーラを出力しません。 各ファイルの最終行を出力した後、そのページの最終行まで行送りをせず、 出力を停止します。

`-w` *width*
: 行の幅を *width* で示す数のカラム位置に設定します。 この指定は複数カラム出力に対してのみ有効です。 このオプションを省略したときのデフォルトの行幅は、 `-s` オプションも省略されていれば `72` となり、 `-s` オプションが指定されていれば `512` となります。

  シングルカラム出力の場合には、 入力行の内容が出力幅の制限によって切り捨てられることはありません。

/usr/bin/pr

以下のオプションは、 `/usr/bin/pr` でのみ使用できます。

`-F`
: 入力ファイルの行を折り返します。 `-a` または `-m` オプション指定による複数カラムモード使用時は、 現在のカラム幅に合うように行を折り返します。 複数カラムモードでなければ、現在の行幅 (80 カラム) に合うように行を折り返します。

`-i [` *char* `][` *gap* `]`
: 出力上で 1 つ以上の連続した空白文字が *gap+1*、 *2*gap+1*、 *3*gap+1*. . . のカラム位置に到達した場合、それらの空白文字を タブ文字で置き換えます。 *gap* を省略するかあるいは `0` を指定した場合、デフォルトとして 8 カラムごとにタブ位置が設定されていると見なされます。 数字でない文字 *char* を指定すると、それが出力タブ文字として使用されます。

/usr/xpg4/bin/pr

以下のオプションは、 `/usr/xpg4/bin/pr` でのみ使用できます。

`-F`
: 改ページの際に、デフォルトである一連の復帰改行文字の代わりに、 用紙送り文字を使用します。

`-i [` *char* `][` *gap* `]`
: 出力上で 2 つ以上の連続した空白文字が *gap+1*、 *2*gap+1*、 *3*gap+1*. . . のカラム位置に到達した場合、それらの空白文字を タブ文字で置き換えます。 *gap* を省略するかあるいは `0` を指定した場合、デフォルトとして 8 カラムごとにタブ位置が設定されていると見なされます。 数字でない文字 *char* を指定すると、それが出力タブ文字として使用されます。

オペランド

以下のオペランドを指定できます。

file 内容を印刷するファイルのパス名。このオペランドを省略するか、または - を指定すると、標準入力が用いられます。

使用例

例 **1** 現在のディレクトリ中の全ファイルの 番号付き一覧を出力する

```
ls -a | pr -n -h "Files in $(pwd)."
```

例 **2** `file1` と `file2` を、 “file list” というヘッダーを付けた、 3 カラムからなる 2 倍の改行幅のリストに出力する

```
pr -3d -h "file list" file1 file2
```

例 **3** タブ位置をカラム 10 、 19 、 28. . . に設定して、 `file1` を `file2` に書き出す

```
pr -e9 -t <file1 >file2
```

環境

`pr` の実行に影響を与える環境変数 `LC_CTYPE` 、 `LC_TIME` 、 `LC_MESSAGES` 、 `NLSPATH` 、 `TZ` についての詳細は、 `environ`(5) を参照してください。

終了ステータス

次の終了ステータスが返されます。

`0` ファイルはすべて正常に書き出された。

`>0` エラーが発生した。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/pr

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/pr

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

`expand`(1), `lp`(1), `attributes`(5), `environ`(5), `XPG4`(5)

名前 | print – 画面またはウィンドウに文字を出力するシェル組み込み関数

ksh | **print** [-Rnprsu [*n*]] [*arg*...]

ksh | シェルの出力機構です。 オプションを省略した場合、あるいは – または – – オプションを指定した場合には、 echo(1) で述べるように標準出力上に引数を表示します。

オプション | 以下のオプションを指定できます。

-n	復帰改行 (new-line) の出力を抑止します。
-R	
-r	(raw モード) echo のエスケープ規則を無視します。 -R オプションは、 -n を除く後続の引数およびオプションすべてを表示します。
-p	標準出力の代わりに |& で生成されたプロセスのパイプ上に引数を出力します。
-s	標準出力の代わりに 履歴ファイル上に引数を書き込みます。
-u [*n*]	出力を格納するファイル記述子番号を、 1 桁の数値 *n* で指定します。 デフォルトは 1 です。

終了ステータス | 以下の終了ステータスが返されます。

0	正常終了
>0	書き込み用出力ファイルが開かない

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | echo(1), ksh(1), attributes(5)

priocntl(1)

名前 | priocntl – 指定したプロセスのスケジューリングパラメータの表示または設定

形式 | **priocntl** -l

priocntl -d [-i *idtype*] [*idlist*]

priocntl -s [-c *class*] [*class-specific options*] [-i *idtype*] [*idlist*]

priocntl -e [-c *class*] [*class-specific options*] *command* [*argument(s)*]

機能説明 | priocntl コマンドは、指定した 1 つまたは複数のプロセスの スケジューリングパラメータを表示または設定します。 また、システムのプロセススケジューラの 現在の設定情報を表示したり、 指定したスケジューリングパラメータを使って コマンドを実行したりする目的にも使用できます。

プロセスはいくつかのクラスに分けられます。 各クラスにはそれぞれ異なった スケジューリング方針が適用されます。 現在サポートされているクラスは、 リアルタイム、タイムシェアリング、対話型、フェアシェア、固定優先順位の 5 種類です。 各クラスの特徴およびクラス固有のオプションについては、 後述の「リアルタイムクラス」、「タイムシェアリングクラス」、「対話型クラス」、「フェアシェアクラス」、「固定優先順位クラス」の項で説明します。 ユーザーは、正しいアクセス権を持っていれば、 priocntl コマンドを使って、稼動中のプロセスのクラスや 対応するスケジューリングパラメータを 変更することができます。

デフォルトの設定では、 実行可能状態にあるリアルタイムプロセスが 他のプロセスに先だって実行されます。 したがってリアルタイムプロセスの使い方を誤ると、 システムのパフォーマンスが 著しく損なわれてしまう恐れがあります。

priocntl コマンドに *idlist* 引数を指定する場合は、 コマンド行の最後に記述してください。 また *idlist* 中の要素は、空白で区切る必要があります。 *idlist* 引数を省略するときは、 *idtype* 引数として pid、 ppid、 pgid、 sid、 class、 uid、 gid を使って、 priocntl コマンド自身のプロセス ID、親プロセス ID、 プロセスグループ ID、セッション ID、クラス ID、 ユーザー ID、グループ ID をそれぞれ指定します。

priocntl -d [-i *idtype***] [***idlist***]**

上記の priocntl コマンドは、 *idtype* および *idlist* 引数で指定したプロセス群の クラスとクラス固有のスケジューリングパラメータを表示します。

priocntl -s [-c *class***] [***class-specific options***] **
[-i *idtype***] [***idlist***]**

上記の priocntl コマンドは、指定したプロセスのクラスおよび クラス固有のパラメータを、 コマンド行で指定した値に設定します。 -c *class* オプションが、設定するクラスを表します。 現在サポートされている *class* 引数値は、リアルタイムクラス用の RT、 タイムシェアリングクラス用の TS、対話型クラス用の IA、フェアシェア用の FSS、固定優先順位用の FX のいずれかです。

各クラス固有のパラメータは、 対応するクラス固有のオプションを使って指定します。 詳しくは各クラスの項で後述します。 -c *class* オプションを省略する場合には、 *idtype* と *idlist* を使って、同じクラスに属する 一群のプロセスを指定してください。 そうしないとエラーが発生します。 クラス固有のオプションをすべて省略すると、

当該プロセスのクラス固有パラメータの値は、 -c *class* で指定したクラスのデフォルト値に設定されます。 なお、 -c *class* オプションも省略した場合には、 そのプロセスの現在のクラスのデフォルト値に設定されます。

priocntl を使ってプロセスのスケジューリングパラメータを変更できるのは、 priocntl を実行したユーザーの実ユーザー ID または実効ユーザー ID (それぞれのグループ ID) が当該プロセスの実ユーザー ID または実効ユーザー ID (それぞれのグループ ID) と 一致している場合、もしくは ユーザーの実効ユーザー ID がスーパーユーザーを示している場合だけです。 アクセス権に関するこの要件は、 全クラスに適用される最低限の必要条件です。 クラスによっては、プロセスを設定する場合やクラス固有のスケジューリングパラメータを設定する場合に、 他のアクセス権に関する要件が追加されることもあります。

idtype と *idlist* の両引数を使っていくつかのプロセスを指定すると、 priocntl はそれらのプロセスに対して処理を実行します。 ただし、処理する順序は実装状態により異なります。 エラーを検出した場合、そのエラーの種類により、 以降のプロセスの処理を続行する場合と コマンドを終了する場合とがあります。

検出したエラーがアクセス権に関連するものであれば、 priocntl はエラーメッセージを出力しますが、 指定されたプロセスのうちユーザーが適正な アクセス権を持っているものについては、 パラメータをリセットして処理を続行します。 エラーがアクセス権に関連したものでなければ、 エラーメッセージを表示してただちに実行を終了します。

特殊なシステムプロセス (たとえばスワッパプロセス) の実行をスケジューリングする目的のため、 sys という特殊なスケジューリングクラスが存在しています。 どのプロセスに対しても、クラスを sys に変更することはできません。 また *idtype* や *idlist* 引数で sys クラスに属するプロセスを指定した場合、 priocntl はそれらを無視します。 たとえば *idtype* として uid を指定し *idlist* に 0 (ゼロ) を指定すると、 UID の値が 0 の全プロセスのうち sys クラスに属するものを除いたプロセス、および (-s オプションを使ってパラメータを変更する場合には) init プロセスが処理の対象となります。

init プロセス (プロセス ID は 1) は特殊なケースです。 priocntl コマンドを使って init プロセスのクラスやその他のスケジューリングパラメータを変更するためには、 *idtype* として pid を指定し、 *idlist* として 1 だけを指定してください。 init プロセスは、システム上に存在する どのクラスに割り当てても構いませんが、ほとんどの場合タイムシェアリングクラスを選択するのが適切です。 他のクラスに割り当てると、 システムに悪影響を及ぼすことがあります。詳細は『*Solaris* のシステム管理 *(*基本編*)*』を参照してください。

priocntl -e [-c *class***] [***class-specific options***]** *command* ****
[*argument . . .***]**

この priocntl コマンドは、 command で示すコマンドを、指定したクラスと スケジューリングパラメータを使って実行します。 *arguments* は、 command に対する引数です。 -c *class* オプションを省略すると、指定したコマンドは ユーザーの現クラスで実行されます。

オプション　次のオプションを指定できます。

`-c` *class*
: 設定するクラスを指定します。 *class* 引数には `RT` (リアルタイムクラスの場合)、 `TS` (タイムシェアリングクラスの場合)、`IA` (対話型クラスの場合)、`FSS` (フェアシェアの場合)、`FX` (固定優先順位の場合) のいずれかを指定できます。 指定したクラスが構成されていない場合には、 自動的に構成されます。

`-d`
: 指定したプロセスに対応したスケジューリングパラメータを表示します。

`-e`
: 指定したコマンドを、指定したプロセスに 対応したスケジューリングパラメータを使って実行します。

`-i` *idtype*
: このオプションは、 *idlist* 引数 (もし存在していれば) とともに、`priocntl` コマンドの処理対象となるプロセス群を指定します。 *idlist* がどのように解釈されるかは、 *idtype* の値により異なります。 *idtype* として指定可能な値、およびそれぞれの値に対応した *idlist* の解釈は以下のとおりです。

  `-i pid`
  : *idlist* はプロセス ID です。 `priocntl` コマンドの処理対象は、指定された全プロセスです。

  `-i ppid`
  : *idlist* は親プロセス ID です。 `priocntl` コマンドの処理対象は、 指定された親プロセス ID を持つすべてのプロセスです。

  `-i pgid`
  : *idlist* はプロセスグループ ID です。 `priocntl` コマンドの処理対象は、 指定されたグループに属するすべてのプロセスです。

  `-i sid`
  : *idlist* はセッション ID です。 `priocntl` コマンドの処理対象は、 指定されたセッション中のすべてのプロセスです。

  `-i taskid`
  : *idlist* はタスク ID です。コマンドの処理対象は、指定されたタスクのすべてのプロセスです。

  `-i class`
  : *idlist* は単一のクラス名です (リアルタイム クラスなら `RT` 、タイムシェアリングクラスなら `TS` 、対話型クラスなら `IA`、フェアシェアクラスなら `FSS`、固定優先順位クラスなら `FX`)。コマンドの処理対象は、 指定されたクラスに属するすべてのプロセスです。

  `-i uid`
  : *idlist* はユーザー ID です。 `priocntl` コマンドの処理対象は、指定された ID と 同一の実効ユーザー ID を持つすべてのプロセスです。

  `-i gid`
  : *idlist* はグループ ID です。 `priocntl` コマンドの処理対象は、指定された ID と 同一の実効グループ ID を持つすべてのプロセスです。

`-i projid` *idlist* はプロジェクト ID です。コマンドの処理対象は、指定された ID と同一の実効プロジェクト ID を持つすべてのプロセスです。

`-i all` `priocntl` コマンドの処理対象は、存在しているすべてのプロセスです。 この場合、 *idlist* 引数は指定できません。指定しても無視されます。 また以下に述べるアクセス権上の制限事項は、 この `all` 指定の場合でも有効です。

`-d` または `-s` オプションを指定して `-i` *idtype* オプションを省略すると、デフォルトの *idtype* である `pid` を指定したものとみなされます。

`-l` 現在システム上に定義されているクラスの一覧を、 各クラス固有の情報とともに表示します。 クラス固有情報の出力形式は「使用法」の項で説明します。

`-s` 指定したプロセスに対応したスケジューリングパラメータを設定します。

リアルタイムクラスのパラメータを設定する場合には、 以下のクラス固有のオプションを指定できます。

`-p` *rtpri* 指定したプロセスのリアルタイム優先順位を、 *rtpri* に設定します。

`-t` *tqntm* [`-r` *res*] 指定したプロセスのタイムクアンタムを、 *tqntm* に設定します。 タイムクアンタムの単位を指定することも可能です (詳しくは後述)。

`-q` *tqsig* 指定したプロセスのリアルタイムのタイムクアンタムシグナルを *tqsig* に設定します。

タイムシェアリングクラスのパラメータを設定する場合には、 以下のクラス固有のオプションを指定できます。

`-m` *tsuprilim* 指定したプロセスのユーザー優先順位の上限値を、 *tsuprilim* に設定します。

`-p` *tsupri* 指定したプロセスのユーザー優先順位を、 *tsupri* に設定します。

対話型クラスのパラメータを設定する場合には、 以下のクラス固有のオプションを指定できます。

`-m` *iauprilim* 指定したプロセスのユーザー優先順位の制限を、 *iauprilim* に設定します。

`-p` *iaupri* 指定したプロセスのユーザー優先順位を、 *iaupri* に設定します。

フェアシェアクラスのパラメータを設定する場合には、 以下のクラス固有のオプションを指定できます。

-m *fssuprilim* 指定したプロセスのユーザー優先順位の上限値を、 *tsuprilim* に設定します。

-p *fssupri* 指定したプロセスのユーザー優先順位を、 *fssupri* に設定します。

固定優先順位クラスのパラメータを設定する場合には、 以下のクラス固有のオプションを指定できます。

-m *fxuprilim* 指定したプロセスのユーザー優先順位の上限値を、 *fxuprilim* に設定します。

-p *fxupri* 指定したプロセスのユーザー優先順位を、 *fxupri* に設定します。

-t *tqntm* [-r *res*] 指定したプロセスのタイムクアンタムを、 *tqntm* に設定します。タイムクアンタムの単位を指定することも可能です (詳しくは後述)。

リアルタイムクラス

リアルタイムクラスは、 高速でかつ決定力のある応答を必要とし、 スケジューリング方針に関して ユーザーやアプリケーションに対する 絶対的な制御権を必要とするようなプロセスに対して、 固定された高い優先順位を持つ スケジューリング方針を与えるものです。 システム中にリアルタイムクラスが定義されている場合には、 システム上の最上位のスケジューリング優先順位群を 制御できる唯一のクラスである必要があります。 これにより、実行可能状態にあるリアルタイムプロセスは、 他のクラスに属するあらゆるプロセスよりも先に CPU のサービスを受けられることが保証できます。

リアルタイムクラスには、 一連のリアルタイム優先順位値 (*rtpri*) が与えられていて、そのクラスに属するプロセスに割り当てることができます。 リアルタイム優先順位値の範囲は 0 から *x* までで、この *x* の値はリアルタイムクラスのスケジューラを 構成済みのシステムごとに設定可能であり、 以下のコマンドを使って表示することができます。

```
priocntl -l
```

リアルタイムクラスのスケジューリング方針とは、 固定優先順位を割り当てることです。 つまり、ユーザーやアプリケーション側からのプロセスの *rtpri* 値を変更する明示的な要求を実行しない限り、 リアルタイムプロセスのスケジューリング優先順位は変わりません。

リアルタイムクラスのプロセスに関して、 *rtpri* の値はプロセスのスケジューリング優先順位と実用上同じです。 リアルタイムクラスのあるプロセスの、 同じクラス内の他のプロセスに対する 相対的なスケジューリング優先順位値は、 *rtpri* 値により決定されます。 *rtpri* 値は数値で示され、大きい数が高い優先順位を表します。 リアルタイムクラスはシステム中で最も高い スケジューリング優先順位群を制御しているので、 実行可能状態にあるリアルタイムプロセスのうち最高の *rtpri* 値を持つプロセスが システム中の他のあらゆるプロセスに先だって選ばれる、 ということが保証されています。

`priocntl` は、優先順位に関する制御権だけでなく、 リアルタイムクラス中のプロセスに割り当てられるタイムクアンタムを制御する権利も提供しています。 このタイムクアンタム値は、プロセスがリソースやイベントの 待ち状態 (休眠状態) に陥らないと

いう前提で、 プロセスの実行時間の最大値を指定するものです。 なお、あるプロセスの稼動中に、 より高い優先順位を持つ他のプロセスが 実行可能状態になった場合、 現在稼動中のプロセスは割り当てられたタイムクアンタム値に達していなくても取り上げられてしまうことがあります。

priocntl -d [-i *idtype***] [***idlist***]**

このコマンドは、 *idtype* と *idlist* で指定した一群のリアルタイムプロセスの 個々のリアルタイム優先順位、時間 (単位はミリ秒)、およびタイムクアンタムシグナル値を表示します。

リアルタイムクラス用の `priocntl -s` または `priocntl -e` コマンドには、`-p`、`-t` [`-r`]、および `-q` を任意の組み合わせで指定できます。 どれかを省略した場合、 当該プロセスが現在リアルタイムであれば、 省略されたオプションに対応したパラメータには影響ありません。 あるプロセスを他のクラスからリアルタイムクラスに変更するような `priocntl` コマンド中でいずれかのオプションを省略すると、 そのオプションに対応したパラメータはデフォルト値に設定されます。 *rtpri* のデフォルト値は `0` です。タイムクアンタムのデフォルトは、 *rtpri* の値およびシステムの構成により異なります (`rt_dptbl`(4) 参照)。

`-t` *tqntm* オプションを使用する際、タイムクアンタムの単位を `-r` *res* オプションを使って指定することもできます。 単位のデフォルトはミリ秒です。 *res* には、 `1` から `1,000,000,000` までの正の整数を指定してください。 この場合、使用される単位は *res* 分の 1 秒となります。たとえば、 `-t 10 -r 100` と指定すると、単位は 100 分の 1 秒となり、 タイムクアンタム長として指定されている 10 は、 10/100 つまり 10 分の 1 秒を意味することになります。 ナノ秒などの細かい単位も指定可能ですが、 実際にはシステムクロックの精度に合うように タイムクアンタム長は丸められてしまいます。 タイムクアンタムとして 0 を指定した場合、 およびシステムに固有の最大クアンタムを超える 大きな値を指定した場合には、エラーとなります。

リアルタイム時間シグナルを使用すると、実行中のリアルタイムプロセスに自身の時間消費量を知らせることができます。リアルタイム時間シグナルはリアルタイムプロセスを監視し、その実行時間が上限に達した時点で、設定されているシグナルをリアルタイムプロセスに送信します。時間シグナル *tqsig* のデフォルト値 (`0`) は、シグナルを送信しないことを示します。*tqsig* に正の値を指定すると、その値ごとに設定されているシグナルが送信されます。`kill`(1) などのシグナルを扱う コマンドの場合と同様に、`-q` *tqsig* オプションを指定して `XCPU` や `KILL` のようなシンボルを使用したシグナル名も扱うことができます。

プロセスのクラスを他のものからリアルタイムクラスに変更するには、 `priocntl` コマンドを実行するユーザーがスーパーユーザー特権を 持つ必要があります。 また、リアルタイムプロセスの *rtpri* 値やタイムクアンタムを変更するには、 `priocntl` コマンドを実行するユーザーはスーパーユーザーであるか、 あるいは現在リアルタイムクラスにいて (リアルタイムプロセスとして稼動中のシェル) その実ユーザー ID または実効ユーザー ID が 処理対象のプロセスの実ユーザー ID または実効ユーザー ID と一致している必要があります。

fork(2) や exec(2) などのシステムコールを使った場合、リアルタイム優先順位、タイムクアンタム、およびタイムクアンタムシグナルは、これらのシステムコールに受け継がれます。exec(2) システムコールで、ユーザー定義のシグナルハンドラを持つ時間シグナルが使用された場合、指定された時間の上限に達する前に、新しいイメージの中にユーザーが定義のシグナルハンドラがインストールされている必要があります。時間内にインストールできない場合は、予測できない動作が発生します。

タイムシェアリングクラス

タイムシェアリングのスケジューリング方針とは、 CPU の消費に関して異なった性質を持つプロセスに対して、 公平で有効な CPU リソースの割り付けを行うことです。 タイムシェアリング方針の目的は、 対話型プロセスには速いレスポンスを提供し、 CPU 消費型のジョブには高いスループットを提供し、 さらにスケジューリングの制御権を ユーザーやアプリケーションにもある程度与えることにあります。

タイムシェアリングクラスには、 一連のタイムシェアリングユーザー優先順位値 (*rtpri*) が与えられていて、 そのクラスに属するプロセスに割り当てることができます。 ユーザー優先順位値の範囲は -x から +*x* までで、この *x* の値はシステムごとに設定可能であり、 以下のコマンドを使って表示することができます。

priocntl -l

ユーザー優先順位の目的は、 タイムシェアリングクラスのプロセスのスケジューリングを、 ユーザーやアプリケーションがある程度制御できるようにすることです。 タイムシェアリングクラスのプロセスの *tsupri* 値を増減させれば、 そのプロセスのスケジューリング優先順位が上下します。 ただし、高い *tsupri* 値を持つタイムシェアリングプロセスが、低い *tsupri* 値を持つプロセスよりも先に実行されるとは限りません。なぜなら *tsupri* 値は、タイムシェアリングプロセスの スケジューリング優先順位を決定する際の 要素の 1 つにしかすぎないからです。 システムは最近のシステム利用率など他の要素を考慮の上、 タイムシェアリングプロセスの 内部スケジューリング優先順位を動的に調整します。

システム内でのユーザー優先順位の上限値 (priocntl -l で表示できる) に加え、プロセスごとにユーザー優先順位上限値 (*tsuprilimf1*) があります。 これは各プロセスに指定できる*tsupri* の最大値を表します。

priocntl -d [**-i** *idtype*] [*idlist*]

このコマンドは、 *idtype* と *idlist* で指定した一群のタイムシェアリングプロセスの個々のユーザー優先順位と 優先順位上限値を表示します。

タイムシェアリングプロセスは、 自身の (および同じユーザー ID を持つ他のプロセスの) *tsuprilim* 値を下げることができます。 *tsuprilim* 値を上げることができるのは、スーパーユーザー特権を持つプロセスだけです。 あるプロセスのクラスをタイムシェアリングから他のクラスに変更する場合、 *tsuprilim* の初期値を 0 より大きい値に設定するには、スーパーユーザー特権が必要となります。

どのタイムシェアリングプロセスも、 自身の (および同じユーザー ID を持つ他のプロセスの) *tsupri* 値を設定できますが、そのプロセスの *tsuprilim* 値を超えることはできません。 *tsuprilim* を超える値に *tsupri* を設定しようとした場合、または *tsupri* 未満の値に *tsuprilim* を設定しようとした場合には、結果として *tsupri* が *tsuprilim* と等しい値に設定されます。

タイムシェアリングクラス用の `priocntl -s` または `priocntl -e` コマンド中に、`-m` と `-p` の両オプションを任意の組み合せで指定できます。 どちらかを省略した場合、 当該プロセスが現在タイムシェアリングであれば、 省略されたオプションに対応したパラメータには 通常影響ありません。 ただし、 `-p` オプションを省略し、 `-m` を使って *tsuprilim* の値を *tsupri* 未満に設定しようとした場合は例外です。 このような設定を試みると、 *tsupri* は現在の *tsuprilim* の値に設定されます。 あるプロセスを他のクラスから タイムシェアリングクラスに変更する際に いずれかのオプションを省略すると、 そのオプションに対応したパラメータはデフォルト値に設定されます。 *tsuprilim* のデフォルト値は `0` です。 *tsupri* のデフォルト値は、現在設定されている *tsuprilim* の値です。

`fork`(2) や `exec`(2) などのシステムコールを使った場合、 タイムシェアリングユーザー優先順位 およびユーザー優先順位の上限値は受け継がれます。

対話型クラス

対話型のスケジューリング方針とは、 CPU の消費に関して異なった性質を持つプロセスに対して、 公平で有効な CPU リソースの割り付けを行い、 さらに対話用に速いレスポンスを提供することです。 対話型方針の目的は、 対話型プロセスには速いレスポンスを提供し、 CPU 消費型のジョブには高いスループットを提供することにあります。 対話型クラスにある処理の優先順位は、 タイムシェアリングクラスにある処理と同じように変更できます。 ただし、修正された優先順位はユーザーとの対話の速い応答性を 提供するために継続します。

対話型クラスのユーザー優先順位の上限値 *iauprilim* は *tsuprilim*と同等です。対話型クラスの、プロセスごとのユーザー優先順位 *iaupri* は *tsupri* と同等です。

iamode (対話型モード) のビットが設定されている対話型クラスのプロセスには、最優先順位を示す値 `10` が割り当てられます。この値を持つプロセスは、優先順位の計算が行われるとき、つまり、プロセスの優先順位が再調整されるたびに、そのプロセスのユーザーモード優先順位に組み込まれます。この機能は、X ウィンドウシステムで使用されています。X ウィンドウシステムでは、このビットを現在アクティブな ウィンドウ内で実行中のプロセスに割り当てて、そのプロセスにより高い優先順位を与えています。

フェアシェアクラス

フェアシェアのスケジューリング方針とは、所有するプロセスの数に関係なく、各プロジェクト間に公平に CPU リソースを割り付けることです。プロジェクトには、CPU リソースの使用権を制御するための「割り当て数」が与えられます。一定時間ごとに CPU リソースの使用量が通知されるため、他のプロジェクトより使用量の多いプロジェクトの使用権は縮小され、使用量の少ないプロジェクトの使用権は拡大されます。CPU 時間は、各プロジェクトが所有するプロセスの数とは無関係に、プロジェクトごとの使用権に応じてスケジュールされます。

FSS スケジューリングクラスは、タイムシェアリングスケジューラとの互換性を保持するために、プロセス単位のユーザー優先順位およびユーザー優先順位上限という概念を取り入れています。フェアシェアスケジューラTは、ユーザープロパティの全領域について、均等な効果を提供することを目的としています。正の *fssupri* 値を持つプロセスには、通常より短いタイムスライスが配分され、負の *fssupri* 値を持つプロセスには、通常より長いタイムスライスが配分されます。ユーザープロパティが割り当て数に影響することはありません。つまり、プロセスの *fssupri* 値を変更しても。そのプロセスの属するプロジェクト全体の CPU 使用量は変更されません。CPU 使用量に影響するものは、他のプロジェクトとの比較によって割り付けられる割り当て数の配分のみです。

フェアシェアクラス内のプロセスのプロパティは、タイムシェアリングクラスのプロセスのプロパティと同じ方法で変更できます。

固定優先順位クラス

固定優先順位クラスは、システムによってプロセスのスケジューリング優先順位が調整されることなく、ユーザーまたはアプリケーションがプロセスのスケジューリング優先順位を制御できる、優先順位が固定された事前定義のスケジューリング方針を提供します。

固定優先順位クラスは、デフォルトではタイムシェアリングクラスとスケジューリング優先順位の同じ範囲を共有しています。固定優先順位クラスには、ユーザー優先順位値 (*fxupri*) の範囲が与えられ、 そのクラスに属するプロセスに割り当てることができます。 ユーザー優先順位値の範囲は 0 から *x* までで、この *x* の値はリアルタイムクラスのスケジューラを 構成済みのシステムごとに設定可能であり、 以下のコマンドを使って表示することができます。

```
priocntl -l
```

ユーザー優先順位の目的は、 固定優先順位クラスのプロセスのスケジューリングを、ユーザーやアプリケーションが制御できるようにすることにあります。 固定優先順位クラスのプロセスに関して、 *fxupri* の値はプロセスのスケジューリング優先順位と実用上同じです。 固定優先順位のあるプロセスの、 同じクラス内の他のプロセスに対する 相対的なスケジューリング優先順位値は、 *fxupri* 値により決定されます。*fxupri* 値は数値で示され、大きい数が高い優先順位を表します。

`priocntl -l` で表示される、システム全体でのユーザー優先順位上限のほかに、プロセスごとにユーザー優先順位上限 (*fxuprilim*) があります。この値は、プロセスの *fxupri* として設定できる最大の値を示します。

固定優先順位プロセスは、 自身の (および同じユーザー ID を持つ他のプロセスの) *fxuprilim* 値を下げることができます。 *fxuprilim* 値を上げることができるのは、 スーパーユーザー特権を持つプロセスだけです。 あるプロセスのクラスを固定優先順位から他のクラスに変更する場合、 *fxuprilim* の初期値を 0 より大きい値に設定するには、スーパーユーザー特権が必要となります。

どの固定優先順位プロセスも、 自身の (および同じユーザー ID を持つ他のプロセスの) *fxupri* 値を設定できますが、そのプロセスの *fxuprilim* 値を超えることはできません。 *fxuprilim* を超える値に *fxupri* を設定しようとした場合、または *fxupri* 未満の値に *fxuprilim* を設定しようとした場合には、結果として *fxupri* が *fxuprilim* と等しい値に設定されます。

`priocntl` は、優先順位に関する制御権だけでなく、 固定優先順位クラス中のプロセスに割り当てられるタイムクアンタムを制御する権利も提供しています。 このタイムクアンタム値は、プロセスがリソースやイベントの 待ち状態 (休眠状態) に陥らないという前提で、 プロセスの実行時間の最大値を指定するものです。 なお、あるプロセスの稼動中に、 より高い優先順位を持つ他のプロセスが 実行可能状態になった場合、 現在稼動中のプロセスは割り当てられたタイムクアンタムに達していなくても取り上げられてしまうことがあります。

固定優先順位クラス用の `priocntl -s` または `priocntl -e` コマンドには、`-m`、`-p`、および `-t` を任意の組み合わせで指定できます。 どれかを省略した場合、 当該プロセスが現在固定優先順位であれば、 省略されたオプションに対応したパラメータには影響ありません。 ただし、`-p` を省略して、`-m` オプションで *fxuprilim* の値を現在の *fxupri* の値より小さな値に設定した場合は例外です。この場合、 *fxupri* は、設定された *fxuprilim* の値と同じ値になります。あるプロセスを他のクラスから固定優先順位クラスに変更するときにオプションを省略すると、 そのオプションに対応したパラメータはデフォルト値に設定されます。 *fxuprilim* のデフォルト値は `0` です。 *fxupri* の値は、設定されている *fxuprilim* の値とデフォルトで同じになります。タイムクアンタムのデフォルトは、設定されている *fxupri* の値およびシステムの構成により異なります (`fx_dptbl`(4) 参照)。

固定優先順位クラス内のプロセスのプロパティは、リアルタイムクラスのプロセスのプロパティと同じ方法で変更できます。

`fork`(2) や `exec`(2) などのシステムコールを使った場合、固定優先順位のユーザー優先順位、ユーザー優先順位上限、およびタイムクアンタムは、これらのシステムコールに受け継がれます。

使用例

リアルタイムクラスの例を示します。

例 1 非リアルタイムプロセスのクラスを設定する

idtype と *idlist* で指定された非リアルタイムのプロセスのクラスを リアルタイムに変更し、 リアルタイム優先順位値をデフォルトの `0` に設定します。 現在リアルタイムクラスに属しているプロセスの リアルタイム優先順位値は変わりません。 指定されたプロセスのタイムクアンタムは、すべて 1/10 秒に設定されます。

```
example% priocntl -s -c RT -t 1 -r 10 -i idtype idlist
```

例 2 リアルタイムでコマンドを実行する

command で指定したコマンドをリアルタイムクラスで実行します。 リアルタイム優先順位値は `5` で、 タイムクアンタムは `20` ミリ秒となります。

例 **2** リアルタイムでコマンドを実行する　(続き)

```
example% priocntl -e -c RT -p 15 -t 20 command
```

例 **3** 時間シグナルを指定して、リアルタイムでコマンドを実行する

command で指定したコマンドをリアルタイムクラスで実行します。 リアルタイム優先順位値は 11 、 タイムクアンタムは 250 ミリ秒で、リアルタイム時間シグナルは SIGXCPU となります。

```
example% priocntl -e -c RT -p 11 -t 250 -q XCPU command
```

タイムシェアリングクラスの例を示します。

例 **4** 非タイムシェアリングプロセスのクラスを設定する

idtype と *idlist* で指定した非タイムシェアリングのプロセスのクラスを タイムシェアリングに変更し、 ユーザー優先順位値と優先順位上限値を 0 に設定します。 現在タイムシェアリングクラスに属しているプロセスには 影響を与えません。

```
example% priocntl -s -c TS -i idtype idlist
```

例 **5** タイムシェアリングクラスでコマンドを実行する

command で指定したコマンドを、 *arguments* で指定した引数を使って タイムシェアリングクラスで実行します。 ユーザー優先順位の上限値は 0 で、ユーザー優先順位は -15 となります。

```
example% priocntl -e -c TS -m 0 -p -15 command [arguments]
```

例 **6** 固定優先順位クラスでコマンドを実行する

command で指定したコマンドを固定優先順位クラスで実行します。 ユーザー優先順位値上限は 20、 ユーザー優先順位は 10 で、タイムクアンタムは 250 ミリ秒となります。

```
example% priocntl -e -c FX -m 20 -p 10 -t 250 command
```

終了ステータス

以下の終了ステータスが返されます。

-d 、 -l 、 -s オプションの場合:

0	正常終了
1	エラーが発生した

-e オプションの場合:

実行されたコマンドの終了ステータスが返された	正常終了
1	コマンドは指定された優先順位で実行できなかった

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目　kill(1), nice(1), ps(1), exec(2), fork(2), priocntl(2), fx_dptbl(4), rt_dptbl(4), attributes(5), FSS(7)

『*Solaris* のシステム管理 *(*基本編*)*』

診断　priocntl は次に示すエラーメッセージを出力します。

Process(es) not found
　指定されたプロセスが 1 つも存在しません。

Specified processes from different classes
　-s オプションを使ってパラメータを設定しようとしたが、 -c *class* オプションが省略されていて、 複数のクラスのプロセスが指定されていました。

Invalid option or argument
　指定されたオプションまたは引数の中に、 認識できないものまたは不正なものがあります。

名前 | ps – プロセスの状態報告

形式 | **ps** [-aAcdefjlLPy] [-g *grplist*] [-n *namelist*] [-o *format*]... [-p *proclist*] [-s *sidlist*] [-t *term*] [-u *uidlist*] [-U *uidlist*] [-G *gidlist*]

機能説明 | ps コマンドは、活動中のプロセスに関する情報を出力します。オプショ ンを指定しないと、ps は、「コマンドを実行したユーザー」と同じ実効ユーザー ID および同じ制御端末に関連するプロセスについての情報を出力します。出力される情報は、プロセス ID、端末識別子、累積実行時間およびコマンド名のみです。オプションを指定すると、出力される情報はオプションによって制御されます。

オプションのなかには、リストを引数として使うものがあります。リストの項目は、コンマで区切られるか、あるいは引用符で囲まれてコンマか空白で区切られます。*proclist* および *grplist* には、数値を指定してください。

オプション | 以下のオプションを指定できます。

-a — 最も頻繁に要求されるすべてのプロセスに関する情報を表示します。ただし、プロセスのセッションリーダーおよび端末に関連していないプロセスは除きます。

-A — すべてのプロセスの情報を表示します。後述の -e と同じです。

-c — priocntl(1) で説明するように、スケジューラ特性を反映する形式で情報を出力します。以下で説明するように、-c オプションは、-f オプションと -l オプションの出力に影響します。

-d — セッションリーダーを除くすべてのプロセスに関する情報を表示します。

-e — 現在実行中のすべてのプロセスに関する情報を表示します。

-f — 完全リストを作成します (完全リストにおけるカラムの意味については、以下を参照)。

-g *grplist* — *grplist* にグループリーダーの ID 番号が指定されているプロセスの情報を表示します (グループリーダーとは、プロセス ID 番号がプロセスグループ ID 番号と同じプロセスのこと)。

-G *gidlist* — *gidlist* に実グループの ID 番号が指定されているプロセスの情報を出力します。*gidlist* は、複数の ID を空白またはコンマで区切って記述するリストで、単一の引数として指定する必要があります。

-j — セッション ID およびプロセスグループ ID を出力します。

-l — 長い形式のリストを作成します (下記を参照)。

-L — 選択された各プロセス中の各軽量プロセス (lwp) についての情報を出力します (後述参照)。

-n *namelist* — で示すシステムファイルをデフォルトの代わりに使用します。このオプションは、互換のために受け付けられますが無視されます。

`-o` *format*　*format* が示す書式指定に従って出力情報を表示します。詳細は「表示書式」の項を参照してください。`-o` オプションは複数回指定することが可能です。その場合、複数の書式指定引数は、空白文字で区切られて継続していると見なされます。

`-p` *proclist*　プロセス ID 番号が *proclist* で指定されたプロセスの情報のみを表示します。

`-P`　追加のカラムヘッダー `PSR` (もしある場合) をつけて、プロセスまたは `lwp` が接続されたプロセッサ数を出力します。

`-s` *sidlist*　ID が *sidlist* にあるすべてのセッションリーダーの情報を表示します。

`-t` *term*　*term* に関連するプロセスの情報のみを表示します。 端末識別子は、デバイスのファイル名および識別子として指定されます。たとえば、`term/a` や `pts/0` などです。

`-u` *uidlist*　実効ユーザー ID またはログイン名が *uidlist* で指定されたプロセスデータのみを表示します。リスト出力時に、ログイン名を表示する `-f` オプションをユーザーが指定しない限り、ユーザー ID は数字で出力されます。

`-U` *uidlist*　実グループ ID またはログイン名が *uidlist* に存在するプロセスの情報を表示します。*uidlist* は、複数の ID を空白またはコンマで区切って記述するリストで、単一の引数として指定する必要があります。

`-y`　長い形式のリストで (`-l`)、旧式の `F` と `ADDR` カラムが省略され、プロセスの常駐の設定サイズを示す `RSS` カラムが含まれます。`-y` オプションを指定すると、`RSS` と `SZ` (後述参照) の両方が、ページの代わりに K バイト単位で報告されます。

上記の多くのオプションは、対象とするプロセスをリストに選択するために使用します。これらのオプションが指定された場合、`ps` はデフォルトリストを無視し、オプションの選択条件の論理和により示されるプロセスを選択します。

表示書式

`-f` オプションが指定された場合、`ps` は、ユーザーブロックを検索することによって、プロセスが作成されたときに指定されたコンド名および引数を特定しようとします。これができない場 合には、`-f` オプションを指定しない場合のコマンド名が角括弧に囲まれて出力されます。

`ps` の実行結果として表示されるリストのカラムの見出しおよびカラムの意味を説明します。文字 `f` および `l` は、それに相当する見出しを作成するオプション (それぞれ完全形式または長い形式) を示します。`all` は、見出しを常に出力することを意味します。これら 2 つのオプションは、プロセスのどの情報が表示されるかを決定するだけで、どのプロセスが表示されるかということは決定しません。

`F` (l)　プロセスに関連するフラグ (16 進数表現の論理和) 。これらのフラグは過去においては意味を持っていたが、現在では何の意味も持たない

S (l)	プロセスの状態 O　プロセスは、プロセッサ上で実行中 S　休止状態。プロセスは、イベントが完了するのを待っている R　実行可能状態。プロセスは、実行待ち行列上にある Z　ゾンビ状態。プロセスは終了していて、親プロセスは待っていない T　ジョブ制御シグナルにより、もしくはトレース状態にあるため、プロセスは停止されている
UID (f,l)	プロセスの実効ユーザー ID (-f オプションではログイン名が出力)
PID (all)	プロセスの実効ユーザー ID (このデータはプロセスを停止させるために必要)
PPID (f,l)	親プロセスのプロセス ID
C (f,l)	スケジューリングのためのプロセッサ利用率 (旧仕様) -c オプションを使用した場合は、出力されない
CLS (f,l)	スケジューリングクラス。-c オプションを使用した場合にのみ、出力される
PRI (l)	プロセスに対する優先順位。-c オプションを指定しないと、数が大きいほど優先順位が低くなる。-c オプションを指定すると、数が大きいほど優先順位が高いことを意味する
NI (l)	優先順位の計算に使用される nice 値。-cオプションを使用した場合は、出力されない。特定のスケージューリングクラスのプロセスのみが、nice 値を持つ
ADDR (l)	プロセスのメモリーアドレス
SZ (l)	割り当てられたすべてのファイルとデバイスを含む、仮想メモリー内のプロセスの合計サイズ (ページ単位)。pagesize(1) を参照
WCHAN (l)	休止状態、または SXBRK 状態になっているプロセスが対象としているイベントのアドレス (空白文字の場合は、プロセスは実行中)
STIME (f)	時間、分、秒で示されるプロセスの開始時間 (ps が実行される 24 時間以上前に開始したプロセスは、月および日で示される)
TTY (all)	プロセスを制御している端末 (制御端末がない場合は、メッセージ ? が出力される)
TIME (all)	プロセスの累積実行時間
CMD (all)	コマンド名 (-f オプションでは、完全なコマンド名と引数を最大 80 文字まで出力)

-j オプションを指定すると、次の 2 つのカラムも出力されます。

PGID　プロセスグループリーダーのプロセス ID

SID　セッションリーダーのプロセス ID

-L オプションを指定すると、次の 2 つのカラムも出力されます。

LWP　情報が出力されている lwp の lwp ID

NLWP　プロセス中の lwp 数 (-f も指定した場合)

-L オプションを指定すると、プロセス中の各 lwp が 1 行に 1 つずつ出力され、プロセスではなく lwp に対する時間のフィールド STIME と TIME を示します。従来のシングルスレッドプロセスは 1 つの lwp だけを含みます。

すでに終了し、親プロセスを持ち、ただし、その親プロセスが待ち状態になっていないプロセスは、<defunct> と符号で示されます。

-o format　-o オプションを使用すると、出力書式をユーザーが指定することができます。

出力書式は、空白またはコンマで区切った引数のリストで指定します。各変数にはデフォルトのヘッダーがあります。等号と新しいヘッダーのテキストを追加することによって、デフォルトのヘッダーを上書きできます。引数の残りの文字は、ヘッダーテキストとして使用されます。コマンド行で指定した順番でフィールドが書き込まれ、カラムとして出力されます。フィールド幅は、少なくともヘッダーを表示できる幅 (デフォルト値または指定値) がシステムによって選択されます。-o *user=,* のようにヘッダーテキストが NULL である場合、フィールド幅は少なくともデフォルトのヘッダーテキストを表示できる幅になります。すべてのヘッダーテキストが NULL である場合、ヘッダー行が出力されません。

POSIX ロケールでは、次の名前が認識されます。

user　プロセスの実効ユーザー ID。テキストのユーザー ID を取得でき、フィールド幅が足りる場合は、テキストのユーザー ID になります。そうでない場合は、10 進数の ID になります。

ruser　プロセスの実ユーザー ID。テキストのユーザー ID を取得でき、フィールド幅が足りる場合は、テキストのユーザー ID になります。そうでない場合は、10 進数の ID になります。

group　プロセスの実効グループ ID。テキストのユーザー ID を取得でき、フィールド幅が足りる場合は、テキストのユーザー ID になります。そうでない場合は、10 進数の ID になります。

rgroup　プロセスの実グループ ID。テキストのユーザー ID を取得でき、フィールド幅が足りる場合は、テキストのユーザー ID になります。そうでない場合は、10 進数の ID になります。

pid　プロセス ID の 10 進数値。

ppid　親プロセス ID の 10 進数値。

`pgid` プロセスグループ ID の 10 進値。

`pcpu` 最近のある期間において、使用できる CPU 時間に対して実際に使用された CPU 時間の割合 (単位: %)。「最近」および「使用できる CPU 時間」は不定で、場合によって異なります。

`vsz` 仮想メモリー中のプロセスの合計サイズ (単位: キロバイト)。

`nice` プロセスの、システムスケジューリング優先順位を表す 10 進数値。`nice`(1) を参照してください。

`etime` POSIX ロケールにおける、プロセスが開始されてからの経過時間。書式は次のとおりです。

[[*dd*-] *hh*:] *mm*:*ss*

dd 日

hh 時間

mm 分

ss 秒

dd フィールドの値は 10 進の整数です。*hh*、*mm*、*ss* フィールドは、ゼロによって左揃えされた 2 桁の 10 進整数です。

`time` POSIX ロケールにおける、プロセスの累積 CPU 時間。書式は次のとおりです。

[*dd*-] *hh*:*mm*:*ss*

dd、*hh, mm*、*ss* フィールドは、`etime` と同様です。`etime` の説明を参照してください。

`tty` プロセスの制御端末の名前。書式は `who`(1) コマンドによって使用されるものと同じです。

`comm` 実行されているコマンドの、文字列としての名前 (`argv[0]` の値)。

`args` 文字列としての、コマンドとそのすべての引数。実装によってフィールド幅までに切り捨てられることがあります。これは実装に依存します。この文字列が、コマンドが起動されたときにコマンドに渡された引数であるか、またはアプリケーションによって変更された引数であるかは、不定です。アプリケーションは、引数リストを変更できて、その変更を `ps` の出力に反映することに依存しません。Solaris 実装では、この文字列は 80 文字以内に限定され、コマンドが起動されたときにコマンドに渡された引数となります。

Solaris 実装では、次の名前が認識されます。

`f` プロセスに関連付けられたフラグ (16 進数、加法的)。

s	プロセスの状態。
c	スケジューリングのためのプロセッサ利用 (旧仕様)
uid	プロセスの実効ユーザー ID (10 進整数)
ru+id	プロセスの実ユーザー ID (10 進整数)
gid	プロセスの実効グループ ID (10 進整数)
rgid	プロセスの実グループ ID (10 進整数)
projid	プロセスのプロジェクト ID (10 進整数)
project	プロセスのプロジェクト ID (取得できる場合はテキスト。そうでない場合は 10 進整数)
sid	セッションリーダーのプロセス ID
taskid	プロセスのタスク ID
class	プロセスのスケジューリングクラス
pri	プロセスの優先順位 (数値が大きいほど優先順位が高い)
opri	プロセスの廃止優先順位 (数値が小さいほど優先順位が高い)
lwp	lwp ID (10 進数)。1 行に 1 つの進行中の軽量プロセスが示されます。
nlwp	軽量プロセスの数
psr	プロセスまたは軽量プロセスがバインドされているプロセッサの数
pset	プロセスまたは軽量プロセスがバインドされているプロセッサセットの ID
addr	プロセスのメモリーアドレス
osz	仮想メモリー中のプロセスの合計サイズ (単位: ページ)
wchan	プロセスが休止中のイベントのアドレス (- の場合、プロセスは動作中)
stime	プロセスの開始時刻または開始日。空白なしで出力されます。
rss	プロセスの常駐の設定サイズ (単位: キロバイト)
pmem	マシン上の物理メモリーに対する、プロセスの常駐の設定サイズの割合 (単位: %)
fname	プロセスの実行可能ファイルのベース名の先頭 8 バイト

comm および args だけに、空白文字を含めることができます。Solaris 実装の変数およびその他すべての名前には、空白文字を含めることができません。

POSIX ロケールで使用されるデフォルトヘッダーとそれに対応する書式指定子は、次のとおりです。

書式指定子	デフォルトヘッダー	書式指定子	デフォルトヘッダー
args	COMMAND	ppid	PPID
comm	COMMAND	rgroup	RGROUP
etime	ELAPSED	ruser	RUSER
group	GROUP	time	TIME
nice	NI	tty	TT
pcpu	%CPU	user	USER
pgid	PGID	vsz	VSZ
pid	PID		

Solaris 実装における書式指定子と対応するデフォルトヘッダーは、次のとおりです。

書式指定子	デフォルトヘッダー	書式指定子	デフォルトヘッダー
addr	ADDR	projid	PROJID
c	C	project	PROJECT
class	CLS	psr	PSR
f	F	rgid	RGID
fname	COMMAND	rss	RSS
gid	GID	ruid	RUID
lwp	LWP	s	S
nlwp	NLWP	sid	SID
opri	PRI	stime	STIME
osz	SZ	taskid	TASKID
pmem	%MEM	uid	UID
pri	PRI	wchan	WCHAN

使用例

例 1 ps コマンドの例

```
example% ps -o user,pid,ppid=MOM -o args
```

上記のコマンドによって、POSIX ロケールでは以下の内容が出力されます。

```
 USER  PID   MOM   COMMAND
helene  34    12   ps -o uid,pid,ppid=MOM -o args
```

切り捨てが発生する可能性があるため、COMMAND の内容は必ずしも上記の例と一致しません。

環境

ps の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、NLSPATH についての詳細は、environ(5) を参照してください。

COLUMNS　画面の水平方向のサイズとして、システムが決定する値の代わりに使用する値を定義します。この値により、表示するテキストカラム数が決まります。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

ファイル

/dev/pts/*

/dev/term/*　端末 (tty) 名を検索するファイル

/etc/passwd　UID 情報を提供

/proc/*　プロセス制御ファイル

/tmp/ps_data　内部のデータ構造体

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み (「使用法」参照)

関連項目

kill(1), nice(1), pagesize(1), pgrep(1), priocntl(1), who(1), getty(1M), proc(4), ttysrch(4), attributes(5), environ(5)

注意事項

ps の実行中に状況が変化することがあります。ps が提供する情報は、ある瞬間だけに当てはまるものであり、ユーザーがそれを参照するときには正確ではないことがあります。すでに存在しないプロセスについてのデータなど、該当しなくなった情報が ps の出力結果に含まれていることもあります。

プロセスを選択するオプションを 1 つも指定しない場合、ps はその制御端末に関連するプロセスすべてについて報告します。制御端末がない場合には、ヘッダー以外には何も報告されません。

`ps -ef` および `ps -o stime` は、tty ログインセッションの実際の開始時間を報告せず、実際よりも少し早い時間、つまり getty が最後に tty 行に再生成された時間を報告します。

`ps` はログイン名 (ユーザー名) を除いて CSI 対応が可能です。

名前 | cd, chdir, pushd, popd, dirs – 現在の作業用ディレクトリの変更

形式 | **/usr/bin/cd** [*directory*]

sh | **cd** [*argument*]

chdir [*argument*]

csh | **cd** [*dir*]

chdir [*dir*]

pushd [+ | *n* | *dir*]

popd [+ *n*]

dirs [-l]

ksh | **cd** [*arg*]

cd *old new*

/usr/bin/cd | /usr/bin/cd ユーティリティは、cd ユーティリティ自身だけの現在のディレクトリを変更します。これは、後述するシェル組み込みの cd とは対照的です。/usr/bin/cd はプロセスの呼び出しには影響しませんが、あるディレクトリを現在のディレクトリとして設定できるかどうかを決定するのに使用できます。

sh | Bourne シェルに組み込まれている cd は、現在のディレクトリを *argument* で指定されたディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *argument* になります。 シェル変数 CDPATH は、 *argument* を含むディレクトリの検索パスを定義します。代替ディレクトリ名は、 コロン (:) で区切ります。 デフォルトのパスは空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは空のパス名で指定します。 このパス名は、等号の直後か、 パスリスト内にある区切り文字のコロンの間に指定します。*argument* の先頭文字が / 、 . 、または . . の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *argument* を検索します。 cd は、 *argument* 中で実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は効率が悪くなります。 そのため、 cd コマンドは、シェルに組み込まれています。 (pwd(1) 、 sh(1) 、 chdir(2) を参照)

chdir は、cd を呼び出すもうひとつの方法です。

csh | *dir* 引数を省略すると、 C シェルに組み込まれている cd は、シェル変数 HOME の値を新たな作業用ディレクトリとして 使用します。 *dir* を指定した場合、それが / 、 . 、 または . . で始まる完全なパス名であれば、その *dir* が新たな作業用ディレクトリとなります。 それ以外の場合は、シェル変数 CDPATH が指定するパスと相対関係を持つディレクトリの中から 該当するものを探し出します。 CDPATH の構文は PATH シェル変数と同一で、セマンティクスも似ています。 cd は *dir* に対する実行 (検索) 権を持っていなければなりません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、C シェルに組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

chdir はシェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。 引数を指定しないと、 そのユーザーのホームディレクトリに変更します。 *dir* が現在のディレクトリからは見つからない相対パス名の場合、変数 cdpath 内のディレクトリリストを検索します。 *dir* が / で始まる値を持つシェル変数の名前である場合、 その変数の値が示すディレクトリに変更します。

pushd はディレクトリスタックにディレクトリをプッシュ (押し込む) します。 引数を指定しないと、スタックにある先頭の 2 つの構成要素を交換します。

+n *n* 番目のエントリがスタックの先頭になるよう回転し、 そのディレクトリに移ります。

dir 現在の作業用ディレクトリをスタックにプッシュし、 そのディレクトリに移ります。

popd はディレクトリスタックからポップして (取り出して)、新たに先頭となったディレクトリへ cd します。ディレクトリスタックの構成要素の先頭番号は、 0 となります。

+n スタック内の *n* 番目のエントリを破棄します。

dirs はディレクトリスタックを出力します。 現在のディレクトリが最も左に現れるように 時間順に出力されます。 -l 引数を指定すると、 ~ を使った省略形ではなく、完全な形式で出力されます。

ksh Korn シェルに組み込まれた cd コマンドは、上記 2 つの形式の いずれかで入力します。第 1 の形式は、現在のディレクトリを *arg* に変更します。 *arg* が – の場合、ディレクトリを直前のディレクトリに変更します。 シェル変数 HOME の値がデフォルトの *arg* になります。PWD 変数は、現在のディレクトリに設定されます。シェル変数 CDPATH は、*arg* を含むディレクトリの検索パスを定義します。 ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定)。 なお、現在のディレクトリは NULL のパス名で指定します。 このパス名は、 等号の直後か、パスリスト内にある区切り文字のコロンの間に指定します。 *arg* の先頭文字が / 、 . 、または . . の場合、 検索パスは使用しません。 それ以外の場合は、 パス中の各ディレクトリで *arg* を検索します。 cd の第 2 の形式は、 PWD 中の現在のディレクトリ名における *old* という文字列を *new* という文字列に置換し、 この新規のディレクトリへ変更しようとします。 cd コマンドは rksh では実行できません。 コマンドを実行するたびに新しいプロセスが生成されるため、 cd を通常のコマンドとして実装した場合は、効率が悪くなります。 そのため、 cd コマンドは、 ksh に組み込まれています。 詳しくは pwd(1) 、 sh(1) 、 chdir(2) を参照してください。

オペランド 以下のオペランドを指定できます。

directory 新たな作業用ディレクトリとなるディレクトリの絶対または相対パス名。 cd が相対パス名をどのように解釈するかは、環境変数 CDPATH の設定により異なります。

出力 CDPATH に設定されている空でないディレクトリ名が用いられる場合、 新たな作業用ディレクトリの絶対パス名が 以下のような形式で標準出力に出力されます。

"%s\n", <*new directory*>

それ以外の場合には、何も出力されません。

環境

cd の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

CDPATH　コロンで区切られた、ディレクトリを示すパス名のリスト。 *directory* オペランドの先頭文字がスラッシュ (/) でなく、 先頭部分が . でも .. でもない場合には、 cd はこのリスト内のパス名を順番に検索し、環境変数 CDPATH に指定されている名前のディレクトリから *directory* を探します。その結果、最初に見つかったディレクトリ名が 新たな作業用ディレクトリとなります。 ディレクトリのパス名として空の文字列を指定すると、 それは現在のディレクトリと見なされます。 CDPATH は、設定されていないときには空の文字列として扱われます。

HOME　*directory* オペランドが省略されたときに用いるホームディレクトリの名前

PWD　現在の作業用ディレクトリのパス名。 この変数は、そのディレクトリに移った後に cd により設定されます。

終了ステータス

以下の終了ステータスが返されます。

0　ディレクトリが正常に変更された。

>0　エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), pwd(1), sh(1), chdir(2), attributes(5), environ(5)

名前 | pwd – 作業中のディレクトリの出力

形式 | **/usr/bin/pwd**

機能説明 | pwd は、現在の作業用 ディレクトリの絶対パス名を標準出力に書き出します。

Bourne シェル sh(1) と Korn シェル ksh(1) にも pwd 組み込みコマンドがあります。

環境 | pwd の実行に影響を与える環境変数 LC_MESSAGES 、 NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　　正常終了

>0　　エラーが発生した

エラーが検出された場合、 出力は標準出力に書き込まれず、 診断メッセージが標準エラーに書かれます。 終了ステータスは 0 にはなりません。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | cd(1), ksh(1), sh(1), shell_builtins(1), attributes(5), environ(5)

診断 | "Cannot open .." および "Read error in .." は、ファイルシステムに異常がある可能性を示しています。 UNIX のシステム管理者にお尋ねください。

注意事項 | 現在のディレクトリまたはその 1 つ上のディレクトリに移動する場合、 pwd が正しく応答しないことがあります。 cd(1) コマンドにフルパス名を付けて使用し、 この状態を修正してください。

名前 | rcp – リモートファイルコピー

形式 | **rcp** [-p] *filename1 filename2*

rcp [-pr] *filename... directory*

機能説明 | rcp コマンドは、マシン間でファイルをコピーします。*filename* 引数、*directory* 引数にリモートファイルを指定する場合は、次のような書式を使用します。

hostname:*path*ローカルファイルを指定する場合は、コロン (:) を使用しないか、コロンの前にスラッシュ (/) を付けます。

hostname には、IPv4 または IPv6 のアドレス文字列を指定できます (inet(7P)、inet6(7P) を参照)。IPv6 のアドレスにはコロンが含まれているので、*hostname* を角括弧で囲む必要があります。 *hostname* を角括弧で囲まないと、最初のコロンが *hostname* と *path* の区切り文字と見なされてしまいます。正しくは、次のような書式になります。

```
[1080::8:800:200C:417A]:tmp/file
```

また、フルパス名を指定しないと、*filename* はホスト (*hostname*) 上のホームディレクトリから見た相対パスと解釈されます。リモートホスト上のパス名と解釈させるには、パス名 (*path*) を \ 、" 、' などで囲みます。

rcp コマンドの使用時、パスワードを入力する必要はありません。ただし、*hostname* で指定したホスト上に現在のローカルユーザー名が存在し、 rsh(1) を使用してリモートコマンドを実行できなくてはなりません。

rcp では、ソースファイルもターゲットファイルも現在のマシン上に存在しないような「サードパーティコピー」も処理できます。 ホスト名は、次の形式になります。

username@*hostname*:*filename*この場合、現在のローカルユーザー名でなく、*username* に指定されたユーザー名が リモートホスト上のユーザー名として使用されます。また、 rcp は、リモートホストの インターネットドメインアドレス指定をサポートしており、次の形式で、使用するユーザー名、ホスト名、ホストの所属するドメインを指定できます。

username@*host*.*domain*:*filename**filename* にフルパス名を指定しないと、*username* に指定したリモートホスト上のユーザーのホームディレクトリから見た相対アドレスと解釈されます。

オプション | 次のオプションを指定できます。

-p　必要に応じて、各コピーに、オリジナルのファイルと同じ変更時間、アクセス時間、 モード、 ACL を設定します。

-r　指定された *filename* のサブツリーをコピーします。 この場合、コピー先はディレクトリでなければなりません。

使用法 | ファイルサイズが 2 ギガバイト (2^{31} バイト) 以上の場合の rcp の動作については、largefile(5) を参照してください。

rcp コマンドでは、IPv6 を使用できます。ip6(7P) のマニュアルページを参照してください。

ファイル

$HOME/.profile

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

cpio(1)、ftp(1)、rlogin(1)、rsh(1)、setfacl(1)、tar(1)、hosts.equiv(4)、attributes(5)、largefile(5)、inet(7P)、inet6(7P), ip6(7P)

注意事項

rcp の目的は、別々のホスト間でファイルをコピーすることです。このため、 たとえば次のような指定で、あるファイルをそのファイル自体にコピーしようとすると、ファイルが壊れてしまいます。

```
example% rcp tmp/file myhost:/tmp/file
```

コピーの対象がディレクトリではなくファイルである場合、 rcp が正常に失敗しないことがあります。

また、リモートホスト上の $HOME/.profile ファイルに指定されているコマンドからの出力によって、処理が正しく行われないこともあります。

rcp でサードパーティコピーを行うには、ソースホストがリモートホスト上で コマンドを実行することを許可されていなければなりません。

rcp では、シンボリックリンクの処理が正しく行われません。 シンボリックリンクや名前付きパイプが含まれているディレクトリをリモートコピーする場合は、 tar または cpio コマンドを rsh コマンドにパイプして使用してください。詳細については、tar(1) と cpio(1) を参照してください。

リモートホストを指定する際のメタキャラクタを引用符で囲まないと、 解釈不可を示すエラーメッセージが出力されます。

ACL をサポートしていないファイルシステムに ACL をコピーすると、 rcp は失敗します。

rcp は、 ユーザー名、 ホスト名、 ドメインの処理を除いて CSI 対応が可能です。

名前 | rdist – リモートファイル配布プログラム

形式 | **rdist** [-b] [-D] [-h] [-i] [-n] [-q] [-R] [-v] [-w] [-y] [-d *macro* = *value*] [-f *distfile*] [-m *host*] ...

rdist [-b] [-D] [-h] [-i] [-n] [-q] [-R] [-v] [-w] [-y] -c *pathname*... [*login* @] *hostname* [: *destpath*]

機能説明 | rdist ユーティリティは、複数のホスト上でファイルのコピーを保守します。マスターコピーの所有者、グループ、モード、更新時刻はそのまま保持しながら、実行中のプログラムを更新できます (ファイルの内容が変更されなかった場合、rdist は所有者やモードの変更を反映しないことに注意してください)。通常、リモートホスト上のコピーは、そのサイズと更新時刻がローカルホスト上のオリジナルと異なっていれば更新されます。-y オプション (更新時刻モード) を使用すると、更新時刻だけがチェックされます。サイズの比較は行われません。下記を参照してください。

rdist には 2 つの形式があります。「形式」の項に示されている 1 つ目の形式の rdist は、-f オプションの *distfile* 引数で示されたファイルから、ファイルやディレクトリを更新する手順を読み込みます。*distfile* 引数の値が '–' の場合は、標準入力から読み込みます。-f が省略された場合、rdist は初めに自分の作業中のディレクトリ内で distfile を探し、次に Distfile を探して更新用の手順を得ようとします。

「形式」の項に示されている 2 つ目の形式の rdist では、コマンド行オプションとして -c オプションを使用してパスを指定します。

複数のマシン間で rdist を使用するには、各ホストマシンに /etc/host.equiv ファイルが用意されているか、またはユーザーがホームディレクトリ内の .rhosts 中にエントリを持っていなければなりません。詳細については hosts.equiv(4) を参照してください。

オプション | 次のオプションを指定できます。

-b
バイナリ比較。単に日付とサイズを比較するのではなく、バイナリ比較を行い、内容が異なっていればファイルを更新します。

-c *pathname* **...** **[***login* **@]***hostname***[:***destpath* **]**
指定したホストへ *pathname* で示すパス名をコピーします。*destpath* を指定した場合は、そのホスト上の *pathname* を更新しません。相対ファイル名は、ユーザーのホームディレクトリに相対であると見なされます。'login @' を先頭に記述した場合には、その login のユーザー ID を使って更新処理が行われます。':*destpath*' を記述した場合は、リモートファイルはその パス名としてインストールされます。

-d *macro=value*
macro で示すマクロに対して *value* で示す値を設定します。このオプションは、*distfile* 中のマクロ定義を変更して使いたい場合に使用します。*value* として指定できるのは、空の文字列、1 個の名前、または全体をカッコで囲んで複数の名前を空白で区切って記述したものです。

-D
デバッグ処理を実施します。

`-f` *distfile*
distfile が示すファイルに、更新処理の手順が記述されていることを指定します。*distfile* 引数として '`-`' を指定すると、標準入力とみなされます。

`-h`
シンボリックリンクをたどります。つまりリンク自身をコピーするのではなく、リンクが指すファイルをコピーします。

`-i`
未解決のリンクを無視します。`rdist` は転送対象のファイルのリンク構造を保守しようとし、すべてのリンクが見つけられなかったときユーザーに警告を出します。

`-m` *host*
更新対象とするホストマシンを *host* が示すマシンだけに限定します。上記の *distfile* 中に複数のホストが記述されていて、その一部に対してだけ更新を行いたい場合、この `-m` オプションを必要な回数だけ指定して対象ホストを限定できます。

`-n`
コマンドを実行しないで単に表示します。このオプションは、*distfile* をデバッグする場合に便利です。

`-q`
静寂モード。更新するファイルを標準出力に出力しません。

`-R`
関連していないファイルを削除します。つまりディレクトリを更新する際、リモートホスト上にあるファイルのうち、マスター (ローカル) ディレクトリに対応するものがないファイルを削除します。このオプションは、ディレクトリのコピーで内容がまったく同一のものを保守する際に便利です。

`-v`
すべてのホスト上でファイルが最新状態にあることを確認します。最新の日付でないファイルが見つかると、それが表示されます。ただし、どのファイルも更新されず、またメールも送られません。

`-w`
ファイル名全体が、宛先ディレクトリ名の終わりに付加されます。通常、リネームを行う際には 名前の最後の構成要素だけが付加されます。このオプションを使えば、コピーされるファイルのディレクトリ構造を平にする代わりに、その構造を保持します。たとえば、(`dir1/f1 dir2/f2`) というファイルに対して `dir3` にリネームする旨を指定すると、`dir3/f1` と `dir3/f2` ではなく、 `dir3/dir1/f1` と `dir3/dir2/f2` という名のファイルが作成されます。文字 `~` で始まるファイル名があるときに `-w` オプションを使用すると、ホームディレクトリ以外の すべてが宛先名に付加されます。

`-y`
マスターコピーよりも日付の古いリモートコピーが見つかったとき、更新しないで警告メッセージを発行します。更新時刻だけがチェックされます。サイズの比較は行われません。

空白　復帰改行、タブ、空白文字の各文字は、いずれも空白として扱われます。入力行が変わっても、次の対応指定が検出されるまでは現在の対応が継続されます。対応指定の始まりは、ファイル名の後に '→' が続いたもの、またはファイル名のリストの始まりを示すカッコ (で示されます。

注釈　注釈は # で始まり復帰改行で終わります。

Distfiles　*distfile* には、コピー元のファイル、コピー先のファイル、コピー先のホスト、更新時に実行する処理、等を指定する一連のエントリが記述されています。各エントリは次に示すいずれかの形式で記述されています。:

variable_name '=' *name_list*
[label:] *source_list* '->' *destination_list command_list*
[label:] *source_list* '::' *time_stamp_file command_list*1 つ目の形式は、変数を定義するために使用します。2 つ目の形式は、他のホストへファイルを配布するために使用します。3 つ目の形式はある特定の日時以降に変更があったファイルをリストするために使用します。*source_list* には、配布するもののマスターコピーとして使用する、ローカルホスト上のファイルやディレクトリを記述します。*destination_list* は、ファイルをコピーする宛先ホストを記述します。*source_list* に記述した各ファイルは。更新中のホスト上で変更が加えられている場合 、または *time_stamp_file* よりもファイルが新しい場合に (3 つ目の形式)、変更があったファイルのリストに追加されます。ラベルは指定してもしなくても構いません。ラベルは、部分的に更新を行う場合のコマンドを識別するために使用します。コロン (:) は、ラベル (省略可) の後に使用します。二重コロン (::) は、特定の日時 (ファイル *time_stamp* の日時で指定) 以降に変更されたファイルをリストするために使用します。通常 ::' 形式のコマンド指定行で使用するのは notify だけです。

マクロ　rdist のマクロ機能には制限があります。マクロが展開されるのは、ファイル名またはホスト名のリスト内、および特定のプリミティブの引数リスト内に限られます。マクロを使ってプリミティブやそのオプションを表したり、'→' や '::' 記号を表したりすることはできません。

マクロの定義は、以下の形式の行で記述します。

macro = *value*

マクロの参照は、以下のような文字列で表します。

${*macro*}

ただし make(1S) の場合と同様に、マクロ名が 1 文字で構成されている場合には中カッコは省略できます。

メタキャラクタ　シェルのメタキャラクタである [、]、 {、 }、 *、 ? は、 csh(1) の場合と同様に、認識されて (ローカルホスト上でのみ) 展開されます。メタキャラクタの先頭にバックスラッシュを 付加すれば、エスケープすることができます。文字 ~ も、 csh の場合と同様に展開されますが、ローカルホスト上と宛先ホスト上では別個に展開されます。

ファイル名

ファイル名の先頭が / でも ~ でもない場合、各宛先ホスト上においてユーザーのホームディレクトリに相対しているとみなされます。この場合、現在の作業中のディレクトリに相対ではない点に注意してください。複数のファイル名を指定する場合には、全体を括弧で囲まなければなりません。

プリミティブ

rdist が各ファイルのリモートコピーを更新する際にどのような動作を実行するかを、以下のプリミティブを使って指定できます。

install [-b] [-h] [-i] [-R] [-v] [-w] [-y] [*newname*]
日付の古いファイルとディレクトリを再帰的にコピーします。*newname* オペランドが指定されていないときは、ローカルファイルの名前がリモートホスト上のコピーに与えられます。ファイル名のパス中の親ディレクトリが、リモートホスト中に存在していなければ、新たに生成されます。安全のため rdist は、宛先ホスト上にある空でないディレクトリを通常ファイルやシンボリックリンクには置き換えません。ただし -R オプションが指定された場合には、空でないディレクトリも、対応するファイル名がマスターホスト上に存在していないと削除されます。

install 用のオプションの意味論は、コマンド行で指定する対応するオプションのものと同じです。ただしその有効範囲は 特定のマップに限定されています。宛先ホストで使用されるログイン名は、宛先名の形式が *login@host.* である場合を除き、ローカルホストでのログイン名と同じです。宛先名が上記の形式のときは、ユーザー名 *login* で更新処理が実行されます。

notify *address . . .*
address で示す TCP/IP アドレスにメールを送信します。アドレスは以下の形式で記述します。

user@host

メールは、更新されたファイルと 検出されたエラーから構成されます。指定されたアドレスに接尾辞 *'@host '* が付加されていなければ、rdist は宛先ホストの名前を付加して完全な形にします。

except *filename . . .*
filename 引数で示すファイル群を更新の対象から除くことを指定します。

except_pat *pattern . . .*
pattern 引数で示す正規表現に一致するファイルを更新の対象から省くことを指定します。正規表現に関する詳細は、ed(1) を参照してください。なお *distfile* 中では、\ と $ の両文字はエスケープしなければなりません。*pattern* 中にシェル変数を指定することは可能ですが、シェルのファイル名展開機能はサポートされていません。

special [*filename*] . . . "*command-line* "
Bourne シェル sh(1) のコマンド行を指定して、各ファイルの更新後にそれらの コマンドをリモートホスト上で実行するよう指示します。*filename* 引数を 1 つも指定しなければ、どのファイルの更新後でも *command-line* 引数で示すコマンド行が実行されます。このときシェル変数 FILE は、ローカルホスト上のファイル名に設定されます。*command-line* 中に疑問符を記述すれば、入力行を *distfile* から読ませることができます。複数のシェルコマンドを記述するには、コマンド間をセミコロン (;) で区切ります。

command-line を実行するシェル用の、デフォルトの作業用ディレクトリは、リモートホスト上でのユーザーのホームディレクトリです。

IPv6

ftp コマンドは IPv6 に対応しています。ip6(7P) のマニュアルページを参照してください。

使用例

例 1 distfile ファイルの例

以下に示す distfile の例は、hermes と magus という 2 つのホスト上で、共有ライブラリの同一コピー、共有ライブラリで初期化したデータファイル、インクルードファイル、ディレクトリを保守するように rdist に対して指示しています。magus 上では、コマンドはスーパーユーザーとして実行されます。rdist は、タイムスタンプファイルと異なるローカルファイルを見つけるたびに、merlin@druid に報告します。括弧 ( ) は、コピー元リストまたはコピー先リスト中に、空白で区切られた名前が 0 個またはそれ以上含まれている場合に使用します。

```
HOSTS = ( hermes root@magus )
FILES = ( /usr/local/lib/libcant.so.1.1
         /usrlocal/lib/libcant.sa.1.1 /usr/local/include/{*.h}
         /usr/local/bin )

(${FILES}) → (${HOSTS})
      install –R ;
${FILES} :: /usr/local/lib/timestamp
         notify merlin@druid ;
```

ファイル

~/.rhosts	接続を許可するホスト名 / ユーザー名リスト (ユーザー用)
/etc/host.equiv	ユーザー名を共有するホスト名リスト (システム用)
/tmp/rdist*	更新リスト用の一時ファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ed(1), make(1S), sh(1), stat(2), hosts.equiv(4), attributes(5), ip6(7P)

診断

rdist のバージョン番号が一致しない、というメッセージが出力された場合、シェル起動時に何らかの問題 (たとえばそのユーザーが属しているグループの数が多すぎる) があったことに起因している可能性が高いと考えられます。

警告

スーパーユーザーは、NFS がマウントしたファイルシステム上では、通常のアクセス特権を持っていません。rdist を使ってそのようなファイルシステムをコピーしようとしても、失敗するか、またはコピーの所有者が “nobody” になります。

使用上の留意点

ソースファイルは、ローカルホスト上に存在するかマウントされていなければなりません。

ディレクトリ中のすべてのファイルを更新した後で特殊コマンドを 1 回だけ実行する、という処理を簡単に行う方法がありません。

変数の展開ができるのは、名前のリストに対してだけです。汎用的なマクロ機能は提供されていません。

ファイルの更新時刻が負の値つまり 1970 年 1 月 1 日より古い場合、`rdist` は異常終了します。

空でないディレクトリを通常ファイルや シンボリックリンクで置き換えるための、“force” オプションは提供されていません。また、ファイルモードと所有者だけが異なるファイルに関して、その 2 種類の情報だけを更新する方法も必要です。

名前 | read – 標準入力から 1 行を入力

形式 | **/usr/bin/read** [-r] *var*...

sh | **read** *name*...

csh | **set** *variable* = $<

ksh | **read** [-prsu [*n*]] [*name* ? *prompt*] [*name*...]

/usr/bin/read

read ユーティリティは、標準入力から 1 行を読み込みます。

デフォルトでは、 -r オプションが指定されない限り、バックスラッシュ文字 (\) をエスケープ文字と見なします。 標準入力が端末装置で、呼び出し側のシェルが対話型のとき、 read は以下の条件が存在していれば プロンプトを表示し、 継続行の入力を要求します。

- -r オプションが指定されておらず、バックスラッシュで終わる行を シェルが読み込んだとき。
- 復帰改行文字の入力後、 here-document (各種シェルで << を使用する機能) が終了しないとき。

シェル内と同じように、入力行はいくつかのフィールドに分割されます。 先頭のフィールドは最初に指定された変数 *var* に、次のフィールドは 2 番目の変数 *var* に、というように割り当てられます。 フィールドの総数より *var* オペランドの数が少ない場合、余ったフィールドおよびそれらの 区切り文字はすべて最後に指定された *var* に割り当てられます。フィールドより *var* の方が多い場合には、余った *var* には空の文字列が設定されます。

var オペランドで指定された変数の設定は、現在のシェルの 実行環境に影響を及ぼします。 read ユーティリティが、以下に示す例のように、サブシェル内や 別のユーティリティ実行環境で呼び出された場合には、 呼び出し側の環境中のシェル変数には影響を与えません。

```
(read foo)
nohup read ...
find . -exec read ... \;
```

標準入力はテキストファイルでなければなりません。

sh

標準入力から 1 行を読み取り、内部フィールド区切り文字の IFS（通常は空白文字またはタブ） を用いてワード境界を区切り、 最初のワードを最初の *name* に、 2 番目のワードを 2 番目の *name* に、というように 順次割り当てます。 残ったワードは最後の *name* に割り当てます。 \ に続いて復帰改行 (NEWLINE) を入力すれば、行を継続できます。 復帰改行以外の文字の前にバックスラッシュを付加すれば、 その文字を引用できます。 このバックスラッシュは、ワードが *name* に割り当てられる前に削除され、 バックスラッシュの後に位置する文字は解釈されません。 ファイルの終わりに到達した場合を除き、 戻り値は 0 となります。

csh

以下の表記は、 標準入力の 1 行を *variable* 値としてロードします (csh(1) 参照) 。

```
set variable = $<
```

ksh | シェルの入力機構です。 1 つの行を読み取り、 IFS が示す文字を、区切り文字として使用して 行の内容をいくつかのフィールドに分割します。 エスケープ文字（ \）は、次の文字の特別な意味または 行の継続に関する意味を取り除くために使用します。 -r で指定する raw モードでは、 \ が持つこの特殊な意味は無視されます。 第 1 フィールドを 1 番目の *name* に、 第 2 フィールドを 2 番目の *name* に、 という順番で割り当てていき、 余ったフィールドがあれば最後の *name* に割り当てます。 -p オプションは、 シェルが |& を使用して生成したプロセスの入力パイプから入力行を取り出します。 -s フラグは、 入力をコマンドとして履歴ファイルに保存します。 -u フラグは、 読み取り元となるファイル記述子番号を 1 桁の数値 *n* で指定します。 ファイル記述子は、 exec という特殊コマンドでオープンできます。 *n* のデフォルト値は 0 です。 *name* を省略すると、 REPLY の値をデフォルトとして使用します。 入力ファイルが読み込み用にオープンされていない場合と ファイルの終わりに到達した場合を除き、 終了ステータスは 0 です。 -p オプションが指定されていてファイルの終わりを検出すると、 このプロセスをクリアし別のプロセスを作成可能にします。 最初の引数が ? を含んでいると、 シェルが対話型のとき、 このワードの残りを標準エラーに対する プロンプトとして使用します。 ファイルの終わりに到達しないかぎり、 終了ステータスは 0 です。

オプション

以下のオプションがを指定できます。

`-r` バックスラッシュ文字を特別な文字とせず、 単なる入力行の一部として扱います。

オペランド

以下のオペランドを指定できます。

var 存在している、あるいはしていないシェル変数の名前。

使用例

例 **1** read コマンドの例

以下に示す /usr/bin/read の例は、入力ファイルの内容を、各行の先頭フィールドを最後尾に 移動して出力するものです。

```
while read -r xx yy
do
        printf "%s %s\n" "$yy" "$xx"
done < input_file
```

環境

read の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

`IFS` フィールドを区切るのに使われている、 内部フィールド区切り文字を定義します。

`PS2` 対話型シェルが標準エラー出力に出力するプロンプトの文字列を 定義します。プロンプトが出力されるのは、 -r オプションが指定されておらずバックスラッシュで終わる行を 読み込んだとき、または復帰改行文字の入力後に here-document が終了しないときです。

終了ステータス

以下の終了ステータスが返されます。

0 正常終了

>0	ファイルの終わりを検出した、またはエラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), line(1), set(1), sh(1), attributes(5), environ(5)

名前 | readonly – 再表示の場合、もとの値の書き換えを防ぐシェル組み込み関数

sh | **readonly** [*name*...]

ksh | ****readonly** [*name* [= *value*]...]

sh | 指定された各 *name* に「読み取り専用」のマークを付け、 これらの名前が後続の割り当てでは変更できないようにします。 引数を省略すると、 読み取り専用と指定された名前がすべて一覧表示されます。

ksh | *name* に「読み取り専用」のマークを付け、 これらの名前が後続の割り当てでは変更できないようにします。

1 つまたは 2 つの ** (アスタリスク) が先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | ksh(1), sh(1), typeset(1), attributes(5)

名前 | ed, red – テキストエディタ

形式 | **/usr/bin/ed** [-s | -] [-p *string*] [-x] [-C] [*file*]

/usr/xpg4/bin/ed [-s | -] [-p *string*] [-x] [-C] [*file*]

/usr/bin/red [-s | -] [-p *string*] [-x] [-C] [*file*]

機能説明 | ed ユーティリティは標準のテキストエディタです。 file が指定されていると ed は e コマンド (下記参照) を指定されたファイルについてシミュレートします。 その結果、このファイルは ed のバッファに読み込まれ、編集できるようになります。

ed ユーティリティは、編集されるファイルのコピーに 対して操作を行います。 つまり、コピーに対して行われた変更は w (write) コマンドが実行されるまで、そのファイルに対して有効になりません。 編集されるテキストのコピーは バッファという一時ファイルに存在します。バッファは 1 つしかありません。

red ユーティリティは ed の制限付きバージョンです。 red は現在のディレクトリ内のファイルしか編集できず、 !*shell command* によるシェルコマンドの実行もできません。 これらの制限を無視しようとするとエラーメッセージ (*restricted shell*) が表示されます。

ed と red は、両方とも fspec(4) フォーマット機能をサポートしています。 デフォルトの端末モードは stty –tabs または stty tab3 で、タブ位置は 8 カラムごとに設定されます (stty(1) を参照) 。ただし file の先頭行にフォーマット指定が記述されていれば、デフォルトモード に優先してその指定が有効となります。 たとえば file の先頭行に次のように指定したとします。

```
<:t5,10,15 s72:>
```

この場合、タブ位置が 5 、 10 、 15 に、そして最大行長が 72 に設定されます。

ed コマンドは単純で、その構造は規則的です。最初に 0 から 2 個までのアドレス、次に 1 文字のコマンド、最後に (あれば) コマンドのパラメータが続きます。アドレスはバッファ内の 1 行または複数行を指定するものです。 アドレスが必要なコマンドはすべてデフォルトのアドレスを持っているので、 アドレスを省略することがよくあります。

一般に、 1 行に 1 つのコマンドだけ指定します。 コマンドの中には、テキストを入力するものもあります。 この時テキストは、バッファの中の適切な場所に格納されます。 ed がテキストを受け付けているときのことを「入力モード」であるといいます。 このモードではコマンドは認識されません。 すべての入力がそのまま受け付けられるだけです。 入力モードから抜けるには、行の先頭でピリオド (.) だけを入力して、キャリッジリターンを押します。

/usr/bin/ed | ed が引数のあるコマンドを実行する場合、 デフォルトのシェル /usr/bin/sh が使われます (sh(1) 参照) 。

/usr/xpg4/bin/ed | ed が引数のあるコマンドを実行する場合、/usr/xpg4/bin/sh が使われます (ksh(1) 参照) 。

正規表現

ed ユーティリティでは「正規表現」規則が使用できますが、 ある程度制限されています。 正規表現は、アドレスの中では行を指定するために、 また、いくつかのコマンド (たとえば、 s) の中では行のうちの置換される部分を指定するために用いられます。 ed におけるアドレス指定方法を理解するには、 常に「現在行」が存在することを認識する必要があります。 一般に、現在行はコマンドによって影響を受けた最後の行です。 現在行が受ける影響については、 各コマンドの説明の箇所で述べます。

国際化された標準の正規表現は、 システムに与えられたすべてのロケールで 使用されます。 詳細については regex(5) を参照してください。

ed コマンド

コマンドには 0 、 1 、または 2 個のアドレスを必要とします。 アドレスを必要としないコマンドにアドレスを指定するとエラーになります。 アドレスが必要なコマンドに必要な数のアドレスを指定しないと、 デフォルトのアドレスが採用されます。 必要以上にアドレスが指定されると、最後のアドレスの方から使用されます。

通常、アドレスはコンマ (,) によってお互いに区切られます。 セミコロン (;) によっても区切られます。 後者の場合、最初のアドレスが計算され、 現在行 (.) がその値に設定されます。 その後、 2 番目のアドレスが計算されます。 この機能は、順方向および逆方向検索の開始行を決定するのに使用できます (上記規則 5 および 6 を参照) 。

以下に示す ed コマンドのリストでは、コマンドの前の括弧はアドレスの一部ではなく、 デフォルトのアドレスを示します。

各アドレス部分の先頭には任意の 2 個の空白文字を付加できます。 またコマンド文字の先頭にも任意の数の空白文字を付加できます。 接尾文字 (l 、 n 、または p) を指定するのであれば、コマンドの直後に記述しなければなりません。

e 、 E 、 f 、 r 、 w の各コマンドには、省略可能な file パラメタがあります。これを指定する場合には、コマンド文字との間に最低 1 個の空白文字を置くことが必要です。

バッファ全体を書き換えた最後の w コマンド実行後にバッファの内容が変更されているとき、 e または q コマンドによりエディタバッファを破壊しようとすると、 ed は警告を発します。具体的には以下の文字列を標準出力に書き出します。

"?\ n"

なお H コマンドにより「ヘルプモード」が起動されていれば、 状況を説明するメッセージが続いて出力されます。 この警告出力後も、 ed はコマンドモードのままで、現在の行番号は変わりません。 ここで続けて e または q コマンドを再度入力すれば、そのコマンドが実行されます。

標準入力から次のコマンドを読み込もうとして ファイルの終わりを検出した場合、ed ユーティリティは q コマンドが入力された場合と同じように動作します。

一般に、 1 行に 2 つ以上のコマンドを指定するとエラーです。 しかし、すべてのコマンド (e 、 f 、 r 、 w を除く) は、 l 、 n 、または p コマンド (それぞれ、現在行をリストする、番号付けする、出力する) を接尾辞として付けることができます (l 、 n 、および p コマンドを参照) 。

(.)a

<*text*>

.

append コマンドは、 0 行以上のテキスト行を受け入れ、それをバッファ内 のアドレスが示す行の直後に追加します。現在行 (.) は、最後に挿入された行に設定されます。 挿入行がない場合は、アドレスが示す行に設定されます。 a コマンドではアドレス 0 は有効です。この場合、バッファの先頭にテキストを追加します。端末から入力できる最大文字数は 1 行あたり 256 文字です (復帰改行文字 (NEWLINE) も含む) 。

(.)c

<*text*>

.

change コマンドは、バッファからアドレスで指定された行を削除し、 0 行以上のテキスト行をバッファ内に受け入れ、 削除された行と置換します。現在行 (.) は最終入力行に、または入力行がない場合、 削除された最後の行の直後の行に設定されます。
バッファの最終行を削除した場合、 現在の行番号は新たに最終行となった行のアドレスに設定されます。 削除によりバッファ内に行が残っていなければ、 現在の行番号はゼロに設定されます。

C

基本的に、後述する x コマンドと同じです。ただし、 NULL キーが入力されないかぎり、 e および r コマンドによって読み込まれたすべてのテキストを 暗号化されているものと見なします。

(. , .)d

delete コマンドは、バッファからアドレスで指定された行を削除します。 削除された最後の行の直後の行が現在行に設定されます。 バッファの最終行を削除した場合、 新たに最終行となった行が現在行になります。 削除によりバッファ内に行が残っていなければ、 現在の行番号は ゼロとなります。

e *file*

edit コマンドは、バッファの全内容を削除して file の内容をバッファ内に読み込みます。 現在行 (.) は、バッファの最終行に設定されます。 file が指定されないと、 (もし、あれば) 現在記憶されているファイル名を使用します (f コマンド参照) 。 -s オプションが指定された場合を除き、読み込んだバイト数が以下の形式で 標準出力に書き出されます。

"%d\ n" 読み込んだバイト数

file は、後で使用する e 、 E 、 r 、および w コマンドのデフォルトのファイル名として記憶されます。 file の代わりに ! を指定すると、 ! 以降の文字列はシェル (sh(1)) コマンドと見な

され、その出力が読み込まれます。 このようなシェルコマンドは現在のファイル名として記憶されません。 後述の「診断」も参照してください。 e コマンドが正常に終了すると、すべてのマークは捨てられます。 バッファ全体が最後に書き換えた後でその内容が変更されているとき、 前述したように警告が発せられます。

E *file* Edit コマンドは、 基本的に、 e コマンドと同じです。ただし、最後の w コマンドを実行してからバッファの内容が変更されたか否かをチェックしません。

f *file* file が指定されると、 f コマンドは、現在記憶されているパス名を file に変更します。 次に、パス名を変更したか否かに関わらず、現在記憶されているパス名 (変更した場合は新しいパス名) を以下の形式で標準出力に書き出します。

```
"%s\ n" パス名
```

現在の行番号は変わりません。

(1 , $) g/*RE*/*command list* global コマンドは、まず *RE* で指定された正規表現と一致する各行をマークします。 そして、該当するすべての行に対して、 まず現在行 (.) をその行に設定してから、 *command list* で指定したコマンドリストを実行します。 g コマンドが終了したとき、現在の行番号の値は、コマンドリスト中の 最後のコマンドが指定した値となっています。 一致する行が見つからなかった場合には、 現在の行番号の値は変わりません。 単一コマンドまたはコマンドリストの最初のコマンドは global コマンドと同一行に現われなければなりません。 コマンドが複数行にまたがる場合、 最終行以外の各行は バックスラッシュ (\ ) で終わらなければなりません。 a 、 i、および c コマンドおよび関連する入力を複数行に指定できます。 入力モードを終了する . が command *list* の最後の行の場合、省略できます。空の command *list* は p コマンドと同じです。 g 、 G 、 v 、 V 、および ! コマンドは、 *command list* 内に書くことができません。 「注意事項」および「ファイル」 の前の最後のパラグラフも参照してください。 *RE* を区切る文字として、スラッシュの代わりに、 空白と復帰改行以外の任意の文字を使うことができます。 また区切り文字を *RE* 中で実際の文字として使いたければ、その前にバックスラッシュを 付加してください。

(1 , $) G/*RE*/ 対話型 Global コマンドでは、最初に、 *RE* で指定された正規表現と一致する各行をマークします。 そして、該当する各行について、その行が標準出力に書き出され、 現在行 (.) がその行に変更され、この時点で、 1 つのコマンド (a 、 c 、 i 、 g 、 G 、 v 、および V コマンドをのぞく) が入力でき、入力されたコマンドは実行されます。 コマンドが実行されると、次のマークされた行が出力され、 以下同じ処理を繰り返します。 改行は無効なコマンドとして扱います。&は、現在起動中の G 内で、 最後に実行された NULL でないコマンドを再実行します。 注意 : G コマンド 実行中のコマンド入力はバッファ内の どの行でもアドレス指定したり、

影響を与えたりできます。 現在の行番号の最終的な値は、最後に正常終了したコマンド が指定した値です (なお G で入力したコマンドが失敗した場合、または NULL コマンドが入力された場合は、 G 自身が「最後に正常終了したコマンド」となります)。 一致する行が見つからなければ、現在の行番号は変わりません。 G コマンドは SIGINT シグナルによって終了させることができます。また、 割り込みシグナル (ASCII DEL または BREAK) でも終了できます。 *RE* を区切る文字として、スラッシュの代わりに、 空白と復帰改行以外の任意の文字を使うことができます。 また区切り文字を *RE* 中で実際の文字として使いたければ、その前にバックスラッシュを 付加してください。

`h`

help コマンドは、最後の ? 診断の理由を説明する短いエラーメッセージを出力します。 現在の行番号は変わりません。

`H`

Help コマンドは、 以後発生するすべての ? 診断のエラーメッセージを出力するモードにします。 もしあれば前の ? も説明します。 H コマンドは、このモードを交互にオンおよびオフにします。 最初はオフです。 現在の行番号は変わりません。

`( . )i`

<*text*>

`.`

insert コマンドは、 0 またはそれ以上の行のテキストを受け入れ、 バッファ内のアドレス指定された行の前に挿入します。 現在行 (.) は最後に挿入された行に設定されます。 挿入行がない場合は、アドレス指定された行に設定されます。 i コマンドと a コマンドは、入力テキストの挿入位置だけ異なります。 このコマンドにはアドレス 0 は有効ではありません。 端末から入力できる最大文字数は 1 行あたり 256 文字です (復帰改行文字も含む) 。

`( . , .+1 )j`

join コマンドは、 適当な復帰改行文字を削除することによって連続する行を結合します。 1 つのアドレスしか指定されないと j コマンドは何もしません。 行の結合が行われると、現在の行番号は結合された側の行のアドレスに 設定されます。結合が行われなければ、現在の行番号は変わりません。

`( . )k`*x*

mark コマンドは、 *x* という名前でアドレス行をマークします。 *x* は ASCII の小文字 (a–z) でなければなりません。 ′*x* を指定すると、このマークされた行を指すようになります。 現在行 (.) は変更しません。

`( . , . )l`

list コマンドは、アドレスされた行をすべての情報が見える形で標準出力へ書き出します。 \\ 、 \ a 、 \ b 、 \ f 、 \ r 、 \ t 、 \ v はそれぞれ対応するエスケープシーケンスとして書き出されます。 テーブル中にある \ n は適用外です。テーブル中にない非印字文字に関しては、各バイトごと (最上位ビットから) に 3 桁の 8 進数で、前にバックスラッシュが付加された形式で出力されます。 長い行は折り返されます。折り返しの発生地点にはバックスラッシュと 復帰改行文字が表示されます。 折り返し地点

の長さは不定ですが、出力装置に適した値に なっています。各行の終わりは $ でマークされます。 l コマンドは、 e 、 E 、 f 、 q 、 Q 、 r 、 w 、 ! 以外のすべてのコマンドの後に付けられます。現在の行番号は、最後に出力された行のアドレスに設定されます。

(. , .)m*a*　move コマンドは、アドレス行を *a* で示される行の後に移動します。アドレス 0 を *a* に書いても有効で、その場合アドレス行をファイルの先頭に移動します。 アドレス *a* が移動する行の範囲内にあるとエラーになります。 現在行 (.) は移動した最後の行に設定されます。

(. , .)n　number コマンドは、行番号とタブの後に、アドレス行を出力します。 現在行 (.) は出力した最後の行に設定されます。 n コマンドは e 、 E 、 f 、 q 、 Q 、 r 、 w 、または ! 以外のすべてのコマンドの後に続きます。

(. , .)p　print コマンドは、アドレス行を標準出力に出力します。 現在行 (.) は出力した最後の行に設定されます。 p コマンドは、 e 、 E 、 f 、 q 、 Q 、 r 、 w 、または ! 以外のすべてのコマンドの後に付けられます。 たとえば、 dp というコマンドは、現在行を削除して新しい現在行を出力します。

P　コマンドは、 後続のすべてのコマンドにアスタリスク (*) (-p 指定時は 文字列) によるプロンプトをつけて入力を促します。 P コマンドは、このモードを交互にオンまたはオフします。 初期値は、 -p オプション指定時はオン、省略時はオフです。 現在行は変わりません。

q　quit コマンドは、 ed を終了します。バッファ全体が書き換えられた後でバッファの内容が 変更されていると、警告が発せられます。 後述の「診断」を参照してください。

Q　エディタは、最後の w コマンドの後、バッファが変更されたか否かをチェックしないで終了します。

($)r *file*　read コマンドは、 file の内容をバッファ内に読み込みます。 file が指定されないと、(もしあれば) 現在記憶されているファイル名を使用します (e および f コマンド参照)。 file が ed 起動後参照された 最初のファイル名でないかぎり、現在記憶されているファイル名は 変更されません。 r コマンドにはアドレス 0 は有効です。この場合、バッファの先頭にファイルを読み込みます。読み込みに成功すると、 -s オプションが省略されていれば、 読み込まれた文字数が以下の形式で標準出力に書き出されます。

```
%d\ n 読み込んだバイト数
```

現在行 (.) は読み込まれた最後の行に設定されます。 file の代わりに ! を指定すると、 ! 以降の文字列はシェルコマンド (sh(1) 参照) と見なされ、その出力が読み込まれます。 たとえば、 $r

`!ls` は編集中のファイルの最後に現在のディレクトリを追加します。 このようなシェルコマンドは、現在のファイル名として記憶されません。

(. , .)s/*RE***/***replacement***/**
(. , .)s/*RE*/*replacement*/*count*, *count*=[1-512]
(. , .)s/*RE*/*replacement*/g
(. , .)s/*RE*/*replacement*/l
(. , .)s/*RE*/*replacement*/n
(. , .)s/*RE*/*replacement*/p

substitute コマンドは、各アドレス行について、 *RE* で示された正規表現を検索します。 これらの置換コマンドは、任意の数だけ指定できます。 一致が発生した各行に対して、 グローバル置換指示子 g がコマンドの後にあれば、すべての (重ならない) 一致した文字列を *replacement* に置換します。 グローバル指示子がなければ、 一致した文字列の最初のものだけを置換します。 数字 *count* がコマンドの後にあれば、各アドレス行内で一致した文字列のうち *count* 番目のものだけを置換します。 すべてのアドレス行について置換が失敗するとエラーになります。 正規表現 *RE* と *replacement* を区切るには、スラッシュ (/) の代わりに空白文字と復帰改行以外のすべての文字が使用できます。 現在行 (.) は置換が発生した最後の行に設定されます。 *RE* の区切り文字を *RE* 中で実際の文字として使いたければ、その前にバックスラッシュを付加してください。 「ファイル」 の前の最後のパラグラフも参照してください。 *replacement* 内のアンパサンド (&) は、現在行上で正規表現 *RE* と一致した文字列に置き換えられます。 この場合の & の特別な意味は、 \ を前につけることによって抑止できます。 さらに一般的な機能として、文字列 *n* (*n* は数字) は、 指定された正規表現 *RE* の \ （ と \ ） で囲まれた *n* 番目のサブ正規表現と一致するテキストに置換されます。 ネストされた括弧付きサブ正規表現が存在する場合、 *n* は左から数えた \ （ の発生回数によって決まります。 文字 % が *replacement* 内の唯一の文字であるとき、最後の置換コマンドで使用した *replacement* を現在の置換コマンドの *replacement* として使用します。 ただしそれ以前に置換コマンドがなかった場合、 このような % の使い方はエラーとなります。 % は、複数の文字の置換文字内にあるとき、または \ が前に付くときには、その特別な意味を失います。 *replacement* を先頭から終端まで走査する際にバックスラッシュ (\) が検出されると、後続の文字は特殊な意味を持っていたとしても その意味を失います。 なお & 、 \ 、 % および数字以外の文字については、どのような特殊な意味が与えられているか は不定です。 1 行を分割するには、復帰改行文字で置換します。 *replacement* 内の復帰改行文字は、 \ を前に付けてエスケープしなければなりません。 このような置換は、 g または v コマンドリストの一部としては実行できません。 現在の行番号は、置換が行われた最後の行のアドレスに設定されます。 置換がまったく行われないと、現在の行番号は変わりません。 行が分割されると、新たな現在の行番号を決定するため、 分割で発生した両方の行で置換が行われたものと見なされます。 置換文字列が置換対象となる文字列と同一の場合でも、 置換は発生したと見なされます。 置換コマンドは以下に示す指示子をサポートします。

count　各アドレス行で見つかった *RE* のうち、 *count* 番目のものだけを置換します。 *count* は 1 から 512 までの値でなければなりません。

g　最初のものだけでなく、すべての重なっていない *RE* を一括して置換します。 g と *count* の両方を指定した場合、結果は保証できません。

l　置換を行なった最後の行の内容を標準出力に書き出します。 出力形式は l コマンドのものと同じです。

n　置換を行なった最後の行の内容を標準出力に書き出します。 出力形式は n コマンドのものと同じです。

p　置換を行なった最後の行の内容を標準出力に書き出します。 出力形式は p コマンドのものと同じです。

(. , .)t*a*
t コマンドは、 m コマンドと同じように動作します。ただし、 アドレス行のコピーがアドレス a (0 でもよい) の後に置かれます。 現在行 (.) はコピーされた最後の行に設定されます。

u
undo コマンドは、 バッファの内容を変更した最後のコマンドの実行結果を無効にします。 無効にできるコマンドは、最後に実行した a 、 c 、 d 、 g 、 i 、 j 、 m 、 r 、 s 、 t 、 u 、 v 、 G 、または V コマンドです。 グローバルコマンドの g 、 G 、 v 、または V でバッファを変更していた場合、一括してその変更を無効にします。 グローバルコマンドで変更が行われていない (たとえば g/*RE* /p) 場合、 u コマンドは何も意味を持ちません。 現在の行番号は、無効にしたコマンドの開始直前に設定されていた値に戻ります。

(1 , $)v/*RE*/*command list*
v コマンドは、基本的に、グローバルコマンド g と同じです。ただし、 最初の段階でマークされる行は *RE* で示す正規表現に一致しない ものです。

(1 , $)V/*RE*/
V コマンドは、基本的に、対話型グローバルコマンド G と同じです。ただし、 最初にマークされる行は *RE* で示す正規表現に一致しない ものです。

(1 , $)w *file*
write コマンドはアドレス行を file に書き込みます。 file が存在しない場合は、ファイル生成マスクが他のモードで指定されていないかぎり、 モード 666 (すべてのユーザーが読み込み、書き込み可能) で作成します。 sh(1) 上での特殊コマンド umask の説明を参照してください。 file が ed 起動後参照された 最初のファイル名でないかぎり、現在記憶されているファイル名は 変更されません。 file が指定されないと、(もし、あれば) 現在記憶されているファイル名を使用します (e および f コマンド参照)。 現在行 (.) は変更されません。 コマンドが正常終了すると、 -s オプションが省略されていれば、書き込まれた文字数が以下の形式で 出力されます。

```
"%d\ n", 書き込んだバイト数
```

file の代わりに ! を指定した場合、 ! 以降のテキストは、アドレス行が標準入力であるシェル (sh(1) 参照) コマンドと見なされ、その出力が読み込まれます。 このようなシェルコマンドは、現在のパス名として記憶されません。 このような ! を伴った w コマンドは、「バッファ全体を書き換えた最後の w コマンド」と見なされます。

(1 , $ W *file*
: W コマンドは基本的に上述の write コマンドと同じです。ただし、 アドレス行を file (存在する場合) の最後に追加します。 file が存在しない場合、上述の w コマンドで述べたようにファイルを作成します。

X
: e および r コマンドで編集するために 読み込まれたテキストが暗号化されているか否かを判定します。 キーとして空文字列を与えると暗号化を無効にします。これ以後の e 、 r 、および w コマンドは、このキーをテキストの暗号化または復合化に使用します 。 明示的に空文字列をキーとして指定した場合は 暗号化は無効になります。 ed のオプション -x も参照してください。

($)=
: アドレス行の行番号が、次に示す形式で標準出力に表示されます。

```
"%d\ n" 行番号
```

このコマンドによって、現在の行番号は変更されません。

!*shell command*
: ! 以降のテキストを UNIX システムシェル (sh(1) 参照) に送信し、コマンドとして解釈します。 コマンドテキストにエスケープされていない % 文字を指定すると、記憶されているファイル名に置換されます。 ! がシェルコマンドの最初の文字として現われる場合、 それは前のシェルコマンドのテキストで置換されます。 つまり、 !! は最後のシェルコマンドを繰り返します。 % または ! による置換が実行されると、変更された行の内容が command の実行前に標準出力に書き出されます。 -s オプションが省略されていれば、 ! コマンドは終了時に以下のメッセージを標準出力に書き出します。

```
"!\ n"
```

現在の行番号は変更されません。

(.+1) <new-line>
: 行にアドレスだけを指定すると、そのアドレス行を出力します。 復帰改行文字だけの場合、 .+1p と同じです。 つまり、バッファ内を進むのに使用します。 現在の行番号は、書き出した行のアドレスに設定されます。

割り込みシグナル (ASCII DEL または BREAK) が送信されると、 ed は "?\ n" を出力して、 ed のコマンドレベルに戻ります。 ed ユーティリティはすべてのシグナルに対して標準的な動作を行います。 ただし次の 2 つのシグナルは例外です。

SIGINT
: ed ユーティリティは現在の動作を中断し、文字列 "?\ n" を標準出力に書き出し、コマンドモードに戻ります。

SIGHUP
: バッファが空でなく、最後の書き込み処理以降に変更されている場合、 ed ユーティリティはファイル中にバッファのコピーを生成しようとします。 その対象ファイルとして、まず現在のディレクトリ中の ed.hup というファイルが選ばれます。それが失敗すると、環境変数 HOME が示すディレクトリ中の ed.hup というファイルが選ばれます。 いずれの場合も ed は、コマンドモードに戻らないで終了します。

いくつかのサイズ制限があります。 1 行は 512 文字以下、グローバル・コマンドリストは 256 文字以下、ファイルのパス名は 255 文字以下です (スラッシュを含む) 。 行数の制限はユーザーのメモリ容量によって異なります。 1 行には 1 ワード必要です。

ファイルを読むとき、 ed は ASCII と NULL 文字を破棄します。

ファイルが復帰改行文字で終了していないとき、 ed はそれを追加して、その旨を説明するメッセージを表示します。

正規表現 RE または置換文字列の終端区切り文字 (たとえば、/) が復帰改行文字の直前の文字のとき、区切り文字は省略できます。 いずれの場合もアドレスされた行が出力されます。 次の各組のコマンドは同じものと見なされます。

```
s/s1/s2         s/s1/s2/p

g/s1            g/s1/p

?s1             ?s1?
```

不正なコマンドが投入されると、 ed は以下の文字列を標準出力に書き出します。

```
"?\ n"
```

このとき H コマンドにより「ヘルプモード」が有効になっていれば、 状況を説明するメッセージが付加されます。 上記文字列出力後、 ed はコマンドモードを継続します。現在の行番号は変わりません。

オプション

`-C` 暗号化オプション。 基本的に、 -x オプションと同じです。ただし、 ed は、 C コマンドをシミュレートします。 C コマンドは、基本的に、 X コマンドと同じです。ただし、 読み込まれたすべてのテキストが暗号化されていると見なされます。

`-p` *string* ユーザーがプロンプト文字列を指定するのを許可します。 デフォルトではプロンプト文字列はありません。

`-s` | `-` e 、 r 、および w コマンドによる文字カウント、 e および q コマンドからの診断、および !*shell command* の後の ! プロンプトを出力しません。

`-x` 暗号化オプション。 ed は、 X コマンドをシミュレートして、ユーザーにキーの入力を要求してきます。 X コマンドは読み込まれたテキストが 暗号化されているか否かを判定するのに高度な推測を行います。 一時バッファファイルも、 -x オプションで入力したキーを変形したバージョンを用いて暗号化されます。 本マニュアルページの最後の節「注意事項」 も参照してください。

オペランド

以下のオペランドを指定できます。

file この file 引数を指定すると、 ed は、標準入力からコマンドを読み込む前に、パス名 file で示されるファイルに e コマンドを適用したかのように動作します。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の ed と red の動作については、largefile(5) を参照してください。

環境

ed の実行に影響を与える環境変数 HOME、 LC_CTYPE、 LC_COLLATE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	ファイルにもコマンドにもエラーはなく、正常終了した
>0	エラーが発生した

ファイル

$TMPDIR	この環境変数が NULL でない場合、 その値は一時作業ファイルのディレクトリ名として /var/tmp の代わりに使用されます。
/var/tmp	/var/tmp が存在する場合、一時作業ファイルのディレクトリ名として使用されます。
/tmp	環境変数 TMPDIR が存在しないか NULL で、さらに /var/tmp が存在しない場合、 /tmp が一時作業ファイルのディレクトリ名として使用されます。
ed.hup	端末がハングアップしたとき、作業ファイルがここに保存されます。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/red

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/ed

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

bfs(1), edit(1), ex(1), grep(1), ksh(1), sed(1), sh(1), stty(1), umask(1), vi(1), fspec(4), attributes(5), environ(5), largefile(5), regex(5), XPG4(5)

診断

?	コマンドエラー
?file	アクセス不可能なファイル (詳細については help および Help コマンドを使用してください)。

最後に w コマンドを使用して全バッファを書き出した後に バッファの内容が変更された場合、 e または q コマンドによってバッファの内容を破棄しようとすると、 ed はユーザーに対して 警告を発し、 ? を出力して編集を続行するか否かを聞いてきます。ここで再度 e または q コマンドを入力すると、これらのコマンドは実行されます。-s コマンド行オプションは、上述の機能を禁止します。

注意事項

– オプションは、サポートはされていますが、ドキュメント内では コマンド構文規格に準拠する -s オプションで置き換えました (intro(1) 参照) 。

g または v コマンドに対して ! コマンドは無効です。

! コマンドと e 、 r 、および w コマンドからのエスケープコマンド ! は、エディタが制限付きシェル (sh(1) 参照) から起動されているときには使用できません。

正規表現 RE 内の \ n シーケンスは復帰改行文字と一致しません。

エディタの入力がコマンドファイル (たとえば、 ed file < *ed_cmd_file*) からの場合、最初にエラーが発生した時点でエディタは終了します。

名前 | regcmp – 正規表現のコンパイル

形式 | **regcmp** [-] *filename...*

機能説明 | regcmp コマンドは、 regcmp 関数と同様の機能を果たします。これにより C プログラムでの regcmp 関数の呼び出しがほとんど必要なくなります。 regcmp 関数をバイパスすることで、実行時間とプログラムサイズの両方を節約できます。 regcmp コマンド は、 *filename* 内の正規表現をコンパイルし、その結果を *filename*.i に出力します。

オプション | - – オプションが使用されている場合、その結果は *filename*.c に出力されます。 *filename* 内のエントリの形式は、名前(C 変数)、その後ろに 1 つ以上の空白文字をあけて、二重引用符に囲まれた 1 つ以上の正規表現が続きます。 regcmp の出力は、 C ソースコードです。 コンパイルされた正規表現は、 extern char ベクトルとして表されます。したがって、 *filename*.i ファイルが C プログラムに #include されることも、あるいは *filename*.c ファイルがコンパイルされ、その後ロードされることもあります。 regcmp 出力を使用する C プログラムでは、 regex(abc,line) は abc という名の正規表現を line に適用します。 診断は、自明です。

使用例 | 例 **1** regcmp コマンドの例

```
name    "([A-Za-z][A-Za-z0-9_]*)$0"
telno   " \({0,1}([2-9][01][1-9])$0\){0,1} *"
        "([2-9][0-9]{2})$1[ -]{0,1}"
        "([0-9]{4})$2"
```

上記の telno に対する 3 つの引数はすべて、 1 行に入力しなければなりません。

regcmp 出力を使用する C プログラムでは、

```
regex(telno, line, area, exch, rest)
```

は telno という名の正規表現を line に適用します。

環境 | LC_* 環境変数の使用法の一般的な説明は environ(5) を参照してください。

LC_CTYPE – regcmp の文字の処理方法を決定します。 LC_CTYPE に有効な値が設定されていると、 regcmp は、その ロケール にあった文字を含むテキストやファイル名を表示および処理できます。

LC_MESSAGES – 診断メッセージや情報メッセージの表示方法を決定します。 また、メッセージの言語とスタイル、 そして肯定応答および否定応答の正しい形も決定します。 C ロケール においては、 メッセージはプログラム自身が使用しているデフォルトの形で表示されます (通常、米語)。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWtoo
CSI	対応済み

関連項目 `regcmp`(3C), `attributes`(5), `environ`(5)

名前 | hash, rehash, unhash, hashstat – ディレクトリの内容の内部ハッシュテーブルの評価

形式 | **/usr/bin/hash** [*utility*]

/usr/bin/hash [-r]

sh | **hash** [-r] [*name*...]

csh | **rehash**

unhash

hashstat

ksh | **hash** [*name*...]

/usr/bin/hash | /usr/bin/hash ユーティリティは、見つかったユーティリティの位置を現在のシェル環境が どのように記憶するか、その記憶方法を変更します。 具体的には、指定された引数に従って、新たなユーティリティを ユーティリティ位置リストに追加したり、リストの内容を消去したりします。引数を指定しないと、リストの内容が報告されます。

シェルの組み込みユーティリティとして提供されているものについては、 hash は報告対象としません。

sh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。-r オプションを指定すると、シェルは記憶したすべての位置を破棄します。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。 この出力表示において、 *Hits* の列はシェルプロセスによってコマンドが呼び出された回数を表します。*Cost* は、検索パスのコマンドを見つけるのに必要な作業量です。 コマンドが検索パスの「相対」ディレクトリにある場合、 そのディレクトリの変更後にそのコマンドが格納された位置が再計算されます。 この再計算が行われる予定のコマンドに対しては、*Hits* 情報の隣にアスタリスク (*) が示されます。 *Cost* の値は、再計算が行われるたびに増加します。

csh | rehash は新しく追加されたコマンドに合わせて、 path 環境の変数に記憶しているディレクトリの内容の内部ハッシュテーブルを再計算します。

unhash は内部ハッシュテーブルを使用不能にします。

hashstat は内部ハッシュテーブルがコマンドの検索 (実行を伴わない) にどの程度有効に働いて いるかを示す統計情報を出力します。 ハッシュ関数がヒットの可能性を示しているパス の各構成要素と、' / ' で始まらない各構成要素について、実行しようとします。

ksh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。引数が与えられない場合、hash は記憶されたコマンドに関する情報を表示します。

オペランド | 以下のオペランドを指定できます。

utility | 位置のリスト内で検索する、あるいはリストに 追加するユーティリティ名。

出力

引数を 1 つも指定しないと、hash の標準出力が使用されます。その形式は決まっていませんが、 現在のシェル環境用の位置リストにある各ユーティリティの パス名は含まれています。このリストは、以前に実行された hash 呼び出し中に指定されていたユーティリティをすべて含んでおり、さらに通常のコマンド検索プロセスで呼び出され見つけられた ユーティリティを含んでいることもあります。

環境

hash の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、 NLSPATH についての詳細は、environ(5) を参照してください。

PATH　　*utility* の位置を指定します。

終了ステータス

以下の終了ステータスが返されます。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), sh(1), attributes(5), environ(5)

名前 | rsh, remsh, remote_shell – リモートシェル

形式 | **rsh** [-n] [-l *username*] *hostname command*

rsh *hostname* [-n] [-l *username*] *command*

remsh [-n] [-l *username*] *hostname command*

remsh *hostname* [-n] [-l *username*] *command*

hostname [-n] [-l *username*] *command*

機能説明 | rsh は、 *hostname* が示すホストに接続し、 *command* が示すコマンドを実行します。 rsh は自身の標準入力のデータをリモートコマンドにコピーし、 リモートコマンドの標準出力を自身の標準出力にコピーし、 さらに、リモートコマンドの標準エラー出力を 自身の標準エラー出力にコピーします。 割り込み、停止、および終了シグナルは、 リモートコマンドに伝えられます。通常 rsh は、リモートコマンドが終了したときに終了します。

command を省略すると、 rsh は単一のコマンドを実行する代わりに、 rlogin(1) を使ってそのユーザーをリモートホストにログインします。

rsh は *command* の終了ステータスを返しません。

シェルのメタキャラクタのうち、 引用符で囲まれていないものはローカルマシン上で解釈されます。 引用符で囲まれているものは、リモートマシン上で解釈されます。「使用例」の項にある例を参照してください。

特定のユーザーのログインシェルの初期化ファイル (.cshrc など) でロケールが設定されていない場合、rsh はコマンドを実行するときに、常に C ロケールを使用します。リモートマシンのデフォルトのロケールは使用しません。

オプション | 以下のオプションを指定できます。

-l *username* — リモートユーザー名として、 ユーザー自身のローカルユーザー名の代わりに *username* を使用します。このオプションを省略すると、 リモートユーザー名はローカルユーザー名と同じになります。

-n — rsh の入力先を /dev/null に変更します。このオプションは、rsh とそれを呼び出したシェルとの間での、 予期できない干渉を防ぐ上で便利です。 たとえば、 rsh を実行していて、さらにバックグラウンドで rsh を呼び出した場合、その入力先を端末以外に変更しないと、 リモートコマンドの読み取りがない場合でも処理が妨げられてしまいます。 -n オプションを指定すれば、このような事態は避けられます。

リモートシェルの種類 (sh や rsh など) は、リモートシステム上の /etc/passwd ファイル中のユーザーのエントリにより決められます。

オペランド | 以下のオペランドを指定できます。

command — 指定された *hostname* で実行するコマンド

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の rsh と remsh の動作については、 largefile(5) を参照してください。

rsh および remsh コマンドは、IPv6 に対応しています。ip6(7P) のマニュアルページを参照してください。

ホスト名は *hosts* データベース (/etc/hosts ファイルに含めることができる)、 インターネットドメイン名データベース、 あるいはその両方に書かれています。 各ホストには 1 つの正式名 (データベースエントリの最初の名前) があり、 さらにいくつかのニックネームが存在することもあります。 *hostname* 引数には、正式なホスト名とニックネームの いずれかを指定します。

rsh を実行するファイル名が rsh でなければ、 rsh はそのファイル名を *hostname* 引数として使用します。これによりユーザーは、ホスト名で rsh に対するシンボリックリンクを生成でき、 実行時にそのホスト上のリモートシェルを呼び出せます。ディレクトリを作成し、それを共通に使われる ホスト名でシンボリックリンクにつなぎ、 さらにそのディレクトリをユーザー自身の シェルの検索パスに含めることにより、シェルに *hostname* を入力するだけで rsh を実行することができます。

rsh を remsh で呼び出した場合、 rsh は /usr/bin/remsh の存在を確認します。このファイルが存在 する場合、 rsh は remsh を rsh の別名として処理します。/usr/bin/remsh が存在しない場合、 rsh は remsh をホスト名として処理します。

各リモートマシンには /etc/hosts.equiv という名のファイルが存在することがあります。 このファイルには、そのマシンとユーザー名を共有する ホスト名の一覧が記録されています。 ローカルマシン上とリモートマシン上での ユーザー名が同一のユーザーは、 リモートマシンの /etc/hosts ファイルにリストされているマシンから rsh を実行することができます。 個々のユーザーは、このような同等名リストを個人用の .rhosts ファイルとして、自身のホームディレクトリに 作成することができます。 このファイル中の各行には 2 つの名前、 *hostname* と *username* が含まれ、両者は空白で区切られます。 *username* で示すユーザーが *hostname* で示すホストにログインしていれば、そのユーザーは rsh を使って、リモートユーザーとしてリモートマシンにアクセスできます。 ローカルホスト名がリモートマシン上の /etc/hosts.equiv ファイル中に見つからず、ローカルのユーザー名とホスト名がリモートユーザーの .rhosts ファイル中に見つからない場合、アクセスは拒否されます。 /etc/hosts.equiv または .rhosts ファイルに記録されているホスト名は、 hosts データベースに登録されている 正式なホスト名である必要があります。つまりこの両ファイル中には、 ニックネームは指定できません。

リモートマシン上でアクセスが拒否されたときは、 *command* 引数が省略されていない限り、 rsh はパスワードの入力を要求するプロンプトを出力しません。

使用例

例 **1** rsh でファイルを追加する

次のコマンドは、lizard というマシンからファイル lizard.file というリモートファイルを、example というマシン上の example.file に付加します。

```
example% rsh lizard cat lizard.file >> example.file
```

例 1 rsh でファイルを追加する　(続き)

一方、次のコマンドは、 lizard というマシン上の lizard.file というファイルを、同じマシン上の lizard.file2 ファイルに付加します。

```
example% rsh lizard cat lizard.file ">>" lizard.file2
```

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

1　エラーが発生した

ファイル

/etc/hosts	インターネットホストテーブル
/etc/hosts.equiv	信頼性のあるリモートホストとユーザー
/etc/passwd	システムパスワードファイル

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

on(1), rlogin(1), telnet(1), vi(1), in.named(1M), hosts(4), hosts.equiv(4), ipnodes(4), attributes(5), largefile(5), ip6(7P)

注意事項

hosts.equiv に登録されているシステムのセキュリティは、 少なくともローカルシステムのセキュリティと 同レベルである必要があります。 セキュリティレベルの低いシステムが hosts.equiv 中に 1 つでも存在していると、 システム全体のセキュリティが損なわれる可能性があります。

vi(1) のような対話型コマンドは実行できません。 対話型コマンドは rlogin を使って実行してください。

停止シグナルは、ローカルの rsh プロセスだけを停止させます。 これはバグだという議論があるかもしれませんが、 現在のところ修正が大変困難です。 その理由は複雑なので、ここでは説明を省きます。

以下を実行すると、シェルの状態がおかしくなります。

```
example% rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf -
```

つまり rsh の前に tar が終了してしまい、そのあとで rsh が「壊れたパイプ」にデータを書き出そうとします。 そのため正常に終了するのではなく、 標準入力をアクセスしようとしてシェルと競合してしまいます。

-n オプションを指定して rsh を実行すれば、このような事態を防ぐことができます。

```
example% tar cf - . | rsh sundial dd of=/dev/rmt0 obs=20b
```

この場合には、前述のような現象は起こりません。 この場合に -n オプションを指定すると、 rsh はパイプから読み込む代わりに、誤って /dev/null からの読み込みを試みます。

名前 | rsh, remsh, remote_shell – リモートシェル

形式 | **rsh** [-n] [-l *username*] *hostname command*

rsh *hostname* [-n] [-l *username*] *command*

remsh [-n] [-l *username*] *hostname command*

remsh *hostname* [-n] [-l *username*] *command*

hostname [-n] [-l *username*] *command*

機能説明 | rsh は、 *hostname* が示すホストに接続し、 *command* が示すコマンドを実行します。rsh は自身の標準入力のデータをリモートコマンドにコピーし、 リモートコマンドの標準出力を自身の標準出力にコピーし、 さらに、リモートコマンドの標準エラー出力を 自身の標準エラー出力にコピーします。 割り込み、停止、および終了シグナルは、 リモートコマンドに伝えられます。通常 rsh は、リモートコマンドが終了したときに終了します。

command を省略すると、 rsh は単一のコマンドを実行する代わりに、 rlogin(1) を使ってそのユーザーをリモートホストにログインします。

rsh は *command* の終了ステータスを返しません。

シェルのメタキャラクタのうち、 引用符で囲まれていないものはローカルマシン上で解釈されます。 引用符で囲まれているものは、リモートマシン上で解釈されます。「使用例」の項にある例を参照してください。

特定のユーザーのログインシェルの初期化ファイル (.cshrc など) でロケールが設定されていない場合、rsh はコマンドを実行するときに、常に C ロケールを使用します。リモートマシンのデフォルトのロケールは使用しません。

オプション | 以下のオプションを指定できます。

-l *username* — リモートユーザー名として、 ユーザー自身のローカルユーザー名の代わりに *username* を使用します。このオプションを省略すると、 リモートユーザー名はローカルユーザー名と同じになります。

-n — rsh の入力先を /dev/null に変更します。このオプションは、rsh とそれを呼び出したシェルとの間での、 予期できない干渉を防ぐ上で便利です。 たとえば、 rsh を実行していて、さらにバックグラウンドで rsh を呼び出した場合、その入力先を端末以外に変更しないと、 リモートコマンドの読み取りがない場合でも処理が妨げられてしまいます。 -n オプションを指定すれば、このような事態は避けられます。

リモートシェルの種類 (sh や rsh など) は、リモートシステム上の /etc/passwd ファイル中のユーザーのエントリにより決められます。

オペランド | 以下のオペランドを指定できます。

command — 指定された *hostname* で実行するコマンド

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の rsh と remsh の動作については、 largefile(5) を参照してください。

rsh および remsh コマンドは、IPv6 に対応しています。ip6(7P) のマニュアルページを参照してください。

ホスト名は *hosts* データベース (/etc/hosts ファイルに含めることができる)、 インターネットドメイン名データベース、 あるいはその両方に書かれています。 各ホストには 1 つの正式名 (データベースエントリの最初の名前) があり、 さらにいくつかのニックネームが存在することもあります。 *hostname* 引数には、正式なホスト名とニックネームの いずれかを指定します。

rsh を実行するファイル名が rsh でなければ、 rsh はそのファイル名を *hostname* 引数として使用します。これによりユーザーは、ホスト名で rsh に対するシンボリックリンクを生成でき、 実行時にそのホスト上のリモートシェルを呼び出せます。ディレクトリを作成し、それを共通に使われる ホスト名でシンボリックリンクにつなぎ、 さらにそのディレクトリをユーザー自身の シェルの検索パスに含めることにより、シェルに *hostname* を入力するだけで rsh を実行することができます。

rsh を remsh で呼び出した場合、 rsh は /usr/bin/remsh の存在を確認します。このファイルが存在 する場合、 rsh は remsh を rsh の別名として処理します。/usr/bin/remsh が存在しない場合、 rsh は remsh をホスト名として処理します。

各リモートマシンには /etc/hosts.equiv という名のファイルが存在することがあります。 このファイルには、そのマシンとユーザー名を共有する ホスト名の一覧が記録されています。 ローカルマシン上とリモートマシン上での ユーザー名が同一のユーザーは、 リモートマシンの /etc/hosts ファイルにリストされているマシンから rsh を実行することができます。 個々のユーザーは、このような同等名リストを個人用の .rhosts ファイルとして、自身のホームディレクトリに 作成することができます。 このファイル中の各行には 2 つの名前、 *hostname* と *username* が含まれ、両者は空白で区切られます。 *username* で示すユーザーが *hostname* で示すホストにログインしていれば、そのユーザーは rsh を使って、リモートユーザーとしてリモートマシンにアクセスできます。 ローカルホスト名がリモートマシン上の /etc/hosts.equiv ファイル中に見つからず、ローカルのユーザー名とホスト名がリモートユーザーの .rhosts ファイル中に見つからない場合、アクセスは拒否されます。 /etc/hosts.equiv または .rhosts ファイルに記録されているホスト名は、 hosts データベースに登録されている 正式なホスト名である必要があります。つまりこの両ファイル中には、 ニックネームは指定できません。

リモートマシン上でアクセスが拒否されたときは、 *command* 引数が省略されていない限り、 rsh はパスワードの入力を要求するプロンプトを出力しません。

使用例

例 **1** rsh でファイルを追加する

次のコマンドは、lizard というマシンからファイル lizard.file というリモートファイルを、example というマシン上の example.file に付加します。

```
example% rsh lizard cat lizard.file >> example.file
```

例 1 rsh でファイルを追加する　(続き)

一方、次のコマンドは、 lizard というマシン上の lizard.file というファイルを、同じマシン上の lizard.file2 ファイルに付加します。

```
example% rsh lizard cat lizard.file ">>" lizard.file2
```

終了ステータス

以下の終了ステータスが返されます。

0　　正常終了

1　　エラーが発生した

ファイル

/etc/hosts	インターネットホストテーブル
/etc/hosts.equiv	信頼性のあるリモートホストとユーザー
/etc/passwd	システムパスワードファイル

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

on(1), rlogin(1), telnet(1), vi(1), in.named(1M), hosts(4), hosts.equiv(4), ipnodes(4), attributes(5), largefile(5), ip6(7P)

注意事項

hosts.equiv に登録されているシステムのセキュリティは、 少なくともローカルシステムのセキュリティと 同レベルである必要があります。 セキュリティレベルの低いシステムが hosts.equiv 中に 1 つでも存在していると、 システム全体のセキュリティが損なわれる可能性があります。

vi(1) のような対話型コマンドは実行できません。 対話型コマンドは rlogin を使って実行してください。

停止シグナルは、ローカルの rsh プロセスだけを停止させます。 これはバグだという議論があるかもしれませんが、 現在のところ修正が大変困難です。 その理由は複雑なので、ここでは説明を省きます。

以下を実行すると、シェルの状態がおかしくなります。

```
example% rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf -
```

つまり rsh の前に tar が終了してしまい、そのあとで rsh が「壊れたパイプ」にデータを書き出そうとします。 そのため正常に終了するのではなく、 標準入力をアクセスしようとしてシェルと競合してしまいます。

-n オプションを指定して rsh を実行すれば、このような事態を防ぐことができます。

```
example% tar cf - . | rsh sundial dd of=/dev/rmt0 obs=20b
```

この場合には、前述のような現象は起こりません。 この場合に -n オプションを指定すると、 rsh はパイプから読み込む代わりに、誤って /dev/null からの読み込みを試みます。

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | exit, return, goto – シェルの連続した処理を分岐して実行するためのシェル組み込み関数

sh | **exit** [*n*]

return [*n*]

csh | **exit** [(*expr*)]

goto *label*

ksh | ***exit** [*n*]

***return** [*n*]

sh | `exit` はシェルまたはシェルスクリプトを *n* で指定した終了状態で終了させます。 *n* を省略すると、 最後に実行されたコマンドの終了状態 がシェルの終了状態になります。 ファイルの終わりを検出した場合もシェルが終了します。

`return` は関数を、 *n* が示す戻り値で終了させます。 *n* を省略すると、戻り値は最後に実行された コマンドの終了状態になります。

csh | `exit` はシェルまたはシェルスクリプトを終了させ、 状態変数の値または式 *expr* で指定された値が返されます。

`goto` 組み込み関数は *label* をコマンド中で検索の引数として指定します。 シェルは可能なかぎり入力をさかのぼり、 *label*: という形式の行を探します。 *label*: の前には空白文字またはタブ文字がある可能性もあります。 指定された行の次から実行が再開します。 `while` または `for` 組み込みコマンドと、対応する `end` との間にあるラベルへジャンプするとエラーになります。

ksh | `exit` はシェルまたはシェルスクリプトを *n* で指定した終了状態で終了させます。 具体的には、指定した値の最下位 8 ビットが終了状態の値となります。 *n* を省略すると、 最後に実行されたコマンドの終了状態 がシェルの終了状態になります。 トラップ実行中に `exit` が発生した場合、ここで言う最後に実行されたコマンドとは、トラップ呼び出し直前に実行されたコマンドを指します。 なお、 `ignoreeof` オプション (後述の `set` を参照) が有効になっているシェルを除き、 ファイルの終わりを検出した場合もシェルが終了します。

`return` はシェル関数またはドット (.) スクリプトを、 *n* で指定された戻り値で 呼び出し側スクリプトに戻します。 *n* で指定した値の最下位 8 ビットが戻り値となります。 *n* を省略すると、 戻り値は最後に実行された コマンドの戻り値になります。 関数やドット (.) スクリプト実行中以外で `return` を起動すると、 結果は `exit` と同一になります。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている `ksh`(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。

3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

break(1), csh(1), ksh(1), sh(1), attributes(5)

名前 | ksh, rksh – Korn シェル。標準/制限付きコマンドとプログラミング言語

形式 | **/usr/bin/ksh** [± abCefhikmnoprstuvx] [± o *option*...] [*arg*...]

/usr/bin/ksh -c [± abCefhikmnoprstuvx] [± o *option*...] *command_string* [*command_name* [*arg*...]]

/usr/xpg4/bin/sh [± abCefhikmnoprstuvx] [± o *option*...] [*arg*...]

/usr/xpg4/bin/sh -c [± abCefhikmnoprstuvx] [± o *option*...] *command_string* [*command_name* [*arg*...]]

/usr/bin/rksh [± abCefhikmnoprstuvx] [± o *option*...] [*arg*...]

/usr/bin/rksh -c [± abCefhikmnoprstuvx] [± o *option*...] *command_string* [*command_name* [*arg*...]]

機能説明 | /usr/xpg4/bin/sh ユーティリティは標準に準拠したシェルです。このユーティリティは /usr/bin/ksh のすべての機能を提供します。ただし、後述するように、動作が異なる場合を除きます。詳細については、「算術展開」の項を参照してください。

/usr/bin/ksh は、端末またはファイルから読み取られたコマンドを 実行するコマンドおよびプログラミング言語です。rksh は、標準コマンドインタプリタである ksh と比べて、機能の一部が制限されており、標準シェルよりも制限された機能を持った ログイン名や実行環境を設定します。シェルへの引数の意味については、後述の「呼び出し」を参照してください。

定義 | メタキャラクタ (*metacharacter*) は、次の文字のいずれかです。

; & ( ) | < > 復帰改行(NEWLINE) 空白文字 タブ

ブランク (*blank*) とは、タブ (TAB) または空白文字 (SPACE) のことです。識別子 (*identifier*) とは、英文字、数字、または下線の並びのことで、その先頭の文字は英文字または下線です。識別子は関数 (*function*) や変数 (*variable*) の名前として使用します。ワード (*word*) とは、引用符がつかない 1 つまたは複数のメタキャラクタで区切られた、文字の並びのことです。

コマンド (*command*) とは、シェル言語の文法にそった文字の並びのことです。シェルは各コマンドを読み取り、指定された動作を直接実行するか、または動作を実行するユーティリティを起動します。特殊コマンド (*special-command*) とは、個別のプロセスを作成しなくても、シェルが実行してくれるコマンドです。本書で述べる副作用が発生する場合を除き、ほとんどの特殊コマンドは それぞれ個別のユーティリティとして実装できます。

コマンド | 単純コマンド (*simple-command*) は、ブランクで区切られたワードの並びで、その前に変数代入リストを指定できます (後述の「環境」を参照) 。先頭のワードは、実行するコマンド名を指定します。残りのワードは、以下に述べる場合を除き、呼び出されたコマンドに引数として渡されます。コマンド名は引数 0 として渡されます (exec(2) を参照) 。単純コマンドの値 (*value*) は、正常終了した場合は終了状態の値、シグナルを

受け取って異常終了した場合は、シグナル番号に 128 を足した値になります。シグナルの値については、signal(3HEAD) のリストを参照してください。なお、正常な終了状態の値 129 ～ 255 と、シグナル番号 1 ～ 127 を受け取って異常終了した場合の値を見分けることはできません。

パイプライン (*pipeline*) は、パイプ (|) で区切られた 1 つ以上のコマンドの並びです。最後のコマンドを除き、各コマンドの標準出力は pipe(2) によってその次のコマンドの標準入力と結合されます。各コマンドは、別々のプロセスとして実行されます。シェルは最後のコマンドが終了するのを待ちます。最後のコマンドの終了状態がパイプライン全体の終了状態となります。

リスト (*list*) は、;、&、&&、または | | で区切られた 1 つ以上のパイプラインの並びです。その並びの終わりに ;、&、または |& を記述することもできます。この 5 つの記号の中で、;、&、および |& の優先度は同じで、&& と | | の優先度より低くなります。&& と | | の優先度は同じです。セミコロン (;) によって、直前のパイプラインが順次実行されます。アンパサンド記号 (&&) によって、直前のパイプラインが非同期的に実行されます (つまりシェルはパイプラインが終了するのを待ちません)。|& という記号によって、親シェルに対して双方向パイプが確立された、直前のコマンドまたはパイプラインが非同期的に実行されます。

生成されたコマンドの標準入出力は、親シェルが特殊コマンドの read および print (後述「特殊コマンド」を参照) の -p オプションを使用して書き込み、読み取ることができます。&& (または | |) という記号によって、直前のパイプラインが 0 (またはゼロ以外) の終了ステータスを返した場合にだけ、その後のリストが実行されます。コマンドの区切りとして、セミコロンの代わりに任意の数の復帰改行を *list* に指定できます。

コマンドは、単純コマンドまたは以下のいずれかです。特に断わりのないかぎり、コマンドが返す値は、そのコマンド中で最後に実行された単純コマンドの値です。

for *identifier* [in *word* . . .] ; do *list* ; done
: for コマンドが実行されるたびに、*identifier* は in *word* リストから次に得られる *word* に設定されます。in *word* . . . を省略すると、for コマンドは、設定された各定位値パラメータに対して、do *list* を 1 回実行します (後述の「パラメータ置換」を参照)。in *word* リストの *word* がなくなると、実行は終了します。

select *identifier* [in *word* . . .] ; do *list* ; done
: select コマンドは、標準エラー (ファイル記述子 2) に、一群のワードを各々の前に番号を付けて出力します。in *word* . . . を省略すると、定位値パラメータが使用されます (後述の「パラメータ置換」を参照)。PS3 プロンプトが出力され、標準入力から行が読み取られます。この行が、リストに示された *word* のいずれかの番号からなる場合、*identifier* が示す変数の値はこの番号に該当する *word* に設定されます。この行が空の場合は、再び選択リストを出力します。空でない場合は、*identifier* 変数の値を NULL に設定します (後述の「ブランクの解釈」を参照)。標準入力から読み取った行の内容は、REPLY というシェル変数に保存します。break またはファイルの終わり (EOF) に行き当たるまで、選択が発生するたびに *list* が実行されます。*list* の実行によって REPLY 変数が NULL に設定されると、2 番目の選択をプロンプトする PS3 の表示前に選択リストが出力されます。

`case` *word* `in` [*pattern* [| *pattern*]) *list* `;;`] ... `esac`
`case` コマンドは、*word* に一致する最初の *pattern* に対応した *list* を実行します。*pattern* の形式は、ファイル名生成に使用される形式と同じです (「ファイル名生成」の項を参照) 。

`if` *list* `;` `then` *list* `;` [`elif` *list* `;` `then` *list* `; ...`] [`else` *list* `;`] `fi`
`if` の後の *list* を実行後、*list* が `0` の終了ステータスを返すと、最初の `then` の後の *list* を実行します。それ以外の場合、`elif` の後の *list* を実行します。この値が `0` の場合、2 番目の `then` の後の *list* を実行します。これが失敗すると、`else` *list* を実行します。`else` *list* も `then` *list* も実行しない場合、`if` コマンドは `0` の終了ステータスを返します。

`while` *list* `;` `do` *list* `;` `done`
`until` *list* `;` `do` *list* `;` `done`
`while` コマンドは、`while` *list* を繰り返し実行し、*list* 中の最後のコマンドの終了ステータスが `0` の場合、`do` *list* を実行します。それ以外の場合、ループは終了します。`do` *list* 中のコマンドを実行しない場合、`while` コマンドは `0` の終了ステータスを返します。ループ終了条件の判定を逆にするには、`while` の代わりに `until` を使用します。

(*list*)
別の環境で *list* を実行します。なお、入れ子において 2 つの開いた括弧を連続して記述する場合、後述の算術評価を避けるために空白を 挿入する必要があります。

{ *list* }
単に *list* を実行します。なお、メタキャラクタの (と) とは異なり、{ と } は「予約語」なので、認識されるためには行の始めまたは ; の後に現れる必要があります。

`[[` *expression* `]]`
expression が示す式を評価し、その値 が真のとき `0` の終了ステータスを返します。*expression* の説明については、後述の「条件式」を参照してください。

`function` *identifier* { *list* `;` }
identifier `(` `)` { *list* `;` }
identifier で参照される関数を定義します。{ と } の間のコマンド群 (*list*) が関数の本体となります (後述の「関数」参照) 。

`time` *pipeline*
pipeline を実行し、標準エラーに経過時間、ユーザー時間、およびシステム時間を出力します。

下記のワードは、コマンドの最初に現れたとき、および引用符をつけずに記述されたときに 「予約語」として認識されます。

```
!          if       then     else    elif    fi      case
esac       for      while    until   do      done    {   }
function   select   time     [[  ]]
```

コメント

でワードを始めると、そのワードおよび以降の 復帰改行までの文字がすべて無視されます。

別名

各コマンドの最初のワードに別名 (alias) が定義されている場合、そのワードは別名のテキストに置き換えられます。別名は任意の数の文字で構成されます。別名に使用できない文字は、メタキャラクタ、引用符、ファイル展開文字、パラメータ置換文字、コマンド置換文字、= です。代入する文字列としては、前述のメタキャラクタを含む有効な シェルスクリプトを指定できます。置換されたテキスト内にある各コマンドの最初のワードは、置換対象のものを除き、別名についてチェックされます。別名の最後の文字がブランクの場合、別名の後のワードも別名置換についてチェックされます。別名を使用すれば、特殊組み込み型コマンドを再定義できますが、前述の予約語の再定義はできません。別名は `alias` コマンドで作成、表示、エクスポートでき、`unalias` コマンドで削除できます。エクスポートされた別名は、名前指定で起動されたスクリプトに対して引き続き有効ですが、シェルを起動 (後述の「呼び出し」を参照) するたびに初期化し直す必要があります。再帰的に別名をつける際に無限ループの発生を防ぐため、シェルが現在その同じ名前の別名を処理中でなければ、ワードは別名の値に置換されます。処理中であれば置換されません。

別名化はスクリプト読み取り時に実行されますが、スクリプト実行中には行われません。したがって別名を有効にするには、別名を参照するコマンドの読み取り前に `alias` コマンドで定義しておく必要があります。

別名は、完全パス名の省略形としてよく使用されます。別名機能のオプションを使えば、別名の値を、該当するコマンドの完全パス名に自動的に設定できます。これらの別名を検索済み (*tracked*) 別名と呼びます。検索済み別名の値は、対応するコマンドを最初に検索するときに定義され、`PATH` 変数を再設定するたびに未定義になります。これらの別名は検索済みのままとなり、次回の参照時に値が再定義されます。複数の検索済み別名がシェル中にコンパイルされます。`set` コマンドの `-h` オプションは、各々の参照されたコマンド名を 検索済み別名にします。

以下に示す「エクスポート済み別名」 はシェルにコンパイルされ組み込まれますが、設定解除または再定義が可能です。

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

後方のブランク文字と予約語に関する例を示します。ユーザーが次のように入力したとします。

```
$ alias foo="/bin/ls "
$ alias while="/"
```

ここで次のコマンドを実行します。

```
$ while true
> do
```

```
> echo "Hello, World"
> done
```

すると Hello, World という文字列が画面上に無限に現れます。一方、次のように入力すると、結果は / の ls 出力となります。

```
$ foo while
```

foo に対する別名置換は空白文字で終わりとなるため、次のワードが置換用にチェックされます。次のワード while も別名化されているので、置換が行われます。つまり while の位置がコマンド名として正しくないために、予約語とは認識されないのです。

以下のように入力すれば、while は通常どおり予約語と見なされます。

```
$ foo; while
```

チルド置換

別名置換を実行すると、各ワードは引用符なしの ~ で始まっているかどうかチェックされます。結果が真なら、/ までのワードに一致するユーザー名が 存在するかどうかチェックされます。存在すれば、~ および一致したワード (ログイン名) は、一致したユーザーのログインディレクトリに置き換えられます。これをチルド置換と呼びます。一致するユーザー名が見つからない場合、元のテキストは変更を受けません。~ が単独で指定された場合、または後ろに / を伴って指定された場合には、$HOME に置き換えられます。~ のあとに + または - を指定すると、それぞれ $PWD または $OLDPWD に置き換えられます。

また変数に代入する値が ~ で始まるときにも、チルド置換が試されます。

チルド展開

チルド接頭辞 (*tilde-prefix*) はワードの先頭の引用符なしのチルド文字と、それに続く文字列からなります。この文字列とは、ワード中に引用符がつかないスラッシュが 含まれていれば、その最初のスラッシュ直前までのすべての文字です。スラッシュが含まれていなければ、ワード中のすべての文字です。代入においては複数のチルド接頭辞を使用できます。つまりワードの先頭 (代入を示す等号の直後) 、引用符がつかない コロンのあと、およびその両方にも指定できます。代入におけるチルド接頭辞の終わりは、引用符がつかないコロンまたは スラッシュが最初に現れた地点です。チルド接頭辞中に引用符付きの文字が 1 つもなければ、先頭のチルドを除いた部分文字列は、ユーザーデータベース中に 登録されているログイン名の可能性があると見なされます。

移植性のあるログイン名には、環境変数 LOGNAME の項に記載されている文字以外の文字を含めることはできません。ログイン名が NULL のとき、すなわちチルド接頭辞がチルドだけからなるとき、チルド接頭辞は変数 HOME の値に置き換えられます。HOME が設定されていないと、その結果は予測できません。ログイン名が NULL でなければ、チルド接頭辞は getpwnam 関数を使って得られるログイン名に対応したホームディレクトリの パス名に置き換えられます。システムがログイン名を認識できない場合、結果は未定義です。

チルド展開は、通常はワードの先頭でのみ発生しますが、従来からの使用法に基づいた例外的な使い方もあります。次の例を見てください。

```
PATH=/posix/bin:~dgk/bin
```

これもチルド展開の対象となります。チルドがコロンの直後にあり、それに続く文字が 1 つも引用符で囲まれて いないためです。以下のような置換も発生しうるため、このようなチルド展開は抑止する方向も検討されました。

```
PATH=$(printf %s ~karels/bin : ~bostic/bin)
for Dir in ~maart/bin ~srb/bin .
do
     PATH=${PATH:+$PATH:}$Dir
done
```

最初のコマンドでは、各ディレクトリにコロンが明示的に指定されています。いずれの場合も、シェルはすべてのディレクトリ名に対してチルド展開を実行します。どの名前も別個のワードとして認識するためです。

次にオペランド中の式の例を見てみましょう。

```
make -k mumble LIBDIR=~chet/lib
```

このような代入式は、シェル変数の代入と認識される条件を 満たしていないので、チルド展開は行われません。ただし、コマンドが展開を自身で行う場合は除きます (make は行いません) 。

将来的な仕様向けに、どのワード中でもチルド展開を強制的に実行する方法として、$~ という特殊な文字列が提供されています。

ワードは引用符をつけてはいけない、という規則なので、以下の指定は同等にはなりません。チルド展開が行われるのは、最後の指定だけです。

```
\~hlj/   ~h\lj/   ~"hlj"/   ~hlj\/   ~hlj/
```

Korn シェルの ~+ と ~– 構造は、この規則を利用しているため、未定義のログイン名に チルドを付加した場合、処理の結果は予測できません。なお、一般に、誤ったログイン名にチルドを付加するとエラーになります。また HOME が未設定の場合、従来のシェルの中にはそれをエラーとするものもあるため、結果は予測できません。

コマンド置換

コマンドが、ドル記号に続く括弧で囲まれている場合 (つまり $(*command*) の形式)、または一対の逆引用符 (`) で囲まれている場合には、その標準出力をワードの一部または全体として使用できます。その場合に標準出力に含まれる復帰改行は削除されます。2 番目の引用符で囲む形式では、コマンドの実行前に、引用符間の文字列に含まれる特殊引用符文字が処理されます (後述の「クォート」を参照) 。$(<*file*) というコマンド置換は、同じ動作で実行速度の速い $(<*file*) で代用することもできます。入出力リダイレクトを行わない特殊コマンドの多くは、別のプロセスを作成せずに実行されるからです。

コマンド置換を使うと、コマンド名をコマンドの出力で置き換えて しまうこともできます。コマンド置換は、コマンドを以下のように囲んで 記述すると行われます。

$(*command*)

または逆引用符を使って 次のように 指定します。

`` `command` ``

シェルは *command* をサブシェル環境で実行し、コマンド置換指定 (*command* のテキストと、それを囲んでいる `$()` または逆引用符) をコマンドの標準出力で置き換え、置換指定の最後にあるいくつかの連続した復帰改行文字を除去する ことにより、コマンド置換を展開します。出力の途中に埋め込まれている復帰改行文字は除去されません。ただし、IFS の値および現在有効な引用符によっては、それらの復帰改行文字はフィールド区切り文字とみなされて、フィールド分割時に削除されることもあります。

逆引用符形式のコマンド置換においては、バックスラッシュは文字そのものとしての意味を保持します。ただし以下に示す文字、つまりドル記号、逆引用符、バックスラッシュが直後に続く場合を除きます。

```
$     `     \
```

対の相手となる逆引用符の検索は、直前の文字がバックスラッシュでない 逆引用符が見つかれば満たされます。この検索において、エスケープのない逆引用符がシェルコメント中、here-document (各種シェルで << を使用する機能) 、$(*command*) 形式のコマンド置換中、または引用符つきの文字列中で検出された場合、結果は未定義となります。先頭が ‘. . .‘ の並びで、終わりがこの並びではない、単一引用符または二重引用符で囲まれた文字列がある場合、結果は未定義となります。

$(*command*) 形式の指定では、左括弧の次の文字から、対応する右括弧までの文字が *command* を構成します。この *command* には、シェルのスクリプトとして正しいものであれば自由に使用できます。ただし、

- 入出力先のリダイレクト指定だけからなるスクリプトの場合、結果は予測できません。
- 単一のサブシェルの場合には、後述するような制限があります。

コマンド置換によって得られた結果は、あとでチルド展開、パラメータ展開、コマンド置換、または 算術展開を行うためにフィールド分割やパス名展開されることはありません。二重引用符に囲まれた内部でコマンド置換が発生しても、二重引用符の中で通常起こる 置換の結果に対しては行われません。

コマンド置換は入れ子にできます。逆引用符形式の内部で入れ子を 指定する場合には、内側の両方の逆引用符の直前にバックスラッシュを 記述する必要があります。以下に例を示します。

`` `\`command \` ``

逆引用符を使ったときの動作が一律でない、という問題は、$() 形式のコマンド置換を使えば解決できます。次の例を見てください。

コマンド	出力
echo '\$x'	\$x
echo `echo '\$x'`	$x
echo $(echo '\$x')	\$x

また、逆引用符形式は、埋め込まれたコマンドの内容に対して、従来から制限を持っていました。新しい $() 形式がどんな種類のスクリプトでも処理できるのに対し、逆引用符形式は、逆引用符を含んだ正しいスクリプトを扱えないことがあります。たとえば以下に示す例は、いずれも正しいスクリプトですが、右側は正しく処理されるのに対し、左側は処理されません。

echo `	echo $(
cat <<eeof	cat <<eeof
a here-doc with `	a here-doc with)
eof	eof
`	)
echo `	echo $(
echo abc # a comment with `	echo abc # a comment with)
`	)
echo `	echo $(
echo '`'	echo ')'
`	)

このように逆引用符形式のコマンド置換は動作に一貫性がないため、コマンド置換を入れ子にしたり、複雑なスクリプトを埋め込もうとするアプリケーションには、使わない方がいいでしょう。

コマンド置換が以下のように単一のサブシェルからなる場合、

$((*command*))

移植性のあるアプリケーションであれば、$(と (を 2 つのトークンに (空白を使って) 分離させる必要があります。これは算術展開と混同するのを避けるためです。

算術展開

ドル記号に続く二重括弧で囲まれた算術式 (つまり ($ ((*arithmetic-expression*))) の形式) は、二重括弧内の算術式の値に置き換えられます。算術展開は、算術式を評価して値を代入するための メカニズムを提供します。算術展開の形式は次のとおりです。

$((*expression*))

式すなわち *expression* は、あたかも二重引用符で囲まれているかのように扱われます。ただし、式の内部の二重引用符は特別な意味を持つとは見なされません。シェルは、パラメータ置換、コマンド置換、引用符削除のために、式の中のトークンをすべて展開します。

次にシェルはこの式を算術式と見なし、その値を置き換えます。算術式は、以下に述べる例外を除き、ISO C の規則に従って処理されます。

- 整数の算術だけが必須です。
- `sizeof()` 演算子、および先頭と後尾の ++ と – – 演算子は必須ではありません。
- 選択、繰り返し、ジャンプの各ステートメントは サポートされていません。
- `/usr/bin/ksh` は先頭の 0 から 9 までを 10 進定数として扱います。次の例を参照してください。

コマンド	**/bin/ksh** の結果	**/usr/xpg4/bin/sh** の結果
`echo $((010+10 ))`	20	18
`echo $((010+10 ))`	29	エラー
`[ 10 –le $((011)) ]`	真	偽

拡張機能として、上記リストに挙がっている算術式でも シェルが認識できる場合があります。式が不正な場合、展開は失敗に終わり、シェルはそのことを示す メッセージを標準エラー出力に書き出します。

算術展開を行う簡単な例を以下に示します。

```
# repeat a command 100 times
x=100
while [ $x –gt 0 ]
do
     command
     x=$(($x–1))
done
```

プロセス置換

この機能は SunOS が提供するもので、UNIX オペレーティングシステムのバージョンが、オープンしたファイルに名前をつけるための `/dev/fd` ディレクトリをサポートしている 場合にだけ使用できます。< (*list*) または > (*list*) 形式の各コマンド引数は、*list* が示すプロセスを実行します。このプロセスは、`/dev/fd` 中のファイルに非同期的に接続されています。このファイルの名前が当該コマンドの引数となります。> が

付いた形式を使用した場合、このファイルに書き出すことにより *list* への入力が可能となります。< の形式を用いると、引数として渡されたファイルには *list* プロセスからの出力が含まれます。次の例を見てください。

```
paste <(cut -f1 file1) <(cut -f3 file2) | tee >(process1) >(process2)
```

このコマンドは *file1* からフィールド 1 をカットし、*file2* からフィールド 3 をカットし、その両者をペーストし、その結果を *process1* と *process2* に送り、さらに標準出力上に書き出します。なおこのファイルは、引数としてコマンドに渡されますが、UNIX の `pipe`(2) になっているので、ファイル上で `lseek`(2) を行おうとするプログラムは動作できません。

パラメータ置換

パラメータ(*parameter*) は、識別子 (*identifier*)、1 つまたは複数の数字、または `*`、`@`、`#`、`?`、`-`、`$`、`!` の文字のいずれかです。変数 (*variable*) は識別子が示すパラメータで、1 つの値 (*value*) といくつかの属性 (*attributes*) を持ちます。属性がない場合もあります。特殊コマンド `typeset` を使用すれば、変数に値と属性を代入できます。シェルがサポートする属性については、`typeset` 特殊コマンドの項で後述します。エクスポートされた変数は、値と属性を環境に渡します。

シェルは一次元配列機能をサポートします。配列変数の要素は、添字 (*subscript*) によって参照されます。添字を指定するには、最初に `[` 、次に算術式 (後述の「算術評価」を参照) 、その次に `]` を記述します。配列に値を割り当てるときは、`set -A` *name value* `. . . .` を使用してください。添字の値は常に 0 から 4095 の範囲内で指定してください。配列を宣言する必要はありません。有効な添字を伴う変数参照は正当であり、必要に応じて配列が作成されます。添字なしで配列を参照するのは、`0` 番目の要素を参照することと同じ意味です。配列の *identifier* に添字として `*` または `@` を用いると、個々の要素の値が置換されます (フィールド区切り文字で区切られます)。

以下の記述により変数に値を 代入することもできます。

name=*value* [*name*=*value*] . . .

name に `-i` という整数属性を設定すると、*value* は後述の算術評価を受けます。

定位値パラメータは数値により指定されるパラメータで、`set` 特殊コマンドで値を代入できます。`$0` パラメータには、シェル起動時に 0 番目の引数から値を設定します。1 つまたは複数の数字からなるパラメータは、定位値パラメータになります。複数の数字からなる定位値パラメータは中括弧で囲む必要があります。

パラメータの展開

パラメータの展開は以下の形式で記述します。

${*expression*}

ここで *expression* は、対応する } の直前までのすべての文字を含みます。なおバックスラッシュによりエスケープされている } や引用符で囲まれた文字列内の }、および埋め込まれた算術展開、コマンド置換、変数展開の中の文字は、対応する } の検索の対象とはなりません。

パラメータ展開の最も単純な形式は次のとおりです。

${*parameter*}

parameter が値を持っていれば置き換えられます。

パラメータの名前と記号は、中括弧 ({}) で囲むこともできます。この中括弧は、複数の数字からなる位置パラメータ、および名前の一部として 解釈される恐れのあるような文字が *parameter*．．．に続く場合を除いては、省略可能です。対となる閉じる中括弧は、括弧のレベルを数えながら、引用符で囲まれた文字列やコマンド置換をスキップして見つけられます。

パラメータの名前や記号が中括弧で囲まれていない場合、正当な名前となりうる最長のものが展開に使われます。その名前が表す記号が存在しているかどうかは問いません。シェルが名前の境界を決定するために入力を走査するとき、どの名前がすでに定義されているかをシェルが知っていても、それに影響されることはありません。たとえば F というシェル変数が定義されているとき、以下のコマンドを実行したとします。

```
echo $Fred
```

このとき、$F のあとに red が表示されるようなことはありません。正しい名前となりうる最長の文字列は Fred であり、そのような名前は定義されていないためです。

二重引用符内でパラメータ展開が発生した場合、

- 展開結果に対してはパス名展開は行われません。
- 展開結果に対しては、@ の場合を除きフィールド分割は行われません。

また、以下に示す形式を使って、パラメータ展開を変更することが可能です。*word* の値が必要となる場合 (後述するように *parameter* の状態による) には、*word* チルド展開、パラメータ展開、コマンド置換、および算術展開の対象となります。*word* が必要でなければ、展開は行われません。なおパラメータ展開の変更指定を区切る文字 } は、先ほど述べた方法、さらに dquote でも述べるような方法で決定されます。たとえば、${foo-bar}xyz} という指定は、foo が設定済みであれば foo の展開のあとに文字列 xyz} が続き、foo が設定されていなければ文字列 barxyz} となります。

${*parameter*:–*word*}	「デフォルト値の使用」。*parameter* が未設定または NULL の場合、*word* の展開結果が代入されます。それ以外の場合には *parameter* の値が代入されます。
${*parameter*:=*word*}	「デフォルト値の割当」。*parameter* が未設定または NULL の場合、*word* の展開結果が *parameter* に割り当てられます。どのような場合でも、*parameter* の最終的な値が代入されます。この割当方法は変数だけに使用可能で、位置パラメータや特殊パラメータには使用できません。
${*parameter*:?[*word*]}	「未設定または NULL のときエラー表示」。*parameter* が未設定または NULL の場合、*word* の展開結果 (また

は *word* 省略時は未設定を表すメッセージ) が標準エラー出力に書き出され、シェルは ゼロ以外の終了ステータスで終了します。それ以外の場合には、*parameter* の値が代入されます。対話型シェルでは終了しません。

${*parameter*:+[*word*]} 「代替値の使用」。*parameter* が未設定または NULL の場合、NULL が代入されます。それ以外の場合には、*word* の展開結果が代入されます。

前述のパラメータ展開では、形式中にコロンを使うとパラメータが未設定または NULL であることのテストとなり、コロンを省略すると パラメータが未設定であることのテストとなります。コロンの役割を以下の表にまとめます。

	パラメータが NULL以外に設定	パラメータが NULLに設定	パラメータが 未設定
${*parameter*:-*word*}	*parameter* に置換	*word* に置換	*word* に置換
${*parameter*–*word*}	*parameter* に置換	NULL に置換	*word* に置換
${*parameter*:=*word*}	*parameter* に置換	*word* を代入	*word* を代入
${*parameter*=*word*}	*parameter* に置換	*parameter* に置換	NULL を代入
${*parameter*:?*word*}	*parameter* に置換	エラー 終了	エラー 終了
${*parameter*?*word*}	*parameter* に置換	NULL に置換	エラー 終了
${*parameter*:+*word*}	*word* に置換	NULL に置換	NULL に置換
${*parameter*+*word*}	*word* に置換	*word* に置換	NULL に置換

この表で、「．．．に置換」は、式が表に示すものに置き換えられることを表します。また「．．．を代入」は、パラメータにその値が代入され、さらに式を置き換えることを表します。

${#*parameter*}　「文字列の長さ」。パラメータの値の長さを文字単位で示します。*parameter* が * または @ のとき、$1 を始めとしてすべての定位置パラメータ (フィールド区切り文字で区切られている) が置き換えられます。

以下に示す 4 種類のパラメータ展開は、いずれも部分文字列処理用のものです。どの場合でも、パターンの評価には、正規表現ではなく パターンマッチング表現 (patmat を参照) が使われます。*parameter* が * または @ のとき、$1 を始めとしてすべての定位置パラメータ (フィールド区切り文字で区切られている) が置き換えられます。完全パラメータ展開文字列を二重引用符で囲んでも、以下の 4 種類のパターン文字列は引用符付きとはなりません。ただし中括弧の内部で文字列を囲んだ場合には、引用符付きとなります。

${*parameter*%*word*}　「最小の接尾辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接尾辞中の *pattern* と一致する最小部分が削除されます。

${*parameter*%%*word*}　「最大の接尾辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接尾辞中の *pattern* と一致する最大部分が削除されます。

${*parameter*#*word*}　「最小の接頭辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接頭辞中の *pattern* と一致する最小部分が削除されます。

${*parameter*##*word*}　「最大の接頭辞パターンの削除」。*word* が展開されてパターンを生成します。そのあと *parameter* に対してパラメータ展開が行われ、接頭辞中の *pattern* と一致する最大部分が削除されます。

使用例:

${*parameter*:–*word*}

次の例では、x が NULL または未設定の場合にのみ ls が実行されます。なお $(ls) コマンド置換の表現方法については、前述の 「コマンド置換」の項で説明してあります。

```
${x:-$(ls)}
```

${*parameter*:=*word*}

```
unset X
echo ${X:=abc}
abc
```

${*parameter*:?*word*}

```
unset posix
echo ${posix:?}
sh: posix: parameter null or not set
```

${*parameter*:+*word*}

```
set a b c
echo ${3:+posix}
posix
```

${#*parameter*}

```
HOME=/usr/posix
echo ${#HOME}
10
```

${*parameter*%*word*}

```
x=file.c
echo ${x%.c}.o
file.o
```

${*parameter*%%*word*}

```
x=posix/src/std
echo ${x%%/*}
posix
```

${*parameter*#*word*}

```
x=$HOME/src/cmd
echo ${x#$HOME}
/src/cmd
```

${*parameter*##*word*}

```
x=/one/two/three
echo ${x##*/}
three
```

シェルが設定するパラメータ

次のパラメータは、シェルが自動的に設定します。

#	定位置パラメータ数 (10 進数)
-	呼び出し時に、または `set` コマンドによってシェルに与えられたフラグ
?	最後に実行されたコマンドが返した 10 進数
$	このシェルのプロセス番号

_	初期設定時、_ というパラメータは、実行中のシェルまたはスクリプトの絶対パス名で、環境 (*environment*) に引き渡される値。そのあと、このパラメータには直前のコマンドの最後の引数が代入される。このパラメータは、非同期式のコマンドに関しては設定されない。このパラメータは、メールのチェック時にも、一致する MAIL ファイルの名前を保持するために使用する
!	最後に呼び出されたバックグラウンドコマンドのプロセス番号
ERRNO	最後に失敗したシステムコールにより設定された *errno* の値。この値はシステムに依存し、デバッグ目的で使用される
LINENO	実行中のスクリプトまたは関数内での現在行の行番号
OLDPWD	cd コマンドで設定された、直前の作業ディレクトリ
OPTARG	getopts 特殊コマンドで処理された最後のオプション引数の値
OPTIND	getopts 特殊コマンドで処理された最後のオプション引数のインデックス
PPID	シェルの親のプロセス番号
PWD	cd コマンドで設定された、現在の作業ディレクトリ
RANDOM	この変数を参照するたびに、0 から 32767 間に均一に分散した乱整数を生成する。乱数の並びは、RANDOM に数値を代入すれば初期化できる
REPLY	この変数は、引数指定のない select 文または read 特殊コマンドにより設定される
SECONDS	この変数を参照するたびに、シェルを起動してからの秒数が返される。この変数に値を代入すると、その値と代入処理実行時からの秒数との合計値が 参照時に返される

シェルが用いる変数

次の変数はシェルによって使用されます。

CDPATH	cd コマンドの検索パスを指定します。
COLUMNS	この変数を設定すると、シェル編集モードと select リストの出力用の編集ウィンドウの幅がその値によって、定義されます。
EDITOR	この変数の値が emacs、gmacs、または vi で終わり、VISUAL 変数が設定されていないと、該当するオプション (後述の「特殊コマンド」中の set の項を参照) が有効になります。
ENV	この変数は、シェルが呼び出されたときに シェルによるパラメータ展開の対象となり、それにより得られた値が、現環境で実行するシェルコマンドを含んでいるファイルのパス名として用いられます。このファイルは実行可能形式でなくてもかまいません。ENV を展開した値が絶対パス名でない場合、結果は予測できませ

	ん。ユーザーの実ユーザー ID と実効ユーザー ID が異なっていたり、実グループ ID と実効グループ ID が異なっていたりするとき、ENV は無視されます。 この変数を使って、そのシェル呼び出しだけに有効な 別名や他の項目を設定することができます。ENV が参照するファイルは $HOME/.profile とは異なります。つまり .profile が通常はセッション立ち上げ時に実行されるのに対し、ENV ファイルはシェル呼び出しの度に冒頭で実行されます。ENV の値はドットスクリプトと同じように解釈されます。つまりコマンドは現環境で実行され、ファイルは実行可能でなくてもかまいませんが、読み取り可能で ある必要があります。なお、ドットスクリプトでは PATH の検索が行われますが、ENV では行われません。これは「トロイの木馬」型セキュリティ侵入から保護するために使われます。
FCEDIT	fc コマンドのデフォルトのエディタ名を指定します。
FPATH	関数定義の検索パスを指定します。デフォルトでは、PATH 変数に設定されているディレクトリを検索したあと、FPATH に設定されているディレクトリを検索します。実行可能ファイルを見つけると、現在の環境中で実行します。-u 属性を持つ関数を参照した場合、PATH 変数の前に FPATH のディレクトリを検索します。あらかじめ設定されている別名の autoload を使えば、-u 属性を持つ関数を生成できます。
IFS	内部フィールド区切り文字。通常は空白文字、タブ、復帰改行です。コマンドまたはパラメータの置換によって生じる コマンドワードを区切るときと、read 特殊コマンドでワードを区切る場合に使用します。$* 置換において引数を区切るときは、IFS 変数の最初の文字を使用します (後述の 「クォート」 を参照) 。
HISTFILE	シェル起動時にこの変数が設定されていると、その値はコマンド履歴 (後述の「コマンド再入力」を参照) を格納するために使用されるファイルのパス名になります。
HISTFILE	シェル起動時にこの変数が設定されていると、このシェルで使用可能な入力済みコマンドの数が、この値以上になります。デフォルト値は 128 です。
HOME	cd コマンドのデフォルト引数 (ホームディレクトリ) を指定します。
LC_ALL	LC_* 変数のデフォルト値を提供する変数を指定します。
LC_COLLATE	この変数は、パターンマッチングにおける範囲式、同等クラス、および 複数バイト文字照合要素の動作を定義します。
LC_CTYPE	シェルが文字を処理する方法を決定します。LC_CTYPE に有効な値が設定されていると、シェルは、そのロケールに合った文字を含む テキストやファイル名を表示および処理できます。環境に

LC_CTYPE (environ(5) を参照) が設定されていなければ、シェルの動作は環境変数 LANG によって決定されます。LC_ALL が設定されていれば、その内容が LANG 変数やその他の LC_* 変数より優先されます。

LC_MESSAGES　メッセージをどの言語で出力するかを表す変数を指定します。

LANG　未設定もしくは NULL の国際化変数に対するデフォルト値を指定します。国際化変数のいずれかが不正な値に設定されていると、ユーティリティは変数が 1 つも定義されていないように動作します。

LINENO　この変数は、各行のコマンド実行前に、スクリプトまたは関数内での現在の 行番号 (1 から始まる連続した番号) を表す 10 進数に、シェルにより設定されます。ユーザーが LINENO を設定解除もしくは再設定すると、現在のシェルの動作中は、この変数が持つ特殊な意味は失われます。シェルが現在スクリプトも関数も実行していない場合、LINENO の値は予測できません。

LINES　この変数を設定すると、その値は選択 (select) リスト出力用のカラム長の決定に使用されます。選択リストは、LINES が示す行数の 3 分の 2 が一杯になるまで垂直に出力されます。

MAIL　この変数をメールファイルの名前に設定し、さらに MAILPATH 変数を設定しない場合、シェルは指定されたファイルにメールが到着すると ユーザーに通知します。

MAILCHECK　この変数は、MAILPATH 変数または MAIL 変数で指定されるファイルが更新されたかどうかを、シェルが何秒ごとにチェックするかを指定します。デフォルト値は 600 秒です。この時間が経過すると、シェルは次のプロンプトを出す前に チェックします。

MAILPATH　コロン (:) で区切ったファイル名のリストを指定します。この変数を設定すると、シェルは直前の MAILCHECK 秒間に発生した、指定ファイルに対する変更について ユーザーに通知します。各ファイル名のあとには ? と、出力されるメッセージを指定できます。メッセージは、出力に先立ち、$_ 変数に変更されたファイルの名前を代入する パラメータ置換処理を受けます。デフォルトメッセージは you have mail in $_ です。

NLSPATH　LC_MESSAGES 処理用のメッセージカタログの場所を示す変数を指定します。

PATH　コマンド用の検索パスを指定します。(後述の「実行」を参照) rksh の下で実行する場合 (.profile の場合を除く) 、PATH を変更することはできません。

PPID　この変数は、シェルを呼び出したプロセスのプロセス ID (10 進数) に対して、シェルにより設定されます。サブシェル中では、PPID は現在のシェルの親の値と同じ値に設定されます。たとえば echo $PPID と（echo $PPID）は同じ値を生成します。

PS1	この変数の値はパラメータ置換用に展開され、一次プロンプト文字列を定義します。デフォルトは $ です。一次プロンプト文字列内の文字 ! は、コマンド番号で置換されます (後述の「コマンド再入力」を参照) 。! を 2 つ連続して指定すると、プロンプト文字列の出力時に ! が 1 つ表示されます。
PS2	二次プロンプト文字列を指定します。デフォルトは > です。
PS3	select ループ内で使用する選択プロンプト文字列を指定します。デフォルトは #? です。
PS4	この変数の値は、パラメータ置換用に展開され、実行トレースの各行の前に出力されます。省略すると、実行トレースプロンプトは + になります。
SHELL	シェルのパス名は環境内に保持されます。起動時に、この変数のベース名が rsh、rksh、または krsh の場合、シェルの機能は制限されます。
TMOUT	0 より大きい値に設定すると、PS1 プロンプトの発行後、指定された秒数以内に コマンドが入力されない場合、シェルは終了します。シェルのコンパイル時に、この値がユーザーが 超えられないような大きい値に設定されていることも あります。
VISUAL	この変数の値が emacs、gmacs、または vi で終わる場合、該当するオプション (後述の「特殊コマンド」中の set の項を参照) が有効になります。

シェルは、PATH、PS1、PS2、PS3、PS4、MAILCHECK、FCEDIT、TMOUT、および IFS にデフォルト値を割り当てますが、HOME、SHELL、ENV、および MAIL は、シェルによって設定されることはありません。ただし HOME は login(1) により設定されます。また一部のシステムでは、MAIL や SHELL も login で設定されることがあります。

ブランクの解釈

パラメータとコマンドの置換後、置換の結果内でフィールド区切り文字 (IFS で発見されるもの) を検索し、その文字が現れた位置で分割します。分割された各々が引数となります。明示的な NULL 引数 ("" または '' は保持されます。暗示的な NULL 引数 (値を持たないパラメータにより生じるもの) は削除されます。

ファイル名の生成

置換後、各コマンドワードは、*、?、[を含んでいないかチェックされます。ただし -f オプションを設定していない場合にかぎります。これらの文字のいずれかがあると、そのワードはパターンとみなされます。このワードは、パターンと一致する、辞書編集方式の順にソートされたファイル名に置換されます。パターンと一致するファイル名が見つからない場合、ワードは変更されません。パターンをファイル名の生成に使用する場合、ファイル名の先頭のピリオド (.) または スラッシュ (/) 直後のピリオドは、明示的に一致させる必要があります (後者の場合はスラッシュ自体をも含む)。ピリオドで始まるファイル名は 括弧の中でピリオドのある次のようなパターンとは 一致しません。

```
ls .@(r*)
```

これは .restore というファイル名を検出しますが、ls @(.r*) は検出しません。パターンマッチングの他のケースでは、/ に続く . は特殊文字とはみなされません。

*	NULL 文字列を含め、任意の文字列と一致します。
?	任意の単一文字と一致します。
[...]	囲まれた文字のいずれかと一致します。2 つの文字を - で区切ると、その間にある任意の文字 (その 2 つの文字も含む) に一致します。先頭の [の次の文字が ! である場合、[ ] で囲まれていない任意の文字と一致します。- は、最初の文字または最後の文字として 文字セットに挿入できます。

pattern-list は、| で区切られた 1 つまたは複数のパターンのリストです。複合パターンは、次のうちの 1 つまたは複数で形成することができます。

?(*pattern-list*)	指定されたパターンのいずれかと任意に一致します。
*(*pattern-list*)	指定されたパターンの 0 回以上の発生と一致します。
+(*pattern-list*)	指定されたパターンの 1 回以上の発生と一致します。
@(*pattern-list*)	指定されたパターンのうち 1 つだけと一致します。
!(*pattern-list*)	指定されたパターンのうち 1 つだけを除き、あらゆるものと一致します。

クォート

前述の「定義」の項で示したメタキャラクタ の各々は、シェルに対して特別の意味を持ち、クォートしないかぎり、ワードの終わりを表します。文字は、その前に \ を指定すればクォートされます。つまりその文字自身を 示すことができます。\ NEWLINE の対は削除されます。一対の単一引用符 (' ') で囲まれた文字はすべてクォートされます。単一引用符内にさらに単一引用符を指定することはできません。一対の二重引用符 ("") で囲まれた文字列内では、パラメータとコマンドの置換が発生し、\ は \ 、 ` 、 " 、 $ をクォートします。$* と $@ の意味は、クォートされていないとき、あるいはパラメータの代入値もしくはファイル名として 使用されるときは同一です。ただしコマンド引数として使用するとき、$* は $1*d* $2*d* . . . と同じになります ( *d* は IFS 変数の最初の文字です)。一方、$@ は $1 $2 ... と同じになります。一対の逆引用符 ( ` ` ) で囲まれた中では、\ は \ 、 ` 、 $ といった文字をクォートします。逆引用符を二重引用符内で指定すると、\ も " という文字をクォートします。

予約語や別名中のいずれかの文字をクォートすれば、その予約語が持つ特別な意味はなくなります。なお以下に説明する関数や特殊コマンドに関しては、その名前をクォートしても、関数名またはコマンド名としての 認識を変えることはできません。

算術評価

let という特殊コマンドには、整数演算を実行する機能が提供されています。評価は *long* 演算を使用して実行します。定数は [*base*#] *n* 形式とします。ここで *base* は底を表す 2 から 36 の範囲の 10 進数で、*n* はその底における数です。*base* を省略すると、底は 10 となります。

算術式は、C 言語での式と同一の構文、優先度、および結合規則を使用します。++、-- -、?:、, 以外のすべての整数演算子がサポートされています。算術式内では、パラメータ置換構文を使用しなくても、名前で変数を参照できます。変数を参照すると、その値は算術式として評価されます。

変数の内部整数表記は、typeset 特殊コマンドの -i オプションで指定できます。算術評価は、-i 属性を備えた変数に対する各代入値について実行されます。底を指定しないと、変数に対する最初の代入が底を決定します。この底は、パラメータ置換が発生する際に使用されます。

算術演算子の多くはクォートしなければならないので、代替形式の let コマンドが提供されています。((で始まるコマンドについては、対応する)) までの文字はすべてクォートされた表現とみなします。具体的に言うと、((...)) は let "..." と同じ意味です。

プロンプト

シェルは、対話的に使用すると、コマンドを読み取る前に PS1 のパラメータ展開値によるプロンプトを発行します。復帰改行を入力したあとで、コマンドを完了するためにさらに入力が必要な場合は、2 次プロンプト (つまり PS2 の値) が出力されます。

条件式

条件式 (*conditional expression*) は、ファイルの属性をテストしたり文字列を比較するときに、複合コマンドの [[とともに使用します。[[と]] の間のワードについてはワード分割とファイル名生成を実行しません。各条件式は、次の単項式または 2 項式をいくつか組み合わせて構築できます。

-a *file*	*file* が存在する場合、真です。
-b *file*	*file* が存在し、ブロック型特殊ファイルである場合、真です。
-c *file*	*file* が存在し、文字型特殊ファイルである場合、真です。
-d *file*	*file* が存在し、ディレクトリである場合、真です。
-e *file*	*file* が存在していれば、真です。
-f *file*	*file* が存在し、通常ファイルである場合、真です。
-g *file*	*file* が存在し、setgid ビットがセットされている場合、真です。
-k *file*	*file* が存在し、スティッキ・ビットがセットされている場合、真です。
-n *string*	*string* の長さが 0 ではない場合、真です。
-o *option*	*option* という名前のオプションが有効の場合、真です。
-p *file*	*file* が存在し、FIFO 特殊ファイルまたはパイプである場合、真です。
-r *file*	*file* が存在し、現在のプロセスで読み取り可能な場合、真です。

`-s` *file*	*file* が存在し、サイズが 0 より大きい場合、真です。
`-t` *fildes*	*fildes* が示すファイル記述子番号がオープンされていて、端末デバイスに対応している場合、真です。
`-u` *file*	*file* が存在し、setuid ビットがセットされている場合、真です。
`-w` *file*	*file* が存在し、現在のプロセスで書き込み可能な場合、真です。
`-x` *file*	*file* が存在し、現在のプロセスで実行可能な場合、真です。*file* が存在し、ディレクトリである場合、現在のプロセスにはそのディレクトリに対する 検索権があります。
`-z` *string*	*string* の長さが 0 の場合、真です。
`-L` *file*	*file* が存在し、シンボリックリンクである場合、真です。
`-O` *file*	*file* が存在し、このプロセスの実効ユーザー ID が所有している場合、真です。
`-G` *file*	*file* が存在し、そのグループがこのプロセスの実効グループ ID と一致する場合、真です。
`-S` *file*	*file* が存在し、ソケットである場合、真です。
file1 `-nt` *file2*	*file1* が存在し、*file2* より新しい場合、真です。
file1 `-ot` *file2*	*file1* が存在し、*file2* より古い場合、真です。
file1 `-ef` *file2*	*file1* と *file2* が存在し、同一のファイルを参照する場合、真です。
string	*string* が NULL 文字列でない場合、真です。
string `=` *pattern*	*string* が *pattern* と一致する場合、真です。
string `!=` *pattern*	*string* が *pattern* と一致しない場合、真です。
string1=*string2*	*string1* と *string2* が同じ場合、真です。
string1`!` =*string2*	*string1* と *string2* が同じでない場合、真です。
string1 `<` *string2*	カテゴリ `LC_COLLATE` に設定されるロケールに適切であると 解釈された文字列を比較し、*string1* が *string2* より小さい場合、真です。
string1 `>` *string2*	カテゴリ `LC_COLLATE` に設定されるロケールに適切であると 解釈された文字列を比較し、*string1* が *string2* より大きい場合、真です。
exp1 `-eq` *exp2*	*exp1* が *exp2* と等しい場合、真です。
exp1 `-ne` *exp2*	*exp1* が *exp2* と等しくない場合、真です。

exp1 `-lt` *exp2*	*exp1* が *exp2* 未満である場合、真です。
exp1 `-gt` *exp2*	*exp1* が *exp2* より大きい場合、真です。
exp1 `-le` *exp2*	*exp1* が *exp2* 以下の場合、真です。
exp1 `-ge` *exp2*	*exp1* が *exp2* 以上の場合、真です。

上記の式の各々において、*file* が `/dev/fd/`*n* という形式の場合 (*n* は整数)、記述子番号が *n* でオープンされているファイルでテストします。

以下のいずれかを使用すれば、これらの基本式から複合式を構築することができます。優先度の高いものから順に並べてあります。

(*expression*)	*expression* が真の場合、真です。式のグループ化に使用します。
`!` *expression*	*expression* が偽の場合、真です。
expression1 `&&` *expression2*	*expression1* と *expression2* が両方とも真の場合、真です。
expression1 `||` *expression2*	*expression1* と *expression2* のどちらかが真の場合、真です。

入出力

コマンド実行前に、シェルが解釈する特殊表記法によって入出力先を変更 (リダイレクト) できます。以下は、単純コマンド内の任意の位置および コマンドの前後に指定することができ、起動されたコマンドには引き渡されません。以下に示す場合を除き、*word* または *digit* を使用する前にコマンドとパラメータの置換が発生します。ファイル名生成が発生するのは、パターンが 1 つのファイルだけと一致し、さらにブランク解釈が実行されない場合だけです。

<*word*	*word* というファイルを標準入力 (ファイル記述子 0) として使用します。
>*word*	*word* というファイルを標準出力 (ファイル記述子 1) として使用します。ファイルが存在しない場合は作成します。ファイルが存在し、かつ `noclobber` オプションが有効の場合、エラーになります。その他の場合は、ファイルの長さが 0 になります。
>|*word*	`>` と同一です。ただし、これは `noclobber` オプションを無視します。
>>*word*	*word* というファイルを標準出力として使用します。ファイルが存在する場合、(EOF までシークしたあと) そのファイルに出力を追加します。ファイルが存在しない場合は、ファイルを作成します。
<>*word*	*word* というファイルを読み取りおよび書き込み用に 標準入力としてオープンします。
<< `[-]`*word*	シェルへの入力として *word* と同一の行まで、または EOF まで読み取ります。*word* に対しては、パラメータ

	置換やコマンド置換や ファイル名生成を実行しません。*here-document* が生成されて標準入力になります。*word* のいずれかの文字がクォートされている場合、ドキュメントの文字はいっさい解釈されません。クォートされている文字がなければ、パラメータとコマンドの置換が発生し、\NEWLINE が無視されます。また \ を使用して、\ 、$、`、および *word* の最初の文字をクォートす必要があります。– を << のあとに付加すると、*word* とドキュメントから先行タブをすべて取り除きます。
<&*digit*	*digit* が示すファイル記述子から複製したものを 標準入力として使用します (dup(2) を参照) 。同様に、標準出力への複製には >&*digit* を使用します。
<&–	標準入力をクローズします。同様に、標準出力については >&– を使用します。
<&p	並行プロセスからの入力を標準入力へ移動します。
>&p	並行プロセスへの出力を標準出力へ移動します。

上記のいずれかの前に数字が付く場合、その値が (デフォルトの 0 または 1 のかわりに) 該当ファイルに対応した ファイル記述子となります。

```
. . . 2>&1
```

ファイル記述子 2 をファイル記述子 1 から複製して、書き込み用にオープンします。

リダイレクトを指定する場合、記述する順序が重要になります。シェルは、リダイレクト指定を左から右へ評価します。

```
. . . 1>fname 2>&1
```

上記の例では、まず *fname* というファイルにファイル記述子 1 を関連付けます。次に、ファイル記述子 1 に関連するファイル (つまり *fname*) に、ファイル記述子 2 を関連付けます。リダイレクトの向きが逆であれば、まずファイル記述子 2 を端末に関連付け (ファイルを記述子 1 がすでに端末に関連付けられているとみなし) 、次にファイル記述子 1 をファイル *fname* に関連付けます。

ジョブ制御が有効でない場合に コマンドのあとに & を指定すると、コマンドにおけるデフォルトの標準入力は /dev/null という空ファイルになります。その他の場合、コマンドを実行するための環境には、起動側シェルのファイル記述子 (入出力指定で変更可能) が含まれます。

環境

環境は、通常の引数リストが 実行されるプログラムに引き渡される場合と同様の方法で引き渡される、名前と値の対です (environ(5) を参照) 。名前は識別子、値は文字列である必要があります。シェルが環境と対話する方法はいくつかあります。シェルは、起動されると、環境を走査して、見つけた名前ごとに変数を作成し、対応する値

を設定し、さらに export というマークを付けます。実行されるコマンドは環境を引き継ぎます。ユーザーがこれらの変数の値を変更したり 新しい変数を作成したときには、export コマンドまたは typeset -x コマンドを使用すれば それらの値が環境の一部になります。したがって、実行されるコマンドが参照する環境は、シェルが最初に引き継いだ「名前 = 値」の対 (その値は現在のシェルで変更可能) に、export コマンドまたは typeset -x コマンドで指定したものを加えたものになります。

1 つまたは複数の変数代入を先頭に付加すれば、単純コマンドおよび関数の環境を 拡張できます。変数代入引数は、*identifier=value* という形式のワードです。

TERM=450 *cmd args*

および

(export TERM; TERM=450; *cmd args)*

上記 2 つの例は同じことを意味します。これは *cmd* の実行に関してだけにかぎります。ただし、*cmd* が「特殊コマンド」の項で示す * 印の付いたコマンドの場合を除きます。

-k フラグを設定すると、変数代入引数はすべて環境に格納されます。これらの引数がコマンド名のあとに指定された場合も同様です。以下の例では、まず a=b c を、次に c を表示します。

```
echo a=b c
set –k echo
a=b c
```

この機能は、シェルの初期バージョン用に作成された スクリプトで使用するためのものです。新しいスクリプトには使用しないでください。将来、この機能がなくなる可能性があるからです。

関数

前述の 「コマンド」 の項で説明した function は予約語であり、シェル関数の定義に使用します。シェル関数は内部で読み取られ、保存されます。別名は、関数を読み取る際に解釈されます。関数はコマンドと同様に実行され、引数は定位置パラメータとして渡されます (後述の「実行」を参照) 。

関数は、呼び出し側と同一のプロセスにおいて実行され、すべてのファイルと現在の作業ディレクトリを呼び出し側と共有します。呼び出し側が受け取るトラップは、関数内部でそのデフォルトの動作に再設定されます。関数が受け取らないかあるいは無視するトラップ条件があると、関数は終了し、その条件は呼び出し側に引き渡されます。

関数内部で設定された EXIT に基づくトラップは、関数が呼び出し側の環境で完了したあとに実行されます。これは、非 POSIX スタイルの関数についてのみ真です。非 POSIX スタイルの関数は次のように宣言されます。

```
function func
```

一方、POSIX スタイルの関数は次のように宣言されます。

```
func()
```

通常、変数は呼び出し側プログラムと関数間で共有されます。ただし、関数内で使用される typeset 特殊コマンドは、適用範囲として 現在の関数とそれが呼び出す関数すべてを含む 局所変数を定義します。

return という特殊コマンドは、関数の呼び出しから戻るときに使用します。関数の中でエラーが起こると、呼び出し側に制御が戻ります。

全関数の名前を一覧表示するには typeset +f と入力します。全関数の名前とともに関数のテキストも表示するには typeset -f と入力します。特定の関数のテキストだけを表示するには typeset -f *function-names* と入力します。unset 特殊コマンドの -f オプションを使えば、関数を未定義状態にすることができます。

通常、シェルがシェルスクリプトを実行するとき、関数は設定を解除されます。typeset コマンドの -xf オプションによって、別個にシェルを起動しなくても 実行されるスクリプトに関数をエクスポートできます。シェルを起動して定義する必要がある関数は、typeset の -xf オプションで ENV ファイルに指定する必要があります。

関数定義コマンド

関数とは、一群のコマンド (compound command) を、新たな定位置パラメータを指定して 単純コマンドとして呼び出すために、ユーザーが 定義した名前です。関数は「関数定義コマンド」を使って定義します。

関数定義コマンドの形式を以下に示します。

fname() *compound-command*[*io-redirect . . .*]

fname は名前でなければならず、これが関数名となります。実装によっては拡張機能として、他の文字を関数名に使うことを 許しているものもあります。その場合、関数と変数とを別個の名前領域で管理します。

関数定義コマンド中の () は、2 つの演算子で構成されます。したがって、fname、(、および) をブランク文字で区切ることもできますが、省略も可能です。

引数 *compound-command* は、関数呼び出しにより実行する一群のコマンドです。

関数が宣言されたとき、wordexp の展開は *compound-command* や *io-redirect* に対しては行われません。すべての展開処理は、通常のように、関数が呼び出されるたびに行われます。同様に、*io-redirect*(省略可能) が示す入出力のリダイレクトや、*compound-command* 中の変数割当は、関数定義時ではなく関数の実行時に行われます。

関数を実行すると、特殊組み込みユーティリティの項で説明するように、構文エラーや変数割当用の機能が与えられます。

関数名が単純コマンドとして指定されると、それに対応した *compound-command* が実行されます。その単純コマンドに指定したオペランドが、*compound-command* の実行中、一時的に位置パラメータとなります。特殊パラメータ # も、オペランド数を示す値に変更されます。特殊パラメータ 0 は変更されません。関数の実行が終了すると、

定位置パラメータや # の値は、関数実行前の値に復元されます。*compound-command* 中で特殊組み込みコマンド return が実行された場合、関数実行は終了し、関数呼び出しの次のコマンドから 処理が再開されます。

単純コマンドが記述できるところであれば、関数定義を記述できます。以下に例を示します。

```
# If variable i is equal to "yes",
# define function foo to be ls −l
#
[ "$i" = yes ] && foo() {
      ls −l
}
```

関数定義コマンドの終了ステータスは、関数が正常に宣言されれば 0 で、そうでなければゼロより大きい値です。関数呼び出しの終了ステータスは、関数によって最後に実行されたコマンドの 終了ステータスです。

ジョブ

set コマンドの monitor オプションを有効にすると、対話型シェルが job を各パイプラインと関連付けます。このオプションは、jobs コマンドが表示する現在のジョブのテーブルを維持し、これらのジョブに整数番号を割り当てます。ジョブを & で非同期に起動すると、シェルは次の形式の行を表示します。

```
[1] 1234
```

非同期に起動されたジョブはジョブ番号 1 であり、プロセス ID が 1234 であるプロセスが 1 つ (トップレベル) あることを示します。

ジョブを実行中であり、別に実行したいジョブがある場合、^Z (CTRL-Z) キーを押せば、現在のジョブに STOP シグナルが送信されます。そうするとシェルは通常、ジョブが「停止」されたことを示し、プロンプトを表示します。これで、このジョブの状態を bg コマンドでバックグラウンドで処理するか、あるいは他のコマンドを実行してから、fg というコマンドで ジョブをフォアグラウンドに移すことができます。^Z は直ちに有効になります。つまり ^Z は、保留中の出力や読み取られていない入力が 直ちに中止されるという点で、割り込みに似ています。

バックグラウンドで実行中のジョブは、端末から読み取ろうとすると停止します。通常バックグラウンドジョブは出力を生成できますが、stty tostop というコマンドを指定すればこの出力生成も抑止することができます。この tty オプションを設定すると、バックグラウンドジョブは、入力の読み取り時と同様に出力を生成しようとすると停止します。

シェル内のジョブを参照する方法はいくつかあります。ジョブは、そのジョブのプロセス ID または以下のいずれかで参照できます。

%*number*	*number* が示す番号のジョブ
%*string*	コマンド行が *string* で始まっていたジョブ
%?*string*	コマンド行が *string* を含んでいたジョブ

%%	現在のジョブ
%+	%% と同じ
%-	直前のジョブ

シェルは、プロセスの状態が変更すると、直ちにそれを検出します。ジョブがブロックされてそれ以上進めない状態になると、シェルはそのことをユーザーに通知します。ただし、ユーザーの作業の妨げにならないよう、この通知はプロンプトを発行する直前にだけ行います。

モニターモードが有効のとき、完了した各バックグラウンドジョブは、CHLD に設定されているトラップを起こします。

ジョブの実行中または停止中にシェルを終了しようとすると、「停止中 (実行中) のジョブがある (You have stopped（running）jobs.)」という警告を受けます。jobs コマンドを使用すれば、どのジョブが該当するのかを確認できます。これを実行するか、あるいは直ちにシェルを再終了しようとすると、シェルは 2 度目の警告は出さず、停止中のジョブは終了します。nohup で起動したジョブの実行中に ログアウトを行うと、次のような警告メッセージを受けます。

```
You have jobs running.
```

実際にログアウトを行うには、ログアウトを 2 回行う必要がありますが、その場合でもバックグラウンドジョブは 実行し続けます。

シグナル

起動されたコマンドに対する INT シグナルと QUIT シグナルは、コマンドの後ろに & が指定され、ジョブの monitor オプションが有効でない場合、無視されます。その他の場合、シグナルは、シェルが親から引き継いだ値を持ちます (ただし、後述の trap コマンドの説明を参照) 。

実行

コマンドが実行されるたびに、上記の置換が実行されます。コマンド名は、後述の「特殊コマンド」 のいずれかと一致する場合、現在のシェルプロセス内で実行されます。次に、コマンド名がユーザー定義関数のいずれかと 一致するかどうかチェックされます。一致する場合、定位置パラメータが保存され関数呼び出しの引数に再設定されます。関数が完了するか return を発行すると、定位置パラメータリストが復元され、関数内の EXIT に設定されているトラップが実行されます。関数の値は、最後に実行されたコマンドの値です。関数は現在のシェルプロセスでも実行されます。コマンド名が特殊コマンドやユーザー定義関数を示していない場合、プロセスが作成され、exec(2) を介してコマンドが実行されます。

PATH というシェル変数は、コマンドが置かれている ディレクトリの検索パスを定義します。ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは /bin:/usr/bin: です (/bin、/usr/bin、および現在のディレクトリの順で指定) 。現在のディレクトリは、複数のコロンを連続して記述するか、パスリストの始めか終わりにコロンを付ければ指定できます。コマンド名に / が含まれている場合は、検索パスは使用されません。/ が含まれていなければ、パスにおける各ディレクトリに実行可能ファイルがあるか検索します。ファイルが実行権を持っているが、ディレクトリや a.out ファイルでない場合、シェルコマンドの入ったファイルとみなされま

す。そのファイルを読み取るときは、サブシェルが生成されます。この場合、エクスポートされていない別名、関数、および変数すべてが削除されます。括弧で囲まれたコマンドは エクスポートされていないものを削除することなく、サブシェルで実行されます。

コマンド再入力

端末装置から最近入力された、HISTSIZE が示す個数 (デフォルトは 128 個) のコマンドのテキストは、履歴ファイルに保存されています。$HOME/.sh_history というファイルは、HISTFILE 変数が設定されていない場合、または変数が示すファイルが書き込み不可能な場合に使用されます。シェルは、同じ名前の HISTFILE を使用する対話型シェルすべてのコマンド履歴を使用できます。fc という特殊コマンドは、このファイルの一部をリスト表示または編集するときに使用します。編集またはリスト表示すべきファイルの部分は、番号か、またはコマンドの最初の文字 (1 つまたは複数) を 指定することによって選択できます。単一のコマンドを指定することも、コマンドの範囲を指定することも可能です。fc の引数としてエディタプログラムが指定されていないと、FCEDIT という変数の値が使用されます。FCEDIT が未定義の場合は、/bin/ed が使われます。編集されたコマンドは、エディタを終了した時点で 表示および再実行されます。エディタ名に - を指定すると、編集段階がスキップされ、コマンドが再実行されます。この場合、*old=new* という形式の代入パラメータを使用すれば、実行前にコマンドを変更できます。たとえば、r が 'fc -e -' の別名として定義されているとき 'r bad=good c' と入力すると、c という文字で始まるコマンドのうち最新のコマンドが、その記述中の最初の bad という 文字列が good に置き換えられて再実行されます。

インライン編集オプション

通常、端末装置から入力されるコマンド行は、単に復帰改行 (RETURN または LINEFEED) をあとに伴います。emacs、gmacs、vi のいずれかのオプションが有効な場合、ユーザーはコマンド行を編集できます。いずれかのオプションを set すれば、対応する編集モードになります。編集オプションは、オプション名のいずれかで終了する値を VISUAL 変数または EDITOR 変数に代入するたびに、自動的に選択されます。

編集機能では、ユーザーの端末が RETURN を改行のないキャリッジリターンとして扱うことができ、空白文字 がスクリーン上の現在の文字を上書きする必要があります。

これらの編集モードは、ユーザーがウィンドウごしに 現在の行を見るという概念を実現します。ウィンドウ幅は、COLUMNS が定義されていればその値に、未定義の場合は 80 になります。ウィンドウ幅が小さすぎて、プロンプトを表示すると入力用に 8 カラム以上残すことができなくなる場合には、プロンプトは左端から切り捨てられます。行がウィンドウ幅から 2 引いたものより長いと、ウィンドウの終わりにマークを表示してユーザーに通知します。カーソルが移動し、ウィンドウの境界に達すると、ウィンドウはカーソルを中心としてセンタリングされます。行がウィンドウの右端を超えている場合、表示されるマークは > となります。同様に、左端を超えていれば < が、左右両端を超えていれば * が表示されます。

各編集モードでは検索コマンドにより 履歴ファイルを使用できます。パターンではなく文字列だけがマッチングされます。ただし文字列の先頭に ^ があると、マッチング開始位置は行の先頭に限定されます。

emacs 編集モード

emacs または gmacs オプションを有効にすると、それぞれに対応する編集モードに入ります。この 2 つのモードは、^T の扱い方が異なるだけです。編集を行うには、訂正が必要な位置にカーソルを移動し、文字やワードを挿入または削除します。編集用のコマンドは、制御文字またはエスケープシーケンスで 入力します。本項の説明では、山型記号 (^) を使って制御文字を示してあります。たとえば ^F というのは CTRL–F を表します。つまり、CTRL (コントロール) キーを押した状態で f キーを入力します。このときシフトキーは押しません。なお、^? は DEL (削除) キーを表します。

エスケープシーケンスの入力は M- を使って示してあります。たとえば M-f(メタ f と呼ぶ) は、ESC キー (ASCII コード 033) のあとに f を入力することを表します。同様に M-F は、ESC キーのあとに大文字の F を入力することを表します。

編集コマンドは、行のどこからでも実行できます。行の先頭だけではありません。特に断わりのないかぎり、編集コマンドの終わりを表すために改行キー (RETURN キーまたは LINEFEED キー) を入力する必要はありません。

^F	カーソルを 1 文字分前進 (右方向) します。
M-f	カーソルを 1 ワード分前進 します。emacs エディタにおける「ワード」とは、英文字、数字、下線からなる文字列を意味します。
^B	カーソルを 1 文字分後退 (左方向) します。
M-b	カーソルを 1 ワード分後退 します。
^A	カーソルを現在の行の先頭に移動します。
^E	カーソルを現在の行の終端に移動します。
^] *char*	現在の行において文字 *char* が次に現れる位置にカーソルを移動します。
M-^] *char*	現在の行において文字 *char* が前に現れた位置にカーソルを移動します。
^X^X	カーソルとマークを入れ替えます。
erase	(ユーザーが stty(1) で定義した削除キー。通常は ^H または #) 直前の文字を削除します。
^D	現在の文字を削除します。
M-d	現在のワードを削除します。
M-^H	(メタ –Back Space) 直前のワードを削除します。
M-h	直前のワードを削除します。
M-^?	(メタ –DEL) 直前のワードを削除します。なお、割り込み文字を ^?(つまりデフォルトの DEL) と定義している場合、このコマンドは動作しません。

^T	emacs モードでは、現在の文字と次の文字とを入れ換えます。gmacs モードでは、直前の 2 つの文字を入れ換えます。
^C	現在の文字を大文字にします。
M-c	現在のワードを大文字にします。
M-l	現在のワードを小文字にします。
^K	現在カーソルが置かれている文字から 行の終端までをすべて削除します。なお、^K の前に数値パラメータを入力した場合には処理が異なります。その値が現在の位置の番号より小さければ、その値が示す位置の文字から現在の文字までを削除します。その値が現在の位置の番号より大きければ、現在の文字からその値が示す位置の文字までを削除します。
^W	カーソルのある文字からマークまでを抹消します。
M-p	カーソル位置からマークまでの 領域をスタックにプッシュします。
kill	(ユーザーが stty(1) で定義した抹消キー。通常は ^G または @) 現在の行を抹消します。2 つの kill 文字を連続して入力すると、そのあとの kill 文字は、改行を意味することになります。この機能は、プリンタ端末 (印字式端末) を使用している場合に便利です。
^Y	最後に削除されたデータを復元します。
^L	改行して、現在の行を印刷します。
^@	(NULL 文字) マークを設定します。
M–*space*	(メタ–空白文字) マークを設定します。
J	(NEWLINE) 現在の行を実行します。
M	(RETURN) 現在の行を実行します。
eof	現在の行が NULL の場合のみ、end-of-file 文字 (ファイルの終わりを示す。通常は ^D) を end-of-file として処理します。
^P	直前のコマンドを取り出します。^P を入力するたびに、1 つずつ逆方向に コマンドの履歴を取り出します。複数行にわたるコマンドで最初の行でない場合は、1 行戻ります。
M-<	最も古いコマンドの履歴を取り出します。
M->	最も新しいコマンドの履歴を取り出します。
^N	次のコマンド行を取り出します。^N を入力するたびに、1 つずつ順方向に コマンドの履歴を取り出します。
^R*string*	履歴を逆上って、*string* が示す文字列を含むコマンドを検索します。パラメータとして 0 を指定すると、順方向に検索します。*string* の終わりは、復帰改行 (RETURN または NEWLINE) で示

します。*string* の先頭に ^ を付加すると、その文字列で始まっている コマンドだけが検索されます。*string* を省略すると、直前に指定した文字列を指定したものとして 検索します。この場合、パラメータとして 0 を指定すると検索方向が逆になります。

`^O` 現在の行を実行し、現在の行に関連する次の行を 履歴ファイルから取り出します。

`M-`*digits* (エスケープ) 数値パラメータを定義します。*digits* は次のコマンドに対するパラメータと見なされます。パラメータを受け付けるコマンドは、`^F`、`^B`、*erase*、`^C`、`^D`、`^K`、`^R`、`^P`、`^N`、`^]`、`M-.`、`M-^]`、`M-_`、`M-b`、`M-c`、`M-d`、`M-f`、`M-h`、`M-l`、および `M-^H` です。

`M-`*letter* (ソフトキー) ユーザーの別名リスト中で、*_letter* という名前の別名を検索します。その別名が定義されていると、その値が入力待ち行列に挿入されます。*letter* は、前述のメタ関数に用いられている文字であってはなりません。

`M-[`*letter* (ソフトキー) ユーザーの別名リスト中で、*_letter* という名前の別名を検索します。この名前の別名が定義されていれば、処理待ちの入力待ち行列上にその値を挿入します。この機能は、多くの端末において、ファンクションキーをプログラムするために 使用できます。

`M-.` 直前のコマンドの最後のワードを 行に挿入します。数値パラメータを指定すると、最後のワードではなく その数値が表すワードが挿入されます。

`M-_` `M-.` と同じです。

`M-*` アスタリスクがワードの最後に付加され、ファイル名が展開されます。

`M-ESC` ファイル名の補完を行います。現在のワードにアスタリスクを付加したものと 一致するすべてのファイル名の最長の 前方一致部分で、現在のワードを 置き換えます。一致するものが 1 つしかない場合、ファイルがディレクトリなら / を付加し、ファイルがディレクトリでないときは空白を付加します。

`M-=` 現在のワードのあとにアスタリスクを付加すれば一致する ファイル名をリスト表示します。

`^U` 次のコマンドの数値パラメータを 4 倍します。

`\` 次の文字をエスケープします。編集用文字、ユーザーが設定した消去文字、抹消文字、割り込み文字 (通常は `^?`) は、その前に \ を指定すれば、コマンド行または検索文字列に入力できます。\ は、次の文字の編集機能 (もしあれば) を無効にします。

`^V` シェルのバージョンを表示します。

M-#	# を行の先頭に挿入後、行を実行します。これによりコメントを履歴ファイルに挿入することができます。

vi 編集モード

2 つの処理モードがあります。初期状態では、コマンドを入力すると、入力モードになります。編集するときは、エスケープキー (ESC(033)) を入力すれば制御モードになるので、訂正が必要な箇所にカーソルを移動し、必要に応じて文字やワードを挿入または削除することができます。大半の制御コマンドでは、コマンドの先頭に繰り返し数を指定できます。

多くのシステムでは、vi モードにいる時ははじめに標準処理モード (行入力モード) になっています。端末の回線速度が 1200 ボー以上であり、しかも制御文字を含んでいるか、または プロンプトが表示されてから 1 秒以下の経過時間であれば、コマンドは再び表示されます。ESC 文字を入力することにより、コマンドの残りについての標準処理を終了します。このときユーザーはコマンド行を変更できます。この機構により、標準処理モードは、raw モード (文字入力モード) の先行入力表示という利点を持ちます。

viraw オプションも設定すると、端末は標準処理モードを常に無効にすることができます。このモードは、行の終わりの区切りを示す代替記号を 2 つサポートしていないシステムでは暗に設定されています。ある種の端末においては便利な機能です。

入力編集コマンド

デフォルトでは、エディタは入力モードになります。

erase	(stty(1) コマンドで定義するユーザー定義消去文字、通常は ^H または #) 直前の文字を削除します。
^W	直前の、ブランクで区切られたワードを削除します。
^D	シェルを終了します。
^V	次の文字をエスケープします。編集用の文字、ユーザーの定義した消去文字または抹消文字は、その前に ^V を入力すれば、コマンド行または検索文字列に入力できます。^V は、次の文字の編集機能 (もしあれば) を無効にします。
\	次の *erase* または末消文字をエスケープします。

移動編集コマンド

次のコマンドはカーソルを移動させます。

[*count*]l	カーソルを 1 文字分右に移動します。
[*count*]w	1 つ先の英数字のワードに カーソルを移動します。
[*count*]W	ブランクがあとに続く次のワードの 先頭にカーソルを移動します。
[*count*]e	カーソルを ワードの終わりに移動します。
[*count*]E	ブランクで区切られている直前のワード の終わりにカーソルを移動します。
[*count*]h	カーソルを 1 文字分左に移動します。
[*count*]b	カーソルを 1 つ前のワードに移動します。

[*count*]B	ブランクで区切られている 直前のワードにカーソルを移動します。
[*count*]|	カーソルを *count* で示すカラムへ移動します。
[*count*]f*c*	現在の行において文字 *c* が次に現れる位置にカーソルを移動します。
[*count*]F*c*	現在の行において文字 *c* が前に現れる位置にカーソルを移動します。
[*count*]t*c*	f と h を連続して実行した場合と同じ結果です。
[*count*]T*c*	F と l を連続して実行した場合と同じ結果です。
[*count*];	直前の単一文字検索コマンドの f、F、t、または T を *count* の数だけ繰り返します。
[*count*],	直前の単一文字検索コマンドを *count* の数だけ逆方向に行います。
0	カーソルを、行の始めまで移動します。
^	カーソルを、行における最初のブランク以外の文字まで移動します。
$	カーソルを、行の終わりまで移動します。
%	現在の位置にある括弧記号 (、)、{、}、[、] に対応する括弧記号にカーソルを移動します。現在の文字がいずれの括弧でもない場合、現在の行を前方向に検索し、最初に現れた括弧に対応する括弧の位置に移動します。

検索編集コマンド　以下のコマンドはコマンド履歴を使用します。

[*count*]k	直前のコマンドの履歴を取り出します。k を入力するたびに、1 つずつ逆方向に コマンドの履歴を取り出します。
[*count*]-	k と同じです。
[*count*]j	次のコマンドを取り出します。j を入力するたびに、1 つずつ順方向に コマンドの履歴を取り出します。
[*count*]+	j と同じです。
[*count*]G	*count* という番号のコマンドの履歴を取り出します。省略時は最も古いコマンドの履歴を取り出します。
/*string*	履歴をさかのぼって、*string* が示す文字列を含むコマンドを検索します。*string* の終わりは復帰改行 (RETURN または NEWLINE) で示します。*string* の先頭に ^ を付加すると、その文字列で始まっている コマンドだけを検索します。*string* が NULL の場合は、直前に指定された文字列を使用します。
?*string*	/ と同じです。ただし、検索は順方向になります。

n	直前の / または ? コマンドで指定した文字列と 次に一致するコマンドの履歴を検索します。
N	直前の / または ? コマンドで指定した文字列と 次に一致するコマンドの履歴を逆方向に検索します。

テキスト変更編集コマンド

以下のコマンドは行を変更します。

a	入力モードにして、現在の文字のあとにテキストを入力します。
A	行の終わりにテキストを追加します。$a と同じです。
c[*count*]*motion*	[*count*]c*motion* 現在の文字から *motion* によりカーソルが移動する先までの文字を削除し、入力モードにします。*motion* が c の場合、行全体を削除し入力モードにします。
C	現在の文字から行の終わりまでを削除し、入力モードにします。c$ と同じです。
[*count*]s	*count* で指定した数の文字を削除して 入力モードにします。
S	cc と同じです。
D	現在の文字から行の終わりまでを削除します。d$ と同じです。
d[*count*]*motion*	[*count*]d*motion* 現在の文字から *motion* によりカーソルが移動する先までの文字を削除します。*motion* が d の場合、行全体を削除します。
i	入力モードにして、現在の文字の前にテキストを挿入します。
I	行の先頭にテキストを挿入します。0i と同じです。
[*count*]P	カーソルの前に、直前のテキスト変更を挿入します。
[*count*]p	カーソルのあとに、直前のテキスト変更を挿入します。
R	入力モードにして、スクリーン上の文字を重ね打ちした文字に 置き換えます。
[*count*]r*c*	現在位置から始まる *count* 個の文字を *c* に置き換え、カーソルを前進させます。
[*count*]x	現在の文字を削除します。
[*count*]X	カーソル直前の文字を削除します。
[*count*].	直前のテキスト変更コマンドを繰り返します。

[*count*]≈	現在のカーソル位置から始まる *count* 個の文字を、大文字の場合は小文字に、小文字の場合は大文字に変換して、カーソルを前進させます。
[*count*]_	直前のコマンドの *count* 個のワードを追加し、入力モードにします。*count* を省略すると、最後のワードを使用します。
*	* を現在のワードのあとに付けたしたものと見なし、ファイル名を生成しようとします。一致するものが見つからない場合、ベルを鳴らします。見つかった場合は、ワードを一致した文字列で置換し、入力モードにします。
\	ファイル名の補完を行います。現在のワードにアスタリスクを付加したものと 一致するすべてのファイル名の最長前方一致部分で、現在のワードを置き換えます。一致するものが 1 つしかない場合、ファイルがディレクトリであれば / を付加し、ファイルがディレクトリでなければ 空白を付加します。

他の編集コマンド　その他のコマンドを以下に説明します。

[*count*]y*motion* y[*count*]*motion*	現在の文字から *motion* によりカーソルが移動する先までの文字を、削除用バッファに入れます。テキストとカーソルは変わりません。
Y	現在の位置から行の終わりまでの文字をバッファに入れます。y$ と同じです。
u	直前のテキスト変更コマンドを取消 (undo) します。
U	現在の行で実行されたテキスト変更コマンドすべてを取消 (undo) します。
[*count*]v	fc -e ${VISUAL:-${EDITOR:-vi}} *count* コマンドを入力バッファに戻します。*count* を省略すると、現在の行を使用します。
^L	復帰改行して現在の行を表示します。制御モードでのみ有効です。
J	復帰改行 (NEWLINE) 。モードと無関係に現在の行を実行します。
M	復帰改行 (RETURN) 。モードと無関係に現在の行を実行します。
#	コマンドの先頭文字が # であれば、その # およびそのあとの 復帰改行 (RETURN) に続く # を削除します。先頭文字が他の文字であれば、# を各行の先頭に挿入後、行を送ります。現在の行を注釈として履歴に挿入

したり、履歴ファイル中にある以前の注釈付きコマンドから 注釈を削除したりする際に便利です。

= 現在のワードのあとにアスタリスクを付加すれば、一致する ファイル名をリスト表示します。

@*letter* 別名リストで _*letter* という名前の別名を検索します。この名前の別名が定義されていれば、処理待ちの入力待ち行列上にその値を挿入します。

特殊コマンド

以下の単純コマンドは、シェルプロセス中で実行されます。入出力のリダイレクトが可能です。特に断わりのないかぎり、出力はファイル記述子 1 上に書き込まれ、構文エラーがなければ終了ステータスは 0 です。1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、以下のような特殊な処理を受けます。

1. コマンドが完了しても、コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、それを含むスクリプトは中止されます。
4. 変数代入形式で、** を先頭に持つコマンドに続くワードは、変数代入と同一の規則で展開されます。つまり、チルド置換は = 符号のあとに実行され、ワード分割とファイル名生成は実行されません。

* : [*arg* . . .]
パラメータの展開だけを行います。

* . *file* [*arg* . . .]
`file` 全体を読み取ってからコマンドを実行します。コマンドは現在のシェル環境において実行されます。`PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。引数の *arg* は (指定されていれば) 定位置パラメータになります。引数を指定しないと定位置パラメータは変更されません。終了状態は、最後に実行されたコマンドの終了状態です。

** `alias` [`-tx`] [*name*[=*value*]] . . .
引数なしの場合、このコマンドは標準出力上に *name=value* という形式の別名のリストを表示します。*value* が指定された名前に対しては別名を定義します。*value* の後方に空白があると、次のワードが別名置換指定かどうかをチェックします。`-t` フラグは、検索済みの別名を設定またはリスト表示します。検索済み別名の値は、指定した *name* に対応する完全パス名になります。`PATH` の値を再設定すると この値は未定義になりますが、別名は検索済みのままです。`-t` フラグを省略すると、*value* が指定されていない引数リスト内の各 *name* について、別名の名前と値を表示します。`-x` フラグは、エクスポートされた別名を設定または表示します。エクスポートされた別名は、名前で起動されるスクリプト用に定義されます。*name* が指定されているが、*value* は指定されておらず、*name* に対しての別名も定義されていない場合は、終了ステータスは ゼロ以外になります。

bg [%*job*. . .]
このコマンドが有効なのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をバックグラウンドで実行します。*job* が省略された場合は、現在のジョブをバックグラウンドで実行します。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。

* break [*n*]
for ループ、while ループ、until ループ、または select ループがあれば終了します。*n* を指定すると、*n* レベル分だけ ループを終了します。

* continue [*n*]
for ループ、while ループ、until ループ、または select ループの次の繰り返しを実行します。*n* を指定すると、*n* 番目のループから実行します。

cd [*arg*]
cd *old new*
このコマンドは上記 2 つの形式の いずれかで入力します。第 1 の形式は、現在のディレクトリを *arg* に変更します。*arg* が – の場合、ディレクトリを直前のディレクトリに変更します。シェル変数 HOME の値がデフォルトの *arg* になります。PWD 変数は、現在のディレクトリに設定されます。CDPATH というシェル変数は、*arg* を含むディレクトリの検索パスを定義します。ディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定) 。なお、現在のディレクトリは 空のパス名で指定します。このパス名は、等号の直後か、パスリスト内にある区切り文字のコロンの間に指定します。*arg* の先頭文字が / の場合、検索パスは使用しません。それ以外の場合は、パス中の各ディレクトリで *arg* を検索します。

cd の第 2 の形式は、PWD 中の現在のディレクトリ名における *old* という文字列を *new* という文字列に置換し、この新規のディレクトリへ変更しようとします。cd コマンドは rksh では実行できません。

command [-p] [*command_name*] [argument . . .]
command [-v -V] *command_name*
command ユーティリティは、シェルに対して、シェル関数検索を抑止し 引数を単純コマンドとして扱うようにします。-p フラグは、全標準ユーティリティを検索できる PATH のデフォルト値を使って、コマンド検索を行います。-v フラグは、標準出力に文字列を書き出します。この文字列は、現在のシェル実行環境でシェルが *command_name* を呼び出すために使用するパス名またはコマンドを表します。-V フラグも標準出力に文字列を書き出します。この文字列は、*command_name* オペランドに指定された名前を、シェルが現在のシェル実行環境で どのように解釈するかを表します。

echo [*arg* . . .]
このコマンドの使用法と説明については、echo(1) を参照してください。

* eval [*arg* . . .]
引数をシェルへの入力として読み取り、生成されるコマンドを実行します。

* exec [*arg* . . .]
arg を指定すると、このシェルの代わりに、引数で指定されたコマンドを (新規プロセスは生成せずに) 実行します。入出力引数が指定可能で、現在のプロセスに影

響を及ぼす場合があります。引数を指定しない場合は、ファイル記述子が、入出力リダイレクトリストの指定どおりに 変更されることになります。この場合、この機能によりオープンされた 2 より大きい番号のファイル記述子は、別のプログラムを起動するとクローズされます。

* `exit` [*n*]
呼び出し元のシェル、またはシェルスクリプトを *n* で指定した終了ステータスで終了させます。具体的には、指定した値の最下位 8 ビットが終了ステータスの値となります。*n* を省略すると、最後に実行されたコマンドの終了ステータス がシェルの終了ステータスになります。トラップ実行中に `exit` が発生した場合、ここで言う最後に実行されたコマンドとは、トラップ呼び出し直前に実行されたコマンドを指します。なお、`ignoreeof` オプション (後述の `set` を参照) が有効になっているシェルを除き、EOF を検出した場合もシェルが終了します。

** `export` [*name*[=*value*]] . . .
指定された *name* に対し、あとで実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

`fc` [`-e` *ename*] [`-nlr`] [*first* [*last*]]
`fc -e -` [*old*=*new*] [`command`]
第 1 の形式は、端末から最近入力された `HISTSIZE` 個のコマンドの中から、*first* から `last` までの範囲のコマンドを選択します。*first* と `last` の両引数は、数値または文字列で指定できます。文字列の場合、その文字列で始まる最新のコマンドを見つけます。負の数値は、現在のコマンド番号からのオフセットとなります。`-l` フラグを指定すると、標準出力上にコマンドをリスト表示します。`-l` を指定しないと、これらのキーボードコマンドの入ったファイル上で *ename* というエディタプログラムを起動します。*ename* が省略されていると、変数 `FCEDIT` (デフォルトは `/bin/ed`) の値をエディタとして使用します。編集が完了すると、編集されたコマンドを実行します。`last` を省略すると、*first* と同一値に設定されます。*first* を省略すると、デフォルトは、編集については直前のコマンドに、リスト表示については –16 になります。`-r` フラグはコマンドの順序を逆にします。`-n` フラグはリスト表示時にコマンド番号の出力を抑止します。第 2 の形式では、*old=new* 置換実行後に `command` が再実行されます。`command` 引数を指定しない場合、一番最後に入力したコマンドが実行されます。

`fg` [`%`*job*. . .]
このコマンドが有効なのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をフォアグラウンドで実行します。*job* が指定されないと、現在のジョブをフォアグラウンドで実行します。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。

`getopts` *optstring name* [*arg* . . .]
arg が正当なオプションを示しているかをチェックします。*arg* を省略すると、定位置パラメータが使用されます。オプション引数は + または – で始まります。+ または – 以外の文字で始まっているオプション、あるいは – – 引数があると、オプションの終わりとみなされます。*optstring* には、`getopts` が認識する文字を記述します。文字のあとに : が続く場合、そのオプションには引数があるとみなされます。オプションと引数とはブランクで区切ることができます。次の *arg* のインデックスは `OPTIND` に格納されます。オプション引数がある場合は `OPTARG` に格納されます。*optstring* 内で先頭に : がある場合は、`getopts` は無効なオプション文

字を OPTARG に格納し、*name* を ? (未定義のオプションが指定された場合) または : (必要なオプション引数が省略されている場合) に設定します。: が先頭にない場合には、getopts はエラーメッセージを表示します。オプションがなくなると、終了ステータスはゼロ以外になります。使用法と説明については、getoptcvt(1) を参照してください。

hash [*name* . . .]
シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。-r オプションを指定すると、シェルは記憶したすべての位置を破棄します。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。*Hits* は、シェルプロセスによってコマンドが呼び出された回数を表します。*Cost* は、検索パスのコマンドを見つけるのに必要な作業領域です。コマンドが検索パスの「相対」ディレクトリにある場合、そのディレクトリの変更後に そのコマンドが格納された位置が再計算されます。この再計算が行われる予定のコマンドに対しては、*Hits* 情報の隣にアスタリスク (*) が示されます。*Cost* の値は、再計算が行われるたびに増加されます。

jobs [-lnp] [%*job* . . .]
指定された各々のジョブに関する情報を一覧表示します。*job* 引数を省略すると、活動中のジョブすべてに関する情報を一覧表示します。-l フラグは、通常の情報に加えてプロセス ID も表示します。-n フラグは、前回通知を受けたあとに停止または終了した ジョブだけを表示します。-p フラグは、プロセスグループだけを表示します。*job* の記述形式についての説明は、「ジョブ」の項と jobs(1) を参照してください。

kill [-*sig*] %*job* . . .
kill [-*sig*] *pid* . . .
kill -l
TERM (終了) シグナルまたは指定されたシグナルのいずれかを、指定されたジョブまたはプロセスに送信します。シグナルは、番号または名前 (signal(3HEAD) の場合と同様に SIG という接頭辞を取り除いたもの。ただし、SIGCHD は CHLD という名前となる) で指定します。送信するシグナルが TERM (終了) または HUP (ハングアップ) の場合、停止中のジョブまたはプロセスには CONT (継続) シグナルを送信します。*job* という引数は、活動中のジョブではないプロセスのプロセス ID を指定することもできます。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。第 2 の形式の kill -l は、シグナル番号とシグナル名をリスト表示します。

let *arg* . . .
各 *arg* は、評価の対象となる個々の算術式を表します。評価の方法については、前述の「算術的評価」の項を参照してください。終了ステータスは、最後の式の値が 0 の場合には 1 で、0 以外の場合は 0 です。

login *argument* . . .
exec /bin/newgrp *arg* . .. と同じです。使用法と機能説明については login(1) を参照してください。

* newgrp [*arg* . . .]
exec /bin/newgrp *arg* . . . と同じです。

`print` [`-Rnprsu`[*n*]] [*arg* . . .]
シェルの出力機構です。フラグを省略した場合、あるいは - または - - フラグを指定した場合には、`echo`(1) で述べるように標準出力上に引数を表示します。出力ファイルが書き込み用にオープンされて いない場合を除いて、終了ステータスは `0` となります。

`-n`	復帰改行 (NEWLINE) の出力を抑止します。
`-R` | `-r`	(raw モード) `echo` のエスケープ規則を無視します。`-R` オプションは、`-n` を除く後続の引数およびオプションすべてを表示します。
`-p`	標準出力の代わりに `|&` で生成されたプロセスのパイプ上に引数を出力します。
`-s`	標準出力の代わりに 履歴ファイル上に引数を書き込みます。
`-u` [*n*]	出力を格納するファイル記述子番号を、1 桁の数値 *n* で指定します。デフォルトは `1` です。

`pwd`
`print -r - $PWD` と同じです。

`read` [`-prsu`[*n*]] [*name*`?`*prompt*] [*name* . . .]
シェルの入力機構です。1 つの行を読み取り、IFS が示す文字を区切り文字として使用して、行の内容をいくつかのフィールドに分割します。エスケープ文字 (\) は、次の文字の特別な意味または 行の継続に関する意味を取り除くために使用します。`-r` で指定する raw モードでは、\ が持つこの特殊な意味は無視されます。第 1 フィールドを 1 番目の *name* に、第 2 フィールドを 2 番目の *name* に、という順番で割り当てていき、余ったフィールドがあれば最後の *name* に割り当てます。`-p` オプションは、シェルが `|&` を使用して生成したプロセスの入力パイプから入力行を取り出します。`-s` フラグは、入力をコマンドとして履歴ファイルに保存します。`-u` フラグは、読み取り元となるファイル記述子番号を 1 桁の数値 *n* で指定します。ファイル記述子は、`exec` という特殊コマンドでオープンできます。*n* のデフォルト値は `0` です。*name* を省略すると、`REPLY` の値をデフォルトとして使用します。入力ファイルが読み込み用にオープンされていない場合と EOF に到達した場合を除き、終了ステータスは `0` です。`-p` オプションが指定されていて EOF を検出すると、このプロセスをクリアし別のプロセスを作成可能にします。最初の引数が `?` を含んでいると、シェルが対話型のとき、このワードの残りを標準エラーに対する プロンプトとして使用します。EOF に到達しないかぎり、終了ステータスは `0` です。

** `readonly` [*name*[=*value*]] . . .
name に「読み取り専用」のマークを付け、これらの名前が後続の割り当てでは変更できないようにします。

* `return` [*n*]
シェル関数または　. スクリプトを、*n* で指定された戻り値で 呼び出し側スクリプトに戻します。*n* で指定した値の最下位 8 ビットが戻り値となります。*n* を省略すると、戻り値は最後に実行された コマンドの戻り値になります。関数や　. スクリプト実行中以外で `return` を起動すると、結果は `exit` と同一になります。

`set [ ±abCefhkmnopstuvx ] [ ±o` *option* `]. . . [ ±A` *name* `] [` *arg* `. . . ]`
このコマンドのフラグの意味は以下のとおりです。

`-A` 配列の代入。*name* で示される変数の設定を解除し、*arg* リストから順々に値を割り当てます。`+A` は、最初の変数設定を解除しません。

`-a` 定義される後続の変数すべてを自動的にエクスポートします。

`-b` シェルに対し、バックグラウンドジョブの終了を ユーザーに非同期に通知するよう要求します。以下のメッセージが標準エラー出力に書き出されます。

`"[%d]%c %s%s\n",` <*job-number*>, <*current*>, <*status*>, <*job-name*>

個々のフィールドの意味は次のとおりです。

<*current*> 文字 + は、`fg` または `bg` ユーティリティ用のデフォルトとして用いるジョブを表します。このジョブは、*job_id* `%+` または `%%` を使って指定することもできます。文字 – は、現在のデフォルトジョブが終了したときに デフォルトとなるジョブを表します。このジョブは、*job_id* `%–` を使って指定することもできます。その他のジョブには、このフィールドは空白文字となります。+ が識別できるジョブは最大 1 つで、– が識別できるジョブも最大 1 つです。中断中のジョブがあると、現在のジョブも中断中となります。複数のジョブが中断中だと、1 つ前のジョブも中断中になります。

<*job-number*> `wait`、`fg`、`bg`、`kill` の各ユーティリティ用にプロセスグループを識別するのに用いる番号です。これらのユーティリティを使用する際、ジョブ番号の先頭に `%` を付加してジョブを識別できます。

<*status*> 未定義です。

<*job-name*> 未定義です。

シェルがジョブの終了をユーザーに伝えるとき、そのジョブのプロセス ID を現在のシェル実行環境内のリストから削除することがあります。非同期の通知をデフォルトにすることはできません。

`-C`
シェルのリダイレクト演算子 > によって既存のファイルが上書きされないようにします。リダイレクト演算子 >| は、個々のファイルに対し、`noclobber` オプションに優先して有効となります。

`-e`
コマンドの終了ステータスが 0 でない場合、`ERR` トラップ (設定されていれば) を実行し、終了します。このモードは、プロファイル読み取り時は無効です。

`-f`
ファイル名の生成を無効にします。

-h
各コマンドは、最初に検出された時点で、検索済み別名になります。

-k
コマンド名に先行するものだけでなく、すべての変数代入引数をコマンドの環境に格納します。

-m
バックグラウンドジョブを個別のプロセスグループで実行し、完了時にメッセージを 1 行表示します。バックグラウンドジョブの終了ステータスは 完了メッセージで報告されます。ジョブ制御を備えたシステムでは、このフラグは 対話型シェルに対して自動的に有効になります。

-n
コマンドを読み取り、構文エラーがないかチェックします。実行は行いません。対話型シェルに対しては無視されます。

-o
このフラグのあとに指定する引数は、以下のオプション名のいずれかです。

allexport	-a と同じです。
errexit	-e と同じです。
bgnice	バックグラウンドジョブをすべて低い優先度で実行します。これはデフォルトモードです。
emacs	コマンド入力用に、emacs 形式のインラインエディタを起動します。
gmacs	コマンド入力用に、gmacs 形式のインラインエディタを起動します。
ignoreeof	EOF を検出してもシェルは終了しません。終了させるには exit コマンドを使用する必要があります。
keyword	-k と同じです。
markdirs	ファイル名生成によって生成されるディレクトリ名には、すべて最後に / を付加します。
monitor	-m と同じです。
noclobber	> によるリダイレクトが存在するファイルを切り捨てないようにします。このオプションが有効なとき、ファイルを切り捨てるには >| 指定が必要です。このオプションは -C と同等です。
noexec	-n と同じです。
noglob	-f と同じです。
nolog	履歴ファイルに関数定義を保存しません。
notify	-b と同等です。
nounset	-u と同じです。

privileged	-p と同じです。
verbose	-v と同じです。
trackall	-h と同じです。
vi	vi 形式のインラインエディタの挿入モードになります。033 というエスケープ文字を押すと、挿入モードから制御モードに変わります。リターンキーで行を送信します。
viraw	各文字を vi モードで入力されたときと同様に処理します。
xtrace	-x と同じです。

オプション名を指定しない場合、-o は現在のオプション設定を表示します。

-p
$HOME/.profile ファイルを処理しないようにし、ENV ファイルの代わりに /etc/suid_profile ファイルを使用します。このモードは、実効ユーザー ID が実ユーザー ID と等しくないとき、また実効グループ ID が実グループ ID と等しくないときには 必ず有効になります。このモードを無効にすると、実効ユーザー ID が実ユーザー ID に、実効グループ ID が実グループ ID にそれぞれ設定されます。

-s
定位置パラメータを辞書編集方式の順にソートします。

-t
コマンド 1 つを読み取って実行し、終了します。

-u
置換を行う際に、設定されていないパラメータをエラーとして扱います。

-v
シェルへの入力行を読み取り時に表示します。

-x
コマンドとその引数を実行時に表示します。

-
-x フラグと -v フラグを無効にし、フラグに対する引数の検査を停止します。

- -
どのフラグも変更しません。このフラグは、$1 を - で始まる値に設定する際に便利です。このフラグのあとに引数がない場合、定位置パラメータが設定解除されます。

- の代わりに + を使用すると、これらのフラグが無効になります。これらのフラグはシェル起動時にも使用できます。現在設定されているフラグは、$- で見ることができます。-A を指定しないかぎり、残りの引数は定位置パラメータとなり、$1 $2 . . . に順番に割り当てられます。引数を 1 つも指定しない場合には、すべての変数の名前と値を標準出力上に表示します。

* `shift` [*n*]
  `$`*n*`+1 . . .` の定位置パラメータを `$1 . . .` という名前に変更します。*n* のデフォルト値は 1 です。*n* に指定できる値は、評価結果が `$#` 以下の負でない数になる算術式です。

`stop%`*jobid* `. . .`

`stop` *pid . . .*
  `stop` は、*jobid* (ジョブ ID 番号) を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (`ps`(1)参照) 。

`suspend`
  現在のシェルがログインシェルでない場合、その実行を中断します。

`test` *expression*
  条件式を評価します。使用法と機能説明については 前述の「条件式」の項と `test`(1) を参照してください。

* `times`
  シェルおよびシェルから実行されたプロセスの、ユーザー時間およびシステム時間の累計値を表示します。

* `trap` [*arg sig . . .*]
  arg は、*sig* が示すシグナルをシェルが受信したときに読み取られ、実行されるコマンドです。*arg* は、トラップ設定時と トラップ取り出し時に 1 度ずつ検索されます。*sig* は、シグナル番号またはシグナル名を指定します。`trap` コマンドは、シグナル番号の順序で実行されます。現在のシェルで無視されているシグナル番号にトラップを 設定しようとしても無効となります。*arg* が – の場合、シェルは各 *sig* デフォルト値を再設定します。*arg* が NULL 文字列の場合、シェルは指定された各 *sig* が発生してもそれを無視します。ただし 対応する *sig* が 1 つでも発生した場合、*arg* はシェルにより実行されます。トラップのアクションは、以前のアクション (デフォルトまたは明示的に設定されたもの) より優先して用いられます。トラップのアクション完了後、`$?` の値はトラップが呼び出されたときの値となります。

  sig は `EXIT` または `0` (`EXIT` と同義) 、またはシンボル名を使って指定したシグナルから接頭辞 `SIG` を除いたものです。たとえば `HUP`、`INT`、`QUIT`、`TERM` などです。*sig* が `0` または `EXIT` で、`trap` 文がある関数内部で実行された場合、関数終了後に *arg* の示すコマンドが実行されます。*sig* が `0` または `EXIT` で、トラップが関数の外側で設定されている場合、シェルの終了時に *arg* の示すコマンドが実行されます。*sig* が `ERR` の場合、コマンドが ゼロ以外の終了ステータスで終わると必ず *arg* が実行されます。*sig* が `DEBUG` の場合、各コマンドのあとに *arg* が実行されます。

  シェルが `EXIT` に対してトラップを実行する環境は、`EXIT` のトラップを得る以前に実行された最後の コマンドの直後の環境と同じです。

  トラップが呼び出されるたびに、*arg* 引数は以下のものと同じように処理されます。

```
eval "$arg"
```

非対話型シェルの開始時に無視されたシグナルは、トラップもリセットもできません。ただしトラップやリセットを 試みても、エラーは報告されません。対話型のシェルは、呼び出し時に無視されたシグナルを リセットしたりキャッチしたりできます。トラップは、そのシェルの動作中は、他の trap コマンドにより明示的に変更されないかぎり、有効であり続けます。

サブシェルを立ち上げたときは、トラップはデフォルトに設定されます。ただしこれは、サブシェル内では コマンドを使って新たなトラップを設定できない、ということではありません。

引数なしの trap コマンドは、各シグナルに対応したコマンドの一覧を標準出力に書き出します。その形式は次のとおりです。

```
trap — %s %s  . . . <arg>, <sig> . . .
```

シェルは、この出力の形式 (引用符の使用法も含む) を直し、同様のトラップ結果をもたらすようなコマンドとしてシェルに再入力できる形式にします。次の例を見てください。

```
save_traps=$(trap) . . . eval "$save_traps"
```

トラップの名前や番号が正しくないと、ゼロ以外の終了ステータスが 返されます。正しければ 0 が返されます。対話型のシェルも非対話型のシェルも、シグナル名やシグナル番号が誤りでも構文エラーとはならず、シェルも異常終了しません。

ジョブがフォアグラウンドプロセスを待っている間は、トラップは処理されません。このため CHLD に対するトラップは、フォアグラウンドジョブが終了するまで実行されません。

type *name* . . .
name をコマンド名として使用した場合に どのように解釈されるかを表示します。

** typeset [±HLRZfilrtux[*n*]] [*name*[=*value*]] . . .
シェル変数と関数の属性と値を設定します。関数内で typeset を実行すると、*name* が示す変数の新しいインスタンスが生成されます。関数が完了すると、その変数の値と型が復元されます。このコマンドには、以下の属性を指定できます。

-H　このフラグは UNIX 以外のマシン上で、UNIX とホスト名ファイルとのマッピング情報を提供します。

-L　左詰めを行い、先行するブランクを *value* から取り除きます。*n* は、ゼロ以外であればフィールドの幅を定義します。0 の場合、フィールドの幅は最初に代入される値の幅で決定されます。変数に値を代入したとき、フィールド幅より短ければ右側にブランクが詰められ、長ければ切り捨てられます。-Z フラグも指定されていれば、先行する 0 を削除します。-R フラグは無効になります。

-R　右詰めを行い、先行するブランクを挿入します。*n* は、ゼロ以外であればフィールドの幅を定義します。0 の場合、フィールドの幅は最初に代入される値の幅で決定されます。変数に値を代入したとき、フィールド幅より短ければ左側にブランクが詰められ、長ければ終端が切り捨てられます。-L フラグは無効になります。

`-Z` 最初の、ブランクでない文字が数字で、さらに `-L` フラグが設定されていない場合、右詰めを行い先頭に 0 を詰めます。*n* は、ゼロ以外であればフィールドの幅を定義します。`0` の場合、フィールドの幅は最初に代入される値の幅で決定されます。

`-f` 名前は、変数名ではなく関数名を指します。代入は行われません。このフラグと共に指定できる他のフラグは、`-t`、`-u`、`-x` だけです。`-t` フラグは、この関数の実行トレースを有効にします。`-u` フラグは、この関数に「未定義」を示すマークを付けます。関数が参照されると、関数定義を見つけるために `FPATH` 変数が検索されます。`-x` フラグを指定すると、名前で呼び出されるシェル手続き全体で関数定義が有効になります。

`-i` パラメータを整数とします。これにより算術演算が高速化されます。*n* は、ゼロ以外であればその値を底として定義します。`0` の場合、最初の代入で底が決定されます。

`-l` 大文字をすべて小文字に変換します。大文字への変換を示す `-u` フラグを無効にします。

`-r` 指定された *name* を読み取り専用にします。あとの代入でこれらの名前を変更できないようにします。

`-t` 変数にタグを付けます。タグはユーザーが定義可能で、シェルに対して特別の意味を持ちません。

`-u` 小文字をすべて大文字に変換します。小文字への変換を示す `-l` フラグを無効にします。

`-x` 指定された *name* に対し、あとで実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

`-i` 属性は、`-R`、`-L`、`-Z`、`-f` と同時に指定することはできません。

– の代わりに + を使用すると、これらのフラグは無効になります。*name* 引数をまったく指定せずにフラグを指定すると、これらのフラグが設定されている変数の名前 (および選択により値も) が一覧表示されます。具体的には – を付加すれば名前と値が、+ を付加すれば名前だけが表示されます。*name* 引数とフラグを 1 つも指定しないと、すべての変数の名前と属性が表示されます。

`ulimit [ -HSacdfnstv ] [ limit ]`
資源の制限を表示または設定します。使用可能な資源の制限は以下に説明します。システムによっては、以下に挙げたすべての 資源の制限を提供していないこともあります。`limit` 引数を指定すると、制限値が設定されます。`limit` の値は、各資源に対応した単位 (後述) の数値、または `unlimited` という文字列です。`H` と `S` の両フラグは、資源に対して 強い制限と弱い制限のどちらを 設定するかを表します。強い制限値は、いったん設定したら あとで増加させることはできません。弱い制限値は、強い制限値を超えない範囲で 増加させることが可能です。`H` も `S` も省略すると、指定した制限値が強い制限と弱い制限の 両方に適用されます。`limit` 引数を省略すると、現在の資源制限値が表示されます。このとき、`H` が指定

された場合を除き、表示されるのは弱い制限値です。複数の資源を指定すると、値の前に制限する資源名と単位とが表示されます。

`-a` 現在の資源制限値をすべて表示します。

`-c` コアダンプ時の コアファイルのサイズをブロック (512 バイト) 単位で表します。

`-d` データ領域のサイズを K バイト単位で表します。

`-f` 子プロセスが書き込むファイルのサイズをブロック (512 バイト) 単位で表します。読み込むファイルにはサイズの制限はありません。

`-n` 最大ファイル記述子に 1 を加えた値を表します。

`-s` スタック領域のサイズを K バイト単位で表します。

`-t` 各プロセスが使用する秒数を表します。

`-v` 仮想記憶のサイズを K バイト単位で表します。

オプションをすべて省略すると、`-f` が指定されたものとみなします。

`umask` [`-S`] [*mask*]
ユーザーファイルの作成時のマスクを *mask* 引数が示す値に設定します (`umask`(2) を参照)。*mask* には、`chmod`(1) で説明する記号値または 8 進数を指定できます。記号値を指定すると、新しい `umask` 値は、*mask* を直前の `umask` 値の補数に適用した結果の補数になります。*mask* 引数を省略すると、マスクの現在の値を表示します。`-S` フラグは、シンボリック形式の出力を生成します。

`unalias` *name*`. . .`
name が示す別名を別名リストから削除します。

`unset` [`-f`] *name* `...`
name が示す変数の設定を解除します。つまり、それらの変数の値と属性を消去します。読み取り専用の変数は設定を解除できません。`-f` フラグが指定されていると、*name* 引数は関数名を表します。`ERRNO`、`LINENO`、`MAILCHECK`、`OPTARG`、`OPTIND`、`RANDOM`、`SECONDS`、`TMOUT`、および `_` の設定を解除すると、これらの変数の特殊な意味が削除されます。あとでこれらの変数に値を代入しても、特殊な意味は持ちません。

* `wait` [*job*]
job 引数で指定されたジョブの終了を待ち、その終了ステータスを報告します。*job* を指定しないと、現在実行中の子プロセスすべてを待ちます。待つ対象のプロセスの終了ステータスが、このコマンドの終了ステータスになります。*job* の記述形式についての説明は、「ジョブ」の項を参照してください。

`whence` [`-pv`] *name* `...`
指定された各 *name* について、それをコマンド名として使用した場合 どのように解釈されるかを示します。

`-v` フラグは、より詳細に表示します。

-p フラグは、*name* が別名、関数名、予約語の場合でも、それに対するパス検索を行います。

呼び出し

シェルを exec(2) で呼び出し、0 番目の引数 ($0) の最初の文字が – である場合、シェルをログインシェルとみなし、/etc/profile からコマンドを読み取り、次に、現在のディレクトリ内に .profile が存在するか、または $HOME/.profile がある場合、そのいずれかのファイルからコマンドを読み取ります。次に、環境変数 ENV に設定されている値をパラメータ置換することによって 指定されるファイルが存在すれば、そのファイルからコマンドを読み取ります。-s フラグが省略され、*arg* 引数が指定されている場合、最初の *arg* に対してパス検索を実行し、実行すべきスクリプトの名前を判別します。*arg* が示すスクリプトには読み取り権が必要で、setuid 設定と setgid 設定は無視されます。パス上でスクリプトが見つからない場合は、*arg* は組み込みコマンドまたは組み込み関数の名前を示しているものとして処理されます。次にコマンドは後述する方法で読み取られます。以下のフラグは、起動時にシェルによって解釈されます。

-c　このフラグが指定されると、*command_string* からコマンドを読み取ります。特殊パラメータ 0 の値は、*command_name* オペランドの値と残りの *arg* オペランドにある定位置パラメータ ($1、$2 など) から設定　されます。標準入力から読み取られるコマンドはありません。

-s　このフラグが指定された場合または引数が残っていない場合、標準入力からコマンドを読み取ります。前述の特殊コマンドの出力を除くシェル出力は、ファイル記述子 2 に書き出されます。

-i　このフラグが指定された場合または シェル入出力が端末に接続されている場合 (ioctl(2) で説明) 、このシェルは対話型となります。この場合、kill 0 が対話型シェルを終了しないように TERM を無視し、wait が割り込み可能になるように INTR を捕えて 無視します。いずれの場合も、シェルは QUIT を無視します。

-r　このフラグを指定すると、シェルは制限付きシェルになります。

他のフラグと引数については、前述の set コマンドの箇所で説明されています。

rksh の特記事項

rksh が設定するログイン名と実行環境の機能は、標準シェルの機能よりも制限を受けることになります。rksh の機能は、以下の動作ができない点を除き ksh と同じです。

- ディレクトリの変更 (cd(1) を参照)
- SHELL、ENV、または PATH の値の設定
- / を含むパス名またはコマンド名の指定
- 出力のリダイレクト (>、>|、<>、>>)
- グループの変更 (newgrp(1) を参照)

これらの制限は、.profile ファイルと ENV ファイルの解釈後に有効となります。

実行すべきコマンドがシェル手続きであることがわかると、rksh は ksh を起動し実行します。したがって一般ユーザーに対して、限られたコマンドのメニューを提供しながら、標準シェルの全機能を利用するシェル手続きを提供することが可能になります。この機構は、一般ユーザーが 同じディレクトリへの書き込み権と 実行権の両方を持ってはいないことを前提としています。

つまり、.profile の作者が、確実な設定処理を実行し ユーザーを適切なディレクトリ (おそらく、ログインディレクトリではない) に置くことにより、ユーザーの動作を完全に制御できるという点が、これらの規則の実際の効果となります。

システム管理者は、rksh で安全に起動できるコマンドのディレクトリ (つまり /usr/rbin) を設定することがよくあります。

エラー

構文エラーなどのエラーを検出すると、シェルは ゼロ以外の終了ステータスを返します。エラーがなければ、シェルは、最後に実行されたコマンドの終了ステータスを返します (前述の exit コマンドの説明を参照) 。シェルを非対話型で使用している場合、シェルファイルの実行は中止されます。シェルが検出する実行時エラーは、コマンド名と関数名、およびエラー状態を表示することにより報告されます。エラーが発生した行の番号が 1 より大きい場合、コマンド名または関数名のあとに角括弧 ([]) で囲んで行番号も表示します。

非対話型のシェルの場合、特殊組み込みユーティリティや他の種類のユーティリティがエラー状態を検出すると、シェルは診断メッセージを書き出し、以下の表に示すように終了します。

エラー	特殊組み込みユーティリティ	他のユーティリティ
シェル言語の構文エラー	終了する	終了する
ユーティリティの構文エラー		
(オプションまたはオペランド)	終了する	終了しない
リダイレクトのエラー	終了する	終了しない
変数割当のエラー	終了する	終了しない
展開エラー	終了する	終了する
コマンドが見つからない	該当せず	終了の場合あり
ドットスクリプトが見つからない	終了する	該当せず

展開エラーとは、シェルの展開時に発生するものです (たとえば ${x!y} のようなとき ! は演算子として正しくないのでエラー) 。ただし実装によっては、これらのエラーを展開時ではなく トークン化時に検出できるのであれば、構文エラーとして 扱うことも可能です。

上記の表で「終了する」 (または「終了の場合あり」) と示されているエラーが サブシェル内で発生した場合、サブシェル自身はゼロ以外のステータス で終了します (または終了する場合があります) が、サブシェルを含んでいるスクリプト自体は終了しません。

上記の表のすべての場合において、対話型のシェルは、診断メッセージを標準エラー出力に書き出すだけで終了はしません。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `ksh` と `rksh` の動作については、`largefile`(5) を参照してください。

終了ステータス

各コマンドには、他のシェルコマンドの動作に影響を与える可能性のある 終了ステータスが定義されています。ユーティリティを除く コマンドの終了ステータスは、本項内で説明します。また標準ユーティリティの終了ステータスは、それぞれの 対応する項で説明されています。

コマンドが見つからない場合、終了ステータスは `127` となります。コマンド名は見つかったが実行可能なユーティリティでは ない場合、終了ステータスは `126` となります。シェルを使わないでユーティリティを呼び出すアプリケーションは、これらの終了ステータスコードを使って同様なエラーを 報告してください。

ワードの展開中またはリダイレクト中にコマンドが失敗すると、その終了ステータスはゼロより大きい値となります。

特殊パラメータ `?` 付きの終了ステータスを報告する際、シェルは得られた終了ステータスの 8 ビットすべてを報告します。シグナルを受け取ったために終了したコマンドの終了ステータスは、`128` より大きな値となります。

ファイル

`/etc/profile`

`/etc/suid_profile`

`$HOME/.profile`

`/tmp/sh*`

`/dev/null`

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/rksh

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/ksh

属性タイプ	属性値
使用条件	SUNWxcu4

CSI	対応済み

関連項目

cat(1), cd(1), chmod(1), cut(1), echo(1), env(1), getoptcvt(1), jobs(1), login(1), newgrp(1), paste(1), ps(1), shell_builtins(1), stty(1), test(1), vi(1), dup(2), exec(2), fork(2), ioctl(2), lseek(2), pipe(2), ulimit(2), umask(2), wait(2), rand(3C), signal(3C), a.out(4), profile(4), attributes(5), environ(5), largefile(5), signal(3HEAD), XPG4(5)

Morris I. Bolsky and David G. Korn, *The KornShell Command and Programming Language*, Prentice Hall, 1989

警告

シェルスクリプトを setuid して使用することは避けてください。

注意事項

検索済み別名であるコマンドを実行し、そのあとで同一名のコマンドが、検索パスにおいて元のコマンドがあるディレクトリの前のディレクトリにインストールされた場合、シェルは元のコマンドの方を実行します。新しい方のコマンドを実行させたければ、alias コマンドの -t オプションを使用してください。

きわめて古いシェルスクリプトには、パイプ文字として | の他に ^ を許すものもあります。

複合コマンド内で fc 組み込みコマンドを使用すると、履歴ファイルからコマンド全体が消えます。

. file という組み込みコマンドは、いずれのコマンドを実行する場合でもその前に必ずファイル全体を読み取ります。したがって、ファイル内の alias コマンドと unalias コマンドは、ファイル内に定義されたどの関数にも適用されません。

存在しないコマンドのインタプリタを実行しようとするシェルスクリプトを、シェルが処理した場合、シェルは、シェルスクリプトが存在しないという間違った診断メッセージを返します。

名前 | rlogin – リモートログイン

形式 | **rlogin** [-8EL] [-e*c*] [-l *username*] *hostname*

機能説明 | rlogin は、使用している端末から *hostname* で指定したリモートホストへのリモートログインセッションを確立します。

ホスト名のリストは *hosts* データベースに格納されています。このデータベースは /etc/hosts ファイルと /etc/inet/ipnodes ファイル、 ネットワーク情報サービス (NIS) の hosts マップ、インターネットドメインネームサーバーのいずれかに含まれています。このうちのいくつかに含まれている場合もあります。 各ホストには、正式ホスト名 (データベースエントリに記載されている最初の名前) が 1 つずつ割り当てられていますが、このほかに 1 つ以上のニックネームを割り当てることもできます。rlogin コマンドの *hostname* 引数には、正式ホスト名とニックネームのどちらでも指定できます。

各リモートマシンには、信頼できるホスト名を記述した /etc/hosts.equiv ファイルを割り当てることができます。このファイルに記述されているホストとリモートマシンは、ユーザー名を共有します。このため、ローカルマシンとリモートマシンで同じユーザー名を使用しているユーザーは、リモートマシンの /etc/hosts.equiv ファイルに記述されているマシンから、パスワードを入力しないで rlogin を実行できます。 各ユーザーが、自分のホームディレクトリ中の .rhosts ファイルに、同様のリストを作成することもできます。.rhosts ファイルの各行に、ホスト名 (*hostname*) とユーザー名 (*username*) を空白文字で区切って記述してください。リモートユーザーの .rhosts ファイル内に指定したホスト (*hostname*) にログインしているユーザー (*username*) は、 パスワードを入力しなくてもリモートマシンにログインできます。 ローカルホスト名がリモートマシン上の /etc/hosts.equiv ファイル中に見つからず、ローカルユーザー名とホスト名がリモートユーザーの .rhosts ファイル中に見つからない場合、 リモートマシンはパスワードの入力を要求します。/etc/hosts.equiv ファイル、.rhosts ファイルには、*hosts* データベースに登録されている 正式ホスト名を指定してください。ニックネームは使用できません。

セキュリティを考慮するのであれば、 .rhosts ファイルの所有者はリモートユーザーかスーパーユーザーにする必要があります。

リモート端末のタイプは、ユーザーの TERM 環境変数によって指定されているローカル端末のタイプと同一でなければなりません。サーバーがサポートしている機能によっては、端末のサイズやウィンドウのサイズもリモートシステムにコピーされます。また、サイズの変更も、 リモートシステムに反映されます。 処理内容は常にリモート側にエコーされるので、遅延が発生している場合以外は、 リモートログイン処理であることを意識せずに操作できます。また CTRL-S や CTRL-Q を使ったフロー制御、 割り込み時の入出力のフラッシュも正しく処理されます。

オプション | 次のオプションを指定できます。

-8 — ネットワーク間で、7 ビットデータではなく 8 ビットデータを送信します。

-e*c* — リモートホストとの接続を解除する際のエスケープ文字として、標準の文字ではなく *c* を使用します。

`-E` すべての文字が、エスケープ文字として認識されなくなります。

`-l` *username* リモートログイン時のユーザー名として別の名前 (*username*) を指定します。このオプションを省略すると、ローカルユーザー名がリモートユーザー名になります。

`-L` litout モードでの `rlogin` セッションの実行を許可します。

エスケープシーケンス

端末から入力した行がチルド（~）で始まっている場合、その行は エスケープシーケンスとみなされます。`-e` オプションを使って、 エスケープ文字をチルド以外に変更することもできます。

`~.` リモートホストとの接続を切断します。 ローカルホストはリモート側に警告を出さずに切断を実行します。 この点で通常のログアウトとは異なります。

`~susp` ログインセッションを一時的に中断します。 この指定は、ジョブ制御機能を持つシェルを使用している場合にのみ有効です。 `susp` は中断用の文字であり、通常は CTRL–Z です (`tty`(1) を参照)。

`~dsusp` ログインの途中で入力を一時中断しますが、 出力は表示します。この指定は、ジョブ制御機能を持つシェルを使用している場合にのみ有効です。 `dsusp` は一時的中断用の文字であり、通常は CTRL–Y です (`tty`(1) を参照)。

オペランド

hostname *rlogin* がリモートログインセッションを確立するリモートマシンです。

ファイル

`/etc/passwd` ユーザーアカウントに関する情報が含まれています。

`/usr/hosts/*` コマンドの *hostname* バージョンです。

`/etc/hosts.equiv` ユーザー名を共有する、信頼できるホスト名が記述されています。

`/etc/nologin` マシンのシャットダウン中にログインしようとするユーザーへのメッセージです。

`$HOME/.rhosts` 個々のユーザー用のリストであり、ホスト名とユーザー名の組み合わせがエントリになっています。

`/etc/hosts` ホストのデータベースです。

`/etc/inet/ipnodes` ホストのデータベースです。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`rsh`(1)、`stty`(1)、`tty`(1)、`in.named`(1M)、`hosts`(4)、`ipnodes`(4)、`hosts.equiv`(4)、`nologin`(4)

診断

次のメッセージは、マシンがシャットダウン中で、ログインできないことを表します。

```
NO LOGINS: System going down in N minutes
```

注意事項

`hosts.equiv` ファイルに記述されているホストのセキュリティは、 ローカルホストと同程度に保護されている必要があります。この保護が十分でないホストがあると、システム全体のセキュリティが影響を受けます。

ネットワーク情報サービス (NIS) は、以前は Sun Yellow Pages (YP) として知られていました。 この 2 つは、名前が異なっているだけで、機能上は同じです。

この実装では TCP ネットワークサービス以外は使用できません。

名前 | rm, rmdir – ディレクトリのエントリの削除

形式 | **/usr/bin/rm** [-f] [-i] *file...*

/usr/bin/rm -rR [-f] [-i] *dirname...* [*file...*]

/usr/xpg4/bin/rm [-fiRr] *file...*

/usr/bin/rmdir [-ps] *dirname...*

/usr/bin/rm
/usr/xpg4/bin/rm

rm ユーティリティは、 *file* オペランドで示された各ディレクトリエントリを削除します。 *file* に書き込み権がなく、標準入力が端末の場合、 そのファイルに対するすべてのアクセス権(8 進数)が出力され、後ろに疑問符がつけられます。 これは、確認を求めるプロンプトです。 これに対する解答が y (yes) の場合にはそのファイルは削除され、 そうでない場合にはそのファイルは残されます。

file がシンボリックリンクの場合リンクは削除されますが、 参照するファイルまたはディレクトリは削除されません。 ユーザーがシンボリックリンクを削除する際、 そのディレクトリに書き込み権があれば、 シンボリックリンクに書き込み権は必要ありません。

file オペランドに複数のファイルが指定され、何らかの理由でそのうちの 1 つが削除できなかった場合、 rm は診断メッセージを標準エラー出力に書き出し、 そのファイルにはそれ以上の処理は行わず、 残りの *file* オペランドを処理します。

標準入力が端末でない場合、 rm は -f オプションが指定されている場合と同様に動作します。

/usr/bin/rmdir

rmdir ユーティリティは、 *dirname* オペランドで示されたディレクトリのエントリを削除します。 *dirname* は空のディレクトリを指している必要があります。

ディレクトリは指定された順序で処理されます。 したがって、あるディレクトリとそのディレクトリのサブディレクトリを 1 度の rmdir 呼び出しで削除したい場合、 サブディレクトリを親ディレクトリよりも前に指定して先に削除し、 親ディレクトリの削除のときにディレクトリが空になるよう にしてください。

オプション | 以下のオプションは、 /usr/bin/rm と /usr/xpg4/bin/rm で使用できます。

-r　引数リストにあるディレクトリおよび サブディレクトリを再帰的に削除します。 ディレクトリ中は空にされ、削除されます。 通常、ディレクトリにある書き込み保護された 任意のファイルの削除には、 ユーザーに確認を求めるプロンプトが出されます。ただし、 -f オプションが使用されている場合、または 標準入力が端末ではなく、しかも -i オプションが使用されていない場合、 書き込み保護されたファイルはプロンプトなしで削除されます。

このオプションが使用された場合に シンボリックリンクのエントリが存在しても、 再帰的処理はシンボリックリンク先には及びません。

書き込み保護された空でないディレクトリを削除しようとすると、 このコマンドは常に(-f オプションが使用されている場合でも)失敗に終わり、エラーメッセージが表示されます。

`-R` `-r` オプションと同じです。

/usr/bin/rm 以下のオプションは、 `/usr/bin/rm` でのみ使用できます。

`-f` ユーザーに確認を求めるプロンプトを出さずに、ディレクトリ内のすべてのファイル (書き込み保護のあるなしにかかわらず)を削除します。 しかし、書き込み保護されたディレクトリでは、 (たとえどんなアクセス権があっても)ファイルは決して削除されません。 また、それについてのメッセージは表示されません。 書き込み保護されたディレクトリを削除しようとすると、 このオプションはエラーメッセージを表示します。

`-i` 対話型。 このオプションを使用すると、 `rm` はファイルを削除する前に、確認を求めるプロンプトを出します。これは `-f` オプションより優先され、 標準入力が端末でない場合でもその効力を持続します。

/usr/xpg4/bin/rm 以下のオプションは、 `/usr/xpg4/bin/rm` でのみ使用できます。

`-f` 確認を求めるプロンプトメッセージを出力しません。 存在しないものがオペランドで指定されても、 診断メッセージを出力したり終了ステータスを変更したりしません。 前に `-i` オプションが指定されていれば、それを無視します。

`-i` 確認を求めるプロンプトを出力します。 `-f` オプションが指定されていれば、それを無視します。

/usr/bin/rmdir 以下のオプションは、 `/usr/bin/rmdir` でのみ使用できます。

`-p` ユーザーはディレクトリ `dirname` および空になる親ディレクトリを削除できます。 パスの全体、または一部が削除されていなければ、 メッセージが標準出力に出されます。

`-s` `-p` が有効なときに標準エラーに出力されるメッセージを表示しません。

オペランド 以下のオペランドを指定できます。

file 削除するディレクトリエントリのパス名

dirname 削除する空のディレクトリのパス名

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `rm` と `rmdir` の動作については、 `largefile`(5) を参照してください。

/usr/bin/rm
/usr/xpg4/bin/rm

例 **1** ディレクトリを削除する

次のコマンドは、ディレクトリエントリ `a.out` と `core` を削除します。

```
example% rm a.out core
```

例 **2** 確認を求めずにディレクトリを削除する

次のコマンドは、ディレクトリ `junk` とその下の内容を、確認を求めずに削除します。

例 2 確認を求めずにディレクトリを削除する　(続き)

```
example% rm -rf junk
```

/usr/bin/rmdir

例 3 空のディレクトリを削除する

現在のディレクトリに a というディレクトリがあり、そのディレクトリには b というディレクトリだけがあるとします。また b には、 c というディレクトリだけがあるものとします。次のコマンドは、3 つのディレクトリすべてを削除します。

```
example% rmdir -p a/b/c
```

環境

rm と rmdir の実行に影響を与える環境変数 LC_COLLATE 、 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　-f オプション省略時は、指定されたすべてのディレクトリエントリが 削除されたことを示す。 -f オプション指定時は、指定されたすべての既存のディレクトリエントリが 削除されたことを示す

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/rm
/usr/bin/rmdir

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/rm

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

rmdir(2), unlink(2), attributes(5), environ(5), largefile(5), XPG4(5)

診断

メッセージは、通常はすべて明白で、補足する必要はありません。

うっかりして次のようなことを行わないために、ファイル "." および ". ." を削除することは禁じられています。

```
example% rm -r .*
```

注意事項

-- を使用することにより、ユーザーは任意のコマンド行オプション の最後を明白に記すことができ、 rm は - で始まるファイル名の引数を認識できます。 BSD 移行の補助として、 rm は - を -- の同義語として受け付けます。 この導入補助は、将来のリリースでは提供されない可能性があります。 -- と - の両方が同一コマンド行にある場合は、 2 つめがファイルとして解釈されます。

名前

mail, rmail – メールの読み取りまたはユーザーへのメールの送信

メールの送信

mail [-tw] [-m *message_type*] *recipient...*

rmail [-tw] [-m *message_type*] *recipient...*

メールの読み取り

mail [-ehpPqr] [-f *file*]

デバッグ

mail [-x *debug_level*] [*other_mail_options*] *recipient...*

機能説明

recipient は通常、ドメイン形式 (*"user@machine"*) のアドレス、または`login`(1) が認識するユーザー名です。*recipient* を指定すると、`mail` はメッセージを送信中であるとみなします。このコマンドは、標準入力からファイルの終端 (Control-d) まで、あるいは端末デバイスから読み取っている場合は、ピリオドだけの行に至るまで読み取りを続けます。ファイルの終端またはピリオドだけの行のいずれかを受信すると、`mail` は各 *recipient* に対してレターをメールファイルに追加します。

レターは、最初にヘッダー行が数行、次に空行、さらに本文が続きます。レターのヘッダー行セクションは、1 つまたは複数の UNIX ポストマークから構成されます。

`From` *sender date_and_time* [`remote from` *remote_system_name*]

この後に、以下の形式の 1 つまたは複数の標準メッセージヘッダー行が続きます。

keyword-name`:` [*printable text*]

ここで、*keyword-name* は、 コロン (':') 以外の表示可能な空白以外の文字から構成されます。`Content-Length:` ヘッダー行は本文内のバイト数を示し、常に存在します。ただし、レターがメッセージ内容のないヘッダー行だけから構成されている場合は例外です。`Content-Type:` ヘッダー行は本文 (テキスト、バイナリ、マルチパートなど) のタイプを示し、常に存在します。ただし、レターがメッセージ内容のないヘッダー行だけから構成されている場合は例外です。ヘッダー行は、その次の行が空白で始まっている場合、次の行に継続できます。

メールの送信

以下のコマンド行引数は、メール送信に影響を与えます。

`-m` *message_type*	`Message-Type:` 行を、*message_type* の値とともにメッセージヘッダーに追加します。
`-t`	対象となる *recipient* の各々について、`To:` 行を、メッセージヘッダーに追加します。
`-w`	リモート送信プログラムの完了を待たずに、レターをリモート受信者に送信します。

レターが配送できないことがわかった場合、そのレターは障害の原因と性質を示す診断とともに送信者に返されます。入力中に `mail` が中断された場合、メッセージを `dead.letter` というファイルにセーブするので、後から編集と再送信を実行することもできます。`dead.letter` は常に追加されるので、以前の内容はすべて保存され

ます。dead.letter の追加 (または作成) は、まず、現在のディレクトリで行われます。これが失敗すると、ユーザーのログインディレクトリに dead.letter を追加 (または作成) しようとします。2 度目の試行も失敗した場合は、dead.letter の処理はいっさい行われません。

rmail は、メールの送信だけ可能です。uucp(1C) は rmail をセキュリティ通知として使用します。メールメッセージを生成するアプリケーションプログラムは、メッセージの送信や配送、またはその両方において、必ず、mailではなく rmail を起動する必要があります。

ローカルシステムに Basic Networking Utilities をインストールしてある場合、ローカルシステムで利用可能な送信機構に応じて、様々な方法でリモートシステム上の受信者にメールを送信できます。最も広く利用されているアドレス指定機構は、ドメイン形式とUUCP 形式の 2 つです。

ドメイン形式のアドレス指定
: 受信者名に '@' とドメイン (およびサブドメインもあり得る) 情報を追加してリモート受信者を指定します (たとえば、user@sf.att.com)。(ローカルシステム上でどの規則が利用可能かについての詳細は、ローカルシステム管理者に相談する必要があります)。

UUCP 形式のアドレス指定
: sysa!user のように、受信者名にリモートシステム名と感嘆符を接頭辞として付けてリモート受信者を指定します。csh(1) がデフォルトシェルの場合、sysa\!user を使用する必要があります。一連のシステム名を感嘆符で区切れば、拡張したネットワークを介してレターを出力することができます (たとえば、sysa!sysb!sysc!user または sysa\!sysb\!sysc\!user)。

メールの読み取り

以下のコマンド行引数は、メールの読み取りに影響を与えます。

-e
: メールを表示しません。ユーザーにメールがある場合、0 の終了コードを返します。メールがない場合、1 の終了コードを返します。

-h
: 最初に、最新のメッセージではなく、ヘッダーのウィンドウを表示します。ディスプレイの後に ? プロンプトを表示します。

-p
: 配置を示すプロンプトなしに、すべてのメッセージを表示します。

-P
: デフォルトの選択したヘッダー行を表示する代わりに、ヘッダー行すべてを表示してから、すべてのメッセージを表示します。

-q
: 割り込みを受けた後、mail は終了します。通常、割り込みによって発生するのは、表示中のメッセージの終了だけです。

-r
: 先入れ先出し方式でメッセージを表示します。

-f *file*
: デフォルトのメールファイルの代わりに、mail は、*file* (たとえば、mbox) を使用します。

mail は、コマンド行引数の影響を特に受けないかぎり、後入れ先出し方式でユーザーのメールメッセージを表示します。メッセージ表示のデフォルトのモードは、至急必要なヘッダー行だけを表示することです。これらには、UNIX の From ポストマークおよび >From ポストマーク、From: ヘッダー行、Date: ヘッダー行、Subject: ヘッダー行、および Content-Length: ヘッダー行、ならびに To:、Cc:、Bcc: などの受信者ヘッダー行が含まれますが、これに限定されるわけではありません。ヘッダー行を表示した後、mail は、メッセージに表示できない文字がないかぎり、その内容 (本体) を表示します。表示できない文字がある場合、mail は、メッセージの内容がバイナリである旨の警告文を発行し、その内容を表示しません (これは、p コマンドで無効にすることができます。以下を参照)。

各メッセージにおいて、ユーザーは ? というプロンプトを受け、標準入力から行が読み取られます。メッセージの配置を判別するときは、以下のコマンドを利用できます。

#	現在のメッセージの数を表示します。
–	直前のメッセージを表示します。
<復帰改行>、+、または n	次のメッセージを表示します。
!*command*	シェルにエスケープして、*command* を実行します。
a	mail セッション中に到着したメッセージを表示します。
d または dp	現在のメッセージを削除し、次のメッセージを表示します。
d *n*	*n* 番のメッセージを削除します。次のメッセージを表示しません。
dq	メッセージを削除し、mail を終了します。
h	現在のメッセージを中心としてヘッダーのウィンドウを表示します。
h *n*	*n* 番のメッセージを中心としてヘッダーのウィンドウを表示します。
h a	ユーザーのメールファイルにすべてのメッセージのヘッダーを表示します。
h d	削除予定のメッセージのヘッダーを表示します。
m [*persons*]	指定された *persons* に現在のメッセージを送信 (および削除) します。
n	*n* 番のメッセージを表示します。
p	現在のメッセージを再び表示し、バイナリ (つまり、表示不能) の内容の表示を無効にします。
P	デフォルトの省略モードを無効にし、現在のメッセージを再表示して、ヘッダー行をすべて表示します。

q または CTRL-D	削除を解除したメールをメールファイルに戻し、mail を終了します。
r [*users*]	送信者と他の *users* に応答してから、メッセージを削除します。
s [*files*]	メッセージを、指定された *files* (デフォルトは mbox) にメッセージの内容をヘッダー行なしでセーブし、メッセージを削除します。
u [*n*]	*n* 番のメッセージの削除を解除します(デフォルトは直前に読み取ったメッセージ)。
w [*files*]	指定された *files* (デフォルトは mbox) にメッセージの内容をヘッダー行なしでセーブし、メッセージを削除します。
x	メールすべてを変更を加えずにメールファイルに戻し、mail を終了します。
y [*files*]	-w オプションと同じ。
?	コマンド概要を表示します。

通常、ユーザーがログインする際にメールが存在すれば、それが通知されます。また、mail を使用中に新しいメールが到着した場合でも通知されます。

chmod(1) を使用し、2 種類の方法でメールファイルのアクセス権を処理すれば、mail の機能を変更できます。ファイルのその他のアクセス権を、読み取り書き込みともに可能 (0666)、読み取り専用 (0664)、または読み取り書き込みとも不可 (0660) とすることによって、プライバシーをさまざまに調整できます。デフォルト (0660モード) 以外に変更すると、空の場合でもファイルは保存され、希望のアクセス権が永久的になります (管理者は mailcnfg の DEL_EMPTY_MAILFILE オプションを使用すれば、このファイル保存を無効にできます)。

メールファイルのグループ ID を mail として新しいメッセージを配送可能にし、メールファイルをグループ mail で書き込み可能にする必要があります。

デバッグ

以下のコマンド行引数によって、mail は、デバッグ情報を提供します。

-x *debug_level*	mail は、デバッグ情報の入ったトレースファイルを作成します。

-x オプションを指定すると、mail は /tmp/MLDBG*process_id* という名前のファイルを作成します。このファイルには、mail が現在のメッセージを処理した手順に関するデバッグ情報が入っています。*debug_level* の絶対値はデバッグ情報の長さを制御します。0 はデバッグなしを意味します。*debug_level* が 0 より大きい場合、デバッグファイルが保持されるのは、mail のメッセージ処理中に問題が発生した場合だけです。*debug_level* が 0 未満の場合、常にデバッグファイルを保持します。-x で *debug_level* を指定すると、/etc/mail/mailcnfg 内の DEBUG の指定すべてが無効になります。-x オプションの提供する情報は難解であり、役立つのはシステム管理者だけだと思われます。

配送通知

メールの通知には、いくつかの形式があります。以下の行をメッセージヘッダーに挿入することによって実現します。

`Transport-Options:` [*/options*]

`Default-Options:` [*/options*]

`>To:` *recipient* [*/options*]

ここで "*/options*" には、以下のうちの 1 つまたは複数が使用できます。

`/delivery`	メッセージが *recipient* のメールボックスに正常に配送されたことを送信者に通知します。
`/nodelivery`	配送が成功したことを送信者に通知しません。
`/ignore`	配送が失敗したことを送信者に通知しません。
`/return`	メール配送が失敗したか否かを送信者に通知します。送信者に失敗メッセージを返します。
`/report`	`/return` と同じ。ただし、元のメッセージは返しません。

デフォルトは `/nodelivery/return` です。矛盾のあるオプションを使用すると、最初のオプションを認識し、その後の矛盾する条件を無視します。

オペランド

メール送信には次のオペランドがあります。

recipient	ドメイン形式 ("*user@machine*") のアドレス、または`login`(1) が認識するユーザー名

使用法

ファイルが 2G バイト (2^{31}バイト) 以上ある場合の `mail` と `rmail` の動作については、`largefile`(5) を参照してください。

環境

`mail` の実行に影響を与える環境変数 `LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、`environ`(5) を参照してください。

`TZ`	日付と時間の文字列とともに使用する時間帯を指定する

終了ステータス

以下の終了ステータスが返されます。

`0`	そのユーザー宛のメールがあり、処理は正常に終了した
`1`	ユーザー宛のメールがなかった、または初期化時にエラーが発生した
`>1`	初期化の後でエラーが発生した

ファイル

`dead.letter`	メールできなかったテキスト
`/etc/passwd`	送信者の識別および *recipient* の発見用
`$HOME/mbox`	セーブされたメール
`$MAIL`	メールファイルのパス名を含む変数
`/tmp/ma*`	一時ファイル

/tmp/MLDBG*	デバッグ・トレースファイル
/var/mail/*.lock	メールディレクトリのロック
/var/mail/:saved	一時ファイルを保持し、システムクラッシュ時のデータ損失を防ぐためのディレクトリ
/var/mail/*user*	ユーザーに送られたメール (ユーザーに届いたメール)。すなわち、デフォルトのメールファイル

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目　chmod(1), csh(1), login(1), mailx(1), uucp(1C), uuencode(1C), vacation(1), write(1), attributes(5), environ(5), largefile(5)

『*OpenWindows* ユーザーズガイド *(*上級編*)*』

注意事項　上記の「配送通知」の節で説明したように、ヘッダー行による解釈と結果的な処置が発生するのは、配送 (または失敗) が行われたシステム上に、このバージョンの mail がインストールされている場合だけです。mail の初期のバージョンはどのタイプの配送通知もサポートしません。

条件によっては、ロックファイルの削除が失敗することがあります。

割り込み後、次のメッセージが表示されないことがあります。表示を強制するときは、p を入力します。

名前 | rm, rmdir – ディレクトリのエントリの削除

形式 | **/usr/bin/rm** [-f] [-i] *file...*

/usr/bin/rm -rR [-f] [-i] *dirname...* [*file...*]

/usr/xpg4/bin/rm [-fiRr] *file...*

/usr/bin/rmdir [-ps] *dirname...*

/usr/bin/rm
/usr/xpg4/bin/rm

rm ユーティリティは、 *file* オペランドで示された各ディレクトリエントリを削除します。 *file* に書き込み権がなく、標準入力が端末の場合、 そのファイルに対するすべてのアクセス権(8 進数)が出力され、後ろに疑問符がつけられます。 これは、確認を求めるプロンプトです。 これに対する解答が y (yes) の場合にはそのファイルは削除され、 そうでない場合にはそのファイルは残されます。

file がシンボリックリンクの場合リンクは削除されますが、 参照するファイルまたはディレクトリは削除されません。 ユーザーがシンボリックリンクを削除する際、 そのディレクトリに書き込み権があれば、 シンボリックリンクに書き込み権は必要ありません。

file オペランドに複数のファイルが指定され、何らかの理由でそのうちの 1 つが削除できなかった場合、 rm は診断メッセージを標準エラー出力に書き出し、 そのファイルにはそれ以上の処理は行わず、 残りの *file* オペランドを処理します。

標準入力が端末でない場合、 rm は -f オプションが指定されている場合と同様に動作します。

/usr/bin/rmdir

rmdir ユーティリティは、 *dirname* オペランドで示されたディレクトリのエントリを削除します。 *dirname* は空のディレクトリを指している必要があります。

ディレクトリは指定された順序で処理されます。 したがって、あるディレクトリとそのディレクトリのサブディレクトリを 1 度の rmdir 呼び出しで削除したい場合、 サブディレクトリを親ディレクトリよりも前に指定して先に削除し、 親ディレクトリの削除のときにディレクトリが空になるよう にしてください。

オプション | 以下のオプションは、 /usr/bin/rm と /usr/xpg4/bin/rm で使用できます。

-r　引数リストにあるディレクトリおよび サブディレクトリを再帰的に削除します。 ディレクトリ中は空にされ、削除されます。 通常、ディレクトリにある書き込み保護された 任意のファイルの削除には、 ユーザーに確認を求めるプロンプトが出されます。ただし、 -f オプションが使用されている場合、または 標準入力が端末ではなく、しかも -i オプションが使用されていない場合、 書き込み保護されたファイルはプロンプトなしで削除されます。

このオプションが使用された場合に シンボリックリンクのエントリが存在しても、 再帰的処理はシンボリックリンク先には及びません。

書き込み保護された空でないディレクトリを削除しようとすると、 このコマンドは常に(-f オプションが使用されている場合でも)失敗に終わり、エラーメッセージが表示されます。

`-R` `-r` オプションと同じです。

/usr/bin/rm 以下のオプションは、 `/usr/bin/rm` でのみ使用できます。

`-f` ユーザーに確認を求めるプロンプトを出さずに、ディレクトリ内のすべてのファイル (書き込み保護のあるなしにかかわらず)を削除します。 しかし、書き込み保護されたディレクトリでは、 (たとえどんなアクセス権があっても)ファイルは決して削除されません。 また、それについてのメッセージは表示されません。 書き込み保護されたディレクトリを削除しようとすると、 このオプションはエラーメッセージを表示します。

`-i` 対話型。 このオプションを使用すると、 `rm` はファイルを削除する前に、確認を求めるプロンプトを出します。これは `-f` オプションより優先され、 標準入力が端末でない場合でもその効力を持続します。

/usr/xpg4/bin/rm 以下のオプションは、 `/usr/xpg4/bin/rm` でのみ使用できます。

`-f` 確認を求めるプロンプトメッセージを出力しません。 存在しないものがオペランドで指定されても、 診断メッセージを出力したり終了ステータスを変更したりしません。 前に `-i` オプションが指定されていれば、それを無視します。

`-i` 確認を求めるプロンプトを出力します。 `-f` オプションが指定されていれば、それを無視します。

/usr/bin/rmdir 以下のオプションは、 `/usr/bin/rmdir` でのみ使用できます。

`-p` ユーザーはディレクトリ `dirname` および空になる親ディレクトリを削除できます。 パスの全体、または一部が削除されていなければ、 メッセージが標準出力に出されます。

`-s` `-p` が有効なときに標準エラーに出力されるメッセージを表示しません。

オペランド 以下のオペランドを指定できます。

file 削除するディレクトリエントリのパス名

dirname 削除する空のディレクトリのパス名

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `rm` と `rmdir` の動作については、 `largefile`(5) を参照してください。

/usr/bin/rm
/usr/xpg4/bin/rm

例 **1** ディレクトリを削除する

次のコマンドは、ディレクトリエントリ `a.out` と `core` を削除します。

```
example% rm a.out core
```

例 **2** 確認を求めずにディレクトリを削除する

次のコマンドは、ディレクトリ `junk` とその下の内容を、確認を求めずに削除します。

例 2 確認を求めずにディレクトリを削除する　(続き)

```
example% rm -rf junk
```

/usr/bin/rmdir

例 3 空のディレクトリを削除する

現在のディレクトリに a というディレクトリがあり、そのディレクトリには b というディレクトリだけがあるとします。また b には、 c というディレクトリだけがあるものとします。次のコマンドは、3 つのディレクトリすべてを削除します。

```
example% rmdir -p a/b/c
```

環境

rm と rmdir の実行に影響を与える環境変数 LC_COLLATE 、 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　-f オプション省略時は、指定されたすべてのディレクトリエントリが 削除されたことを示す。 -f オプション指定時は、指定されたすべての既存のディレクトリエントリが 削除されたことを示す

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/rm
/usr/bin/rmdir

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/rm

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

rmdir(2), unlink(2), attributes(5), environ(5), largefile(5), XPG4(5)

診断

メッセージは、通常はすべて明白で、補足する必要はありません。

うっかりして次のようなことを行わないために、ファイル "." および ". ." を削除することは禁じられています。

```
example% rm -r .*
```

注意事項

-- を使用することにより、ユーザーは任意のコマンド行オプション の最後を明白に記すことができ、 rm は - で始まるファイル名の引数を認識できます。 BSD 移行の補助として、 rm は - を -- の同義語として受け付けます。 この導入補助は、将来のリリースでは提供されない可能性があります。 -- と - の両方が同一コマンド行にある場合は、 2 つめがファイルとして解釈されます。

名前 | rmformat – 再書き込み可能な取り外し可能媒体のフォーマット

形式 | **rmformat** [-DeHpUv] [-b *label*] [-c *blockno*] [-Fquick | long | force] [-R enable | disable] [-s *filename*] [-w enable | disable] [-W enable | disable] [*devname*]

rmformat -V read | write *devname*

機能説明 | rmformat ユーティリティは、再書き込み可能な取り外し可能媒体の、フォーマット、ラベル作成、パーティション作成を行なったり、取り外し可能媒体に関するその他のさまざま処理を行います。この「再書き込み可能な取り外し可能媒体」とは、フロッピードライブ、IOMEGA ZIP/Jaz 製品、および PCMCIA メモリーおよび ata カードを指します。rmformat ユーティリティは、検証、表面解析、不良セクターの修復 (ドライブまたはドライバが不良ブロック管理をサポートしている場合) を行うためにも使用できます。

rmformat には、パスワード付きまたはパスワードなしの読み取り/書き込み保護を媒体に設定するための機能があります。IOMEGA ZIP/Jaz 製品など特定の再書き込み可能な媒体に対して、パスワードによる保護を有効にしたり無効にしたりすることができます。

フロッピーディスクや PCMCIA メモリーカードをフォーマットした後、rmformat は fdformat の動作との互換性を保つために、媒体全体を 1 つのスライスとしたラベルを書き込みます。ZIP/JAZ デバイスの場合、デフォルトではドライバがディスク全体を占める 1 つのスライスをエクスポートします。rmformat は、明示的に要求されない限り ZIP/JAZ 媒体にラベルを書き込みません。パーティション情報は、rmformat のオプションを使用して変更することができます。

オプション | 以下のオプションを指定できます。

-b *label*
: 媒体に SUNOS ラベルを作成します。SUNOS ボリュームラベル名は 8 文字以内にする必要があります。DOS ボリュームラベルを書き込むには、mkfs_pcfs(1M) を使用してください。

-c *blockno*
: 指定されたブロックを修正して修復します。このオプションは、rmformat がサポートするすべてのデバイスに適用できるわけではありません。不良ブロック管理機能を持つドライブが装備されているデバイスと、このオプションがドライバによって実装されているデバイスとがあるためです。 不良ブロック管理がサポートされているのがドライブまたはドライバのどちらであっても、不良ブロックは可能なかぎり修正されるように機能します。不良ブロックを修正できなかった場合は、修復に失敗したことを示すメッセージが表示されます。メッセージには、該当するブロック番号が 10 進数、8 進数、16 進数で示されます。

	通常のフロッピーディスクおよび PCMCIA メモリーおよび ata カードは、不良ブロック管理をサポートしていません。
`-D`	720K バイト (3.5 インチ) 倍密度フロッピーディスクをフォーマットします。倍密度タイプのドライブでは、このオプションの動作がデフォルトです。高密度タイプまたは拡張密度タイプのドライブで倍密度フロッピーディスクをフォーマットするには、このオプションを使用する必要があります。
`-e`	処理が終了した時に媒体を取り出します。ドライブが自動で取り出しを行う機能をサポートしていない場合、このオプションは利用できません。
`-F` quick | long | force	媒体をフォーマットします。 `quick` オプションは、検証をしないでフォーマットを開始するか、または媒体の特定のトラックだけを検証してフォーマットを開始します。 `long` オプションは、完全フォーマットを開始します。ドライブ自体によって媒体全体の検証を行うデバイスもあります。 `force` オプションは、フォーマットを開始する前にユーザーによる確認を行わずに `long` オプションによるフォーマットを行います。パスワードによる保護機構を持っているドライブの場合、フォーマット中にパスワードを消去します。これは、パスワードを利用できなくなっている場合に便利です。パスワードによる保護機構を持っていない媒体に対してこのオプションを使用した場合は、`long` オプションによるフォーマットを行います。 フロッピーディスクなど従来の媒体では、上記のどのオプションを指定しても、デフォルトでフロッピーディスクドライブが動作するモード (拡張密度モード、高密度モード、倍密度モード) に応じて、`long` オプションによるフォーマットを行います。PCMCIA メモリーカードでは、上記のどのオプションを指定しても `long` オプションによるフォーマットを行います。
`-H`	1.44M バイト (3.5 インチ) 高密度フロッピーディスクをフォーマットします。高密度タイプのドライブでは、このオプションの動作がデフォルトです。拡張密度タイプのドライブで高密度フロッピーディスクをフォーマットするには、このオプションを使用する必要があります。

`-p`

媒体の保護状態 (書き込み保護、読み取り保護、パスワードによる保護が設定されているかどうか) を出力します。

`-R` enable | disable

読み取り/書き込み保護に対するパスワードを有効または無効にします。セキュリティを保つためにユーザーからのパスワード入力が必要なので、このオプションによる処理は対話式で行われます。

パスワードによる読み取り/書き込み保護をサポートしている IOMEGA 製品で、最長 32 バイトまでのパスワードを使用できます。このオプションは、IOMEGA 製品に対してのみ利用できます。IOMEGA 製品では、パスワードなしの読み取り/書き込み保護は設定できません。ソフトウェアの読み取り/書き込み保護機能を持っていないデバイスでは、この機能は利用できないことを示す警告メッセージが表示されます。

`-s` *filename*

ユーザーが、SUNOS ラベル中のパーティション情報を配置することができるようにします。

ユーザーは、各スライスの情報 (バイトオフセット、必要なサイズ、タグ、フラグ) を記述したファイルを入力として指定する必要があります。各スライスの情報を記述する書式は次のとおりです。

```
slices: n = offset, size [, flags, tags]
```

n はスライス番号です。*offset* はスライス *n* の開始位置で、バイトオフセットです。*size* はスライス *n* のサイズです。*offset* および *size* に指定する値は、512 バイトの倍数にする必要があります。これらの数値は、10 進数、16 進数、または 8 進数で指定することができます。浮動小数点数は使用できません。スライスの最大数についての詳細は、『*Solaris* のシステム管理 *(*第 *1* 巻*)*』を参照してください。

size または *offset* を K バイト、M バイト、G バイト単位で指定するには、それぞれ数値の後に `KB`、`MB`、`GB` を付けてください。これらの単位を表す接尾辞なしの数値は、バイトオフセットとみなされます。指定できるフラグは、次のとおりです。

```
wm = 読み取り/書き込み、マウント可
wu = 読み取り/書き込み、マウント不可
ru = 読み取り専用、マウント不可
```

指定できるタグは、次のとおりです : `unassigned`、`boot`、 `root`、 `swap`、 `usr`、 `backup`、 `stand`、`var`、 `home`、 `alternates`

値を特定する必要がない場合は、タグとフラグの指定を省略することができます。ただし、省略する場合はタグとフラグの両方を省略し、指定する場合はタグとフラグの両方を指定する必要があります。特定のスライスに対するタグとフラグの値が指定されない場合、それぞれのデフォルト値が使用されます。フラグのデフォルト値は wm、タグのデフォルト値は unassigned です。

また、タグの名前は完全名または短縮名で指定できます。短縮名は完全名から最初の 2 文字以上を引用したものです。rmformat は、タグおよびフラグの短縮名を扱う時には大文字・小文字を区別しません。

スライスの指定は、次のように区切って記述します。

例:

```
slices: 0 = 0, 30MB, "wm", "home" :
        1 = 30MB, 51MB :
        2 = 0, 100MB, "wm", "backup" :
        6 = 81MB, 19MB
```

rmformat は、重複パーティションまたは媒体の容量を超える不正な要求を検出するために、必要性検査を行います。1 つのスライス *n* について提供するスライス情報は 1 項目だけにする必要があります。同じスライス *n* について複数のスライス情報がある場合は、エラーメッセージが表示されます。スライス 2 は、ディスク全体を含むバックアップです。# は、入力ファイル中にコメント行を記述するために使用します。 rmformat は、行頭にハッシュ記号 (#) がある行は、その行の末尾までのすべての文字を無視します。

容量がごく小さい媒体にいくつかのパーティションを作成することはできますが、そのようなデバイスに対してこのオプションを使用する場合には注意が必要です。

-U

すべてのタイプのファイルシステムに対して umount を実行し、フォーマットを行います (mount(1M) を参照)。つまり、マウントされていたスライスをすべてマウント解除し、指定したデバイスに対して long オプションによるフォーマットを実行します。

-V read | write

フォーマット後に媒体の各ブロックを検証します。書き込みの検証は、破壊的な機構 (書き込んだら消去するという機構) で行われます。検証を開始する前に、ユー

ザーは検証を始めることを確定するように求められます。このオプションを使用すると、不良ブロックとして検出されたブロックの番号のリストが出力されます。

読み取りの検証は、ブロックを検証して、エラーが発生する傾向にあるブロックのリストが出力されます。

出力されるブロック番号のリストは、修復を行う時に `-c` オプションを使って利用することができます。

`-w` enable | disable

媒体の書き込み保護を有効または無効にします。ソフトウェアの書き込み保護機能を持っていないデバイスでは、この機能は利用できないことを示すメッセージが表示されます。

`-W` enable | disable

媒体の書き込み保護を有効または無効にします。セキュリティを保つためにユーザーからのパスワード入力が必要なので、このオプションによる処理は対話式で行われます。

パスワードによる書き込み保護をサポートしている IOMEGA 製品では、最長 32 バイトまでのパスワードを使用できます。パスワードによる書き込み保護機能を持っていないデバイスでは、この機能は利用できないことを示す警告メッセージが表示されます。

オペランド

以下のオペランドを指定できます。

devname

devname には、絶対パス名または現在のディレクトリからの相対パス名でデバイスのパス名を指定するか、ボリューム管理システムによってエクスポートされるニックネームを指定します。詳細は、`vold`(1M) を参照してください。

フロッピーディスクのデバイスとしては、`/dev/rdiskette0` (ボリューム管理なしのシステム) または `floppy0` (ボリューム管理ありのシステム) を使用することができます。第 1 ドライブを使用するには `/dev/rdiskette1` (ボリューム管理なしのシステム) を、第 2 ドライブを使用するには `floppy1` (ボリューム管理なしのシステム) を指定します。

ボリューム管理が動作していないシステムでは、`/dev/rdsk/c?t?d?s?` のような絶対パス名で、または現在のディレクトリからの相対パス名で、デバイスを指定することができます。

使用例

例 1 フロッピーディスクのフォーマット

```
example$ rmformat -F quick /dev/rdiskette
Formatting will erase all the data on disk.
Do you want to continue? (y/n)y
```

例 2 Zip ドライブのフォーマット

```
example$ rmformat -F quick /vol/dev/aliases/zip0
Formatting will erase all the data on disk.
Do you want to continue? (y/n)y
```

例 3 フロッピーディスクを UFS ファイルシステム用にフォーマットする

以下は、フロッピーディスクをフォーマットして、UFS ファイルシステムを作成する例です。

```
example$ rmformat -F quick /vol/dev/aliases/floppy0
Formatting will erase all the data on disk.
Do you want to continue? (y/n)y
example$ su
# /usr/sbin/newfs /vol/dev/aliases/floppy0
newfs: construct a new file system /dev/rdiskette: (y/n)? y
/dev/rdiskette: 2880 sectors in 80 cylinders of 2 tracks, 18 sectors
         1.4MB in 5 cyl groups (16 c/g, 0.28MB/g, 128 i/g)
 super-block backups (for fsck -F ufs -o b=#) at:
  32, 640, 1184, 1792, 2336,
#
```

例 4 取り外し可能な媒体を PCFS ファイルシステム用にフォーマットする

以下は、代替の `fdisk` パーティションを作成する例です。

```
example$ rmformat -F quick /dev/rdsk/c0t4d0s2:c
Formatting will erase all the data on disk.
Do you want to continue? (y/n)y
example$ su
# fdisk /dev/rdsk/c0t4d0s2:c
# mkfs -F pcfs /dev/rdsk/c0t4d0s2:c
Construct a new FAT file system on /dev/rdsk/c0t4d0s2:c: (y/n)? y
#
```

以下は、`fdisk` パーティションを使用せずに PCFS ファイルシステムを作成する例です。

```
example$ rmformat -F quick /dev/rdiskette
Formatting will erase all the data on disk.
Do you want to continue? (y/n)y
example$ su
# mkfs -F pcfs -o nofdisk,size=2 /dev/rdiskette
Construct a new FAT file system on /dev/rdiskette: (y/n)? y
#
```

例 **5** 読み取り保護または書き込み保護を有効または無効にする

以下は、書き込み保護を有効にして、Zip ドライブにパスワードを設定する例です。

```
example$ rmformat -W enable /vol/dev/aliases/zip0
Please enter password (32 chars maximum): xxx
Please reenter password: xxx
```

以下は、書き込み保護を無効にして、Zip ドライブからパスワードを削除する例です。

```
example$ rmformat -W disable /vol/dev/aliases/zip0
Please enter password (32 chars maximum): xxx
```

以下は、読み取り保護を有効にして、Zip ドライブにパスワードを設定する例です。

```
example$ rmformat -R enable /vol/dev/aliases/zip0
Please enter password (32 chars maximum): xxx
Please reenter password: xxx
```

以下は、読み取り保護を無効にして、Zip ドライブからパスワードを削除する例です。

```
example$ rmformat -R disable /vol/dev/aliases/zip0
Please enter password (32 chars maximum): xxx
```

ファイル

/vol/dev/diskette0	フロッピーディスクドライブ 0 に入っている媒体に対するブロック型デバイスアクセスを提供するディレクトリ。
/vol/dev/rdiskette0	フロッピーディスクドライブ 0 に入っている媒体に対する文字型デバイスアクセスを提供するディレクトリ。
/vol/dev/aliases	別名を使用するボリューム管理によって制御されている異なる媒体に対して、文字型デバイスへのシンボリックリンクを提供するディレクトリ。
/vol/dev/aliases/floppy0	フロッピーディスクドライブ 0 に入っている媒体に対する文字型デバイスへのシンボリックリンク。
/vol/dev/aliases/zip0	Zip ドライブ 0 に入っている媒体に対する文字型デバイスへのシンボリックリンク。
/vol/dev/aliases/jaz0	Jaz ドライブ 0 に入っている媒体に対する文字型デバイスへのシンボリックリンク。
/dev/rdiskette	一次フロッピーディスクドライブ (通常はドライブ 0) に入っている媒体に対する文字型デバイスアクセスを提供するシンボリックリンク。

/vol/dev/dsk	PCMCIA メモリーおよび ata カードや取り外し可能な媒体のデバイスに対するブロック型デバイスアクセスを提供するディレクトリ。
/vol/dev/rdsk	PCMCIA メモリーおよび ata カードや取り外し可能な媒体のデバイスに対する文字型デバイスアクセスを提供するディレクトリ。
/vol/dev/aliases/pcmem*S*	ソケット *S* 中の PCMCIA メモリーカードに対する文字型デバイスへのシンボリックリンク。*S* は、PCMCIA ソケット番号を表しています。
/vol/dev/aliases/rmdisk0	Zip、Jaz、CD-ROM、フロッピーディスク、DVD-ROM、PCMCIA メモリーカード以外の一般的な取り外し可能な媒体へのシンボリックリンク。
/dev/rdsk	PCMCIA メモリーおよび ata カードやその他の取り外し可能な媒体のデバイスに対する文字型デバイスアクセスを提供するディレクトリ。
/dev/dsk	PCMCIA メモリーおよび ata カードやその他の取り外し可能な媒体のデバイスに対するブロック型デバイスアクセスを提供するディレクトリ。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

cpio(1), eject(1), fdformat(1), tar(1), volcancel(1), volcheck(1), volmissing(1), volrmmount(1), format(1M), mkfs_pcfs(1M), mount(1M), newfs(1M), prtvtoc(1M), rmmount(1M), rpc.smserverd(1M), vold(1M), rmmount.conf(4), vold.conf(4), attributes(5), pcfs(7FS), udfs(7FS)

注意事項

SPARC ベースのシステム上で (newfs(1M) を使用して) 作成された ufs ファイルシステムが含まれている、再書き込み可能な媒体、PCMCIA メモリーカード、または PCMCIA ata カードと、x86 ベースのシステム上で作成された ufs ファイルシステムが含まれている再書き込み可能な媒体または PCMCIA メモリーカードとは、同じではありません。SPARC ベースのシステムと x86 ベースのシステムとの間で、ufs ファイルシステムを含む再書き込み可能な媒体を交換しないでください。SPARC ベースのシステムと x86 ベースのシステムとの間で、フロッピーディスクまたはメモリーカード中のファイルを転送するには、cpio(1) または tar(1) を使用してください。相互に交換することが可能なファイルシステムについては、pcfs(7FS) および udfs(7FS) のマニュアルページを参照してください。

使用上の留意点

現在のところ、フロッピーディスクまたは PCMCIA メモリーカード上の不良セクターの場所を検出する機能は、サポートされていません。このため、`rmformat` を実行して `bad sector` というエラーが発生した場合、フロッピーディスクまたはメモリーカードを使用できません。

名前 | rsh, remsh, remote_shell – リモートシェル

形式 | **rsh** [-n] [-l *username*] *hostname command*

rsh *hostname* [-n] [-l *username*] *command*

remsh [-n] [-l *username*] *hostname command*

remsh *hostname* [-n] [-l *username*] *command*

hostname [-n] [-l *username*] *command*

機能説明 | rsh は、 *hostname* が示すホストに接続し、 *command* が示すコマンドを実行します。 rsh は自身の標準入力のデータをリモートコマンドにコピーし、 リモートコマンドの標準出力を自身の標準出力にコピーし、 さらに、リモートコマンドの標準エラー出力を 自身の標準エラー出力にコピーします。 割り込み、停止、および終了シグナルは、 リモートコマンドに伝えられます。通常 rsh は、リモートコマンドが終了したときに終了します。

command を省略すると、 rsh は単一のコマンドを実行する代わりに、 rlogin(1) を使ってそのユーザーをリモートホストにログインします。

rsh は *command* の終了ステータスを返しません。

シェルのメタキャラクタのうち、 引用符で囲まれていないものはローカルマシン上で解釈されます。 引用符で囲まれているものは、リモートマシン上で解釈されます。「使用例」の項にある例を参照してください。

特定のユーザーのログインシェルの初期化ファイル (.cshrc など) でロケールが設定されていない場合、rsh はコマンドを実行するときに、常に C ロケールを使用します。リモートマシンのデフォルトのロケールは使用しません。

オプション | 以下のオプションを指定できます。

-l *username* — リモートユーザー名として、 ユーザー自身のローカルユーザー名の代わりに *username* を使用します。このオプションを省略すると、 リモートユーザー名はローカルユーザー名と同じになります。

-n — rsh の入力先を /dev/null に変更します。このオプションは、rsh とそれを呼び出したシェルとの間での、 予期できない干渉を防ぐ上で便利です。 たとえば、 rsh を実行していて、さらにバックグラウンドで rsh を呼び出した場合、その入力先を端末以外に変更しないと、 リモートコマンドの読み取りがない場合でも処理が妨げられてしまいます。 -n オプションを指定すれば、このような事態は避けられます。

リモートシェルの種類 (sh や rsh など) は、リモートシステム上の /etc/passwd ファイル中のユーザーのエントリにより決められます。

オペランド | 以下のオペランドを指定できます。

command — 指定された *hostname* で実行するコマンド

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の rsh と remsh の動作については、 largefile(5) を参照してください。

rsh および remsh コマンドは、IPv6 に対応しています。ip6(7P) のマニュアルページを参照してください。

ホスト名は *hosts* データベース (/etc/hosts ファイルに含めることができる)、 インターネットドメイン名データベース、 あるいはその両方に書かれています。 各ホストには 1 つの正式名 (データベースエントリの最初の名前) があり、 さらにいくつかのニックネームが存在することもあります。 *hostname* 引数には、正式なホスト名とニックネームの いずれかを指定します。

rsh を実行するファイル名が rsh でなければ、 rsh はそのファイル名を *hostname* 引数として使用します。これによりユーザーは、ホスト名で rsh に対するシンボリックリンクを生成でき、 実行時にそのホスト上のリモートシェルを呼び出せます。
ディレクトリを作成し、それを共通に使われる ホスト名でシンボリックリンクにつなぎ、 さらにそのディレクトリをユーザー自身の シェルの検索パスに含めることにより、シェルに *hostname* を入力するだけで rsh を実行することができます。

rsh を remsh で呼び出した場合、 rsh は /usr/bin/remsh の存在を確認します。このファイルが存在 する場合、 rsh は remsh を rsh の別名として処理します。
/usr/bin/remsh が存在しない場合、 rsh は remsh をホスト名として処理します。

各リモートマシンには /etc/hosts.equiv という名のファイルが存在することがあります。 このファイルには、そのマシンとユーザー名を共有する ホスト名の一覧が記録されています。 ローカルマシン上とリモートマシン上での ユーザー名が同一のユーザーは、 リモートマシンの /etc/hosts ファイルにリストされているマシンから rsh を実行することができます。 個々のユーザーは、このような同等名リストを個人用の .rhosts ファイルとして、自身のホームディレクトリに 作成することができます。 このファイル中の各行には 2 つの名前、 *hostname* と *username* が含まれ、両者は空白で区切られます。 *username* で示すユーザーが *hostname* で示すホストにログインしていれば、そのユーザーは rsh を使って、リモートユーザーとしてリモートマシンにアクセスできます。 ローカルホスト名がリモートマシン上の /etc/hosts.equiv ファイル中に見つからず、ローカルのユーザー名とホスト名がリモートユーザーの .rhosts ファイル中に見つからない場合、アクセスは拒否されます。 /etc/hosts.equiv または .rhosts ファイルに記録されているホスト名は、 hosts データベースに登録されている 正式なホスト名である必要があります。つまりこの両ファイル中には、 ニックネームは指定できません。

リモートマシン上でアクセスが拒否されたときは、 *command* 引数が省略されていない限り、 rsh はパスワードの入力を要求するプロンプトを出力しません。

使用例

例 **1** rsh でファイルを追加する

次のコマンドは、lizard というマシンからファイル lizard.file というリモートファイルを、example というマシン上の example.file に付加します。

```
example% rsh lizard cat lizard.file >> example.file
```

例 1 rsh でファイルを追加する　(続き)

一方、次のコマンドは、 lizard というマシン上の lizard.file というファイルを、同じマシン上の lizard.file2 ファイルに付加します。

```
example% rsh lizard cat lizard.file ">>" lizard.file2
```

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

1　エラーが発生した

ファイル

/etc/hosts	インターネットホストテーブル
/etc/hosts.equiv	信頼性のあるリモートホストとユーザー
/etc/passwd	システムパスワードファイル

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

on(1), rlogin(1), telnet(1), vi(1), in.named(1M), hosts(4), hosts.equiv(4), ipnodes(4), attributes(5), largefile(5), ip6(7P)

注意事項

hosts.equiv に登録されているシステムのセキュリティは、 少なくともローカルシステムのセキュリティと 同レベルである必要があります。 セキュリティレベルの低いシステムが hosts.equiv 中に 1 つでも存在していると、 システム全体のセキュリティが損なわれる可能性があります。

vi(1) のような対話型コマンドは実行できません。 対話型コマンドは rlogin を使って実行してください。

停止シグナルは、ローカルの rsh プロセスだけを停止させます。 これはバグだという議論があるかもしれませんが、 現在のところ修正が大変困難です。 その理由は複雑なので、ここでは説明を省きます。

以下を実行すると、シェルの状態がおかしくなります。

```
example% rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf -
```

つまり rsh の前に tar が終了してしまい、そのあとで rsh が「壊れたパイプ」にデータを書き出そうとします。 そのため正常に終了するのではなく、 標準入力をアクセスしようとしてシェルと競合してしまいます。

-n オプションを指定して rsh を実行すれば、このような事態を防ぐことができます。

```
example% tar cf - . | rsh sundial dd of=/dev/rmt0 obs=20b
```

この場合には、前述のような現象は起こりません。 この場合に -n オプションを指定すると、 rsh はパイプから読み込む代わりに、誤って /dev/null からの読み込みを試みます。

名前 | rusers – リモートマシン上にログインしているユーザーの表示

形式 | **rusers** [-ahilu] *host...*

機能説明 | rusers コマンドは、 who(1) と似た形式の出力をリモートマシン用に作成します。通常この出力では、応答を得た順序でユーザー名がリストされますが、 後述するオプションを使って順序を変更することができます。

デフォルトではログインしているユーザーの名前だけが出力されます。 -l オプションを指定すると、 それに加えて以下の情報が各ユーザーに対して出力されます。

userid hostname : *terminal login_date login_time idle_time login_host*

hostname （ホスト名）と *login host* （ログインホスト）が同じ値の場合には、 *login host* フィールドは表示されません。同様に *hostname* で示すホストがアイドル状態になければ、 *idle_time* （アイドル時間）は表示されません。

各リモートホストは、 rusersd デーモンを実行している場合にのみ本コマンドに対して応答します。 このデーモンは通常 inetd(1M) または listen(1M) により起動されます。

オプション | -a　ログインしているユーザーがいないマシンに関しても報告します。

-h　ホスト名のアルファベット順にソートします。

-i　アイドル時間の順にソートします。

-l　who(1) の出力のような、長い形式のリストを表示します。

-u　ユーザー数の順にソートします。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu

関連項目 | who(1), inetd(1M), listen(1M), pmadm(1M), sacadm(1M), attributes(5)

名前 | script – 端末セッションの記録の生成

形式 | **script** [-a] [*filename*]

機能説明 | script は、そのユーザーの画面上で映し出される全出力の 記録を生成します。 記録は、 *filename* 引数で指定されたファイルに書き出されます。 ファイル名を省略すると、記録は typescript という名のファイルに保存されます。

script コマンドは、 $SHELL の値に従って サブシェルをフォークしたり作成したり、 このセッションからのテキストを記録します。 script が終了するのは、フォークされたシェルが終了したとき、または CTRL-D が入力されたときです。

オプション | -a　セッションの記録を、 *filename* の最後尾に追加します。上書きはしません。

注意事項 | script は、端末の画面上に現れるすべての出力を記録します。 これにはプロンプトも含まれます。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | attributes(5)

名前 | sdiff – 2つのファイル間の違いを並べて出力

形式 | **sdiff** [-l] [-s] [-o *output*] [-w *n*] *filename1 filename2*

機能説明 | sdiff は、 diff コマンドの出力を使用して、2 つのファイルの並列リストを作成して、異なっている行を示します。 2 つのファイルの行が同じ場合、これらの行は間に空白を 入れて出力されます。 また、この行が *filename1* にしかない場合には < が、この行が *filename2* にしかない場合には > が、またこれらの行が異なる場合には | が 1 つ間に入ります(「使用例」を参照)。

オプション

`-l` 同一の任意の行の左側だけを出力します。

`-s` 同一行を出力しません。

`-o` *output* 引数 *output* を *filename1* と *filename2* のユーザー制御のマージとして作成された第 3 ファイルの名前として使用します。 *filename1* と *filename2* の同一行が *output* にコピーされます。 diff が処理した結果を差異の種類によりグループ分けし、出力します。 このとき、この結果は共通のガータキャラクタを共有して表示されます。 それぞれの異なるセットを出力した後、 sdiff はユーザーに % のプロンプトを出し、ユーザーが次のいずれかのコマンドをタイプするのを待ちます。

l	出力ファイルに左側のカラムを追加
r	出力ファイルに右側のカラムを追加
s	サイレントモードにし、同一行を出力しない。
v	サイレントモードを終了
e l	左カラムでのエディタの呼び出し
e r	右カラムでのエディタの呼び出し
e b	左右カラムの連結でのエディタの呼び出し
e	長さ 0 のファイルでのエディタの呼び出し
q	プログラムを終了

エディタの終了時に、結果のファイルを *output* ファイルの最後に連結します。

`-w` *n* 引数 *n* を出力行の幅として使用。 デフォルトの行長は、 130 文字です。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の sdiff の動作については、largefile(5) を参照してください。

使用例 | **例 1** sdiff コマンドの例

sdiff の出力例は、次のとおりです。

```
x   |   y
a       a
```

例 1 sdiff コマンドの例　(続き)

```
b   <
c   <
d       d
    >   c
```

環境

LC_* 変数 (LC_CTYPE、LC_MESSAGES、LC_TIME、LC_COLLATE、LC_NUMERIC、LC_MONETARY) (environ(5) 参照) のいずれも環境に設定されていなければ、 それぞれ対応する ロケール のカテゴリにおける sdiff の動作は、環境変数 LANG によって決定されます。もし、LC_ALL が設定されていれば、その内容が LANG 変数やその他の LC_* 変数より優先されます。上記の変数が環境にまったく設定されていなければ、 C ロケールが sdiff の動作を決定します。

LC_CTYPE　sdiff の文字の処理方法を決定します。 LC_CTYPE に有効な値が設定されていると、 sdiff は、その ロケール にあった文字を含むテキストやファイル名を表示および処理できます。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

diff(1), ed(1), attributes(5), environ(5), largefile(5)

名前 | sed – ストリームエディタ

形式 | **/usr/bin/sed** [-n] *script* [*file*...]

/usr/bin/sed [-n] [-e *script*...] [-f *script_file*...] [*file*...]

/usr/xpg4/bin/sed [-n] *script* [*file*...]

/usr/xpg4/bin/sed [-n] [-e *script*...] [-f *script_file*...] [*file*...]

機能説明 | sed ユーティリティは、テキストファイルを読み込み、編集コマンドのスクリプトに従って内容を編集し、結果を標準出力に書き出すストリームエディタです。スクリプトは、*script* オペランド文字列か、-e *script* と -f *script_file* のオプション引数の組み合わせで指定します。sed ユーティリティはテキストエディタですが、バイナリファイルや、ASCII NUL (\0) 文字を含んだファイル、長い行を含んだファイルは編集できません。

オプション | 次のオプションを指定できます。

-e *script* — *script* は、sed の編集コマンドです。*script* のフォーマットの詳細については、「使用法」を参照してください。-e オプションを 1 つだけ指定し、-f オプションを指定しない場合、フラグ -e は省略できます。

-f *script_file* — *script_file* に指定したファイルからスクリプトを読み取ります。このファイルの各行には、編集コマンドが 1 つずつ記述されています。

-n — デフォルトの出力を抑止します。

-e と -f の両オプションは、それぞれ複数個指定できます。コマンドはすべて、どのオプションで指定されたかに関係なく、指定の順序でスクリプトに追加されます。

オペランド | 次のオペランドを指定できます。

file — 内容を読み込んで編集するファイルのパス名。*file* オペランドを複数個指定すると、指定した順序で読み込まれ、連結されたファイルを編集することになります。*file* を指定しないと、標準入力が読み込まれます。

script — 編集コマンドのスクリプトとして使用する文字列。アプリケーション側で、テキストファイルの規約に反するような *script* を指定することはできませんが、文字列の最後の文字が復帰改行文字でなくてもかまいません。

使用法 | スクリプトは、複数の編集コマンドで構成されます。スクリプト内の各行に、編集コマンドを 1 つずつ記述します。次のような書式を使用します。

[*address* [, *address*]] *command* [*arguments*]

最初のアドレス (*address*) の前と、コマンド (*command*) の前には、任意の数の空白文字を入力できます。また、最初のアドレスの前には、任意の数のセミコロンも入力できます。

通常、sed は、次の動作を繰り返します。

まず、D コマンドの実行後に何も残っていないことを確認し、入力行の 1 行を「パターンスペース」にコピーします。このとき、行末の復帰改行文字は含まれません。さらに、このパターンスペースを選択するアドレスを持つコマンドがあれば、そのコマンドをすべてパターンスペースに適用します。次に、その結果を標準出力にコピーし、パターンスペースを削除します。ただし、-n が指定されているときは、この処理は行いません。このようにしてパターンスペースを標準出力または指定ファイルに書き出すたびに、sed はその後に復帰改行文字を付け加えます。

コマンドのいくつかは、以前に取り出したパターンスペースを取り出しやすくするために、その一部または全部を「ホールドスペース」に保存します。パターンスペースとホールドスペースには、それぞれ最大 8192 バイトのデータを記録できます。

sed のアドレス

address には、10 進数 (すべてのファイルの入力行を累積的にカウントする)、$ (入力の最終行をアドレス指定する)、コンテキストアドレスのほか、空 (区切りのみ) を指定できます。*address* は、regexp(5) のマニュアルページで解説しているように、スラッシュで囲んだ正規表現 (*/regular expression/*) で構成されます。

アドレスのないコマンド行は、すべてのパターンスペースを選択します。

アドレスを 1 つ持つコマンド行は、そのアドレスと一致するパターンスペースを選択します。

アドレスを 2 つ持つコマンド行は、1 つめのアドレスと一致する最初のパターンスペースから 2 つめのアドレスと一致する次のパターンスペースまでの範囲を選択します。以降、同じプロセスが繰り返されます (2 つめのアドレスが、1 つめのアドレスで選択した行番号以下の場合、1 つめのアドレスに対応する行だけが選択されます)。

通常、アドレスとアドレスの間はコンマ (,) で区切りますが、セミコロン (;) を使うこともできます。

sed の正規表現

sed は、regexp(5) のマニュアルページで説明されている正規表現のほか、次の指定をサポートしています。

cREc
: コンテキストアドレスでは、*\cREc* は */RE/* と同じ意味になります (*c* はバックスラッシュと復帰改行文字を除く任意の文字。*RE* は正規表現)。*c* に指定した文字がバックスラッシュの直後に現れた場合、これは正規表現の終了文字ではなく、リテラル文字と見なされます。たとえば、\xabc\xdefx というコンテキストアドレスでは、2 つめの x はリテラル文字 x です。この場合、正規表現は abcxdef になります。

\n
: エスケープシーケンス \n は、パターンスペースに埋め込まれた復帰改行文字と一致します。コンテキストアドレスの正規表現内や置換コマンド内では、復帰改行文字をリテラル文字として処理で

きません。

否定コマンド ! を使用すると、編集コマンドは、選択されていないパターンスペースに適用されます (下記参照)。

sed の編集コマンド

以下の表に、各機能に指定できるアドレスの最大数を一覧します。

r コマンドと w コマンドには、それぞれ *rfile* と *wfile* という任意指定のパラメータがあります。コマンド文字とパラメータの間は、1 つ以上の空白文字で区切ります。

複数のコマンドをセミコロン (;) で区切って、1 つのコマンド行に記述できます。

引数 *text* は、行で構成されます。この引数が複数の行で構成されている場合、最終行以外のすべての行は \ で終了し、復帰改行をエスケープします。復帰改行文字をテキストに埋め込む場合は、その前にバックスラッシュを付加してください。テキストに埋め込まれたその他のバックスラッシュは削除され、その直後の文字がリテラル文字として処理されます。つまり、テキストに埋め込まれたバックスラッシュは、s コマンドの置換文字列中のバックスラッシュと同様に処理されることになります。また、バックスラッシュを使用して、スクリプト行の行頭の空白文字やタブ文字が削除されるのを防ぐこともできます。*rfile* 引数や *wfile* 引数は、その前に空白文字を 1 つ入力して、コマンド行の末尾に置いてください。*wfile* パラメータを指定すると、そのファイルが作成されます。すでにファイルが存在していれば、その内容が書き換えられます。最大 10 個の *wfile* 引数を指定できます。

正規表現のマッチングは、行単位ではなく、文字列全体で行われます。ただし、sed の正規表現では、復帰改行文字は \n と一致します。このため、正規表現内では復帰改行文字は使用できません。また、入力行末尾の復帰改行文字のマッチングに、\n は使用できません。パターンスペースには、復帰改行文字は編集コマンド N の結果として現れます。

コマンドのうち 2 つは *command-list* の形式をとります。これは復帰改行で区切られたいくつかのコマンドの集まりです。以下にその形式を示します。

```
{ command
command
}
```

{ の前後には空白文字を置くことができます。また、*command* の前にも空白文字を置くことができます。最後の } の前は復帰改行文字でなければなりません。この復帰改行文字の前後にも、空白文字を置くことができます。さらに、中括弧の前後にも空白文字を置くことができます。コマンドの前にも空白文字を置くことができますが、後には置けません。

次の表は、機能の一覧です。

最大アドレス数	コマンド	説明
1	a\ *text*	N コマンド実行または新サイクル開始による追加。次の入力行を読み取る前に *text* を出力します。
2	b *label*	*label* を持つ : コマンドに分岐します。*label* が空の場合は、スクリプトの最後に分岐します。ラベルは最大 8 文字が一意であると認識されます。
2	c*text*	変更。パターンスペースを削除します。*text* を出力します。次のサイクルを起動します。
2	d	パターンスペースを削除します。次のサイクルを起動します。
2	D	パターンスペースの先頭セグメントから最初の復帰改行までを削除します。次のサイクルを起動します (下記の N コマンドを参照) 。
2	g	パターンスペースの内容をホールドスペースの内容で置換します。
2	G	ホールドスペースの内容をパターンスペースに追加します。
2	h	ホールドスペースの内容をパターンスペースの内容で置換します。
2	H	パターンスペースの内容をホールドスペースに追加します。
1	i*text*	挿入。*text* を標準出力に出力します。
2	l	usr/bin/sed: 明白な形式でパターンスペースを標準出力に一覧表示します。非印字文字は 8 進数表記法で表示されます。長い行は複数行に分けて表示されます。
		/usr/xpg4/bin/sed: 明白な形式でパターンスペースを標準出力に一覧表示します。非印字文字は 8 進数表記法で表示されます。長い行は複数行に分けて表示されます。文字 \\、\a、\b、\f、\r、\t、\v が対応するエスケープシーケンスとして出力されます。テーブル中にない非印字文字に関しては、各バイトごと (最上位ビットから) に 3 桁の 8 進数で、前にバックスラッシュが付加された形式で出力されます。1 バイトが 10 ビット以上のシステムでは、非印字文字のフォーマットは導入先により異なります。長い行は折り返されます。折り返しの発生地点にはバックスラッシュと復帰改行文字が表示されます。折り返し地点の長さは不定ですが、出力装置に適した値です。各行の終わりは $ でマークされます。

最大 アドレス数	コマンド	説明
2	`n`	デフォルト出力が抑止されていなければ、パターンスペースを標準出力にコピーします。パターンスペースを入力の次の行で置換します。
2	`N`	入力の次の行をパターンスペースに追加し、復帰改行文字を埋め込みます。(現在の行番号が変わります)。入力に次の行がなければ、`N` コマンド動詞はスクリプトの終わりに分岐し、新たなサイクルを開始せずに、パターンスペースを書き出さずに終了します。
2	`p`	出力。パターンスペースを標準出力にコピーします。
2	`P`	パターンスペースの先頭セグメントから最初の復帰改行までを標準出力にコピーします。
1	`q`	終了。スクリプトの最後に分岐します。新しいサイクルを起動しません。
2	`r` *rfile*	*rfile* の内容を読み取ります。次の入力行を読み取る前にこの内容を出力します。*rfile* が存在しないまたは読み取れないときは、エラーとはならず、あたかも空のファイルであるかのように扱われます。
2	`t` *label*	テスト。入力行の最新の読み取りまたは `t` の実行以降に代入が行われている場合は、*label* を持つ `:` コマンドに分岐します。*label* が空の場合は、スクリプトの最後に分岐します。
2	`w` *wfile*	書き込み。*wfile* にパターンスペースを追加します。最初の `w` が 発生すると、*wfile* がクリアされます。後続の `w` が呼び出されると、追加されます。 `sed` コマンドが使用されるごとに、*wfile* が上書きされます。
2	`x`	パターンスペースおよびホールドスペースの内容を交換します。
2	`!` *command*	否定。アドレスによって選択されなかった行だけに *command* (*command* が `{` の場合にはグループ) を適用します。
0	`:` *label*	このコマンドは何もしません。これは分岐するための `b` および `t` コマンド用の *label* を持っています。
1	`=`	現在の行番号を 1 行として標準出力します。
2	{*command-list*}	パターンスペースが選択された場合のみ、*command-list* を実行します。
0		空のコマンドは無視されます。

最大 アドレス数	コマンド	説明
0	#	スクリプトファイルの行の先頭文字が # の場合、その行全体が注釈行として扱われます。ただし、先頭行に # があり、# の後の文字が n の場合は唯一の例外で、デフォルトの出力が抑止されます。#n の後の残りの行も無視されます。スクリプトファイルには 1 つ以上の非注釈行が必要です。

最大アドレス数	コマンド (*strings* を使用) と説明
2	s/*regular expression*/*replacement*/*flags* パターンスペース内の *regular expression* に一致する文字列を *replacement* に置換します。正規表現 (regular expression) と置換文字列 (replacement) を区切るのに、スラッシュの代わりに、バックスラッシュと復帰改行以外の任意の文字を使用できます。正規表現および置換文字列内で、区切り文字を文字列の一部として使用したければ、前にバックスラッシュを付加してください。 置換文字列中のアンパサンド (&) は、正規表現に一致する文字列に置き換えられます。この & が持つ特殊な意味は、前にバックスラッシュを付加すれば無効にできます。数字 *n* の前にバックスラッシュを付加した *n* は、対応する後方参照表現と一致するテキストに置き換えられます。置換文字列を先頭から最後まで走査する間に検出した個々のバックスラッシュについては、その直後の文字に特殊な意味があってもそれは無効となります。&、\、数字以外の文字がそれぞれどのような特殊な意味を持つかは不定です。 行は、その中に復帰改行文字を使用することによって分割できます。アプリケーションは、*replacement* 内の復帰改行文字の前にバックスラッシュを置くことによって、これをエスケープする必要があります。置換文字列が、置換後の文字列と同一である場合も、置換が行われたことになります。 *flags* には以下のものを、0 個以上指定できます。 *n* n= 1 – 512 。*regular expression* の *n* 番目の発生を置換します。 g グローバル。*regular expression* の最初に一致したものだけでなく、重ならないすべての例を置換します。*g* と *n* の両方を指定した場合、結果は保証できません。 p 置換が行われた場合、パターンスペースを出力します。 P パターンスペースの先頭セグメントから最初の復帰改行までを、標準出力にコピーします。 w *wfile* 書き込み。置換が行われた場合は、*wfile* にパターンスペースを追加します。最初の w は *wfile* をクリアします。それ以後の w 呼び出しは、追加処理となります。sed コマンドが使われるたびに、*wfile* は上書きされます。
2	y/ *string1* / *string2* /

最大アドレス数	コマンド (*strings* を使用) と説明
	変形。*string1* にあるすべての文字を *string2* の対応する文字と交換します。*string1* と*string2* の文字は同じ数でなければなりません。*string1* 中に同じ文字が複数回現れた場合の結果は定義されていません。2 つの文字列を区切るのに、スラッシュの代わりに、バックスラッシュと復帰改行以外の任意の文字を使用できます。*string1* と*string2* 内で、区切り文字を文字列の一部として使用したければ、前にバックスラッシュを付加してください。たとえば、y/abc/ABC/は、a を A で、b を B で、および c を C で置き換えます。

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `sed` の動作については、`largefile`(5) を参照してください。

使用例

例 **1** `sed` スクリプトの例

以下に示す `sed` スクリプトは、BSD `cat -s` コマンドをシミュレートし、標準入力から余分な空白行を圧縮します。

```
sed –n '
# Write non-empty lines.
/./     {
        p
        d
        }
# Write a single empty line, then look for more empty lines.
/^$/        p
# Get next line, discard the held <newline> (empty line),
# and look for more empty lines.
:Empty
/^$/        {
        N
        s/.//
        b Empty
        }
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
        p
'
```

環境

`sed` の実行に影響を与える環境変数 `LC_COLLATE`、`LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` の詳細については、`environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 正常終了

`>0` エラーが発生した

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

/usr/bin/sed

属性タイプ	属性値

使用条件	SUNWcsu
CSI	未対応

/usr/xpg4/bin/sed

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目 awk(1)、ed(1)、grep(1)、attributes(5)、environ(5)、largefile(5)、regexp(5)、XPG4(5)

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | set, unset, setenv, unsetenv, export – 現在のシェルおよびそこから起動されたプロセスでの環境変数の特性を決定するシェル組み込み関数

sh | **set** [--aefhkntuvx [*argument*]]...

unset [*name*...]

export [*name*...]

csh | **set** [*var* [= *value*]]

set *var* [*n*] = *word*

unset *pattern*

setenv [*VAR* [*word*]]

unsetenv *variable*

ksh | **set** [±aefhkmnopstuvx] [±o *option*]... [±A *name*] [*arg*...]

unset [-f] *name*...

****export** [*name* [=*value*]]...

sh | set 組み込みコマンドには次のようなオプションがあります。

`--` どのフラグも変更しません。 `$1` に - を設定する際に便利です。

`-a` 修正または作成された変数にエクスポートのマークを付けます。

`-e` コマンドが 0 以外の終了状態で終了した場合、直ちに終了します。

`-f` ファイル名を生成しないようにします。

`-h` 関数の定義時に、関数コマンドを検索しその位置を記憶します (通常、関数コマンドは関数実行時に検索されます)。

`-k` コマンド名の前にあるキーワード引数だけでなく、 すべてのキーワード引数がコマンド用の環境に置かれます。

`-n` コマンドを読み取るが、実行しません。

`-t` 1 つのコマンドを読み取り、実行したあと、終了します。

`-u` 未設定の変数を置換時にエラーとして扱います。

`-v` シェル入力行の読み取り時に、その内容を表示します。

`-x` コマンドの実行時に、コマンドと引数の内容を表示します。

- の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェルの起動時にも使用できます。 現在セットされているフラグは、 `$-` で見つけられます。 *argument* は定位置パラメタで、 `$1`、`$2`、... へ順に割り当てられます。 *argument* が指定されない場合、 すべての名前の値が出力されます。 unset は *name* ごとに、対応する変数または関数値を削除します。 変数 PATH 、PS1、 PS2、MAILCHECK および IF は設定解除できません。

export 組み込みコマンドでは、 指定された *name* に対し、ひきつづき実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。 引数を省略すると、 現在のシェル実行中にエクスポートのマークが付けられた 変数名を一覧表示します。 関数名はエクスポートされません。

csh 引数を指定しないと set はすべてのシェル変数の値を表示します。 複数ワードからなる値は括弧でくくられて表示されます。 引数 *var* だけを指定すると、 set は空 (NULL) の値を *var* が示す変数に割り当てます。 引数を *var* = *value* の形式で指定すると、 set は、変数 *var* に値 *value* を割り当てます。 *value* は次のいずれかです。

word 単一ワード (もしくは引用符付きの文字列)

(*wordlist*) 空白で区切られた、括弧付きワードの並び

値は、割り当てられる前に、コマンドおよびファイル名展開されます。 set *var*[*n*]=*word* 形式は、複数ワードからなる値の *n* 番目のワードを *word* に置き換えます。

unset は *pattern* が示すファイル名置換パターンに一致する名の変数を削除します。 'unset *' と指定すると、すべての変数が削除されます。 ただしこれは、 csh の動作に悪影響をおよぼします。

引数を指定しないと setenv はすべての環境変数を表示します。引数 *VAR* を指定すると、 setenv は環境変数 *VAR* に空の値 (NULL) を設定します (慣習上、環境変数名は大文字で指定されるのが通常)。 *VAR* と *word* の両引数を指定すると、 setenv は、*VAR* に単一ワードまたは引用符付き文字列である値 *word* を設定します。 環境変数 PATH は、コロンで区切られた複数の *word* 引数を指定できます (後述の「使用例」を参照) 。 最もよく使用される環境変数 USER 、 TERM 、 PATH は、自動的に csh 変数 user、term、path から (へ) インポート (エクスポート) されます。 これらの変数を変更する場合には setenv を使用してください。 さらにシェルは、 csh 変数 cwd が変更されるたびに、その値を 環境変数 PWD へ設定します。

環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、LC_COLLATE、LC_NUMERIC、LC_MONETARY は、C シェル内で変更されると新しい値が即座に有効になります。 これらの環境変数の詳細については environ(5) を参照してください。

unsetenv は環境から *variable* が示す変数を削除します。 unset のようなパターンマッチングは行いません。

ksh set コマンドのフラグの意味は以下のとおりです。

-A 配列の代入。 *name* で示される変数の設定を解除し、 *arg* リストから順々に値を割り当てます。 +A は、 最初の変数設定を解除しません。

-a 定義される後続の変数すべてを自動的にエクスポートします。

-e コマンドの終了状態が 0 でない場合、 ERR トラップ (設定されていれば) を実行し、 終了します。 このモードは、 プロファイル読み取り時は無効です。

-f ファイル名の生成を無効にします。

`-h` 各コマンドは、 最初に検出された時点で、 検索済み別名になります。

`-k` コマンド名に先行するものだけでなく、 すべての変数代入引数をコマンドの環境に格納します。

`-m` バックグラウンドジョブを個別のプロセスグループで実行し、 完了時にメッセージを 1 行表示します。 バックグラウンドジョブの終了状態は 完了メッセージで報告されます。 ジョブ制御を備えたシステムでは、 このフラグは 対話型シェルに対して自動的に有効になります。

`-n` コマンドを読み取り、 構文エラーがないかチェックします。 実行は行いません。 対話型シェルに対しては無視されます。

`-o` このフラグの後に指定する引数は、 以下のオプション名のいずれかです。

`allexport`	`-a` と同じです。
`errexit`	`-e` と同じです。
`bgnice`	バックグラウンドジョブをすべて低い優先度で実行します。 これはデフォルトモードです。
`emacs`	コマンド入力用に、 `emacs` 形式のインラインエディタを起動します。
`gmacs`	コマンド入力用に、 `gmacs` 形式のインラインエディタを起動します。
`ignoreeof`	ファイルの終わりを検出してもシェルは終了しません。 終了させるには `exit` コマンドを使用する必要があります。
`keyword`	`-k` と同じです。
`markdirs`	ファイル名生成によって生成されるディレクトリ名には、 すべて最後に `/` を付加します。
`monitor`	`-m` と同じです。
`noclobber`	`>` によるリダイレクトが存在するファイルを切り捨てないようにします。 このオプションが有効なとき、 ファイルを切り捨てるには `>|` 指定が必要です。
`noexec`	`-n` と同じです。
`noglob`	`-f` と同じです。
`nolog`	履歴ファイルに関数定義を保存しません。
`nounset`	`-u` と同じです。
`privileged`	`-p` と同じです。
`verbose`	`-v` と同じです。
`trackall`	`-h` と同じです。

vi	vi 形式のインラインエディタの挿入モードになります。 033 というエスケープ文字を押すと、 挿入モードから制御モードに変わります。 Return で行を送信します。
viraw	各文字を vi モードで入力されたときと同様に処理します。
xtrace	-x と同じです。

オプション名を指定しない場合、 -o は現在のオプション設定を表示します。

-p	$HOME/.profile ファイルを処理しないようにし、 ENV ファイルの代わりに /etc/suid_profile ファイルを使用します。 このモードは、 実効ユーザー ID が実ユーザー ID と等しくないとき、また実効グループ ID が実グループ ID と等しくないときには 必ず有効になります。 このモードを無効にすると、 実効ユーザー ID が実ユーザー ID に、 実効グループ ID が実グループ ID にそれぞれ設定されます。
-s	定位置パラメタを辞書編集方式の順にソートします。
-t	コマンド 1 つを読み取って実行し、 終了します。
-u	置換を行う際に、設定されていないパラメタを エラーとして扱います。
-v	シェルへの入力行を読み取り時に表示します。
-x	コマンドとその引数を実行時に表示します。
–	-x フラグと -v フラグを無効にし、 フラグに対する引数の検査を停止します。
– –	どのフラグも変更しません。 このフラグは、 $1を – で始まる値に設定する際に便利です。 このフラグの後に引数がない場合、 定位置パラメタが設定解除されます。

– の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェル起動時にも使用できます。 現在設定されているフラグは、 $– で見ることができます。 -A を指定しないかぎり、 残りの引数は定位置パラメタとなり、 $1 $2 ... に順番に割り当てられます。 引数を 1 つも指定しない場合には、 すべての変数の名前と値を標準出力上に表示します。

name が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、 MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

unset を使用すると *name* が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、

MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

export 組み込みコマンドでは、 指定された *name* に対し、 後で実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

1 つまたは 2 つの (*) アスタリスクが先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

csh 次の例では、 /bin、/usr/bin、/usr/sbin、/usr/ucb/bin ディレクトリにあるファイルをその順番で検索するために、 環境変数 PATH を設定しています。

```
setenv PATH "/bin:/usr/bin:/usr/sbin:usr/ucb/bin"
```

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), read(1), sh(1), typeset(1), attributes(5), environ(5)

名前 | set, unset, setenv, unsetenv, export – 現在のシェルおよびそこから起動されたプロセスでの環境変数の特性を決定するシェル組み込み関数

sh **set** [--aefhkntuvx [*argument*]]...

unset [*name*...]

export [*name*...]

csh **set** [*var* [= *value*]]

set *var* [*n*] = *word*

unset *pattern*

setenv [*VAR* [*word*]]

unsetenv *variable*

ksh **set** [±aefhkmnopstuvx] [±o *option*]... [±A *name*] [*arg*...]

unset [-f] *name*...

****export** [*name* [=*value*]]...

sh set 組み込みコマンドには次のようなオプションがあります。

-- どのフラグも変更しません。 $1 に - を設定する際に便利です。

-a 修正または作成された変数にエクスポートのマークを付けます。

-e コマンドが 0 以外の終了状態で終了した場合、直ちに終了します。

-f ファイル名を生成しないようにします。

-h 関数の定義時に、関数コマンドを検索しその位置を記憶します (通常、関数コマンドは関数実行時に検索されます)。

-k コマンド名の前にあるキーワード引数だけでなく、 すべてのキーワード引数がコマンド用の環境に置かれます。

-n コマンドを読み取るが、実行しません。

-t 1 つのコマンドを読み取り、実行したあと、終了します。

-u 未設定の変数を置換時にエラーとして扱います。

-v シェル入力行の読み取り時に、その内容を表示します。

-x コマンドの実行時に、コマンドと引数の内容を表示します。

- の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェルの起動時にも使用できます。 現在セットされているフラグは、 $- で見つけられます。 *argument* は定位置パラメタで、 $1、$2、... へ順に割り当てられます。 *argument* が指定されない場合、 すべての名前の値が出力されます。 unset は *name* ごとに、対応する変数または関数値を削除します。 変数 PATH 、PS1、 PS2、 MAILCHECK および IF は設定解除できません。

export 組み込みコマンドでは、 指定された *name* に対し、ひきつづき実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。 引数を省略すると、 現在のシェル実行中にエクスポートのマークが付けられた 変数名を一覧表示します。 関数名はエクスポートされません。

csh 引数を指定しないと set はすべてのシェル変数の値を表示します。 複数ワードからなる値は括弧でくくられて表示されます。 引数 *var* だけを指定すると、 set は空 (NULL) の値を *var* が示す変数に割り当てます。 引数を *var* = *value* の形式で指定すると、 set は、変数 *var* に値 *value* を割り当てます。 *value* は次のいずれかです。

word	単一ワード (もしくは引用符付きの文字列)
(*wordlist*)	空白で区切られた、括弧付きワードの並び

値は、割り当てられる前に、コマンドおよびファイル名展開されます。 set *var*[*n*]=*word* 形式は、複数ワードからなる値の *n* 番目のワードを *word* に置き換えます。

unset は *pattern* が示すファイル名置換パターンに一致する名の変数を削除します。 'unset *' と指定すると、すべての変数が削除されます。 ただしこれは、 csh の動作に悪影響をおよぼします。

引数を指定しないと setenv はすべての環境変数を表示します。引数 *VAR* を指定すると、 setenv は環境変数 *VAR* に空の値 (NULL) を設定します (慣習上、環境変数名は大文字で指定されるのが通常)。 *VAR* と *word* の両引数を指定すると、 setenv は、*VAR* に単一ワードまたは引用符付き文字列である値 *word* を設定します。 環境変数 PATH は、コロンで区切られた複数の *word* 引数を指定できます (後述の「使用例」を参照) 。 最もよく使用される環境変数 USER 、 TERM 、 PATH は、自動的に csh 変数 user、term、path から (へ) インポート (エクスポート) されます。 これらの変数を変更する場合には setenv を使用してください。 さらにシェルは、 csh 変数 cwd が変更されるたびに、その値を 環境変数 PWD へ設定します。

環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、LC_COLLATE、LC_NUMERIC、LC_MONETARY は、C シェル内で変更されると新しい値が即座に有効になります。 これらの環境変数の詳細については environ(5) を参照してください。

unsetenv は環境から *variable* が示す変数を削除します。 unset のようなパターンマッチングは行いません。

ksh set コマンドのフラグの意味は以下のとおりです。

-A	配列の代入。 *name* で示される変数の設定を解除し、 *arg* リストから順々に値を割り当てます。 +A は、 最初の変数設定を解除しません。
-a	定義される後続の変数すべてを自動的にエクスポートします。
-e	コマンドの終了状態が 0 でない場合、 ERR トラップ (設定されていれば) を実行し、 終了します。 このモードは、 プロファイル読み取り時は無効です。
-f	ファイル名の生成を無効にします。

-h　各コマンドは、 最初に検出された時点で、 検索済み別名になります。

-k　コマンド名に先行するものだけでなく、 すべての変数代入引数をコマンドの環境に格納します。

-m　バックグラウンドジョブを個別のプロセスグループで実行し、 完了時にメッセージを 1 行表示します。 バックグラウンドジョブの終了状態は 完了メッセージで報告されます。 ジョブ制御を備えたシステムでは、 このフラグは 対話型シェルに対して自動的に有効になります。

-n　コマンドを読み取り、 構文エラーがないかチェックします。 実行は行いません。 対話型シェルに対しては無視されます。

-o　このフラグの後に指定する引数は、 以下のオプション名のいずれかです。

allexport	-a と同じです。
errexit	-e と同じです。
bgnice	バックグラウンドジョブをすべて低い優先度で実行します。 これはデフォルトモードです。
emacs	コマンド入力用に、 emacs 形式のインラインエディタを起動します。
gmacs	コマンド入力用に、 gmacs 形式のインラインエディタを起動します。
ignoreeof	ファイルの終わりを検出してもシェルは終了しません。 終了させるには exit コマンドを使用する必要があります。
keyword	-k と同じです。
markdirs	ファイル名生成によって生成されるディレクトリ名には、 すべて最後に / を付加します。
monitor	-m と同じです。
noclobber	> によるリダイレクトが存在するファイルを切り捨てないようにします。 このオプションが有効なとき、 ファイルを切り捨てるには >| 指定が必要です。
noexec	-n と同じです。
noglob	-f と同じです。
nolog	履歴ファイルに関数定義を保存しません。
nounset	-u と同じです。
privileged	-p と同じです。
verbose	-v と同じです。
trackall	-h と同じです。

vi	vi 形式のインラインエディタの挿入モードになります。 033 というエスケープ文字を押すと、 挿入モードから制御モードに変わります。 Return で行を送信します。
viraw	各文字を vi モードで入力されたときと同様に処理します。
xtrace	-x と同じです。

オプション名を指定しない場合、 -o は現在のオプション設定を表示します。

-p	$HOME/.profile ファイルを処理しないようにし、 ENV ファイルの代わりに /etc/suid_profile ファイルを使用します。 このモードは、 実効ユーザー ID が実ユーザー ID と等しくないとき、また実効グループ ID が実グループ ID と等しくないときには 必ず有効になります。 このモードを無効にすると、 実効ユーザー ID が実ユーザー ID に、 実効グループ ID が実グループ ID にそれぞれ設定されます。
-s	定位置パラメタを辞書編集方式の順にソートします。
-t	コマンド 1 つを読み取って実行し、 終了します。
-u	置換を行う際に、設定されていないパラメタを エラーとして扱います。
-v	シェルへの入力行を読み取り時に表示します。
-x	コマンドとその引数を実行時に表示します。
–	-x フラグと -v フラグを無効にし、 フラグに対する引数の検査を停止します。
– –	どのフラグも変更しません。 このフラグは、 $1を – で始まる値に設定する際に便利です。 このフラグの後に引数がない場合、 定位置パラメタが設定解除されます。

– の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェル起動時にも使用できます。 現在設定されているフラグは、 $- で見ることができます。 -A を指定しないかぎり、 残りの引数は定位置パラメタとなり、 $1 $2 ... に順番に割り当てられます。 引数を 1 つも指定しない場合には、 すべての変数の名前と値を標準出力上に表示します。

name が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、 MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

unset を使用すると *name* が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、

MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

export 組み込みコマンドでは、 指定された *name* に対し、 後で実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

1 つまたは 2 つの (*) アスタリスクが先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

csh 次の例では、 /bin、/usr/bin、/usr/sbin、/usr/ucb/bin ディレクトリにあるファイルをその順番で検索するために、 環境変数 PATH を設定しています。

```
setenv PATH "/bin:/usr/bin:/usr/sbin:usr/ucb/bin"
```

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), read(1), sh(1), typeset(1), attributes(5), environ(5)

名前 | touch, settime – ファイルのアクセス日時および更新日時の変更

形式 | **touch** [-acm] [-r *ref_file* | -t *time*] *file...*

touch [-acm] [*date_time*] *file...*

settime [-f *ref_file*] [*date_time*] *file...*

機能説明 | touch ユーティリティは、各ファイルのアクセス日時または更新日時を設定します。指定した *file* が存在しない場合には、*file* が作成されます。

新たに設定する日時は、 -t *time*、 -r *ref_file* に指定したファイルの時刻フィールド、または *date_time* オペランドによって、指定することができます。 これらのオプションやオペランドを 1 つも指定しない場合には、 touch は、 time(2) 関数によって返される現在の日時を設定します。

-a オプションと -m オプションの両方が省略された場合、touch は更新日時とアクセス日時の両方を更新します。

ファイルへの書き込み権を持つが、そのファイルの所有者ではなく、スーパーユーザーでもないユーザーは、ファイルの更新日時とアクセス日時を現在の日時にだけ変更できます。touch で特定の日時に設定しようとするとエラーになります。

settime ユーティリティは、touch -c [*date_time*] *file* と同等です。

touch | touch には、以下のオプションを指定できます。

-a　*file* のアクセス日時だけを変更します。 -m オプションも同時に指定された場合を除き、 更新日時は変更しません。

-c　*file* が示す名前のファイルが存在しない場合、そのファイルを作成しません。 また、この状態についての診断メッセージを出力しません。

-m　*file* の更新日時だけを変更します。-a オプションも同時に指定しなければ、アクセス日時は更新しません。

-r *ref_file*　設定する日時として、現在の日時の代わりに *ref_file* が示すファイルの日時を使用します。

-t *time*　設定する日時として、現在の日時の代わりに *time* が示す値を使用します。 *time* は以下の形式の 10 進数です。

[[*CC*] *YY*] *MMDDhhmm* [.*SS*]

各 2 桁の値は次のような意味を持ちます。

MM　月 [01-12]

DD　日 [01-31]

hh　時 [00-23]

mm　分 [00-59]

CC　西暦年の上 2 桁

YY　西暦年の下 2 桁

SS　秒 [00-61]

CC と *YY* はともに省略可能です。両方とも省略すると、現在の年と見なされます。*YY* を指定して *CC* を省略すると、 *CC* は以下に示す値と見なされます。

YY の値	**CC** のデフォルト
69-99	19
00-38	20
39-68	エラー

結果として得られる日時の値は、環境変数 `TZ` の値によって影響を受けます。日時の値が 1970 年 1 月 1 日 0 時 0 分 0 秒 (グリニッジ標準時) 以前を示しているとき、`touch` はエラーとなりただちに実行を終了します。 許される日時の値の範囲は、1970 年 1 月 1 日 0 時 0 分 0 秒 (グリニッジ標準時) 以後から 2038 年 1 月 18 日までです。

SS の範囲は、 [00-59] ではなく [00-61] です。これはうるう秒を考慮しているためです。 *SS* が 60 または 61、`TZ` により加工された後の時刻の値がうるう秒を表していない場合、結果の時刻値は *SS* を 59 とした時刻の 1 秒後 または 2 秒後となります。 *SS* を省略すると 0 と見なされます。

settime　`settime` には、以下のオプションを指定できます。

`-f` *ref_file*　現在の日時の代わりに、*ref_file* に指定されたファイルの日時を使用します。

オペランド　`touch` および `settime` には、 以下のオペランドを指定できます。

file　日時を変更するファイルのパス名

date_time　現在の日時の代わりに *date_time* を使用します。*date_time* は以下の形式の 10 進数です。

MMDDhhmm [*YY*] 各 2 桁の値は次のような意味を持ちます。

file　日時を変更するファイルのパス名。

MM　月 [01-12]

DD　日 [01-31]

hh　時 [00-23]

mm　分 [00-59]

YY　西暦年の下 2 桁。

YY は省略可能です。省略すると、現在の年と見なされます。 *YY* を指定すると、 年は以下に示す値と見なされます。

YY	対応する年
69-99	1969-1999
00-38	2000-2038
39-68	エラー

`-r` オプションと `-t` オプションの両方を省略し、最低 2 つのオペランドが指定され、そのうちの先頭のオペランドの値が 8 または 10 桁の 10 進数であるとき、その先頭オペランドは *date_time* であると見なされます。それ以外の場合には、先頭オペランドは `file` であると見なされます。

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `touch` の動作については、`largefile`(5) を参照してください。

環境 `touch` の実行に影響を与える環境変数 `LANG`、`LC_ALL`、`LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、 `environ`(5) を参照してください。

`TZ` *time* オプション引数 や *date_time* オペランド に適用するタイムゾーンを指定します。

終了ステータス 以下の終了ステータスが返されます。

`0` `touch` の実行が正常終了し、要求されたすべての変更が行われた

`>0` エラーが発生した。 `touch` は、日時の変更を実施できなかったファイルの数を返す

属性 次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 `time`(2), `attributes`(5), `environ`(5), `largefile`(5)

注意事項 BSD 環境に熟知しているユーザーは、 `-f` オプションが `touch` に 受け入れられたのに `-f` オプションは無視されることに気づくでしょう。 ファイル上のアクセス権にかかわりなく、 `touch` は ユーザーが所有するすべてのファイルに対して成功するので、 `-f` オプションは不要です。

名前 | sh, jsh – 標準シェルとジョブ制御シェルおよびコマンドインタプリタ

形式 | **/usr/bin/sh** [-acefhiknprstuvx] [*argument*...]

/usr/xpg4/bin/sh [± abCefhikmnoprstuvx] [± o *option*...] [-c *string*] [*arg*...]

/usr/bin/jsh [-acefhiknprstuvx] [*argument*...]

機能説明 | /usr/bin/sh ユーティリティは、端末またはファイルからコマンドを実行する コマンドプログラミング言語です。

/usr/xpg4/bin/sh ユーティリティは標準に準拠したシェルです。このユーティリティは ksh(1) のすべての機能を提供します。ただし、ksh(1) で説明しているように、動作が異なる場合を除きます。

jsh ユーティリティは、sh のすべての機能を備えた、かつジョブ制御を可能にするシェルへのインタフェースです (下記の「ジョブ制御」を参照) 。

シェルへの引数については、後述の「呼び出し」にリストされています。

定義 | ブランク (*blank*) はタブ (tab) または空白文字 (space) です。名前 (*name*) は ASCII 英文字、 数字、 または下線の並びで、 先頭文字は英文字または下線です。パラメタ (*parameter*) は、 名前、 複数、 または *、 @、 #、 ?、 –、 $、 ! などの文字のいずれかです。

コマンド | 単純コマンド (*simple-command*) は、ブランクで区切られた、ブランクでないワードの並びです。先頭のワードは、実行すべきコマンドの名前を指定します。残りのワードは、以下に述べる場合を除き、呼び出されたコマンドに引数として渡されます。コマンド名は引数 0 として渡されます (exec(2) を参照) 。単純コマンドの値 (*value*) は、正常終了した場合は終了ステータス、異常終了した場合は 200+*status* (8 進数) です。ステータス値の一覧表については、signal(3HEAD) を参照してください。

パイプライン (*pipeline*) は、パイプ (|) で区切られた 1 つ以上のコマンドの並びです。最後のコマンドを除き、各コマンドの標準出力は pipe(2) によってその次のコマンドの標準入力と結合されます。各コマンドは、別々のプロセスとして実行されます。シェルは最後のコマンドが終了するのを待ちます。最後のコマンドの終了ステータスが パイプライン全体の終了ステータスとなります。

リスト (*list*) は、; 、&、&&、または | | で区切られた 1 つ以上のパイプライン の並びです。その並びの終わりに ; または & を記述することもできます。これら 4 つの記号の中で、; と & の優先度は同じで、&& と | | の優先度より低くなります。&& と | | の優先度は同じです。セミコロン (;) によって、直前のパイプラインが順次実行されます。つまりシェルはパイプラインが終了するのを待ってから、セミコロンの後のコマンドを実行します。アンパサンド記号 (&) によって、直前のパイプラインが非同期的に実行されます。つまりシェルはパイプラインが終了するのを待ちません。&& という記号は、直前のパイプラインの終了ステータスが 0 の場合にだけ、後続のリストを実行するものです。反対に (| |) は、終了ステータスが 0 以外の場合にだけ、後続のリストを実行します。リスト中のコマンドを区切るのに、セミコロンの代わりに 任意の数の復帰改行 (newline) を指定することもできます。

コマンド (*command*) は、単純コマンドまたは以下のいずれかです。特に断わりのない限り、コマンドが返す値は、そのコマンド中で最後に実行された単純コマンドの値です。

`for` *name* [`in` *word* . . .] `do` *list* `done`
コマンドが実行されるたびに、 *name* は `in` *word* リストから次に得られる *word* に設定されます。`in` *word* . . . を省略すると、`for` コマンドは、設定された各定位値パラメタに対して、`do` *list* を 1 回実行します (後述の「パラメタ置換」を参照) 。リストの *word* がなくなると、実行は終了します。

`case` *word* `in` [*pattern* [| *pattern*]) *list* `;;`] . . . `esac`
`case` コマンドは、*word* に一致する最初の *pattern* に対応した *list* を実行します。パターンの形式は、ファイル名生成に使用される形式と同じです (「ファイル名の生成」の項を参照) 。ただしスラッシュ、先行するドット、およびスラッシュ直後のドットは、明示的に一致しなくてもかまいません。

`if` *list* `;` `then` *list* `;` [`elif` *list* `;` `then` *list* `;`] . . . [`else` *list* `;`] `fi`

`if` の後の *list* を実行後、*list* が 0 の終了ステータスを返すと、最初の `then` の後の *list* を実行します。それ以外の場合、`elif` の後の *list* を実行します。この値が 0 の場合、次の `then` の後の *list* を実行します。これが失敗すると、`else` *list* を実行します。`else` *list* も `then` *list* も実行しない場合、`if` コマンドは 0 の終了ステータスを返します。

`while` *list* `do` *list* `done`	`while` コマンドは、`while` *list* を繰り返し実行し、*list* 中の最後のコマンドの終了ステータスが 0 の場合、`do` *list* を実行します。それ以外の場合、ループは終了します。`do` *list* 中のコマンドを実行しない場合、`while` コマンドは 0 の終了ステータスを返します。ループ終了条件の判定を逆にするには、`while` の代わりに `until` を使用します。
(*list*)	サブシェル内の *list* の実行
{ *list* ; }	現在の (つまり、親) シェル内での *list* の実行。記号 { の後には空白が必要です。
name ( ) { *list* ; }	*name* が参照する関数を定義します。{ と } の間のコマンド群 (*list*) が関数の本体となります (後述の「関数」参照) 。{ の後には空白が必要です。関数の実行については「実行」の項で後述します。関数の本体が、上記の「コマンド」の項で 定義したようなコマンドの場合、{ および } は必要ではありません。

下記のワードは、コマンドの最初に現れたとき、およびクォートされずに記述されたときに認識されます。

`if then else elif fi case esac for while until do done { }`

注釈行

でワードを始めると、そのワードおよび以降の 復帰改行までの文字がすべて無視されます。

コマンド行置換

シェルは、2 つの逆引用符 (`...`) で囲まれた文字列から コマンドを読み取ります。これらのコマンドからの標準出力は、ワードの一部または全体として使用できます。標準出力上で最後につく復帰改行は削除されます。

文字列は、読み取られる前にはいっさい解釈されません。ただし例外として、文字のエスケープに使用されるバックスラッシュ (\) の削除だけは行われます。バックスラッシュは、逆引用符 (`) または別のバックスラッシュ (\) をエスケープするためにも使用され、コマンド文字列の読み取り前に削除されます。逆引用符をエスケープすることにより、コマンド置換のネストが可能になります。コマンド置換が、一対の二重引用符に囲まれている場合 ("...`...`...") 、二重引用符をエスケープしているバックスラッシュ (\") は削除されますが、それ以外のバックスラッシュはそのまま残されます。

復帰改行文字のエスケープにバックスラッシュを用いた場合は (\newline)、バックスラッシュと復帰改行の両方が削除されます (後述の「クォート」の項の後半を参照)。さらに、ドル記号 (\$) をエスケープしているバックスラッシュも削除されます。コマンド文字列に対するパラメタの置換は 読み取り前には行われないので、バックスラッシュでドル記号をエスケープしようとしても 無意味です。\、`、"、復帰改行 (newline)、および $ 以外の文字の前に付くバックスラッシュは、コマンド文字列の読み取り時にもそのまま残ります。

パラメタ置換

文字 $ は、置換可能なパラメタ (*parameters*) を示します。パラメタには、定位置パラメタとキーワードパラメタの 2 種類があります。パラメタが数字の場合、これは定位置パラメタです。定位置パラメタには、set コマンドによって値を割り当てられます。キーワードパラメタ (変数とも呼ばれる) には、以下の記述により値を代入することもできます。

name=*value* [*name*=*value*] . . .

value に対しては、パターンマッチングは行われません。同じ *name* を持つ関数と変数が存在することはできません。

${*parameter*}
: パラメタの値 (もしあれば) は置換されます。中括弧が必要となるのは、パラメタの後に、その名前の一部として解釈すべきでない文字、数字、または下線を指定するときだけです。*parameter* が * または @ の場合、$1 で始まる定位値パラメタはすべて (空白で区切られて) 置換されます。パラメタ $0 は、シェルが呼び出されたときに、引数 0 から設定されます。

${*parameter*:-*word*}
: *parameter* が NULL 以外の値に設定されている場合、その値に置き換えられます。その他の場合、*word* に置き換えられます。

${*parameter*:=*word*}	*parameter* が設定されていない場合、または NULL に設定されている場合、それを *word* に設定します。次に、パラメタの値を置き換えます。この方法で定位値パラメタに値を代入することはできません。
${*parameter*:?*word*}	*parameter* が NULL 以外の値に設定されている場合、その値に置き換えられます。その他の場合、*word* を出力しシェルを終了します。*word* を省略すると、メッセージ “parameter null or not set” が出力されます。
${*parameter*:+*word*}	*parameter* が NULL 以外の値に設定されている場合、*word* に置き換えられます。その他の場合は置換を行いません。

上記にあるように、*word* は、代入用文字列として使用する場合にだけ評価されます。たとえば次の例では、pwd が実行されるのは、d が設定されていないかあるいは NULL に設定されている場合だけです。

```
echo  ${d:-`pwd`}
```

上記の式からコロン (:) を省略すると、シェルは *parameter* が設定されているかどうかだけをチェックします。

次のパラメタは、シェルが自動的に設定します。

#	定位置パラメタ数 (10 進数)
-	呼び出し時にまたは set コマンドによってシェルに与えられたフラグ
?	最後に同期実行されたコマンドが返した 10 進数
$	このシェルのプロセス番号
!	最後に呼び出されたバックグラウンドコマンドのプロセス番号

次に挙げるパラメタはシェルが使用するもので、環境変数とも呼ばれるものです。

HOME	cd コマンドのデフォルト引数 (ホームディレクトリ) 。login(1) によって、パスワードファイルから ユーザーのログインディレクトリに設定されます (passwd(4) を参照) 。
PATH	コマンド用検索パス (後述の「実行」を参照) 。
CDPATH	cd コマンドの検索パス
MAIL	このパラメタにメールファイルの名前がセットされていて、かつ MAILPATH パラメタが設定されていない場合、シェルは指定されたファイルにメールが到着すると ユーザーに通知します。
MAILCHECK	このパラメタは、MAILPATH または MAIL パラメタで指定されたファイルへメールが到着したか否かを、シェルが何秒ごとにチェックするかを指定します。デフォルト値は 600 秒 (10 分) で

す。このパラメタの値として 0 が設定された場合、シェルは次のプロンプトを出す前に チェックを行います。

MAILPATH	コロン (:) で区切ったファイル名のリスト。このパラメタが設定されると、指定されたいずれかのファイルに メールが到着するたびに、シェルはユーザーに通知します。各ファイル名の後には、% および更新時刻の変更時に出力されるメッセージを 指定することができます。デフォルトのメッセージは you have mail です。
PS1	1 次プロンプト文字列。デフォルトは " $ " です。
PS2	2 次プロンプト文字列。デフォルトは " > " です。
IFS	内部フィールドセパレータ。通常は空白文字、タブ、および復帰改行です (「ブランクの解釈」を参照) 。
SHACCT	このパラメタにユーザーが書き込み可能な ファイル名が設定された場合、シェルは、実行された各シェルプロシージャごとの アカウンティングレコードをこのファイルに書き込みます。
SHELL	シェルは、呼び出されると、このパラメタが示す名前が環境中に存在するかを 確かめます (「環境」の項を参照) 。sh の実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES についての詳細は、environ(5) を参照してください。シェルは、PATH、PS1、PS2、MAILCHECK、および IFS にデフォルト値を割り当てます。HOME および MAIL のデフォルト値は login(1) で設定します。

ブランクの解釈

パラメタとコマンドの置換後、置換の結果内でフィールドセパレータ文字 (IFS で発見されるもの) を検索し、その文字が現れた位置で分割します。分割された各々が引数となります。明示的な NULL 引数 ("" または '') は保持されます。暗示的な NULL 引数 (値を持たないパラメタにより生ずるもの) は削除されます。

入出力のリダイレクト

シェルが解釈する特殊表記法によって、入出力をリダイレクションできます。以下は、単純コマンド内の任意の位置および コマンドの前後に指定することができ、起動されたコマンドには引き渡されません。なお *word* または *digit* を使用する前にコマンドとパラメタの置換が発生するので 注意してください。

<*word*	*word* というファイルを標準入力 (ファイル記述子 0) として使用します。
>*word*	*word* というファイルを標準出力 (ファイル記述子 1) として使用します。ファイルが存在しない場合は作成します。ファイルが存在していれば、ファイルの長さを 0 にします。
>>*word*	*word* というファイルを標準出力として使用します。ファイルが存在する場合、(EOF までシークした後) そのファイルに出力を追加します。ファイルが存在しない場合は、ファイルを作成します。
< >*word*	*word* というファイルを標準入力として読み書き用にオープンします。

<<[−]*word*

word に対するパラメタおよびコマンド置換が行われた後、その結果得られた *word* と文字どおり一致する最初の行が現れるまで、または EOF に達するまでシェルへの入力が読み取られます。ただし << に − が付加されて指定された場合は、以下のようになります。

1. まず、シェルへの入力の読み取り前に (ただしパラメタとコマンドの置換後) 、先行するタブを *word* から取り除きます。
2. 読み取ったシェルへの入力の各行を *word* と比較する前に、その行から先行するタブを取り除きます。そして、
3. その結果得られた *word* と文字どおり一致する最初の行が現れるまで、または EOF に達するまでシェルへの入力が読み取られます。

word 中のいずれかの文字がクォートされている場合 (後述の「クォート」を参照) 、シェルへの入力に対して追加処理は行われません。*word* 中のどの文字もクォートされていない場合は、以下のようになります。

1. まずパラメタとコマンドの置換を実施します。
2. エスケープされた復帰改行 (`\newline`) を削除します。
3. 文字 \、$、および ` に対しては、 \ を使ってクォートしなければなりません。

その結果得られるドキュメントが標準入力となります。

<&*digit*

ファイル記述子 *digit* に対応するファイルを標準入力として使用します。同様に、標準出力の使用には >&*digit* を指定します。

<&−

標準入力をクローズします。同様に、標準出力については >&− を使用します。

上記のいずれかの前に数字が付く場合、その値が (デフォルトの 0 または 1 の代わりに) 該当ファイルに対応した ファイル記述子となります。

```
... 2>&1
```

この例では、現在ファイル記述子 1 に関連しているファイルに、ファイル記述子 2 を関連付けます。

リダイレクションを指定する場合、記述する順序が重要になります。シェルは、リダイレクション記述を左から右へ評価します。

```
... 1>xxx 2>&1
```

上記の例では、まず *xxx* というファイルにファイル記述子 1 を関連付けます。次に、ファイル記述子 1 に関連するファイル (つまり *xxx*) に、ファイル記述子 2 を関連付けます。リダイレクションの向きが逆であれば、まずファイル記述子 2 を端末に関連付け (ファイルを記述子 1 が既に端末に関連付けられているとみなし) 、次にファイル記述子 1 をファイル *xxx* に関連付けます。

最初のページの「コマンド」の項で述べた用語を使って説明すると、以下のようになります。コマンドが複数の単純コマンドで構成される場合、リダイレクションは、個々の単純コマンドに対して行う前に、コマンド全体に対して評価されます。すなわち、シェルはまずリスト 全体に対してリダイレクションを評価し、次にリスト内の各パイプラインに対して評価し、次にパイプライン内の各コマンドに対して評価し、最後にコマンド内の各単純コマンドに対して評価します。

コマンドの後に & を指定すると、コマンドにおけるデフォルトの標準入力は /dev/null という空ファイルになります。その他の場合、コマンドを実行するための環境には、起動側シェルのファイル記述子 (入出力指定で変更可能) が含まれます。

ファイル名の生成

コマンド実行に先立ち、各コマンドワードは、*、?、および [を含んでいないかチェックされます。これらの文字のいずれかがあると、そのワードはパターンとみなされます。このワードは、パターンと一致する、辞書編集方式の順にソートされたファイル名に置換されます。パターンと一致するファイル名が見つからない場合、ワードは変更されません。ファイル名の先頭のピリオド (.) または スラッシュ (/) 直後のピリオドは、明示的に一致しなければなりません (後者の場合はスラッシュ自体をも含む) 。

*	NULL 文字列を含め、任意の文字列と一致します。
?	任意の単一文字と一致します。
[...]	囲まれた文字のいずれかと一致します。2 つの 文字を - で区切ると、その間にある任意の文字 (その 2 つの文字も含む) に一致します。先頭の [の次の文字が ! である場合、 で囲まれていない任意の文字と一致します。

クォートされているすべての文字 (下記「クォート」を参照) は、ファイル名において明示的に一致しなければなりませんので、ご注意ください。

クォート

次の文字はシェルに対しては特別な意味を持ち、クォートしない (後述の説明を参照) 限り ワードの終わりを表します。

; & ( ) | ^ < > 復帰改行 空白文字 タブ

これらの文字をクォートする、つまり文字自身を表すには、文字の前にバックスラッシュ (\) を付けるか、または一対の引用符 (' ' または "") で囲みます。シェルは、特定の文字をクォートして、それらが特別の意味を持たないようにすることがあります。単一の文字をクォートするのに用いるバックスラッシュは、コマンド実行前にワードから取り除かれます。\ と復帰改行との組み合せは、コマンドとパラメタの置換前にワードから取り除かれます。

一対の単一引用符 (' ') で囲まれたすべての文字 (ただし単一引用符は除く) は、シェルによってクォートされます。バックスラッシュは、一対の単一引用符で囲まれていれば特殊な意味を持ちません。単一引用符は、一対の二重引用符で囲めばクォートされますが (例 " '") 、一対の単一引用符で囲んでもクォートされません。

一対の二重引用符 ("") の中では、パラメタとコマンドの置換が実施され、シェルは、その結果をクォートして、ブランクの解釈とファイル名の生成が行われないようにします。$* が一組の二重引用符で囲まれている場合、定位置パラメタは置換され、クォートされ、クォートされた空白で分けられます ("$1 $2 ...") 。しかし $@ が一組の二重引用符で囲まれている場合は、定位置パラメタは置換され、クォートされ、クォートされていない空白 ("$1" "$2" ...) で分けられます。\ は \、'、"、, (コンマ)、$ といった文字をクォートします。\ と復帰改行との組み合せは、コマンドとパラメタの置換前にワードから取り除かれます。バックスラッシュが \、'、"、, (コンマ)、$、および 復帰改行以外の文字の前に付く場合は、バックスラッシュ自体がシェルによってクォートされます。

プロンプト

シェルは、対話的に使用すると、コマンドを読み取る前に PS1 の値によるプロンプトを発行します。復帰改行を入力したあとで、コマンドを完了するためにさらに入力が必要な場合は、2 次プロンプト (PS2 の値) が出力されます。

環境

環境 (*environment*) は、通常の引数リストが 実行されるプログラムに引き渡される場合と同様の方法で引き渡される、名前と値の対の集まりです (environ(5) を参照) 。シェルが環境と対話する方法はいくつかあります。シェルは、呼び出されると、環境を走査して、見つけた名前ごとに変数を作成し、対応する値を設定します。ユーザーがこれらのパラメタの値を変更したり 新しいパラメタを作成したときには、export コマンドを用いてシェルのパラメタを環境に関連付けなければ、これらのパラメタは環境に何の影響も与えません (set -a の説明を参照) 。環境からパラメタを削除するには、unset コマンドを使用します。したがって、実行されるコマンドが参照する環境は、シェルが最初に引き継いだ「名前 = 値」の対のうち 変更されていないものから、unset によって削除された対を引き、変更または追加した対をくわえたものです。これらはいずれも export コマンドで指定する必要があります。

単純コマンドの環境は、いくつかのパラメタ代入指定を 先頭に付加すれば拡張できます。

以下は、*command* が特殊コマンドでなければ、*command* の実行に関するかぎり同じことを意味します。

TERM=450 *command*

および

(export TERM; TERM=450; *command*

command が特殊コマンドの場合、次の指定は、現在のシェル内の TERM 変数を修正します。

TERM=450 *command*

-k フラグを設定すると、すべてのキーワード引数は 環境に格納されます。これらの引数がコマンド名の後に指定された場合も同様です。以下の例では、まず a=b c を、次に c を表示します。

```
echo a=b c

a=b c
```

```
set  –k

echo a=b  c

c
```

シグナル

起動されたコマンドに対する INTERRUPT シグナルと QUIT シグナルは、コマンドの後に & が付く場合には無視されます。その他の場合、シグナルは、シェルが親から引き継いだ値を持ちます。ただし、シグナル 11 だけは例外です (後述の trap コマンドの説明を参照) 。

実行

コマンド実行のたびに、前述の コマンドの置換、パラメタの置換、ブランクの解釈、入出力のリダイレクション、およびファイル名の生成が行われます。コマンド名が定義済みの関数名と一致する場合、その関数がシェルプロセスで実行されます (これと実行時にサブシェルを要求する シェルスクリプトファイルの実行の違いに注意) 。定義済み関数名とは一致しないが、後述の特殊コマンドの いずれかと一致するコマンド名の場合、そのコマンドがシェルプロセスで実行されます。

定位置パラメタの $1、 $2、 ... は関数の引数に設定されます。コマンド名が特殊コマンド とも定義済み関数の名前とも一致しない場合、新しいプロセスが作成され、exec(2) を用いてそのコマンドの実行が試みられます。

PATH というシェルパラメタは、コマンドを含んでいる ディレクトリの検索パスを定義します。2 つのディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは /usr/bin です。現在のディレクトリは NULL パス名によって指定されます。これは等号の直後、パスリスト内の任意の場所にある 2 つの区切り文字のコロンの間、またはパスリストの最後に記述できます。コマンド名に / が含まれている場合は、検索パスは使用されません。/ が含まれていなければ、パスにおける各ディレクトリに実行可能ファイルがあるか検索します。ファイルが実行権を有するが、それが a.out ファイルでない場合、シェルコマンドの入ったファイルとみなされます。そのファイルを読み取るときは、サブシェルが生成されます。括弧で囲まれたコマンドもサブシェル内で実行されます。

シェルは、(あとで不必要な exec を行わなくてもいいように) 検索パス内のコマンドの位置を記憶します。コマンドが相対ディレクトリにあった場合、その位置を 現在のディレクトリの変更のたびに再決定しなければなりません。シェルは、PATH 変数が変更されるか hash -r コマンドが実行されるたびに、記憶していたすべての位置を忘れてしまいます (下記参照) 。

特殊コマンド

以下に示す特殊コマンドに対しては、入出力のリダイレクションが可能です。ファイル記述子 1 は、デフォルトの出力位置です。ジョブ制御が可能なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます (「ジョブ制御」の項を参照) 。

:
コマンドは何もせず、実行による影響は何もありません。終了ステータス 0 が返されます。

. *filename*
filename からコマンドを読み取り実行し、戻ります。PATH によって指定された検索パスを用いて、*filename* を含むディレクトリを探します。

bg [%*jobid* . . .]
ジョブ制御が可能なとき、bg コマンドはジョブの操作用に ユーザー環境に追加されます。停止状態のジョブをバックグラウンドで再び実行します。%*jobid* を省略すると、現在のジョブとみなされます。(詳細については後述の 「ジョブ制御」の項を参照)

break [*n*]
for または while ループがあれば抜け出します。*n* を指定すると、*n* レベル分ブレークします。

cd [*argument*]
現在のディレクトリを *argument* に変更します。シェルパラメタ HOME は、*argument* のデフォルト値です。シェルパラメタ CDPATH は、*argument* を含むディレクトリの検索パスを定義します。2 つのディレクトリ名は、コロン (:) で区切ります。デフォルトのパスは 空の文字列です (現在のディレクトリの指定) 。なお現在のディレクトリは 空のパス名で指定します。このパス名は、等号の直後か、パスリスト内の区切り文字のコロンの間に指定します。*argument* の先頭文字が / の場合、検索パスは使用しません。それ以外の場合は、パス中の各ディレクトリで *argument* を検索します。

chdir [*dir*]
chdir はシェルの作業用ディレクトリを *dir* が示すディレクトリに変更します。引数を指定しないと、そのユーザーのホームディレクトリに変更します。*dir* が現在のディレクトリからは見つからない相対パス名の場合、変数 CDPATH 環境内のディレクトリリストを検索します。*dir* が / で始まる値を持つシェル変数の名前である場合、その変数の値が示すディレクトリに変更します。

continue [*n*]
for または while ループの次の繰り返しを実行します。*n* を指定すると、*n* 番目のループから実行します。

echo [*arguments* . . .]
arguments の文字列が 空白文字に区切られて、シェルの標準出力に書かれます。引数をエコーバックします。使用法と説明については echo(1) を参照してください。

eval [*argument* . . .]
引数をシェルへの入力として読み取り、生成されるコマンドを実行します。

exec [*argument* . . .]
このシェルの代わりに、引数で指定されたコマンドを (新規プロセスは生成せずに) 実行します。入出力引数が指定可能で、その他の引数が指定されない場合は、これによってシェルの入出力が変更されます。

exit [*n*]
呼び出し元のシェルまたはシェルスクリプトを *n* で指定した終了ステータスで終了させます。*n* を省略すると、最後に実行されたコマンドの終了ステータス がシェルの終了ステータスになります。EOF によっても、シェルは終了します。

`export` [*name* . . .]
指定された *name* 群に対し、後で実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。引数を省略すると、現在のシェル実行中にエクスポートのマークが付けられた 変数名を一覧表示します。親シェルからエクスポートされた変数名は、現在のシェル実行中に再びエクスポートされた場合にだけ一覧表示されます。関数名はエクスポートされません。

`fg` [`%`*jobid* . . .]
ジョブ制御が可能なとき、`fg` はジョブの操作用に ユーザー環境に追加されます。このコマンドは、フォアグラウンド状態の停止ジョブを再び実行します。また、実行中のバックグラウンドジョブをフォアグラウンドへ移します。`%`*jobid* を省略すると、現在のジョブとみなされます。(詳細については後述の 「ジョブ制御」の項を参照)

`getopts`
コマンドシンタクス標準のサポート用に、シェルスクリプト内で使用されるコマンドです (`intro`(1) を参照) 。このコマンドは、定位置パラメタを構文解析し、オプションの正当性をチェックします。使用法と説明については、`getoptcvt`(1) を参照してください。

`hash` [`-r`] [*name* . . .]
シェルは、各 *name* ごとに、それが示すコマンドの検索パス内の位置を決定し、記憶します。`-r` オプションを指定すると、シェルは記憶したすべての位置を忘れます。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。この出力表示において、*hits* はシェルプロセスによってコマンドが呼び出された回数を表します。*cost* は、検索パスのコマンドを見つけるのに必要な作業です。コマンドが検索パスの「相対」ディレクトリにある場合、そのディレクトリの変更後に そのコマンドが格納された位置が再計算されます。この再計算が行われる予定のコマンドに対しては、*hits* 情報の隣にアスタリスク (`*`) が示されます。*cost* の値は、再計算が行われるたびに増加されます。

`jobs` [`-p`|`-l`] [`%`*jobid* ...]

`jobs -x` `command` [*arguments*]
停止中またはバックグラウンドで実行中の すべてのジョブを報告します。`%`*jobid* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが報告されます (詳細については後述の 「ジョブ制御」の項を参照) 。

`kill` [*–sig*] `%`*job* . . .

`kill -l`
`TERM` (終了) シグナルまたは指定されたシグナルのいずれかを、指定されたジョブまたはプロセスに送信します。シグナルは、番号または名前 (`signal`(3HEAD) の場合と同様に “SIG” という接頭辞を取り除いたもの (`CHLD` と名付けられた `SIGCHD` 以外)) で指定します。送信するシグナルが `TERM` (終了) または `HUP` (ハングアップ) の場合、停止中のジョブまたはプロセスには `CONT` (継続) シグナルを送信します。*job* という引数は、活動中のジョブではないプロセスのプロセス ID を指定することもできます。後述の「ジョブ制御」の項を参照してください。第 2 の形式の `kill –l` は、シグナル番号とシグナル名をリスト表示します。(`kill`(1) 参照)

`login` [*argument* . . .]
'`exec login` *argument*. . . .' と同機能です。使用法と説明については、`login`(1) を参照してください。

`newgrp` [*argument*]
`exec newgrp` *argument*. と同機能です。使用法と説明については、`newgrp`(1) を参照してください。

`pwd`
現在の作業用ディレクトリを表示します。使用法と機能説明については、`pwd`(1) を参照してください。

`read` *name* . . .
標準入力から 1 行を読み取り、内部フィールドセパレータの `IFS` (通常は空白文字またはタブ) を用いてワード境界を区切り、最初のワードを最初の *name* に割り当て、2 番目のワードを 2 番目の *name* に割り当て、続くワードも順次割り当てます。残ったワードは最後の *name* に割り当てます。\ に続いて復帰改行を入力すれば、行を継続できます。復帰改行以外の文字の前にバックスラッシュを付加すれば、その文字をクォートできます。このバックスラッシュは、ワードが *name* に割り当てられる前に削除され、バックスラッシュの後に位置する文字は解釈されません。EOF に到達した場合を除き、リターンコードは `0` となります。

`readonly` [*name* . . .]
指定された *name* に読み取り専用のマークを付け、これらの名前が後続の割り当てでは変更できないようにします。引数を省略すると、読み取り専用と指定された名前がすべて一覧表示されます。

`return` [*n*]
関数を、*n* が示すリターンステータスで終了させます。*n* を省略すると、リターンステータスは最後に実行された コマンドのリターンステータスになります。

`set` [`-` `-aefhkntuvx` [*argument* . . .]]

`-a` エクスポート用に修正または作成された変数にマークを付けます。

`-e` コマンドが 0 以外の終了ステータスで終了した場合、直ちに終了します。

`-f` ファイル名を生成しないようにします。

`-h` 関数の定義時に、関数コマンドを検索しその位置を記憶します (通常、関数コマンドは関数実行時に検索されます) 。

`-k` コマンド名の前にあるキーワード引数だけでなく、すべてのキーワード引数がコマンド用の環境に置かれます。

`-n` コマンドを読み取るが、実行しません。

`-t` 1 つのコマンドを読み取り、実行し、終了します。

`-u` 未設定の変数を置換時にエラーとして扱います。

`-v` シェルへの入力行の読み取り時に、その内容を表示出力します。

`-x` コマンドおよび引数の実行時に、その内容を表示出力します。

– どのフラグも変更しません。$1 に – を設定する際に便利です。

– の代わりに + を使用すると、これらのフラグがオフになります。これらのフラグはシェル起動時にも使用できます。現在セットされているフラグは、$– で見つけられます。残りの引数は定位置パラメタで、$1、 $2、 ... へ順に割り当てられます。引数が指定されない場合、 すべての名前の値が出力されます。

shift [*n*]
$*n*+1 . . . から始まる一連の定位置パラメタを $1 . . . に再命名 (リネーム) します。*n* を省略すると、1 とみなされます。

stop *pid* . . .
pid (プロセス ID 番号) の実行を停止します (ps(1) 参照) 。

suspend
現在実行中のシェルを停止します。(ログインシェルの場合は停止しません。)

test
条件式を評価します。使用法と説明については、test(1) を参照してください。

times
シェルから実行されたプロセスの ユーザーおよびシステム時間の累積値を出力します。

trap [*argument n* [*n2* . . .]]
argument が示すコマンドを、シェルが数値または記号シグナル (*n*) を受信した時に読み取り実行します。なお *argument* は、トラップ設定時に 一度、トラップ取り出し時に 一度検索されます。トラップコマンドは、シグナル番号または 対応するシンボリック名の順序で実行されます。現在のシェルで無視されているシグナルにトラップを 設定しようとしても無効となります。シグナル 11 (メモリフォールト) にトラップを指定しようとすると、エラーになります。*argument* を省略すると、すべてのトラップ *n* は元の値に再設定されます。*argument* が NULL 文字列の場合、シェルおよびシェルが呼び出したコマンドは、このシグナルを無視します。*n* が 0 の場合、*argument* が示すコマンドはシェル終了時に実行されます。引数なしの trap コマンドは、コマンドの一覧を 各々が対応しているシグナル番号とともに表示します。

type [*name* . . .]
各 *name* ごとに、コマンド名として使用される場合に どのように解釈されるかを指示します。

ulimit [–[HS] [a | cdfnstv]]

ulimit [–[HS] [c | d | f | n | s | t | v]] limit
ulimit は、資源の強い制限値または弱い制限値を 表示または設定します。これらの限界については getrlimit(2) の説明を参照してください。

limit 引数を省略すると、ulimit は指定されている限界値を表示します。限界値は一度にいくつでも表示できます。-a オプションは限界値すべてを表示します。

limit 引数を指定すると、ulimit は指定されたフラグに対応する限界値をその引数の値に設定します。*limit* 引数の値として unlimited という文字列を指定すると、有効な最大値に設定されます。一度に資源 1 つについてだけ限界値を設定できま

す。ユーザーは誰でも、弱い限界値を強い限界値を超えない 任意の値に設定できます。ユーザーは誰でも、強い限界値を下げることができます。強い限界値を上げることができるのはスーパーユーザーだけです。詳しくは su(1M) を参照してください。

-H オプションはハード限界を表し、-S オプションはソフト限界を表します。どちらのオプションも指定しない場合、ulimit は両方の限界値を設定し、弱い限界値を表示します。

以下のオプションは、限界値を表示または設定する資源を指定します。オプションをいっさい指定しないと、ファイルサイズ限界値を表示または設定します。

-c 最大 core ファイルサイズ (512 バイトブロック単位)

-d データセグメントまたはヒープの最大サイズ (K バイト単位)

-f 最大ファイルサイズ (512 バイトブロック単位)

-n 最大ファイル記述子プラス 1

-s スタックセグメントの最大サイズ (K バイト単位)

-t 最大 CPU 時間 (秒単位)

-v 仮想メモリの最大サイズ (K バイト単位)

システムで利用可能な最大上限値を 調べるには sysdef(1M) コマンドを実行してください。表示される値は 16 進数ですが、bc(1) ユーティリティを使って 10 進数に変換できます。swap(1M) を参照してください。

たとえば、0 メガバイトに コアファイルダンプのサイズを 制限するには次のように入力します。

```
ulimit -c 0
```

umask [*nnn*]
ユーザーファイル作成マスクを *nnn* が示す値に設定します (umask(1) を参照) 。*nnn* を省略すると、マスクの現在の値を出力します。

unset [*name* . . .]
name ごとに、対応する変数または関数値を削除します。変数 PATH、 PS1、 PS2、MAILCHECK、 および IFS は設定解除できません。

wait [*n*]
当該ユーザーのバックグラウンドプロセスのうち ID が *n* のプロセスを待ち、その終了ステータスを報告します。*n* が省略された場合、当該ユーザーの現在活動中のすべての バックグラウンドプロセスを待ち、リターンコードは 0 になります。

呼び出し

exec(2) を介してシェルが呼び出される場合で、引数 0 の先頭文字が – のとき、コマンドは、まず /etc/profile から読み込まれ、次に $HOME/.profile から読み込まれます (これらのファイルがある場合) 。その後、コマンドは後述のように読み込まれます。シェルが /usr/bin/sh として呼び出される場合にも、このようになりま

す。以下に述べるフラグは、呼び出し時にのみ、シェルによって解釈されます。なお -c または -s フラグが指定されないかぎり、先頭引数はコマンドを含むファイルの名前であるとみなされ、残りの引数は定位置パラメタとして そのコマンドファイルに引き渡されます。

`-c` *string* このフラグが指定されると、*string* からコマンドを読み取ります。

`-i` このフラグが指定された場合あるいは シェル入出力が端末に接続されている場合、このシェルは対話型となります。この場合、`kill 0` が対話型シェルを終了しないように TERM を無視し、`wait` が割り込み可能になるように INTERRUPT を捕え、無視します。いずれの場合も、シェルは QUIT を無視します。

`-p` このフラグが指定されると、シェルは実効ユーザーおよびグループ ID に、実ユーザーおよびグループ ID を設定しません。

`-r` このフラグを指定すると、シェルは制限付きシェルになります (`rsh`(1M) を参照) 。

`-s` このフラグが指定された場合または引数が残っていない場合、標準入力からコマンドを読み取ります。引数が残っていれば、それらは定位置パラメタを指定します。前述の特殊コマンドの出力を除くシェル出力は、ファイル記述子 2 に書き出されます。

他のフラグと引数については、前述の `set` コマンドの箇所で説明されています。

ジョブ制御 (jsh)

シェルを `jsh` として呼び出すと、`sh` の説明で述べたすべての機能に加えて、ジョブ制御が可能になります。通常、ジョブ制御は対話型シェルに対してだけ可能です。非対話型シェルは、ジョブ制御の機能が追加されても、その恩恵を受けないのが通常です。

ジョブ制御が可能な場合、ユーザーが端末から入力したコマンドまたはパイプラインは、すべてジョブ (*job*) と呼ばれます。どのジョブも、必ずフォアグラウンド、バックグラウンド、または停止のいずれかの状態にあります。これらの用語の定義を次に示します。

1. フォアグラウンド状態にある ジョブは、制御している端末への読み取り および書き込み権を持っています。
2. バックグラウンド状態にある ジョブは、制御している端末への読み取り 権を拒否されていますが、条件付き 書き込み権は持っています (`stty`(1) を参照) 。
3. 停止ジョブは保留状態に置かれた ジョブであり、通常は `SIGTSTP` シグナルにより、この状態になります (`signal`(3HEAD) を参照) 。

シェルが起動するすべてのジョブには、ジョブ番号 (*job number*) と呼ばれる 正の整数が割り当てられます。シェルはこの番号を把握し、特定のジョブを示す識別子として使用します。また、シェルは現在 (*current*) および直前 (*previous*) のジョブも把握しています。現在のジョブとは、最後に起動または再起動されたジョブです。前回のジョブとは、その直前のジョブです。

ジョブ識別子の正しい構文は次のような形式です。

%*jobid*

このうち *jobid* は、次のいずれかの形式で指定できます。

%	または + 現在のジョブ
–	前回のジョブ
?<*string*>	コマンド行が *string* を含んでいるジョブ
n	ジョブ番号が *n* のジョブ
pref	コマンド名の先頭が *pref* のコマンド。たとえば `ls –l name` がバックグラウンドで実行中だった場合、`%ls` と指定すればこのコマンドを示すことができます。*pref* は、クォートしない限り、ブランクを含めることができません。

ジョブ制御が可能なとき、ジョブの操作用に 次のコマンドがユーザー環境に追加されます。

`bg` [`%`*jobid . . .*]
停止状態のジョブをバックグラウンドで再び実行します。`%`*jobid* を省略すると、現在のジョブとみなされます。

`fg` [`%`*jobid . . .*]
停止状態のジョブをフォアグラウンドで再び実行します。また、実行中のバックグラウンドジョブをフォアグラウンドへ移します。`%`*jobid* を省略すると、現在のジョブとみなされます。

`jobs` [`-p`|`-l`] [`%`*jobid . . .*]

`jobs -x command` [*arguments*]
停止中またはバックグラウンドで実行中の すべてのジョブを報告します。`%`*jobid* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが報告されます。次のオプションを使って、ジョブの出力を変更できます。

`-l`	ジョブのプロセスグループ ID および作業ディレクトリを報告します。
`-p`	ジョブのプロセスグループ ID のみを報告します。
`-x`	コマンドまたは引数中に見つかった *jobid* を、対応するプロセスグループ ID に置き換え、コマンドに引数を渡して実行します。

`kill` [`–`*signal*] `%`*jobid*
`kill` コマンドの組み込みバージョン。*jobid* で示すプロセスに対して `kill` コマンドの機能を提供します。

`stop %`*jobid . . .*
バックグラウンドジョブの実行を停止します。

`suspend`
現在のシェルの実行を停止します (ただし、ログインシェルの場合は停止しません)。

`wait` [`%`*jobid* . . .]
`wait` コマンドの組み込みバージョンで、ジョブ識別子の指定を受け入れます。`%`*jobid* が省略された場合、`wait` は、前述の「特殊コマンド」で説明したように動作します。

大規模ファイルの動作

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `sh` と `jsh` の動作については、`largefile`(5) を参照してください。

終了ステータス

構文エラーなどのエラーを検出すると、シェルは 0 以外の終了ステータスを返します。シェルを非対話型で使用している場合、シェルファイルの実行は中止されます。対話型で使用している場合は、シェルは最後に実行されたコマンドの 終了ステータスを返します (上記の `exit` コマンドの説明を参照) 。

jsh のみ

シェルが `jsh` として呼び出された場合、停止ジョブがあるのにシェルを終了させようとすると、シェルは次のような警告を出します。

```
There are stopped jobs.
```

これが唯一のメッセージです。もう一度終了が試みられ、停止ジョブがまだ存在している場合、これらのジョブにカーネルから `SIGHUP` シグナルが送られ、シェルは終了します。

ファイル

```
$HOME/.profile
/dev/null
/etc/profile
/tmp/sh*
```

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/jsh

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/sh

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

`intro`(1), `bc`(1), `echo`(1), `getoptcvt`(1), `kill`(1), `ksh`(1), `login`(1), `newgrp`(1), `ps`(1), `pwd`(1), `set`(1), `shell_builtins`(1), `stty`(1), `test`(1), `umask`(1), `wait`(1), `rsh`(1M), `su`(1M), `swap`(1M), `sysdef`(1M), `dup`(2), `exec`(2), `fork`(2), `getrlimit`(2), `pipe`(2), `ulimit`(2), `setlocale`(3C), `signal`(3HEAD), `passwd`(4), `profile`(4), `attributes`(5), `environ`(5), `largefile`(5), `XPG4`(5)

警告 | シェルスクリプトを `setuid` して使用することは避けてください。

注意事項 | 入出力のリダイレクションでファイル名に使用されたワードは、ファイル名生成では解釈されません (上記の「ファイル名の生成」の項を参照) 。たとえば `cat file1 >a*` という指定は、`a*` という名前のファイルを生成します。

パイプラインにあるコマンド群は それぞれ別個のプロセスとして稼動するので、パイプラインに設定された変数は親シェルには何の影響も与えません。

`cannot fork,too many processes` というエラーメッセージを受け取った場合には、`wait`(1) コマンドを用いてユーザーのバックグラウンドプロセスを クリーンアップしてください。それでも効果がない場合には、おそらくシステム・プロセステーブルが満杯であるか、または活動中のフォアグラウンドプロセスがありすぎるためです。ユーザーログインに結合するプロセス ID の数、およびシステムが把握できる数には 限度があります。

パイプラインの最後のプロセスだけが、待つ対象になり得ます。

あるコマンドを実行し、その後で同一名のコマンドが、検索パスにおいて元のコマンドがあるディレクトリの 前に位置するディレクトリにインストールされた場合、シェルは元のコマンドの方を実行し続けます。新しい方のコマンドを実行させたければ、`hash` コマンドを使用してください。

Bourne シェルにはプロセスの実効ユーザー ID に対して制限があります。このユーザー ID が 100 よりも小さい (さらにプロセスの実ユーザー ID と同等ではない) 場合には、ユーザー ID はプロセスの実ユーザー ID にリセットされます。

同じプロセスグループで フォアグラウンドジョブと バックグラウンドジョブの両方を シェルが実行しているため、ジョブは同じシグナルを受け取り、予期しない結果を招くことがあります。したがって、特に対話型のシェルを動作している場合は、他のジョブ制御のシェルを使用することを おすすめします。

存在しないコマンドのインタプリタを実行しようとするシェルスクリプトを、シェルが処理した場合、シェルはシェルスクリプトが存在しないという間違った診断メッセージを 返します。

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目 `intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | shift – シェルの引数のリスト、またはフィールドで区切られた単語をずらすためのシェル組み込み関数

sh | **shift** [*n*]

csh | **shift** [*variable*]

ksh | *** shift** [*n*]

sh | $*n*+1 . . . から始まる一連の定位置パラメタを $1 . . . という名前に変更します。 *n* を省略すると、1 と見なされます。

csh | *argv* の構成要素 (または *variable* が指定されればその変数の構成要素) を、 左へずらして最初の構成要素を切り捨てます。 未設定の変数および NULL 値に対してはエラーとなります。

ksh | $*n*+1 $*n*+1 ． ． ．の定位置パラメタを $1. . . という名前に変更します。 *n* のデフォルト値は 1 です。 *n* に指定できる値は、 評価結果が $# 以下の負でない数になる算術式です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | csh(1), ksh(1), sh(1), attributes(5)

名前 | sleep – 実行の一定期間保留

形式 | **sleep** *time*

機能説明 | sleep ユーティリティは、少なくとも *time* 秒間、実行を停止します。

オペランド | 以下のオペランドを指定できます。

time 実行を停止する長さを秒数で表す、負でない整数。

使用例 | 例 **1** sleep コマンドの例

ある一定時間後に、コマンドを実行する例を以下に示します。

```
(sleep 105; command)&
```

また、ある時間ごとにコマンドを繰り返し実行するために sleep を使用する例を以下に示します。

```
while true
do
        command
        sleep 37
done
```

環境 | sleep の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0 time で示す秒数以上のあいだ実行を中断することに成功した、または SIGALRM シグナルを受信した （「注意事項」を参照）

>0 エラーが発生した

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | wait(1), alarm(2), sleep(3C), wait(3UCB), attributes(5), environ(5)

注意事項 | SIGALRM シグナルを受信すると、 sleep ユーティリティは以下の動作のうちどちらかを行います。

- 終了ステータス 0 で正常終了する。
- 事実上、 シグナルを無視する。

その他のシグナルを受信した場合には、 sleep は標準の動作を行います。

名前 | sort – テキストファイルのソート、マージ、順序の確認

形式 | **/usr/bin/sort** [-bcdfimMnru] [-k *keydef*] [-o *output*] [-S *kmem*]
[-t *char*] [-T *directory*] [-y [*kmem*]] [-z *recsz*] [+*pos1* [-*pos2*]] [*file...*]

/usr/xpg4/bin/sort [-bcdfimMnru] [-k *keydef*] [-o *output*] [-S *kmem*]
[-t *char*] [-T *directory*] [-y [*kmem*]] [-z *recsz*] [+*pos1* [-*pos2*]] [*file...*]

機能説明 | sort コマンドは、指定されたファイルのすべての行をソートし、その結果を標準出力に書き込みます。

比較は、各入力行から抽出された 1 つまたは複数のソートキーに基づいて行われます。デフォルトでは、1 つのソートキー、つまり入力行全体が使用されます。また、ソートの順は、現在のロケールの照合順序に従います。

オプション | 次にあげるオプションは、デフォルトの動作を変更します。

/usr/bin/sort

-c | 引数で指定された入力ファイルが、現在のロケールの 照合順序に従ってソートされていることを確認します。終了ステータスが設定されますが、正しくソートされていれば何も出力されません。

/usr/xpg4/bin/sort

-c | どんな場合でも何も出力されない、という点を除き /usr/bin/sort と同じです。

-m | マージのみを行います。入力ファイルはソート済みであるものとします。

-o *output* | 標準出力の代わりに使用される出力ファイル名を指定します。このファイルは入力ファイルの 1 つと同じでも構いません。

-S *kmem* | ソートを行うために使用するスワップメモリー容量の最大値を指定します (デフォルトの単位はキロバイト)。*kmem* には、バイト (b)、キロバイト (k)、メガバイト (m)、ギガバイト (g)、テラバイト (t)、の数値を直接指定したり、インストールされている物理メモリーに対する割合パーセント (%) で指定することができます。

-T *directory* | *directory* 引数は、一時ファイルを入れるディレクトリ名です。

-u | 同一行を削除します。同じキーを持つ行の組から 1 行のみを取り出し、他を抑制します。-c オプションと同時に指定すると、入力ファイルがソート済みであることと、同じキーを持つ複数の行が存在しないことの両方を確認します。

-y *kmem* | (旧式オプション) sort が最初に使用する主記憶量を指定するために使用されていました。このオプションは、仮想メモリーシステムには適していません。sort が使用するメモリー容量は、-S オプションで指定するようになりました。

-z *recsz* | (旧式オプション) システム固有の デフォルトバッファサイズを超えるような行が存在した場合でも、異常終了を起こさないようにするためのオプションでした。現在では、最長の行が書き込める大きさのバッファを sort が自動的に割り当てるので、このオプションは意味がなくなりました。

並べ換えオプション

デフォルトのソートの並び換え順は、LC_COLLATE の値に依存します。LC_COLLATE が C に設定された場合、ASCII の並べ換え順でソートされます。LC_COLLATE が en_US に設定された場合、ある 2 つの文字列のうち、一方が他方よりも先に大文字を使用していて その他の点では一様である場合は 大文字・小文字を区別しますが、それ以外は大文字・小文字を区別しません。その他のロケールでは、別の並べ換え順でソートされます。

次にあげるオプションは、デフォルトの並べ換え規則を変更します。ソートキーフィールド指定とは無関係に並べ換えオプションがある場合、要求された並べ換え規則は、すべてのソートキーに適用されます。指定された並べ換えオプションが特定のソートキーに付いている場合 (「ソートキーオプション」の説明を参照)、このオプションはそのキーに対するすべてのグローバルな 並べ換えオプションを無効にします。旧式のキーオプションでは、*+pos1* オプションのあとに以下のオプションを 1 つ以上指定すると、その直前のオプションで指定されたキーフィールド に対してだけ有効になります。

`-d` 辞書編集順にします。比較では、英字、数字および空白 (空白文字およびタブ) だけが有効です。

`-f` 小文字と大文字の区別をしません。

`-i` 非印字文字を無視します。

`-M` 月名とみなして比較します。フィールドの先頭 3 文字 (空白文字以外) は 大文字に変換し、比較します。たとえば、英語ではソートの並び換え順は JAN < FEB < . . . < DEC です。月名でないフィールドは、JAN よりも低く見なされます。`-M` オプションは `-b` オプションの動作を含みます(下記を参照)。

`-n` ソートキーを、省略可能な空白文字、省略可能なマイナス符号、省略可能な小数点および桁区切り文字 (現在のロケールに定義されているもの) を持つ 0 個以上の数値からなる最初の数値列に限定します。キーの値は算術的な数値としてソートされます。空の数値列は、ゼロとみなされます。先行するゼロおよびゼロに付けられた符号は、並べ換えに 影響を与えません。

`-r` 比較の意味を反転します。

フィールド区切り文字オプション

フィールド区切り文字オプションの扱い方は、以下のオプションを使って 変更できます。

`-b` 限定ソートキーの開始および終了位置の決定時に、先行する空白文字を無視します。最初のソートキーオプションの前に `-b` オプションが指定されている場合、このオプションはすべての ソートキーオプションに適用されます。それ以外の場合は、オプション引数の `-k`*field_start* 、*field_end* 、*+pos1* 、*–pos2* の各々に `-b` オプションを個別に指定します (下記を参照) 。

`-t` *char* *char* をフィールド区切り文字として使用します。*char* は、(ソートキーに含まれることがあっても) フィールドの一部とは見なされません。*char* は、繰り返し出てきてもそれぞれが有効です (たとえ

ば、*<char><char>* は空フィールドを区切ります) 。`-t` を省略すると、デフォルトのフィールド区切り文字として空白文字が 用いられます。空白でない文字に続く、最大長の空白文字列が、フィールド区切り文字となります。

ソートキーオプション

ソートキーは以下のオプションを使って指定できます。

`-k`*keydef*　　*keydef* 引数の形式は次のとおりです。

`-k` *field_start* [*type*] [*,field_end* [*type*]]

個々の引数の意味を以下に説明します。

field_start と *field_end*
行の一部分に限定したキーフィールドを定義します。

type
文字群 `bdfiMnr` から得られる修飾子です。`b` 修飾子は `-b` オプションと同じ働きですが、対象とした *field_start* または *field_end* だけに適用されます。フィールド内の文字は、フィールド内の最初の空白でない文字から 数えられます。なお、これは *first_character* と *last_character* に個別に適用されます。その他の修飾子も、同じ文字で示すオプションと 同じ働きですが、対象としたキーフィールドにのみ適用されます。このように動作するのは、*field_start* または *field_end* のどちらか一方、あるいは両方が指定されているときです。*field_start* と *field_end* のいずれかに修飾子が付けられているとき、それらはオプションの対象とはなりません。

複数のキーフィールドがある場合、後にあるキーは、それ以前のキーが等しいと比較された場合にだけ 比較されます。`-u` オプションが指定された場合を除き、等しいと比較された行は、`-d` 、`-f` 、`-i` 、`-n` 、`-k` のいずれもが省略された場合と同じように並べ換えられます (`-r` オプションは、指定されていれば有効) 。また行の中の全バイトを用いて比較されます。

次の表記を見てください。

`-k` *field_start*[*type*][*,field_end*[*type*]]

これは *field_start* で始まり *field_end* で終わるキーフィールドを定義するものです。ただし *field_start* が行の限界を超えていたり、*field_end* のあとに位置している場合には、キーフィールドは空となります。*field_end* 指定を省略すると、行の終わりと見なされます。

フィールドは、区切り文字でない最長の文字列で、`-t` オプションが指定されていなければ先行するフィールド区切り文字も含みます。

keydef オプション引数の *field_start* 部分の形式は次のとおりです。

field_number[*.first_character*]

フィールド、およびフィールド内の文字には、1 から始まる番号が付けられます。*field_number* と *first_character* 指定は、いずれも正の 10 進整数と解釈され、ソートキーの部分として用いられる 先頭の文字を指定します。 *.first_character* 指定を省略すると、フィールドの先頭文字が使用されます。

keydef オプション引数の *field_end* 部分の形式は次のとおりです。

field_number[*.last_character*]

このうち *field_number* の意味は上記の *field_start* のものと同じです。*last_character* 指定は、負でない 10 進整数と解釈され、ソートキーの部分として用いられる 最後の文字を指定します。*last_character* の値の評価結果がゼロのとき、または *.last_character* 指定が省略されたときは、*field_number* が示すフィールドの最終文字が使用されます。

-b オプションまたは b 修飾子が有効なとき、フィールド中の文字はそのフィールドの 最初の空白でない文字から数えられます。これは *first_character* と *last_character* に別々に適用されます。

[+*pos1* [−*pos2*]] -k*keydef* オプションと同等な機能を提供する旧式のオプションです。

pos1 および *pos2* はそれぞれ *m*.*n* という形式をとり、オプションでフラグ bdfiMnr の中から 1 つまたは複数のフラグを後に付けることができます。+*m*.*n* によって指定される開始位置は、*m*+1 番目のフィールドにある *n*+1 番目の文字であると解釈されます。 *.n* がない場合は、*m*+1 番目のフィールドの先頭文字を指す .0 を意味します。b フラグが有効な場合、*m*+1 番目のフィールドの最初の空白文字以外の文字から *n* が数えられます。+*m*.0b は *m*+1 番目のフィールド内の最初の空白文字以外の文字を指します。

−*m*.*n* によって指定される最終位置は、*m* 番目のフィールドの最後の文字から後へ *n* 番目の文字 (区切り文字も含む) を意味するように解釈されます。 *.n* がない場合、*m* 番目のフィールドの最後の文字を指す .0 を意味します。b フラグが有効な場合、*m*+1 番目のフィールドの前についている最後の空白文字から *n* が数えられます。したがって、−*m*.1b は、*m*+1 番目のフィールド内の最初の空白文字以外の文字を指します。

次の指定を見てください。

```
+w.xT -y.zU
```

これは *+pos1 −pos2* の形式をタイプ修飾子 T と U とともに指定したもので、以下の記述と同等です。

```
undefined             (z==0 & U b を含む & -t が存在する)
-k w+1.x+1T,y.0U      (z==0 でその他の場合)
-k w+1.x+1T,y+1.zU    (z > 0)
```

ソートキー (-k オプションおよび旧式の *+pos1* と *–pos2*) は少なくとも 9 個記述できます。複数個ある場合、コマンド行での順序が意味を持ちます。ソートキーを 1 つも指定しないと、行全体のデフォルトのソートキーが使用されます。

オペランド

以下のオペランドを指定できます。

file ソート、マージ、または確認するファイルのパス名。このオペランドを 1 つも指定しない場合、または – を指定した場合には、標準入力が用いられます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の sort の動作については、largefile(5) を参照してください。

使用例

以下の例では、sort キーを指定する方法のうち旧式のものと旧式でないものの両方を示します。これにより両形式の関係が理解しやすくなるはずです。

例 1 第 2 フィールドをキーにしてソートする

第 2 フィールドをソートキーとして用いて、infile の内容をソートします。

```
example% sort -k 2,2 infile
example% sort +1 –2 infile
```

例 2 逆の順序でソートする

第 2 フィールドの 2 番目の文字をソートキーとして、infile1 および infile2 の内容を逆の順序でソートし、outfile に出力します。なお第 2 フィールドの先頭文字はフィールド区切り文字であると想定しています。

```
example% sort -r -o outfile -k 2.2,2.2 infile1 infile2
example% sort -r -o outfile +1.1 –1.2 infile1 infile2
```

例 3 ファイル中の特定の 1 文字をキーとしてソートする

第 2 フィールドの 2 番目の空白でない文字をソートキーとして、infile1 と infile2 の内容をソートします。

```
example% sort -k 2.2b,2.2b infile1 infile2
example% sort +1.1b –1.2b infile1 infile2
```

例 4 ユーザー ID でソートする

ユーザー ID (コロンで区切った 3 番目のフィールド) でソートされたパスワードファイル passwd(4) を出力します。

```
example% sort -t : -k 3,3n /etc/passwd
example% sort -t : +2 –3n /etc/passwd
```

例 4 ユーザー ID でソートする　（続き）

例 5 フィールドが重複する行を除外してソートされた行を出力する

ソート済みファイル infile の行の出力において、同じ第 3 フィールドを持つ行のうち、最初に現われる行だけを出力します。

```
example% sort -um -k 3.1,3.0 infile
example% sort -um +2.0 –3.0 infile
```

例 6 ホスト IP アドレスでソートする

次のコマンドは、どちらも数値形式の IP アドレス (先頭から 4 番目までの数値フィールド) でソートされた hosts(4) ファイル (IPv4 ホストデータベース) を出力します。

```
example$ sort -t . -k 1,1n -k 2,2n -k 3,3n -k 4,4n /etc/hosts
example$ sort -t . +0 -1n +1 -2n +2 -3n +3 -4n /etc/hosts
```

'.' は、フィールドの区切り文字であると同時に、多くのロケールで 10 進数の区切り文字として使用されています。そのため、フィールドの末尾の指定に失敗すると、2 つめのフィールドが 1 つめのフィールドの小数部として認識されてしまうといった問題が発生します。

環境

sort の実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES 、NLSPATH についての詳細は、environ(5) を参照してください。

LC_CTYPE　テキストデータのバイト列を文字 (たとえば、引数や入力ファイル中のシングルバイト文字や複数バイト文字) として解釈するロケールや、-b 、-d 、-f 、-i 、-n の各オプション用に、文字分類の方法を指定する ロケールを定義します。

LC_NUMERIC　-n オプション用に、小数点や桁区切り文字を指定するロケールを定義します。

終了ステータス

以下の終了ステータスが返されます。

0　入力ファイルはすべて正常に出力された、または -c が指定され入力ファイルは正しくソートされていた

1　-c オプションが指定され入力ファイルは指定どおりにソートされていなかった、または -c と -u の両オプションが指定され 2 つの入力行のキーが等しかった

>1　エラーが発生した

ファイル

/var/tmp/stm???

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/sort

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

/usr/xpg4/bin/sort

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目

comm(1), join(1), uniq(1), nl_langinfo(3C), strftime(3C), hosts(4), passwd(4), attributes(5), environ(5), largefile(5), XPG4(5)

診断

さまざまなトラブル (たとえば、入力行が長すぎる) が発生した場合や -c オプションにおいてソート済みでないことを発見した場合、コメントを出力し、0 以外の終了ステータスで終了します。

注意事項

入力ファイルの最終行に復帰改行文字 (NEWLINE) がない場合、sort はこれを追加し、警告メッセージを出力し、処理を継続します。

sort で、等しいキーを持つ行の相対的な順序が保存されるという保証はありません。

-S オプションを使用して、特定の用途に sort の動作を調整することができます。ただし、sort は仮想メモリーシステムよりも、ソート時により適切に膨大な量のメモリーを利用できることに注意してください。このため、-S を使用して、膨大なメモリーを利用するようにしてソートを実行すると、パフォーマンスがかなり低くなることもあります。

前述のように、フィールド修飾子の一部 (-M や -d など) は、入力データをロケール固有の設定と照合して解釈します。ロケール固有の設定を考慮に入れていないと、解釈の結果が予期しないものになることがあります。たとえば月の名前をキーにする場合、「一般に通用する」省略名を使用しても、sort コマンドはそれを正しい名前に読み換えることはしません。sort コマンドが認識する省略名は、nl_langinfo(3C) または strftime(3C) で定義されているものだけです。表示可能順や辞書編集順でソートする場合、このような定義がロケールで適切に設定されていないと、空のソートキーが返されることがあります。結果として、次のキーが適切な並べ替えを決定する有効なキーになります。

名前 | exec, eval, source – 他のコマンドを実行するためのシェル組み込み関数

sh | **exec** [*argument*...]

eval [*argument*...]

csh | **exec** *command*

eval *argument*...

source [-h] *name*

ksh | ***exec** [*arg*...]

***eval** [*arg*...]

sh | exec コマンドは このシェルの代わりに、 *argument* で指定されたコマンドを、 新規プロセスは生成せずに 実行します。 入出力引数が指定可能で、 それら以外の引数を指定しない場合には、 シェルの入出力を変更します。

eval の組み込みの *argument* をシェルへの入力として読み取り、 生成されるコマンドを実行します。

csh | exec は現在のシェルの代わりに *command* を実行します。シェルは終了します。

eval は引数をシェルへの入力として読み取り、 生成されるコマンドを実行します。通常この指定は、コマンドまたは 変数置換の結果として生成されたコマンドを 実行するために使用します。

source は *name* からコマンドを読み取ります。 source コマンドは入れ子にできますが、 あまり深く入れ子にするとシェルのファイル記述子が 不足する可能性があります。 ソースファイル中のエラーは、それがいかなるレベルであろうと、 入れ子にされたすべての source コマンドを終了させます。

-h　*name* が示す、履歴のリスト上のファイル からコマンドを持ってきますが、実行はしません。

ksh | exec 組み込み関数を使用して *arg* を指定すると、 このシェルの代わりに 引数で指定されたコマンドを、 新規プロセスは生成せずに 実行します。 入出力引数が指定可能で、 現在のプロセスに影響を及ぼす場合があります。 引数を指定しない場合は、ファイル記述子が 入出力リダイレクトリストの指定どおりに 変更されることになります。 この場合、 この機能によりオープンされた 2 より大きい番号のファイル記述子は、 別のプログラムを起動するとクローズされます。

eval に続く引数をシェルへの入力として読み取り、 生成されるコマンドを実行します。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されている ksh(1) コマンドは、以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。

3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、チルド置換は = 符号の後に実行され、ワード分割とファイル名生成は実行されません。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), sh(1), attributes(5)

名前 | split – ファイルを複数に分割

形式 | **split** [-linecount | -l *linecount*] [-a *suffixlength*] [*file* [*name*]]

split [-b *n* | *n*k | *n*m] [-a *suffixlength*] [*file* [*name*]]

機能説明 | split ユーティリティは、*file* で示すファイルを読み取り、それを *linecount* で示す数の行に分けて出力ファイル群に書き込みます。最初の出力ファイル名は、*name* に aa を追加したもので、その後辞書編集順どおりに zz までを付けた名前となります (最大 676 ファイル)。*name* の最大長は、ファイルシステムによって認められた最大ファイル名の長さ未満の 2 文字です。 statvfs(2) を参照してください。出力名が指定されていない場合、x をデフォルトとして使用します (出力ファイルは xaa または xab などとなります)。

オプション | 以下のオプションが指定できます。

–*linecount* | -l *linecount*	各断片の行数。デフォルトは 1000 行。
-a *suffixlength*	各出力ファイルの接尾辞部分の長さを *suffixlength* 文字にします。この -a オプションを省略すると、接尾辞のデフォルト長は 2 文字となります。*name* オペランドで指定したファイル名の長さと接尾辞の長さの合計が NAME_MAX バイトを超えてしまう場合には、エラーとなります。このとき split は診断メッセージを発行して処理を終了します。ファイルは生成されません。
-b *n*	各出力ファイルのサイズを *n* バイトとします。
-b *n*k	各出力ファイルのサイズを *n**1024 バイトとします。
-b *n*m	各出力ファイルのサイズを *n**1 048 576 バイトとします。

オペランド | 以下のオペランドを指定できます。

file	入力となる通常ファイルのパス名。このオペランドを省略するか – を指定すると、標準入力とみなされます。
name	split の実行により生成される各出力ファイルの接頭辞を指定します。*name* オペランドを省略すると、x が接頭辞として使用されます。この接頭辞のベース名の長さと *suffixlength* 値の合計は、NAME_MAX バイトを超えることはできません。詳しくは「オプション」の項を参照してください。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の split の動作については、largefile(5) を参照してください。

環境 | split の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 csplit(1), statvfs(2), attributes(5), environ(5), largefile(5)

名前

srchtxt – メッセージデータベースの内容表示、またはそこからのテキスト文字列の検索

形式

srchtxt [-s] [-l *locale*] [-m *msgfile* ,...] [*text*]

機能説明

srchtxt ユーティリティは、 メッセージデータベース内のすべてのテキスト文字列を表示したり、 あるいはメッセージデータベース内のテキスト文字列を検索したりします (mkmsgs(1) を参照) 。 これらのデータベースは、 -m オプションで指定したファイル名に / が含まれていないかぎり、 ディレクトリ /usr/lib/locale/*locale*/LC_MESSAGES のファイルにあります (setlocale(3C) を参照) 。 ディレクトリ *locale* をそのテキスト文字列に使用している言語名とみなすことができます。 -l オプションが指定されていない場合、アクセスされるファイルは、環境変数 LC_MESSAGES の値によって決まります。 LC_MESSAGES が設定されていない場合、 アクセスされるファイルは、環境変数 LANG の値によって決まります。 LANG が設定されていない場合、 アクセスされるファイルは、デフォルト文字列を含むディレクトリ /usr/lib/locale/C/LC_MESSAGES にあるものになります。

text 引数がない場合、 アクセスされたファイル内のすべてのテキスト文字列が表示されます。

-s オプションが指定されていない場合、 表示されたテキストの前に メッセージシーケンス番号が付きます。 メッセージシーケンス番号は、次のように山括弧で囲まれます。<*msgfile*:*msgnum*>.

ここで各要素は以下のような意味を持ちます。

msgfile	表示されたテキストのあるファイル名
msgnum	表示されたテキストのある *msgfile* 内のシーケンス番号

これは、 gettxt(1) および gettxt(3C) で使用されたフォーマットで表示されます。

オプション

-s	表示中のメッセージのメッセージシーケンス番号を表示しません。
-l *locale*	ディレクトリ /usr/lib/locale/*locale*/LC_MESSAGES 内のファイルへアクセスします。 -m *msgfile* も提供された場合、 / を含む *msgfile*s では *locale* が無視されます。
-m *msgfile*	1 つないしは複数の *msgfile* によって指定されたファイルへアクセスします。 *msgfile* に / 文字がある場合、 *msgfile* はパス名として解釈されます。 それ以外の場合、 これは上述のとおりに決定されたディレクトリ内にあると見なされます。 複数の *msgfile* を指定するには、コンマを用いてファイル名を区切ります。
text	*text* によって指定されたテキスト文字列を検索し、 一致する個々の文字列を表示します。 *text* は一般の表現形式をとることができます。 regexp(5) を参照。

使用例

例 1 srchtxt の使用例

mkmsgs(1) を用いて、 french という名のロケールにメッセージファイルがインストールされた場合、 ユーザは次のように入力すると、 frenchのロケール (/usr/lib/locale/french/LC_MESSAGES/*) のテキスト文字列セット全体を表示できます。

```
example% srchtxt –l french
```

例 2 srchtxt の使用例

オペレーティングシステムに関連するエラーメッセージのセットが frenchのロケール用の UX というファイル (/usr/lib/locale/french/LC_MESSAGES/UX) にインストールされている場合、 LANG 環境変数の値を用いて、 検索するロケールを決定し、 次のように入力すると、そのロケール内の該当ファイルから、 ファイルに関するすべてのエラーメッセージを検索できます。

```
example% setenv LANG=french; export  LANG
example% srchtxt -m UX "[Ff]ichier"
```

/usr/lib/locale/french/LC_MESSAGES/UX に次の文字列が含まれていた場合、

```
Erreur E/S\n
Liste d’arguments trop longue\n
Fichier inexistant\n
Argument invalide\n
Trop de fichiers ouverts\n
Fichier trop long\n
Trop de liens\n
Argument hors du domaine\n
Identificateur supprim\n
Etreinte fatale\n
  .
  .
  .
```

次の文字列が表示されます。

```
<UX:3>Fichier inexistant\n
<UX:5>Trop de fichiers ouverts\n
<UX:6>Fichier trop long\n
```

例 3 srchtxt の使用例

オペレーティングシステムに関連するエラーメッセージ・セットがファイル UX にインストールされており、また INGRESS データベース製品に関連するエラーメッセージ・セットがファイル ingress にインストールされていた場合 (両者とも german ロケール内にあるとして)、 次のように入力することによって、 german ロケール内の両ファイル、 UX と ingress でパターン [Dd]atei を検索できます。

```
example% srchtxt -l german -m UX,ingress "[Dd]atei"
```

例 **3** srchtxt の使用例　（続き）

環境

srchtxt の実行に影響を与える環境変数 LC_CTYPE についての詳細は、 environ(5) を参照してください。

ファイル

/usr/lib/locale/C/LC_MESSAGES/*
  mkmsgs(1) によって作成されたデフォルトのファイル

/usr/lib/locale/locale/LC_MESSAGES/*
  mkmsgs(1) によって作成されたメッセージファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWloc

関連項目

exstr(1), gettxt(1), locale(1), mkmsgs(1), gettxt(3C), setlocale(3C), attributes(5), environ(5), locale(5), regexp(5)

診断

srchtxt によって出力されたエラーメッセージは明白で 詳しく解説しなくても理解できるでしょう。 このメッセージは、 コマンド行内のエラーまたは特定のロケールまたはメッセージファイル (あるいは両方) を検索中に発生したエラーを指します。

名前 | jobs, fg, bg, stop, notify – プロセスの実行の制御

sh **jobs** [-p | -l] [% *job_id*...]

jobs -x *command* [*arguments*]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

csh **jobs** [-l]

fg [% *job_id*]

bg [% *job_id*...]

notify [% *job_id*...]

stop % *job_id*...

stop *pid*...

ksh **jobs** [-lnp] [% *job_id*...]

fg [% *job_id*...]

bg [% *job_id*...]

stop % *job_id*...

stop *pid*...

sh ジョブ制御が有効なとき、Bourne シェルに組み込まれた jobs は、停止中またはバックグラウンドで実行中の すべてのジョブを表示します。 %*job_id* を省略すると、停止中またはバックグラウンドで稼動中の すべてのジョブが表示されます。次のオプションを使って、ジョブに関する表示を変更できます。

-l ジョブのプロセスグループ ID および作業ディレクトリを表示します。

-p ジョブのプロセスグループ ID のみを表示します。

-x *command* または *argument* 中に見つかった *job_id* を、対応するプロセスグループ ID に置き換え、*command* に *argument* を渡して実行します。

シェルを jsh として呼び出すと、sh の説明で述べたすべての機能に加えて、ジョブ制御が可能になります。通常、ジョブ制御は対話型シェルに対してだけ可能です。通常、非対話型シェルは、ジョブ制御の機能 を使用しません。

ジョブ制御が可能なとき、ユーザーが端末から入力したコマンドまたはパイプラインは、すべて *job_id* と呼ばれます。どのジョブも、必ずフォアグラウンド、バックグラウンド、または停止のいずれかの状態にあります。 これらの用語の定義を次に示します。

1. フォアグラウンド状態にあるジョブは、 制御している端末への読み取りおよび 書き込みアクセス権を持っています。
2. バックグラウンド状態にあるジョブは、 制御している端末への読み取りアクセスを拒否されていますが、 条件付き書き込みアクセス権は持っています (`stty`(1) を参照)。
3. 停止ジョブは保留状態に置かれたジョブであり、通常は `SIGTSTP` シグナルにより、この状態になります (`signal`(3HEAD) を参照)。

シェルが起動するすべてのジョブには、 ジョブ番号 (*job-id number*) と呼ばれる正の整数が割り当てられます。シェルはこの番号を把握し、 特定のジョブを示す識別子として使用します。また、シェルは現在 (*current*) および前回 (*previous*) のジョブも把握しています。現在のジョブとは、最後に起動または再起動されたジョブです。 前回のジョブとは、その直前のジョブです。

ジョブ識別子の正しい構文は次のような形式です。

`%`*job_id*

job_id は、次のいずれかの形式で指定できます。

`%` または `+`	現在のジョブ
–	前回のジョブ
`?`<*string*>	*string* を含むコマンド行 (一意に表す) に対応したジョブ
n	ジョブ番号が *n* のジョブ
pref	コマンド名の先頭が *pref* のコマンド。 たとえば `ls -l` *name* がバックグラウンドで実行中だった場合、`% ls` と指定すればこのコマンドを示すことができます。*pref* は、引用符で囲まない限り、空白文字を含めることができません。

ジョブ制御が有効なとき、`fg` は中断しているジョブの実行を フォアグラウンドで再開します。 またバックグラウンドで実行中のジョブを フォアグラウンドに移動します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

ジョブ制御が有効なとき、`bg` は中断されているジョブの実行を バックグラウンドで再開します。`%`*job_id* を省略した場合は、 現在のジョブとみなされます。

`stop` は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (`ps`(1) を参照) 。

csh C シェルに組み込まれた `jobs` は、引数なしでジョブ制御下で活動中のジョブを一覧表示します。

`-l` 通常の情報の他に、プロセス ID を表示します。

シェルは、番号の付いた *job_id* を各コマンドシーケンスと対応付けて、バックグラウンドで動作中のコマンド、または TSTP シグナル (通常は Control-Z) によって停止したコマンドの動作を追跡します。コマンドまたはコマンドシーケンス (セミコロンで区切られたリスト) をメタキャラクタ & を使用してバックグラウンドで起動した場合、 シェルは角括弧で囲まれたジョブ番号と 関連するプロセス番号のリストを表示します。以下に例を示します。

[1] 1234現在のジョブリストを見るには、組み込みコマンド jobs を使用します。最後に停止したジョブ (停止したジョブがない場合は、 最後にバックグラウンドに投入されたジョブ) を「現在のジョブ」といい、 '+' で示します。前のジョブは '-' で示します。 現在のジョブが終了したりフォアグラウンドに移された場合、 前のジョブが新しく現在のジョブになります。

ジョブの操作方法については、組み込みコマンド bg、 fg、 kill、 stop、 % の説明を参照してください。

ジョブの参照は '%' で始まります。 パーセント記号だけの指定は、現在のジョブを示します。

% %+ %%	現在のジョブ
%-	前のジョブ
%*j*	'kill -9 %*j*' のようにジョブ *j* を参照します。 *j* はジョブ番号、またはジョブを起動した コマンド行を一意に表す文字列です。 たとえば 'fg %vi' は、停止した vi ジョブをフォアグラウンドに移します。
%?*string*	*string* を含むコマンド行 (一意に表す) に対応したジョブを指定します。

バックグラウンドで動作中のジョブは、端末からの読み取り時に停止します。 バックグラウンドジョブは、通常出力を生成しますが、'stty tostop' コマンドを使用して抑止することも可能です。

fg は現在のジョブまたは指定された *job_id* をフォアグラウンドへ移します。

bg はバックグラウンドで、現在のジョブ または指定されたジョブを実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

notify は現在のジョブまたは指定されたジョブの状態が変わったとき、 その旨非同期にユーザーに知らせます。

ksh jobs は、現在のシェル環境で開始されたジョブの状況を表示します。 jobs がジョブの終了を報告したとき、シェルはそのジョブのプロセス ID を、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除します。

特定のジョブの報告だけが必要なら、*job_id* を使ってジョブを指定します。*job_id* を 1 つも指定しないと、全ジョブに関する情報が出力されます。

以下のオプションは、 `jobs` の出力を変更または拡張するために使用します。

`-l` (文字のエル) 個々のジョブに関して詳細な情報を出力します。具体的には、 ジョブ番号、現在のジョブ、プロセスグループ ID 、状態、ジョブを生成したコマンドを出力します。

`-n` 前回通知を受けた後に停止または終了した ジョブだけを表示します。

`-p` 選択されたジョブのプロセスグループリーダのプロセスグループ ID だけを出力します。

デフォルトでは、`jobs` は、停止しているすべてのジョブの状態、実行中のバックグラウンドジョブの 状態、そして状態が変わったのにシェルによりまだ報告されていない すべてのジョブの状態を表示します。

`set` コマンドの `monitor` オプションを有効にすると、対話型シェルが `job` を各パイプラインと関連付けます。 このオプションは、`jobs` コマンドが表示する現在のジョブのテーブルを維持し、これらのジョブに整数番号を割り当てます。ジョブを `&` で非同期に起動すると、シェルは、`[1] 1234` という形式の行を表示します。 非同期に起動されたジョブはジョブ番号 1 であり、プロセス ID が 1234 であるプロセスが 1 つ (トップレベル) あることを示します。実行中のジョブがあるが、 別に実行したいジョブがある場合、`^Z` (Control-Z) キーを押せば、 現在のジョブに `STOP` シグナルが送信されます。 そうするとシェルは通常、 ジョブが「停止」されたことを示し (後述の「出力」の項を参照)、 プロンプトを表示します。これで、このジョブの状態を `bg` コマンドでバックグラウンドで処理するか、または他のコマンドを実行してから、 `fg` というコマンドで ジョブをフォアグラウンドに移すことができます。`^Z` は直ちに有効になります。つまり `^Z` は、 保留中の出力や読み取られていない入力が 直ちに中止されるという点で、割り込みに似ています。

シェル内のジョブを参照する方法はいくつかあります。そのジョブのいずれかのプロセスの ID を使っても、また以下のいずれかを使っても参照できます。

`%`*number* *number* が示す番号のジョブ

`%`*string* コマンド行が *string* で始まっていたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%?`*string* コマンド行が *string* を含んでいたジョブ。履歴ファイルが有効なとき、 この方法は対話モードでのみ使用できます。

`%%` 現在のジョブ

`%+` `%%` と同じ

`%-` 直前のジョブ

シェルは、 プロセスの状態が変更すると、 直ちにそれを検出します。 ジョブがブロックされてそれ以上進めない状態になると、 シェルはその旨をユーザーに通知します。 ただし、ユーザーの作業の妨げにならないよう、 この通知はプロンプトを発行

する直前にだけ行います。 モニタモードが有効なとき、 完了した各バックグラウンドジョブは、 CHLD に設定されているトラップを起こします。 ジョブの実行中または停止中にシェルを終了しようとすると、 「停止中 (実行中) のジョブがある ('You have stopped (running) jobs.') 」旨の警告を受けます。 jobs コマンドを使用すれば、 どのジョブが該当するのかを確認できます。これを実行するか、 または直ちにシェルを再終了しようとすると、 シェルは 2 度目の警告は出さず、 停止中のジョブは終了します。

fg は、バックグラウンドジョブを、現在の環境からフォアグラウンドへ移します。 fg を使ってジョブをフォアグラウンドへ移した場合、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID のリスト」から 削除されます。 fg コマンドが使えるのは、ジョブ制御をサポートするシステム上だけです。指定された各 *job* をフォアグラウンドで実行します。*job_id* が指定されないと、 現在のジョブをフォアグラウンドで実行します。

bg は、現在の環境で中断されたジョブを、バックグラウンドジョブとして 実行することにより再開します。 *job_id* が示すジョブがすでにバックグラウンドジョブを実行している場合、bg は何も行わず正常に終了します。bg を使ってジョブをバックグラウンドへ移した場合、あたかも非同期リストから起動されたかのように、そのジョブのプロセス ID は、「現在のシェル実行環境で把握しているプロセス ID 」の 1 つとなります。 bg コマンドが使えるのは、 ジョブ制御をサポートするシステム上だけです。*job_id* が省略された場合は、 現在のジョブをバックグラウンドで実行します。

stop は、*job_id* を指定して バックグラウンドジョブの実行を中断、または *pid* (プロセス ID 番号) を指定して すべてのプロセスを中断します (ps(1) を参照) 。

出力

-p オプションを指定すると、各プロセス ID に対して次に示す 1 行の情報が出力されます。

"%d\n", *"process ID"*

-p を省略すると、 -l オプションも省略されていれば、以下の形式の一連の行が出力されます。

"[%d] %c %s %s\n", *job-number*, *current*, *state*, *command*

各フィールドの意味を以下に説明します。

current

文字 + は、 fg および bg コマンド用のデフォルトとして使用するジョブを表します。 このデフォルトジョブは、 *job_id* %+ または %% を使って指定することもできます。文字 – は、現在のデフォルトジョブが終了してしまった場合に デフォルトとなるジョブを表します。このジョブは、 *job_id* %– を使って指定することもできます。 その他のジョブに関しては、このフィールドは空白文字として出力されます。 + や – を使って表せるジョブの数は、どちらも最大 1 つです。停止中の

ジョブがあれば、現在のジョブも停止ジョブとなります。 停止中のジョブが 2 つ以上あれば、以前のジョブも停止ジョブとなります。

job-number

`wait` 、 `fg` 、 `bg` 、 `kill` の各ユーティリティ用にプロセスグループを識別するのに使用する番号。 これらのユーティリティを使うと、ジョブはジョブ番号の後に `%` を付加することにより識別できます。

state

以下の文字列 (POSIX ロケールにある) のいずれかです。

`Running`	ジョブはシグナルによって中断されておらず、 終了もしていないことを表す。
`Done`	ジョブは終了して、ゼロの終了ステータスを返したことを表す。
`Done(`*code*`)`	ジョブは正常に終了し、指定されたゼロ以外の終了ステータス (*code* が示す 10 進数) を返したことを表す。
`Stopped`	
`Stopped (SIGTSTP)`	`SIGTSTP` シグナルがジョブを停止したことを表す。
`Stopped (SIGSTOP)`	`SIGSTOP` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTIN)`	`SIGTTIN` シグナルがジョブを停止したことを表す。
`Stopped (SIGTTOU)`	`SIGTTOU` シグナルがジョブを停止したことを表す。

利用者側で、文字列 `Stopped` の代わりに `Suspended` を使うよう定義することができます。 ジョブをシグナルが終了した場合、 `state` の形式は不定ですが、ここに示した他の `state` 形式とは明確に区別できるものです。その出力上で、 ジョブを終了させたシグナルの名前または説明が与えられます。

command

シェルに与えられた関連コマンド。

`-l` オプションを指定すると、プロセスグループ ID を示すフィールドが `state` フィールドの前に挿入されます。さらに、プロセスグループ内の より多くのプロセスが別の行に出力されることがあります。 その内容は、プロセス ID と `command` フィールドだけです。

環境

jobs 、 fg 、 bg の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

jobs 、 fg 、 bg は、以下の終了ステータスを返します。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), kill(1), ksh(1), ps(1), sh(1), stop(1), shell_builtins(1), stty(1), wait(1), signal(3HEAD), attributes(5), environ(5)

名前 | strchg, strconf – ストリーム構成の変更または照会

形式 | **strchg** -h *module1* [, *module2...*]

strchg -p [-a | -u *module*]

strchg -f *filename*

strconf [-m | -t *module*]

機能説明 | これらのコマンドは、 ユーザーの標準入力に対応するストリームの構成を 変更または照会するために使用されます。 strchg コマンドは、 ストリーム上へモジュールをプッシュ、 もしくはストリームからモジュールをポップします。 strconf コマンドは、 ストリームの構成を問い合わせます。 ストリームの構成を変更できるのは、スーパーユーザーおよび対応する STREAMS デバイスの所有者だけです。

strconf を引数なしで実行した場合、 ストリーム上の全モジュール名と 最上位のドライバ名のリストが返されます。 出力リストは 1 行に 1 つの名前の形式で、 もしストリーム上に最上位モジュールが存在していれば 先頭にその名前が置かれています。最後の項目はドライバ名となります。

オプション | 以下に示すオプションは、 いずれも strchg コマンド用のものです。 なお -h、-f、 -p は相互に排他的なオプションです。

-h *module1* [, *module2*. . .] pus*h* のニモニック。 ストリームへモジュールをプッシュします。 1 つ以上のプッシュ可能なストリームモジュールの名前を 引数として指定します。 これらのモジュールは指定された順序でプッシュされます。 つまり最初に *module1*、次に *module2* というようにプッシュされます。

-p pop のニモニック。 ストリームからモジュールをポップします。 -p だけを記述して strchg を実行すると、 最上位のモジュールがポップされます。

-a *module* 最上位ドライバよりも上位に位置する すべてのモジュールをストリーム上からポップする対象とします。 このオプションは、 必ず -p とともに指定してください。

-u *module* *module* で示したモジュールよりも上位にある すべてのモジュールをストリーム上からポップする対象とします。 *module* 自体は対象とはなりません。 このオプションは、 必ず -p とともに指定してください。

-f *filename* ストリームの構成の変更内容を示す モジュールのリストを含んでいるファイルを *filename* で指定します。 このファイルは、 1 行に 1 モジュール名という形式でなければなりません。 最上位のモジュール名を 1 行目に、 ドライバに最も近いモジュール名を最終行に記述してください。 strchg は、 ストリームの構成が最終的にこのファイルで指定した内容になるよう、 モジュールのプッシュとポップを実行します。

以下に示すオプションは、 いずれも strconf コマンド用のもので、 -m と -t は相互に排他的です。

-m *module*
: *module* で示したモジュールがストリーム上に存在しているかを調べます。 存在していれば、 strconf は yes というメッセージを出力してゼロを返します。 存在しなければ、 no というメッセージを出力してゼロ以外の値を返します。 この -m オプションは、 次の -t とは排他的です。

-t *module*
: 最上位モジュールが存在していればその名前を出力します。 この -t オプションは、 前述の -m とは排他的です。

使用例

例 1 strchg コマンドの例

次の例は、 ユーザーの標準入力に対応するストリーム上に ldterm という名のモジュールをプッシュするものです。

```
example% strchg -h ldterm
```

次の例は、 /dev/term/24 に対応するストリームから 最上位モジュールをポップするものです。 このコマンドを発行するユーザーは、 当該デバイスの所有者もしくはスーパーユーザーでなければなりません。

```
example% strchg -p < /dev/term/24
```

次は、 以下のような内容を含む fileconf というファイルがあるとした場合の例で、

```
ttcompat
ldterm
ptem
```

以下のコマンドを実行すると、

```
example% strchg -f fileconf
```

ユーザーの標準入力ストリームを構成し、 結果として ptem がドライバの上にプッシュされ、 その次に ldterm が置かれ、 ttcompat がストリームの先頭に最も近い位置に置かれます。

引数なしの strconf コマンドは、 ストリーム上の全モジュールと 最上位のドライバのリストを出力します。 たとえばストリームの現在の内容が、 ドライバ zs の上にモジュール ldterm がプッシュされているだけの状態の場合、 引数なしで strconf コマンドを実行すると次のような出力が得られます。

```
ldterm
zs
```

この内容のストリームに対して

```
example% strconf -m ldterm
```

というコマンドを発行すると、 これは ldterm がストリーム上に存在しているかを問い合わせるものなので、 以下のメッセージを出力し、 終了コード 0 を返して実行が終了します。

```
yes
```

例 1 strchg コマンドの例　*(続き)*

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目　attributes(5), streamio(7I)

診断　strchg コマンドは、 実行に成功すれば ゼロを返します。 エラーが発生すると、 エラーメッセージを表示して ゼロ以外の値を返します。 起こりうるエラーとしては、 使用方法の誤り、 不当なモジュール名、 プッシュ対象のモジュールが多すぎる、 ストリーム上の ioctl の失敗、 -f オプションで指定した *filename* がオープンできない、 などが考えられます。

strconf コマンドも、 実行に成功すればゼロを返します（ ここで言う成功とは、 -m オプション使用時には指定したモジュールが 存在していることを、 また -t オプション使用時には最上位モジュールが 存在していることを意味します）。 -m または -t オプションが指定され、 該当するモジュールが存在しない場合、 ゼロ以外の値を返します。 また、使用方法の誤りやストリーム上の ioctl の失敗などのエラーが発生すると、 エラーメッセージを表示してゼロ以外の値を返します。

注意事項　スーパーユーザーでもなく、 ストリームの所有者でもないユーザーは、 strchg コマンドを実行することはできません。 また、スーパーユーザーでもなく、 ストリームに対する読み取り権を持ってもいないユーザーは、 strconf コマンドを実行することはできません。

誤った順序でモジュールをプッシュしてしまうと、 ストリームが期待どおりに機能しなくなる恐れがあります。 また、 tty のストリームで回線規約モジュールを 誤った位置にプッシュしてしまうと、 端末がどのコマンドに対しても応答しなくなる恐れがあります。

名前 | strchg, strconf – ストリーム構成の変更または照会

形式 | **strchg** -h *module1* [, *module2...*]

strchg -p [-a | -u *module*]

strchg -f *filename*

strconf [-m | -t *module*]

機能説明 | これらのコマンドは、 ユーザーの標準入力に対応するストリームの構成を 変更または照会するために使用されます。 strchg コマンドは、 ストリーム上へモジュールをプッシュ、 もしくはストリームからモジュールをポップします。 strconf コマンドは、 ストリームの構成を問い合わせます。 ストリームの構成を変更できるのは、スーパーユーザーおよび対応する STREAMS デバイスの所有者だけです。

strconf を引数なしで実行した場合、 ストリーム上の全モジュール名と 最上位のドライバ名のリストが返されます。 出力リストは 1 行に 1 つの名前の形式で、 もしストリーム上に最上位モジュールが存在していれば 先頭にその名前が置かれています。最後の項目はドライバ名となります。

オプション | 以下に示すオプションは、 いずれも strchg コマンド用のものです。 なお -h、-f、 -p は相互に排他的なオプションです。

-h *module1*
: [, *module2*. . .] pus*h* のニモニック。 ストリームへモジュールをプッシュします。 1 つ以上のプッシュ可能なストリームモジュールの名前を 引数として指定します。 これらのモジュールは指定された順序でプッシュされます。 つまり最初に *module1*、次に *module2* というようにプッシュされます。

-p
: po*p* のニモニック。 ストリームからモジュールをポップします。 -p だけを記述して strchg を実行すると、 最上位のモジュールがポップされます。

-a *module*
: 最上位ドライバよりも上位に位置する すべてのモジュールをストリーム上からポップする対象とします。 このオプションは、 必ず -p とともに指定してください。

-u *module*
: *module* で示したモジュールよりも上位にある すべてのモジュールをストリーム上からポップする対象とします。 *module* 自体は対象とはなりません。 このオプションは、 必ず -p とともに指定してください。

-f *filename*
: ストリームの構成の変更内容を示す モジュールのリストを含んでいるファイルを *filename* で指定します。 このファイルは、 1 行に 1 モジュール名という形式でなければなりません。 最上位のモジュール名を 1 行目に、 ドライバに最も近いモジュール名を最終行に記述してください。 strchg は、 ストリームの構成が最終的にこのファイルで指定した内容になるよう、 モジュールのプッシュとポップを実行します。

以下に示すオプションは、 いずれも strconf コマンド用のもので、 -m と -t は相互に排他的です。

-m *module*
: *module* で示したモジュールがストリーム上に存在しているかを調べます。 存在していれば、 strconf は yes というメッセージを出力してゼロを返します。 存在しなければ、 no というメッセージを出力してゼロ以外の値を返します。 この -m オプションは、 次の -t とは排他的です。

-t *module*
: 最上位モジュールが存在していればその名前を出力します。 この -t オプションは、 前述の -m とは排他的です。

使用例

例 1 strchg コマンドの例

次の例は、 ユーザーの標準入力に対応するストリーム上に ldterm という名のモジュールをプッシュするものです。

```
example% strchg -h ldterm
```

次の例は、 /dev/term/24 に対応するストリームから 最上位モジュールをポップするものです。 このコマンドを発行するユーザーは、 当該デバイスの所有者もしくはスーパーユーザーでなければなりません。

```
example% strchg -p < /dev/term/24
```

次は、 以下のような内容を含む fileconf というファイルがあるとした場合の例で、

```
ttcompat
ldterm
ptem
```

以下のコマンドを実行すると、

```
example% strchg -f fileconf
```

ユーザーの標準入力ストリームを構成し、 結果として ptem がドライバの上にプッシュされ、 その次に ldterm が置かれ、 ttcompat がストリームの先頭に最も近い位置に置かれます。

引数なしの strconf コマンドは、 ストリーム上の全モジュールと 最上位のドライバのリストを出力します。 たとえばストリームの現在の内容が、 ドライバ zs の上にモジュール ldterm がプッシュされているだけの状態の場合、 引数なしで strconf コマンドを実行すると次のような出力が得られます。

```
ldterm
zs
```

この内容のストリームに対して

```
example% strconf -m ldterm
```

というコマンドを発行すると、 これは ldterm がストリーム上に存在しているかを問い合わせるものなので、 以下のメッセージを出力し、 終了コード 0 を返して実行が終了します。

```
yes
```

例 1 strchg コマンドの例　(続き)

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目　attributes(5), streamio(7I)

診断　strchg コマンドは、 実行に成功すれば ゼロを返します。 エラーが発生すると、 エラーメッセージを表示して ゼロ以外の値を返します。 起こりうるエラーとしては、 使用方法の誤り、 不当なモジュール名、 プッシュ対象のモジュールが多すぎる、 ストリーム上の ioctl の失敗、 -f オプションで指定した *filename* がオープンできない、 などが考えられます。

strconf コマンドも、 実行に成功すればゼロを返します（ ここで言う成功とは、 -m オプション使用時には指定したモジュールが 存在していることを、 また -t オプション使用時には最上位モジュールが 存在していることを意味します）。 -m または -t オプションが指定され、 該当するモジュールが存在しない場合、 ゼロ以外の値を返します。 また、使用方法の誤りやストリーム上の ioctl の失敗などのエラーが発生すると、 エラーメッセージを表示してゼロ以外の値を返します。

注意事項　スーパーユーザーでもなく、 ストリームの所有者でもないユーザーは、 strchg コマンドを実行することはできません。 また、スーパーユーザーでもなく、 ストリームに対する読み取り権を持ってもいないユーザーは、 strconf コマンドを実行することはできません。

誤った順序でモジュールをプッシュしてしまうと、 ストリームが期待どおりに機能しなくなる恐れがあります。 また、 tty のストリームで回線規約モジュールを 誤った位置にプッシュしてしまうと、 端末がどのコマンドに対しても応答しなくなる恐れがあります。

名前 | strings – オブジェクトファイルおよびバイナリファイルからの印字可能な文字列の検索

形式 | **strings** [-a | -] [-t *format* | -o] [-n *number* | -number] [*file*...]

機能説明 | strings ユーティリティは、バイナリファイル中の ASCII 文字列を探します。 文字列は、4 バイト以上の印字可能文字で、 復帰改行 (NEWLINE) または NULL 文字で終わらなければなりません。

strings はランダムオブジェクトファイルなどを識別するのに便利です。

オプション | 以下のオプション を指定できます。

-a | –
: ファイル全体を検索の対象とします。 本オプションを指定しないと、 オブジェクトファイルの初期化データスペースのみが対象とされます。

-n *number* | -*number*
: *number* を最短文字列のバイト数とします。 デフォルトは 4 バイトです。

-o
: -t d オプションと同じ機能です。

-t *format*
: ファイル内での先頭からのバイトオフセットの値を 文字列の前に付加して各文字列を出力します。 出力形式は、引数 *format* で示す文字により決まります。

  d　オフセット値を 10 進数で出力する。

  o　オフセット値を 8 進数で出力する。

  x　オフセット値を 16 進数で出力する。

オペランド | 以下のオペランドを指定できます。

file
: 入力に用いる通常ファイルのパス名。 このオペランドを省略すると、strings ユーティリティは標準入力から読み込みます。

環境 | strings の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWtoo
CSI	対応済み

関連項目 | od(1), attributes(5), environ(5)

注意事項 | 文字列を認識するアルゴリズムは、きわめて原始的です。

旧バージョンとの互換性において、 -a の代わりに - を指定することができます。

名前 | stty – 端末用オプションの設定

形式 | **/usr/bin/stty** [-a] [-g]

/usr/bin/stty [*modes*]

/usr/xpg4/bin/stty [-a | -g]

/usr/xpg4/bin/stty [*modes*]

機能説明 | stty コマンドは、現在標準入力に使用しているデバイスに 特定の端末入出力 (I/O) オプションを設定します。 引数を指定しないで実行すると、特定のオプションの設定を報告します。

この報告で、直前にキャレット (^) を伴う文字がある場合、 そのオプションの値は対応する制御文字になります。たとえば、^h は CTRL-H を表します (CTRL-H は Backspace キーと同じ)。^ は、オプションに NULL 値があることを示しています。

このマニュアルページの「制御モード」の項から「ローカルモード」の項で紹介するモードの詳細については、termio(7I) を参照してください。 「ハードウェアフロー制御モード」の項と「クロックモード」の項で紹介するモードの詳細については、termiox(7I) を参照してください。

「組み合わせモード」の項に示すオペランドを使用するときは、これより前の項で説明するオプションが必要です。あまり多くのオプションを組み合わせても意味がありませんが、組み合わせるオプションの数をチェックする機能は特に用意されていません。また、ハードウェアインタフェースの種類によっては、 ハードウェアフロー制御オプションやクロックモードオプションがサポートされない場合があります。

オプション | 次のオプションがサポートされています。

-a	使用端末のオプション設定をすべて標準出力に書き出します。
-g	現在の設定を報告します。この報告は、他の stty コマンドの引数として使用できる形式で出力されます。ドライバが termious をサポートしている場合は termios 型、サポートしていない場合は termio 型の出力になります。

オペランド | 次の *mode* オペランドがサポートされています。

制御モード

parenb(-parenb)	パリティの生成と検出を有効または無効にします。
parext(-parext)	拡張パリティの生成と、マークパリティおよびスペースパリティの検出を有効または無効にします。
parodd(-parodd)	奇数 (偶数) パリティを選択します。parext が有効になっている場合はマーク (スペース) パリティを選択します。
cs5 cs6 cs7 cs8	文字サイズを選択します (termio(7I) を参照)。
0	ただちに回線を切断します。

`hupcl` (`-hupcl`)	最終的な終了時に接続を切断するかしないかを指定します。
`hup` (`-hup`)	`hupcl` (`-hupcl`) と同じです。
`cstopb` (`-cstopb`)	文字ごとに 2 つ (1 つ) の停止ビットを使用します。
`cread` (`-cread`)	受信を有効または無効にします。
`crtscts` (`-crtscts`)	出力ハードウェアフロー制御を有効にします。RTS (送信依頼) モデム制御回線を立てます。CTS (送信可能) 回線が立てられるまで、出力を保留します。
`crtsxoff` (`-crtsxoff`)	入力ハードウェアフロー制御を有効にします。データ受け取りのため RTS (送信依頼) モデム制御回線を立てます。RTS の信号がオフの場合、入力を保留します。
`clocal` (`-clocal`)	回線でモデム制御が行われていない、または行われているものと想定します。
`defeucw`	マルチバイト文字の幅を、`LC_CTYPE` で指定された現在のロケールで定義されている値に設定します。内部的には、文字幅はバイト単位で文字ごとに表現されます。また、スクリーンまたはディスプレイカラムで文字ごとに表現されます。
110 300 600 1200 1800 2400 4800 9600 19200 38400 357600 76800 115200 153600 230400 307200 460800	端末ボーレートに指定された可能な数値を設定します。すべてのハードウェアインタフェースがすべての速度をサポートしているわけではありません。
ispeed 0 110 300 600 1200 1800 2400 4800 9600 19200 38400 57600 76800 115200 153600 230400 307200 460800	端末入力ボーレートに指定された可能な数値を設定します。ただし、すべてのハードウェアが分割ボーレートをサポートしているわけではありません。入力ボーレートを `0` に設定すると、この値は出力ボーレートの値によって指定されます。
ospeed 0 110 300 600 1200 1800 2400 4800 9600 19200 38400 57600 76800 115200 153600 230400 307200 460800	端末出力ボーレートに指定された可能な数値を設定します。ただし、すべてのハードウェアが分割ボーレートをサポートしているわけではありません。出力ボーレートを `0` に設定すると、ただちに回線が切断されます。

入力モード	**ignbrk (-ignbrk)**	入力時のブレークを無視するかどうかを指定します。
	brkint (-brkint)	ブレーク時に INTR シグナルを送るかどうかを指定します。
	ignpar (-ignpar)	パリティエラーを無視するかどうかを指定します。
	parmrk (-parmrk)	パリティエラーをマークするかどうかを指定します (termio(7I) を参照)。
	inpck (-inpck)	入力パリティチェックを有効または無効にします。
	istrip (-istrip)	入力文字を 7 ビットにストリップするか、ストリップしないかを指定します。
	inlcr (-inlcr)	入力時に NL を CR に変換するかどうかを指定します。
	igncr (-igncr)	入力時に CR を無視するかどうかを指定します。
	icrnl (-icrnl)	入力時に CR を NL に変換するかどうかを指定します。
	iuclc (-iuclc)	入力時に大文字のアルファベットを小文字のアルファベットに変換するかどうかを指定します。
	ixon (-ixon)	START/STOP 出力制御を有効または無効にします。STOP 制御文字を送信することによって出力を停止し、 START 制御文字を送信することによって出力を開始します。
	ixany (-ixany)	任意の文字 (DC1 のみ) に出力の再開を許可します。
	ixoff (-ixoff)	入力待ち行列がほとんど空であるか、いっぱいになっているときに、システムから START/STOP 文字が送信されるように、またはされないようにします。
	imaxbel (-imaxbel)	入力行が長すぎるときに、BEL をエコーするかしないかを指定します。
出力モード	opost (-opost)	出力の後処理を行うかどうかを指定します。後処理を行わない場合は、他のすべての出力モードを無視します。
	olcuc (-olcuc)	出力時に小文字のアルファベットを大文字のアルファベットに変換するかどうかを指定します。
	onlcr (-onlcr)	出力時に NL を CR-NL に変換するかどうかを指定します。
	ocrnl (-ocrnl)	出力時に CR を NL に変換するかどうかを指定します。
	onocr (-onocr)	カラム 0 に CR を出力するかどうかを指定します。

	onlret (-onlret)	端末で NL が CR 関数を実行するかどうかを指定します。
	ofill (-ofill)	遅延時に fill 文字を使用します。または、タイミングを使用します。
	ofdel (-ofdel)	fill 文字は DEL (NUL) です。
	cr0 cr1 cr2 cr3	キャリッジリターンの遅延スタイルを選択します (termio(7I) を参照)。
	nl0 nl1	ラインフィードの遅延スタイルを選択します (termio(7I) を参照)。
	tab0 tab1 tab2 tab3	水平タブの遅延スタイルを選択します (termio(7I) を参照)。
	bs0 bs1	バックスペースの遅延スタイルを選択します (termio(7I) を参照)。
	ff0 ff1	フォームフィードの遅延スタイルを選択します (termio(7I) を参照)。
	vt0 vt1	垂直タブの遅延スタイルを選択します (termio(7I) を参照)。
ローカルモード	isig(-isig)	特殊制御文字 INTR、QUIT、SWTCH、SUSP に対する文字のチェックを有効または無効にします。
	icanon (-icanon)	標準的な入力 (ERASE および KILL の処理) を有効または無効にします。 MIN や TIME は設定されません。
	xcase (-xcase)	標準的な (未処理の) 大文字/小文字を表示します。
	echo (-echo)	入力されたすべての文字を表示するかどうかを指定します。
	echoe (-echoe)	ERASE 文字を「バックスペース–空白文字–バックスペース」の文字列として表示するかどうかを指定します。このモードを使用すると、多くの CRT 端末では、ERASE された文字が消去されます。 ただし、このときカラム位置は記録されないので、エスケープした文字、タブ、バックスペースの区別が付きにくくなります。
	echok(-echok)	KILL 文字の後の NL を表示するかどうかを指定します。
	lfkc (-lfkc)	echok (-echok) と同じです。この指定は廃止されました。
	echonl (-echonl)	NL をエコーするかどうかを指定します。
	noflsh (-noflsh)	INTR、QUIT、SUSP の後のフラッシュを無効または有効にします。

	stwrap (-stwrap)	同期回線で 79 文字より長い行の切り捨てを無効または有効にします。
	tostop (-tostop)	バックグラウンドプロセスが端末に書き込むときに、SIGTTOU を送信するかどうかを指定します。
	echoctl (-echoctl)	制御文字を ^*char* としてエコーし、^? として削除します。または、エコーしません。
	echoprt (-echoprt)	消去文字を、消去されたものとしてエコーするかどうかを指定します。
	echoke (-echoke)	行の削除の際、BS-SP-BS で行全体を削除するかどうかを指定します。
	flusho (-flusho)	出力をフラッシュするかどうかを指定します。
	pendin (-pendin)	次の読み取りまたは入力文字で、保留中の入力を再入力するかどうかを指定します。
	iexten (-iexten)	現在、icanon、isig、ixon、ixoff のモード指定によって制御できない特殊な制御文字を有効または無効にします。このような特殊文字には、VEOLZ、VSWTCH、VREPRINT、VDISCARD、VDSUSP、VWERASE、VLNEXT があります。
	stflush (-stflush)	write(2) を実行するたびに、その後で同期回線でのフラッシュを有効または無効にします。
	stappl (-stappl)	同期回線でアプリケーションモードまたは回線モードを使用します。
ハードウェアフロー制御モード	rtsxoff (-rtsxoff)	入力時に RTS ハードウェアフロー制御を有効または無効にします。
	ctsxon (-ctsxon)	出力時の CTS ハードウェアフロー制御を有効または無効にします 。
	dtrxoff (-dtrxoff)	入力時の DTR ハードウェアフロー制御を有効または無効にします。
	cdxon (-cdxon)	出力時の CD ハードウェアフロー制御を有効または無効にします。
	isxoff (-isxoff)	入力時の等時的なハードウェアフロー制御を有効または無効にします。
クロックモード	xcibrg	内部ボーレートジェネレータから送信クロックを取得します。
	xctset	送信シグナル要素のリードタイミング (DCE ソース)、 CCITT V.24 回線 114、EIA-232-D ピン 15 から送信クロックを取得します。
	xcrset	受信シグナル要素のリードタイミング (DCE ソース)、 CCITT V.24 回線 115、EIA-232-D ピン 17 から送信クロックを取得します。
	rcibrg	内部ボーレートジェネレータから受信クロックを取得します。

`rctset` 送信シグナル要素のリードタイミング (DCE ソース)、CCITT V.24 回線 114、EIA-232-D ピン 15 から受信クロックを取得します。

`rcrset` 受信シグナル要素のリードタイミング (DCE ソース)、 CCITT V.24 回線 115、EIA-232-D ピン 17 から受信クロックを取得します。

`tsetcoff` 送信シグナル要素のタイミングクロックが提供されていません。

`tsetcrbrg` 送信シグナル要素のリードタイミング (DTE ソース)、CCITT V.24 回線 113、EIA-232-D ピン 24 上の受信ボーレートジェネレータを出力します。

`tsetctbrg` 受信シグナル要素のリードタイミング (DTE ソース)、CCITT V.24 回線 113、EIA-232-D ピン 24 上の送信ボーレートジェネレータを出力します。

`tsetctset` 送信シグナル要素のリードタイミング (DTE ソース)、CCITT V.24 回線 113、EIA-232-D ピン 24 上の送信シグナル要素のタイミング (DCE ソース) を出力します。

`tsetcrset` 送信シグナル要素のリードタイミング (DTE ソース)、CCITT V.24 回線 113、EIA-232-D ピン 24 上の受信シグナル要素のタイミング (DCE ソース) を出力します。

`rsetcoff` 受信シグナル要素のタイミングクロックが提供されていません。

`rsetcrbrg` 受信シグナル要素のリードタイミング (DTE ソース)、CCITT V.24 回線 128、非 EIA-232-D ピン上の受信ボーレートジェネレータを出力します。

`rsetctbrg` 受信シグナル要素のリードタイミング (DTE ソース)、 CCITT V.24 回線 128、非 EIA-232-D ピン上の送信ボーレートジェネレータを出力します。

`rsetctset` 受信シグナル要素のリードタイミング (DTE ソース)、CCITT V.24 回線 128、非 EIA-232-D ピン上の送信シグナル要素のタイミング (DCE ソース) を出力します。

`rsetcrset` 受信シグナル要素のリードタイミング (DTE ソース)、CCITT V.24 回線 128、非 EIA-232-D ピン上の受信シグナル要素タイミング (DCE ソース) を出力します。

制御の割り当て

control-character c *control-character* を *c* に設定します。*control-character*、*c* には、それぞれ次の値を使用できます。

control-character
`ctab`、`discard`、`dsusp`、`eof`、`eol`、`eol2`、`erase`、`intr`、`kill`、`lnext`、`quit`、`reprint`、`start`、`stop`、`susp`、`swtch`、`werase` (`ctab` は `-stappl` とともに使用。`termio`(7I) を参照)

c
c が単一の文字である場合、制御文字はその文字に設定されます。POSIX ロケールでは、*c* の前に

シェルからのエスケープを表すキャレット (^) が付いている場合、その値は制御文字に対応しています。この対応関係については、次の表を参照してください。たとえば、^d は CTRL-D を表し、^? は DEL と解釈されます。また、^- は未定義です。

^*c*	値	^*c*	値	^*c*	値
a, A	<SOH>	l, L	<FF>	w, W	<ETB>
b, B	<STX>	m, M	<CR>	x, X	<CAN>
c, C	<ETX>	n, N	<SO>	y, Y	<EM>
d, D	<EOT>	o, O	<SI>	z, Z	<SUB>
e, E	<ENQ>	p, P	<DLE>	[	<ESC>
f, F	<ACK>	q, Q	<DC1>	\	<FS>
g, G	<BEL>	r, R	<DC2>	]	<GS>
h, H	<BS>	s, S	<DC3>	^	<RS>
i, I	<HT>	t, T	<DC4>	_	<US>
j, J	<LF>	u, U	<NAK>	?	<DEL>
k, K	<VT>	v, V	<SYN>		

`min` *number*
`time` *number*
: *number* に `min` または `time` の値を設定します。 `MIN` と `TIME` は非標準モードの入力処理 (`-icanon`) で使用されます。

`line` *i*
: 回線制御手順を *i* に設定します (`0`< *i* <`127`)。

組み合わせモード

saved settings
: 現在の端末特性を `-g` オプションで生成され、保存された設定に変更します。

`evenp` または `parity`
: `parenb` と `cs7` を有効にするか、`parodd` を無効にします。

`oddp`
: `parenb`、`cs7`、`parodd` を有効にします。

`spacep`
: `parenb`、`cs7`、`parext` を有効にします。

`markp`
: `parenb`、`cs7`、`parodd`、`parext` を有効にします。

`-parity` または `-evenp`
: `parenb` を無効にし、`cs8` を設定します。

`-oddp`
: `parenb` と `parodd` を無効にし、`cs8` を設定します。

`-spacep`
: `parenb` と `parext` を無効にし、`cs8` を設定します。

	-markp	parenb、parodd、parext を無効にし、cs8 を設定します。

	raw (-raw または cooked)	raw 入力と raw 出力を有効または無効にします。raw モードは次のように設定した場合と同じです。
		`stty cs8 -icanon min 1 time 0 -isig -xcase \`
		`-inpck -opost`

/usr/bin/stty	nl (-nl)	icrnl と onlcr の設定を解除、または設定します。-nl は、inlcr、igncr、ocrnl、onlret の設定も解除します。
/usr/xpg4/bin/stty	nl (-nl)	icrnl を設定、または設定解除します。-nl は、inlcr、igncr、ocrnl、onlret の設定も解除します。つまり、-nl は onlcr を設定し、nl は onlcr の設定を解除します。
	lcase (-lcase)	xcase、iuclc、olcuc を設定、または設定解除します。
	LCASE (-LCASE)	lcase (-lcase) と同じです。
	tabs (-tabs または tab3)	出力時にタブを保存 (空白を拡張) します。
	ek	ERASE 文字、KILL 文字を、通常の #、@ にリセットします。
	sane	すべてのモードを適切な値にリセットします。
	term	すべてのモードを端末タイプ *term* に合うように設定します。ここで、*term* は、tty33、tty37、vt05、tn300、ti700、tek のいずれかです。
	async	通常の非同期通信 (クロック設定が xcibrg、rcibrg、tsetcoff、rsetcoff) を設定します。
ウィンドウサイズ	rows *n*	ウィンドウサイズを *n* 行に設定します。
	columns *n*	ウィンドウサイズを *n* カラムに設定します。
	cols *n*	ウィンドウサイズを *n* カラムに設定します。 cols は、 columns の省略形です。
	ypixels *n*	垂直ウィンドウサイズを *n* ピクセルに設定します。
	xpixels *n*	水平ウィンドウサイズを *n* ピクセルに設定します。

使用法

-g フラグは、 シェルレベルから端末状態の保存や復元を行うために 指定します。次に、プログラムの例を示します。

```
saveterm="$(stty -g)"      # 端末の状態を保存
stty (new settings)        # 新しく状態を設定
. . .           #  . . .
stty $saveterm               # 端末の状態を復元
```

-a を使用すると、表記が冗長になります。このため、端末設定の保存および復元を行うスクリプトには、-g オプションを使用してください。

環境

stty の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH の詳細については、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　入力ファイルがすべて正常に出力された

>0　　エラーが発生した

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

/usr/bin/stty

属性タイプ	属性値
使用条件	SUNWcsu

/usr/xpg4/bin/stty

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目

tabs(1)、ioctl(2)、write(2)、getwidth(3C)、attributes(5)、environ(5)、termio(7I)、termio

名前 | sum – ファイルのチェックサムおよびブロックカウントの出力

形式 | **sum** [-r] [*file...*]

機能説明 | sum ユーティリティは、指定されたファイルの 16 ビットのチェックサムを計算し、出力します。 また、ファイル内の 512 バイトのブロック数も出力します。 通常、これを用いて不良箇所を見つけたり、または ある転送回線で通信されたファイルを検証するのに使用します。

オプション | 以下にオプションを示します。

-r　チェックサムの計算には、代替 (マシン依存型) アルゴリズムを使用。

オペランド | 以下にオペランドを示します。

file　ファイルのパス名。 このオペランドを省略すると、 標準入力とみなされます。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の sum の動作については、largefile(5) を参照してください。

環境 | sum の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　入力ファイルはすべて、正常に出力された

>0　エラーが発生した

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目 | cksum(1), sum(1B), wc(1), attributes(5), environ(5), largefile(5)

診断 | “Read error” は、ほとんどの装置で、ファイルの終わりとの区別がつきません。 ブロックカウントをチェックしてください。

注意事項 | 移植性の必要なアプリケーションには cksum(1) を使用してください。

sum と usr/ucb/sum (sum(1B) 参照) の返すチェックサムは異なります。

名前 | suspend – 現在のシェルを停止させるためのシェル組み込み関数

sh | **suspend**

csh | **suspend**

ksh | **suspend**

sh | 現在のシェルがログインシェルでない場合、 その実行を中断します。

csh | ^Z を使用して停止シグナルが送信されたときと同様に、 シェルを停止します。 このコマンドは、 su コマンドによって開始したシェルを停止するときによく使用します。

ksh | 現在のシェルがログインシェルでない場合、 その実行を中断します。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | csh(1), kill(1), ksh(1), sh(1), su(1M), attributes(5)

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目

`intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | tabs – 端末へのタブ設定

形式 | **tabs** [-n | —*file* [[-code] | -a | -a2 | -c | -c2 | -c3 | -f | -p | -s | -u]] [+m [*n*]] [-T *type*]

tabs [-T *type*] [+ *m* [*n*]] *n1* [, *n2* ,…]

機能説明 | tabs は、以前のタブ設定をクリアした後、タブ仕様 に従って、ユーザー端末にタブストップを設定します。 ユーザー端末には、リモートで設定可能なハードウェアタブがなければなりません。

オプション | 以下のオプションを指定できます。 同じオプションを複数回記述した場合には、 最後に書かれたものが有効です。

-T *type*
: tabs は、タブとマージンを設定するために端末のタイプを知らなければなりません。 type は term(5) で記述されている名前です。 -T フラグが指定されない場合、 tabs は環境変数 TERM の値を使用します。 TERM の値が NULL のとき、または TERM が環境中に定義されていないとき に (environ(5) を参照) 、 tabs は端末タイプとして ansi+tabs を使用し、多くの端末で作動するシーケンスを提供します。

+m[*n*]
: いくつかの端末に対しては、マージン引数を使用することもあります。 これを用いると、 カラム *n+1* を左マージンにすることによって、すべてのタブが *n* カラム移動します。 *n* の値なしで +m が指定された場合、値は 10 と見なされます。 TermiNet では、タブリストの最初の値は 1 でなければなりません。 さもなければ、マージンはさらに右に移動します。 ほとんどの端末上にある通常 (左端) のマージンは、 +m0 によって得られます。 ほとんどの端末のマージンは、 +m フラグが明示的に指定されたときにだけ、リセットされます。

タブ仕様 | 4 種類のタブ仕様が認められています。 それらは、内蔵 (*Canned –code*) 、反復 (-*n*) 、任意 (*n1,n2,...*) 、およびファイル (*––file*) の 4 つです。

タブ仕様が指定されない場合、デフォルト値は –8 、すなわち UNIX システムの「標準」タブです。カラム番号の最小値は 1 です。 注意 : tabs の場合、カラム 1 は常に端末上のいちばん左のカラムを指します。 これは、 DASI 300 、 DASI 300s 、および DASI 450 などのように カラムマーカが 0 で始まる場合でも同様です。

内蔵 | 内蔵されたタブの設定を選択するには下記のコードの 1 つを使用してください。 複数個指定すると、最後に指定したものだけが有効となります。 指定可能なコードとその意味を次に示します。

-a
: 1,10,16,36,72 アセンブラ IBM S/370 の第 1 フォーマット

-a2
: 1,10,16,40,72 アセンブラ IBM S/370 の第 2 フォーマット

-c
: 1,8,12,16,20,55 COBOL 標準フォーマット

-c2
: 1,6,10,14,49 COBOL 圧縮フォーマット (カラム 1-6 を省略) 。 このコードを使用すると、 最初に入力された文字はカードカラム 7 に対応し、

空白を 1 個入れるとカラム 8 となり、 タブ文字を入力するとカラム 12 となります。 このタブ設定を使用するファイルには、 次のようなフォーマット仕様があります (`fspec`(4) を参照)。

```
<:t-c2  m6  s66  d:>
```

`-c3` `1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67` `-c2` より多いタブを伴う COBOL 圧縮フォーマット (カラム 1-6 を省略) 。 COBOL にはこのフォーマットをお勧めします。 適切なフォーマット仕様は次のとおりです (`fspec`(4) を参照)。

```
<:t-c3  m6  s66  d:>
```

`-f` `1,7,11,15,19,23` FORTRAN

`-p` `1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61` PL/I

`-s` `1,10,55` SNOBOL

`-u` `1,12,20,44` UNIVAC 1100 アセンブラ

反復

`-`*n* 反復仕様は、カラム 1+*n* 、 1+2**n* などにタブを要求します。 *n* は 1 桁の 10 進数です。特に重要なのは、値 `8` です。 これは UNIX システムの「標準」タブ設定を表し、 端末でもっとも多くみられるタブ設定です。値 `-0` を指定すると、タブはクリアされ新たな設定は行われません。

任意

「オペランド」の項を参照のこと。

ファイル

–file *file* の名前が指定されると、 `tabs` はフォーマット仕様を検索しながら、そのファイルの先頭行を読み取ります (`fspec`(4) を参照) 。 先頭行にフォーマット仕様があった場合、 その仕様に従ってタブストップを設定します。 そうでない場合、これはタブを `-8` に設定します。 この種の仕様は、 タブ設定されたファイルが正しいタブ設定で出力されているかを 確認する際に用いられます。また `pr` コマンドとともにも用いることもできます。

example% tabs – *file*; `pr` *file*

タブおよびマージンの設定は、標準出力を介して行われます。

オペランド

以下のオペランドを指定できます。

n1[,*n2*, . . .] 任意フォーマットは、コンマまたは空白で区切られた 一連のタブストップ位置指定で構成されます。 タブストップ位置は、 10 進整数を昇順で指定しなければなりません。 最大 40 個まで指定可能です。数値 (最初のものを除く) の前に正の符号が付いている場合、 これは直前の値からの増分と見なされます。 したがって、フォーマット `1,10,20,30` と `1,10,+10,+10` は同じ意味となります。

使用例

例 1 tabs コマンドの使用例

次の例では、 –code (内蔵仕様) を用いて IBM アセンブラが要求する設定、 すなわちカラム 1, 10, 16, 36, 72 にタブを設定します。

```
example% tabs -a
```

次の例では、 -n (反復仕様) (*n* は 8) を用いる使用例で、 これにより、タブは 8 カラムごとに設定されます。 1+(1*8), 1+(2*8), . . . すなわち、カラム 9, 17, . . .

```
example% tabs –8
```

次の例では、 *n1,n2,*. . . (任意仕様) を用いた使用例で、 カラム 1 、8 、および 36 にタブを設定します。

```
example% tabs 1,8,36
```

次の例では *—file* (ファイル仕様) を用いた使用例で、 タブは $HOME/fspec.list/att4425 の先頭行に従って設定されることを示しています (fspec(4) を参照)。

```
example% tabs –$HOME/fspec.list/att4425
```

環境

tabs の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

TERM　端末のタイプを決定します。 この変数が設定されていないかあるいは NULL に設定されていて、 -T オプションが省略された場合、端末タイプとして ansi+tabs が用いられます。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

expand(1), newform(1), pr(1), stty(1), tput(1), fspec(4), terminfo(4), attributes(5), environ(5), term(5)

注意事項

タブの解除および左マージンの設定方法は端末によってそれぞれ異なります。

tabs は (長いシーケンスを要求する端末では) 20 タブしか解除しませんが、 64 タブまで設定できます。

tabs コマンドとともに使用される *tabspec* は、 newform コマンドとともに使用されるものとは異なります。 たとえば tabs においては、 tabs –8 がタブを 8 カラムごとに設定します。 一方 newform においては、 newform –i–8 がタブを 8 カラムごとに設定することを示します。

名前 | tail – ファイルの最終部分の出力

形式 | **/usr/bin/tail** [± *number* [lbcr]] [*file*]

/usr/bin/tail [-lbcr] [*file*]

/usr/bin/tail [± *number* [lbcf]] [*file*]

/usr/bin/tail [-lbcf] [*file*]

/usr/xpg4/bin/tail [-f | -r] [-c *number* | -n *number*] [*file*]

/usr/xpg4/bin/tail [± *number* [l | b | c] [f]] [*file*]

/usr/xpg4/bin/tail [± *number* [l] [f | r]] [*file*]

機能説明 | tail ユーティリティは、指定されたファイルの内容 のうち、指定地点から終わりまでを標準出力にコピーします。 ファイルの指定がない場合は、標準入力を使用します。

コピーの開始地点は、 -c*number* 、 -n*number* 、 ±*number* の各オプションで指定されます。 +*number* は先頭からの距離、 –*number* は終端からの距離を表します。 *number* が NULL のときは 10 と見なされます。 *number* の単位は -c または -n オプションの指定に従い、行またはバイトとなり、さらに l、 b、 c のいずれかのオプションが指定された場合には、それに 従って、行、ブロック、バイトのいずれかとなります。 単位指定がすべて省略された場合には、行と見なされます。

オプション | 以下のオプションは、 /usr/bin/tail と /usr/xpg4/bin/tail で指定できます。 -r と -f オプションは同時に互いに排他的です。 両方がコマンド行に指定された場合、 -f オプションは無効になります。

-b　ブロック単位

-c　バイト単位

-f　継続。 入力ファイルがパイプでない場合、 プログラムは入力ファイルの行がコピーされた後に終了せず、 無限ループに入り、その中で 1 秒間休止し、 入力ファイルからさらにレコードを読み取ったり、コピーしようとします。 したがって、このオプションは、 他のプロセスによって作成中のファイルの成長を監視する際に用います。

-l　行単位

-r　r (reverse) オプションは、ファイルに指定された開始点から行を 逆の順番にコピーします。 r のデフォルトは、ファイル全体を逆順に出力します。

/usr/xpg4/bin/tail | 以下のオプションは、 /usr/xpg4/bin/tail でのみ指定できます。

-c *number*　*number* は、コピーの開始地点をバイト単位で表す 10 進整数です。符号も意味を持ちます。

+　ファイルの先頭からの位置を表す。

–　ファイルの終端からの位置を表す。

符号なし　ファイルの終端からの位置を表す。

カウントは 1 から始まります。つまり `-c+1` はファイルの第 1 バイト目を表し、 `-c-1` は最後のバイトを表します。

`-n` *number*
: 単位がバイトでなく行である点を除き、 `-c` *number* オプションと同じ意味です。 カウントは 1 から始まります。つまり `-n+1` はファイルの第 1 行目を表し、 `-n-1` は最後の行を表します。

オペランド

以下のオペランドを指定できます。

file
: 入力ファイルのパス名。 このオペランドを省略すると、標準入力とみなされます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `tail` の動作については、`largefile`(5) を参照してください。

使用例

例 1 `tail` コマンドの使用

次のコマンドは、ファイル `fred` の最後の 10 行を出力し、 その後には `tail` が初期化されてから削除されるまでの間に `fred` に追加された任意の行が続きます。

```
example% tail  -f  fred
```

別の例として、次のコマンドは、ファイル `fred` の最後の 15 バイトを出力し、 その後には `tail` が初期化されてから削除されるまでの間に `fred` に追加された任意の行が続きます。

```
example% tail  -15cf  fred
```

環境

`tail` の実行に影響を与える環境変数 `LC_CTYPE` 、 `LC_MESSAGES` 、 `NLSPATH` についての詳細は、 `environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0`
: 正常終了

`>0`
: エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/tail

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

/usr/xpg4/bin/tail

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目 | cat(1), head(1), more(1), pg(1), dd(1M), attributes(5), environ(5), largefile(5), XPG4(5)

注意事項 | パイプされた場合、 (ファイルの末尾に相当する) 入力最終部分 は、バッファに格納されるので、その長さには制限があります。 文字型特殊ファイルでは、 さまざまな種類の変則的な動作が起こる可能性があります。

名前 | tar – テープアーカイブの作成およびファイルの追加または抽出

形式 | **tar** c [bBeEfFhiklnopPqvwX [0-7]] [*block*] [*tarfile*] [*exclude-file*] {-I *include-file* | -C *directory* | *file* | *file...*}

tar r [bBeEfFhiklnqvw [0-7]] [*block*] {-I *include-file* | -C *directory* | *file* | *file...*}

tar t [BefFhiklnqvX [0-7]] [*tarfile*] [*exclude-file*] {-I *include-file* | *file...*}

tar u [bBeEfFhiklnqvw [0-7]] [*block*] [*tarfile*] *file...*

tar x [BefFhiklmnopqvwX [0-7]] [*tarfile*] [*exclude-file*] [*file...*]

機能説明 | tar コマンドは、複数のファイルを *tarfile* と呼ばれる 1 つのファイルに保管したり、抽出したりします。 通常、 tarfile は磁気テープですが、 その他のファイルであってもかまいません。 tar の動作は、 キー (*key*) 引数によって制御されます。 キーは、機能文字 (c、r、t、u、または x) 1 つと、使用される機能文字に応じて、これに続く 機能修飾子 (文字または数字) からなる文字列です。 *key* 文字列は空白文字を含みません。 機能修飾子に対する引数は *key* 文字列で使用した順序で コマンド行に指定します。

-I *include-file*、-C *directory file*、*file* 引数は、 どのファイルまたはディレクトリを保管または抽出するかを指定します。 いずれの場合も、ディレクトリ名を使用すると、そのディレクトリのすべてのファイルやサブディレクトリを (再帰的に) 参照します。中括弧 ({ }) で囲まれた引数は、 そのうちのどれか 1 つの引数を指定することを示します。

オプション | 以下にオプションを示します。

-I *include-file*
: 1 行につき 1 つのファイル名からなるファイルの一覧を含んだ *include-file* をオープンし、 各ファイルがコマンド行で 指定されたのと同様に処理します。 *include-file* の行末に不要な空白を置かないように注意してください。 また、各行頭にも不要な空白を置かないように注意してください。 組み込むファイルの最初の文字列に対する一致には、 改行で分かれている行全体が使用されます。 除外されたファイルが存在する場合には (X 機能修飾子を参照) 、 除外されたファイルは 組み込まれたファイルよりも 優先されます。 したがって、あるファイルが *exclude-file* と *include-file* の両方のファイル (またはコマンド行上) で指定されていれば、そのファイルは 除外されます。

-C *directory file*
: *directory* に対して chdir (cd(1) 参照) を実行し、*file* に c (作成) または r (置換) を実行します。*file* には短い相対パス名を使用します。*file* が '.' の場合には *directory* 以下のすべてのファイルを保存します。 このオプションを使うと 共通の親ディレクトリを持たない 複数のディレクトリ中のファイルを保存できます。

オペランド | 以下にオペランドを示します。

file 保存 (c、r、または u 機能を指定した場合) 、 抽出 (x) 、または表示 (t) される通常ファイルまたはディレクトリの パス名。*file* がディレクトリのパス名である場合、動作はそのディレクトリ以下のすべてのファイルと (再帰的に) サブディレクトリに適用します。

ファイルがアーカイブで、 E フラグ (「機能修飾子」を参照) が指定されていない場合、ファイル名は 256 文字を超えることはできません。 さらに、親ディレクトリ名との間を分割して考えると、 ディレクトリ名部分は 155 文字、ファイル名部分は 100 文字を超えることはできません。E フラグが指定されている場合、PATH_MAX で指定されている文字までファイル名に指定できます。

たとえば、ファイル名部分が 100 文字を超えるファイルは、E フラグを指定しないと保存されない可能性があります。ディレクトリ名部分が 200 文字で、 ファイル名部分が 50 文字であるファイルは、ディレクトリ名の 151 文字目から 156 文字目のどこかにスラッシュが 1 つ入っていれば、 E フラグを指定しなくても保存されます。

機能文字 キーの機能部分は、以下の文字のいずれか 1 つで指定します。

c 作成。 tarfile の最初から書き込みます。最後からではありません。

r 置換。指定した *file* を tarfile の最後に書き込みます。 拡張ヘッダーを付けて作成したファイルは、 拡張ヘッダーを付けて更新する必要があります (「機能修飾子」の E フラグの説明を参照)。 拡張ヘッダーなしで作成したファイルは、 拡張ヘッダーを付けて修正することはできません。

t 内容の一覧表示。指定されたファイルが tarfile に検出されるごとに、ファイル名が表示されます。*file* 引数を指定していない場合は、 tarfile のファイル名全部と関連する拡張属性が表示されます。 v 機能修飾子をともに指定すると、指定したファイルの追加情報が表示されます。

u 更新。指定された *file* が tarfile にまだ入っていない場合、または最後に tarfile へ書き込まれて以来変更があった場合は、 *file* は tarfile の最後に追加されます。 更新には時間のかかる場合があります。 SunOS 5.x システム上で作成された tarfile は SunOS 4.x システム上では更新できません。 拡張ヘッダーを付けて作成したファイルは、 拡張ヘッダーを付けて更新する必要があります (「機能修飾子」の E フラグの説明を参照)。 拡張ヘッダーなしで作成したファイルは、 拡張ヘッダーを付けて修正することはできません。

x 抽出または復元。 指定された *file* が tarfile から抽出され、 (現在のディレクトリからの相対パスで) tarfile で指定されたディレクトリに書き込まれます。 そのため、 tarfile にファイルを書き込む際には、ファイルとディレクトリの相対パス名を使用してください。

tar アーカイブに含まれる絶対パス名を復元するときには、絶対パス名を使用します。つまり、先頭のスラッシュ (/) は取り除かれません。

指定されたファイルが、ディレクトリで内容が tarfile に書き込まれている場合は、 このディレクトリは再帰的に抽出されます。 tar コマンドが、ファイルあるいはディレクトリを見つけられないことのないように、 適宜、ファイルやディレクトリの相対パスを使用してください。 可能であれば、所有者、変更時間およびモードを復元します 。所有者を復元するには、スーパーユーザーである必要があります。 文字型特殊デバイスと ブロック型特殊デバイス (mknod(1M) で作成) は、 スーパーユーザーによってのみ抽出できます。*file* 引数が指定されない場合は、 tarfile の全体の内容を抽出します。 tarfile に同じ名前のファイルが複数ある場合は、 各ファイルはパス名通りにディレクトリに書き込まれ、 それ以前のファイルを上書きします。 アーカイブからファイルを抽出する場合は、 ファイル名にワイルドカードを使用することはできません。 この場合には、 次の形式のコマンドを使用してください。

```
tar xvf ... /dev/rmt/0 `tar tf ... /dev/rmt/0 | \
     grep 'pattern' `
```

r または u 機能を使って作成した tarfile から抽出した場合、 ディレクトリ修正時刻が正しく設定されないことがあります。 また、これらの機能は、 バックスペースや追加などの機能がないという テープドライブの制限のために、 多くのテープドライブでは使用できません。

r、u 、または x 機能、 あるいは X 機能修飾子を使用する場合には、 *tarfile* 中の対応するファイルとパス名が 一致している必要があります。 たとえば、 *./thisfile* を抽出するためには、 *thisfile* ではなく *./thisfile* を指定してください。 t 機能はどのように各ファイルが保存されているかを 表示します。

機能修飾子　以下の文字は、使用する機能文字に付けて使用します。

b　ブロック化因数。 生の磁気テープアーカイブに読み取り、または書き込みを する場合に使用します (下記の f を参照) 。*block* 引数では tarfile 上で実行した読み取りや書き込みの各操作で含まれる 512 バイトの テープブロックの 数を指定します。 最小は 1、デフォルトは 20 です。 最大値は使用可能なメモリーの総量と使用する テープデバイス固有のブロック化条件によって決まります (詳細は mtio(7I) を参照) 。 最大値は INT_MAX/512 (4194303) を超えることはできません。

テープアーカイブを読み取る場合は、 実際のブロック化因数が自動的に検出され、 名目上のブロック化因数 (b 修飾子を指定していない場合は、*block* 引数の値かデフォルト値) よりも少ないか、 あるいは等しい値が割り当てられます。 実際のブロック化因数が名目上のブロック化因数よりも 大きい場合、 結果は読み取りエラーになります。 「使用例」の項の例 5 を参照してください。

B　ブロック。 tar は、 (必要であれば) 複数の読み取り操作を実行し、 ブロックを埋めるのに十分なバイト数だけを読み取ります。 パイプやソケットは、 それ以降の入力がある場合でもブロックを部分的に返すので、この機能修飾子は tar がイーサネットを介して動作することを可能にして

います。 標準入力 (–) から読み取る場合、 tar がブロックを埋めるのに十分なバイト数を読み取れるように この機能修飾子が デフォルトで選択されます。

e　エラー。予想外のエラーが発生した場合は、 すぐに終了すると同時に、正の終了ステータスを返します。SYSV3 環境変数はデフォルトの動作を無効にします (「環境」の項を参照)。

E　拡張ヘッダーを付けて tarfile を書き込みます (c、r、u オプションで使用可能、t、x オプションでは無効)。tarfile が拡張ヘッダー付きで書き込まれた場合、 その修正時刻は秒単位ではなく マイクロ秒単位で続けられます。さらにファイル名の文字数が PATH_MAX 文字 (保存には E フラグが必要) 以下で、 ファイルサイズが 8G バイトを超えるファイルの場合もサポートします。 E フラグは、 サイズの大きいファイルや名前の長いファイル、 またはその両方の場合、 あるいはユーザー ID またはグループ ID が 2097151 を超えるときに保存したい場合、 マイクロ秒単位の時間を取りたい場合に 有用です。

f　ファイル。 tarfile として パス名を指定します。f を指定すると、/etc/default/tar を検索しません。 f を省略した場合には、 tar は TAPE 環境変数 (設定されている場合) が示すデバイス を使用します。この変数の設定もなければ、 デフォルトの値は /etc/default/tar に定義されています。 数字 N を使用すると、/etc/default/tar 内で archive*N* (N は数字) に指定されている出力デバイス (ブロック化とサイズの指定付き) を参照できます。次に例を示します。

```
tar -c 2/tmp/*
```

このコマンドは、/etc/default/tar において archive2 に指定されているデバイスに出力を書き込みます。

tarfile の名前が – である場合は、 tar は標準出力への書き込みあるいは 標準入力からの読み取りのいずれか適当な操作を行います。 tar はパイプラインの先頭もしくは末尾として使用することができます。 また、次のコマンドを使えば、 tar は、ディレクトリ階層を移動するためにも使用することができます。

```
example% cd fromdir; tar cf – .| (cd todir; tar xfBp –)
```

F　F 引数を 1 つ指定すると、 tar は tarfile から SCCS および RCS の名前のついたすべてのディレクトリを除外します。 FF のように引数を 2 つ指定すると、 tar は SCCS および RCS の名前のついたすべてのディレクトリ、接尾辞として .o を持つすべてのファイル、および errs、core、a.out という名前のファイルをすべて除外します。SYSV3 環境変数はデフォルトの動作を無効にします (下記を参照)。

h　シンボリックリンクをたどり、 通常ファイルあるいはディレクトリとして扱います。 通常、 tar はシンボリックリンクをたどりません。

i　ディレクトリ・チェックサム・エラーを無視します。

k *size*	サイズの引数を、 K バイト単位の アーカイブサイズとして使用するように要求します。 アーカイブが、フロッピーディスクのような 固定されたサイズのデバイスを指す場合に便利です。 大規模なファイルが、 指定されたサイズに合わない場合には、 ボリュームにまたがって分割されます。
l	リンク。保管されるファイルのリンクで、 tar が解決できないものがある場合、 エラーメッセージを出力します。 l が指定されない場合は、エラーメッセージは表示されません。
m	修正。 ファイルの変更時刻を抽出した際の時刻に設定します。この機能修飾子は、 x 機能とともに用いた場合のみ有効です。
n	テープデバイスにないファイルを読み込みます。 tar は、 アーカイブ内部を読み取り専用で探すことができるため、 アーカイブの読み込みは速くなります。
o	オーナーシップ。 抽出されたファイルに、 tarfile 上のユーザー識別子やユーザーのグループ識別子を使用する代わりに、 プログラムを実行するユーザーのユーザー識別子や グループ識別子を割り当てます。これは、スーパーユーザーではないユーザーに対してデフォルトで行われます。o 機能修飾子が指定されておらず、さらにユーザーが スーパーユーザーである場合、 抽出されたファイルは、 tarfile 上のファイルのグループ識別子とユーザー識別子 を使用します(詳細は、chown(1) を参照)。 o 機能修飾子は、 x 機能とともに用いた場合のみ有効です。
p	現在の umask(1) を無視して、指定されたファイルを元のモード、 および ACL が有効であれば、 ACL を戻します。スーパーユーザーとして x 機能文字を指定して起動した場合は、 これがデフォルトの動作になります。スーパーユーザーである場合は、 SETUID およびスティッキ情報も抽出し、 ファイルは スーパーユーザーの所有ではなく、元の所有者とアクセス権に復元されます。この機能修飾子を、 c 機能とともに使用した場合、 ACL は他の情報と共に tarfile に作成されます。ACL の入った tarfile が tar の前のバージョンによって抽出された場合、 エラーが起こることがあります。
P	アーカイブ中のディレクトリの末尾にスラッシュ (/) を付加しません。
q	名前付きファイルを最初に抽出したあとで 中止します。 通常、 tar はファイルを検索したあともアーカイブの読み取りを続行します。
v	冗長出力。 機能文字に続けて、 各ファイル名を出力します。 t 機能とともに使用すると、 v は、 tarfile エントリに関する詳しい情報も提供します。 表示は ls(1) コマンドの -l オプションによる出力形式に似ています。
w	処理の指示。 とるべき処理とファイル名を出力し、 ユーザーの確認を待ちます。 応答が肯定である場合、この動作が実行されます。 そうでない場合には処理は行われません。この機能修飾子は、 t 機能と同時には使えません。
X	除外。機能 c、x 、または、 t を使用した場合に、 tarfile から除外されるファイル (あるいはディレクトリ) の相対パス名のリストを収めたファイル

として、*exclude-file* 引数を使用します。 *exclude-file* の行末に不要な空白を置かないように注意してください。 また、各行頭にも不要な空白を置かないように注意してください。 除外するファイルの最初の文字列に対する一致には、 改行で分かれている行全体が使用されます。 引数 1 つにつき 1 つの *exclude-file* で、複数の `X` 引数を使用することができます。 この場合、組み込まれたファイルが存在する場合には (`-I` *include-file* オプションを参照) 、 除外されたファイルは 組み込まれたファイルよりも 優先されます。したがって、 あるファイルが *exclude-file* と *include-file* の両方のファイル (またはコマンド行上) で指定されていれば、そのファイルは 除外されます。

`@` 拡張属性をアーカイブに取り込みます。デフォルトでは、`tar` は拡張属性をアーカイブに取り込みません。このフラグを使用すると、`tar` はファイルが拡張属性を持っているかどうかを調べて、持っていれば、拡張属性をアーカイブに取り込みます。アーカイブにおいて、拡張属性は特殊な種類のラベルを持つ特殊なファイルとして格納されます。この修飾子を `x` 機能と一緒に使用すると、拡張属性は通常のファイルのデータと一緒にテープから抽出されます。拡張属性ファイルは、通常のファイルのデータの一部としてのみ抽出できます。拡張属性ファイルだけを明示的に抽出しようとすると無視されます。

`[0-7]` テープをマウントする代替ドライブを選択します。 デフォルトは、`/etc/default/tar` に指定されています。 数字または `f` 機能修飾子が指定されなければ、 `0` を持つ `/etc/default/tar` 中のエントリがデフォルトになります。

使用法

検出するファイルが 2G バイト (2^{31} バイト) 以上ある場合の `tar` の動作については、`largefile`(5) を参照してください。

実際のブロック化因数の自動決定は、パイプやソケットからの読み取り時には正しく行われないことがあります (`B` 機能修飾子を参照)。

1/4 インチのストリームテープは 512 バイト単位のブロック化因数を持つので、 すべてのブロック化因数を使用して、 読み取りまたは書き込みが可能です。

この機能修飾子は、 ディスクファイルや ブロック型特殊デバイス上の アーカイブに対して動作しますが、 主としてテープデバイス用を目的としています。

`tar` のヘッダー形式の情報については `archives`(4) を参照してください。

使用例

例 1 `tar` を使用して、ユーザーのホームディレクトリの アーカイブを作成する

`tar` を使用して、ドライブ `/dev/rmt/0` にマウントされたテープにユーザーのホームディレクトリの アーカイブを作成する例を示します。

```
example% cd
    example% tar cvf /dev/rmt/0 .
    tar からのメッセージ
```

例 **1** tar を使用して、ユーザーのホームディレクトリの アーカイブを作成する　(続き)

c 機能文字は、アーカイブの作成を意味します。 v 機能修飾子は、tar の動作状況を説明するメッセージを出力します。f 機能修飾子は、 tarfile を指定している (この例では /dev/rmt/0) ことを示します。 コマンド行の最後のドット (.) は現在のディレクトリを示し、 f 機能修飾子の引数になります。

次のコマンドで、 tarfile の内容の一覧を表示します。

```
example% tar tvf /dev/rmt/0
```

POSIX ロケールでは、 次のように出力されます。

```
    rw-r--r--   1677/40    2123    Nov  7 18:15 1985    ./test.c
    ...
    example%
```

各カラムは次の意味を持ちます。

- カラム 1 は、./test.c へのアクセス権
- カラム 2 は、./test.c の ユーザー ID またはグループ ID
- カラム 3 は、./test.c のバイトサイズ
- カラム 4 は、./test.c の修正時刻。 LC_TIME カテゴリが POSIX ロケールに設定されていない場合、 形式や日付順フィールドが異なって使用されることがあります。
- カラム 5 は、./test.c のファイル名

アーカイブから、ファイルを 抽出するためには、 次のコマンドを使用します。

```
example% tar xvf /dev/rmt/0
    tar からのメッセージ
    example%
```

テープ上に複数のアーカイブファイルがある場合は、 それぞれのファイルは、 EOF マーカによって次のファイルと区切られます。 tar を使用して複数のアーカイブファイルが入っているテープから 1 番目と 2 番目のアーカイブファイルを読ませる場合、 以下のように f 機能修飾子として渡すテープデバイス名は *non-rewind* 指定のある方を使用してください。

```
example% tar xvfp /dev/rmt/0n テープからの、最初のアーカイブの読み取り
    tar からのメッセージ
    example% tar xvfp /dev/rmt/0n テープからの、次のアーカイブの読み取り
    tar からのメッセージ
    example%
```

以前のリリースでは、 上記の処理が正しく動作しなかったり、 mt(1) と tar の実行との間に調整が必要になったりする場合が ありますので注意してください。 以前のリリースでの動作をエミュレートするには、 BSD 用の動作を行う b 文字を含んだ non-rewind のデバイス名を 使用してください。 詳細については mtio(7I) マニュアルページの 「 Close Operations 」の項を参照してください。

例 **2** tar を使用して、デフォルトのテープドライブ 0 に /usr/include のファイルおよび /etc のファイルを保存する

デフォルトのテープドライブ 0 に /usr/include のファイルおよび /etc のファイルを保存するためには、 次のコマンドを使用します。

```
example% tar c -C /usr  include -C /etc .
```

このコマンドによって保存された tarfile からの内容の一覧を表示すると、 たとえば次のようになります。

```
include/
    include/a.out.h
    /usr/include ... の下のすべてのファイル
    ./chown /etc にあるその他すべてのファイル
```

include の下にあるすべてのファイルを抽出するためには、次のコマンドを使用します。

```
example% tar xv include
    x include/, 0 bytes, 0 tape blocks \
    include ... の下のすべてのファイル
```

例 **3** ネットワークを介してファイルを転送する

tar を使用して、 ネットワークを介してファイルを転送する例を示します。 最初に、ローカルマシン (example) からリモートシステム (*host*) 上のテープへファイルを保管する方法を示します。

```
example% tar cvfb  -  20 files |  rsh host dd of=/dev/rmt/0  obs=20b
tar からのメッセージ
example%
```

上記の例では、 c 機能文字を使用して tarfile を作成し、 v 機能修飾子により tar からの冗長出力を要求し、 f 機能修飾子を用いて出力 tarfile の名前を指定し (- で、標準出力を指定)、b 機能修飾子によりブロックサイズ (20) を指定しています。 ユーザーがブロックサイズを変更したい場合は、ユーザーは、 tar コマンドおよび dd コマンド両方のブロックサイズ引数を変更する必要があります。

例 **4** リモートシステム上のテープから ローカルシステムへファイルを戻す

次に、 tar を使用して、 リモートシステム上のテープから ローカルシステムへファイルを戻す例を示します。

```
example% rsh -n host dd if=/dev/rmt/0 bs=20b | \
    tar xvBfb - 20 files
tar からのメッセージ
example%
```

上記の例では、x キー文字を使用して tarfile からファイルの抽出を行い、v 機能修飾子により tar からの冗長出力を要求し、B 機能修飾子によりパイプから読み取りを行うように tar に指示し、 f 機能修飾子を用いて入力 tarfile の名前を指定し (- で、標準入力を指定) 、b 機能修飾子によりブロックサイズ (20) を指定しています。

例 **5** ホームディレクトリのアーカイブを作成する

次に 実際のブロック化因数を 19 にして /dev/rmt/0 上にホームディレクトリのアーカイブを作成する例を示します。

```
example% tar cvfb /dev/rmt/0 19 $HOME
```

b 機能修飾子を使用しないでアーカイブの実際のブロック化因数を 認識するためには、 次のコマンドを使用します。

```
example% tar tvf /dev/rmt/0
tar: blocksize = 19
...
```

実際のブロック化因数 よりも大きい名目上のブロック化因数を使用して、 アーカイブの実際のブロック化因数を 認識するためには、 次のコマンドを使用します。

```
example% tar tvf /dev/rmt/0 30
tar: blocksize = 19
...
```

実際のブロック化因数に対して 小さすぎる名目上のブロック化因数を使用して、アーカイブの実際のブロック化因数を 認識しようとした場合は次のようになります。

```
example% tar tvf /dev/rmt/0 10
tar: tape read error
```

環境

SYSV3 この環境変数は、 デフォルトの tar の動作を無効にして、INTERACIVE UNIX システムと SCO UNIX のインストールスクリプトとの 互換性を提供するために使用します。 新しいスクリプトでは使用しないでください (互換性だけを目的とした環境変数です)。 設定した場合、次のオプションの動作が異なります。

-F *filename* コマンド行スイッチと対象となるファイルのリストを 獲得するために *filename* を使用します。

-e ファイルが、ボリュームにまたがって分割されないようにします。 ひとつのボリュームに十分な空きがない場合、 tar は新しくボリュームを入力するようプロンプトを出します。 ファイルが新しいボリュームに修正されない場合、 tar はエラーで終了します。

tar の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、TZ、NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 入力ファイルはすべて、正常に出力された

>0 エラーが発生した

ファイル

/dev/rmt/[0-7][b][n]

/dev/rmt/[0-7]l[b][n]

/dev/rmt/[0-7]m[b][n]

/dev/rmt/[0-7]h[b][n]

/dev/rmt/[0-7]u[b][n]

/dev/rmt/[0-7]c[b][n]

/etc/default/tar　設定は以下のようになります。

```
archive0=/dev/rmt/0
archive1=/dev/rmt/0n
archive2=/dev/rmt/1
archive3=/dev/rmt/1n
archive4=/dev/rmt/0
archive5=/dev/rmt/0n
archive6=/dev/rmt/1
archive7=/dev/rmt/1n
```

/tmp/tar*

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み
インタフェースの安定性	安定

関連項目

ar(1), basename(1), cd(1), chown(1), cpio(1), csh(1), dirname(1), ls(1), mt(1), pax(1), setfacl(1), umask(1), mknod(1M), vold(1M), archives(4), attributes(5), environ(5), fsattr(5), largefile(5), mtio(7I)

診断

キー文字の誤りとテープ読み取り/書き込みエラー、 およびリンクテーブルを保持する十分なメモリーがないという 診断メッセージが出力されます。

注意事項

ファイルの *n* 番目に現れるものをアクセスする方法がありません。

テープエラーの処理は不十分です。

ボリューム管理プログラムデーモンが実行中である場合、 従来のデバイス名 (たとえば /dev/rdiskette) でフロッピーディスクデバイスへアクセスすると 失敗する可能性があります。 詳細については vold(1M) を参照してください。

tar アーカイブ形式では アーカイブヘッダーにユーザー ID およびグループ ID を 2097151 まで格納することができます。 この値よりも大きいユーザー ID およびグループ ID を持つファイルは 60001 のユーザー ID およびグループ ID で格納されます。

アーカイブを作成するときに、 複数のロケールで実行している処理によってファイル名を作成したファイルを含んでいる場合、アーカイブの作成とアーカイブからのファイルの抽出のどちらも、フル 8 ビットコードセットを使用するロケール (たとえば、 en_US ロケール) 上で使用すべきです。

1/4 インチのアーカイブテープ用のテープドライブはバックスペースできないので、このようなテープでは -r オプションと -u オプションは使用できません。

名前 | tbl – nroff または troff 用のテーブルの書式化

形式 | **tbl** [-me] [-mm] [-ms] [*filename*]...

機能説明 | tbl は、 nroff(1) または troff(1) 用のテーブルを書式化するプリプロセッサです。 入力ファイル *filename* は、標準出力にコピーされます。 この時、 コマンド行 .TS と .TE の間にある行だけが、 テーブルを記述するものとみなされ、 書式化し直されます。

引数を省略すると、 tbl は標準入力を読み取ります。 したがって tbl はフィルタとして使用することができます。 tbl を eqn(1) または neqn とともに使用する場合は、 tbl コマンドを最初に指定し、 パイプを通して渡すデータを最小限に抑えるようにします。

オプション | -me　　-me マクロパッケージを出力ファイルの前にコピーします。

-mm　　-mm マクロパッケージを出力ファイルの前にコピーします。

-ms　　-ms マクロパッケージを出力ファイルの前にコピーします。

使用例 | 例 **1** tbl の使用

例として、 次のような入力ファイルがあるとします (実際には '@' はタブ文字を入力する) 。

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town@Households
@Number@Size
Bedminster@789@3.26
Bernards Twp.@3087@3.74
Bernardsville@2018@3.30
Bound Brook@3425@3.04
Branchburg@1644@3.49
.TE
```

この出力は次のようになります。

Household Population		
Town	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30

例 **1** tbl の使用　(続き)

```
Bound Brook             3425                    3.04
Branchburg              1644                    3.49
```

ファイル

/usr/share/lib/tmac/e	-me マクロ
/usr/share/lib/tmac/m	-mm マクロ
/usr/share/lib/tmac/s	-ms マクロ

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目　eqn(1), nroff(1), troff(1), attributes(5)

名前 | tee – 標準出力の複写

形式 | **tee** [-ai] [*file*...]

機能説明 | tee ユーティリティは、 0 個または複数のファイルを作成して、 標準入力を標準出力にコピーします。 tee は出力をバッファに入れません。 オプションでは、指定したファイルを上書きするか、 または追加するかを決定します。

オプション | 以下にオプションを示します。

-a　出力は、 ファイルを上書きするのではなく、 ファイルに追加されます。

-i　割り込みを無視します。

オペランド | 以下にオペランドを示します。

file　出力ファイルのパス名。 少なくとも 13 の *file* オペランドが処理できます。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の tee の動作については、largefile(5) を参照してください。

環境 | tee の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　標準入力は、正常にすべての 出力ファイルにコピーされた

>0　オープンできない、 または状態が得られないファイルの数

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | cat(1), attributes(5), environ(5), largefile(5)

名前 | test – 条件の評価

形式 | **/usr/bin/test** [*condition*]

[[*condition*] **]**

sh | **test** [*condition*]

[[*condition*] **]**

csh | **test** [*condition*]

[[*condition*] **]**

ksh | **test** [*condition*]

[[*condition*] **]**

機能説明 | `test` ユーティリティは、*condition* が示す条件を評価し、その結果を終了ステータスで表します。評価結果が真のときは終了ステータスが 0 に、偽のときは 1 になります。

test ユーティリティの第 1 の形式は次のとおりです。

```
test [ condition ]
```

角括弧は *condition* が省略可能なオペランドであり、必ずしもコマンド行で指定する必要がないことを示します。

test ユーティリティの第 2 の形式は次のとおりです。

```
[ [ condition ] ]
```

最初の開く角括弧 (`[`) は必須のユーティリティ名です。内側の角括弧は *condition* がオプションであることを示しています。最後の閉じる角括弧 (`]`)は必須のオペランドです。

2G バイト (2^{31} バイト) 以上のファイルに対する `test` の動作については、`largefile`(5) のマニュアルページを参照してください。

`test` と [ユーティリティは条件 *condition* を評価して、その値が真である場合は終了ステータスを 0 に設定し、そうでない場合はゼロ以外の値 (偽) に設定します。引数がない場合は、`test` と `[` ユーティリティは終了ステータスをゼロ以外の値に設定します。アクセス権をテストするときは、プロセスの実効ユーザー ID が使用されます。

test と [ユーティリティに指定する引数、つまり、オペレータ、フラグ、および (最後の行を示す) 括弧はすべて、区切り文字 (通常はスペース文字) で区切る必要があります。

オペランド | 2 つの要素を持つ、次の形式のプライマリのことを「単項プライマリ」と呼びます。

-primary_operator primary_operand

3 つの要素を持つ、次のいずれかのプライマリのことを「2 項プライマリ」と呼びます。

primary_operand -primary_operator primary_operand
primary_operand primary_operator primary_operand

ファイルオペランド (-h と -L プライマリを除く) がシンボリックリンクを示している場合、シンボリックリンクは展開され、test は展開後のファイルに実行されます。

自分のファイルをテストした場合でも (-r、-w、または -x のテスト)、テストしたファイルのアクセス権に所有者ビットが設定されていない場合は、ファイルのアクセス権に「グループ」ビットまたは「その他」のビットが設定されていても、ゼロ以外 (偽) の終了ステータスが返されます。

= と != プライマリは単項プライマリよりも優先されます。= と != プライマリは常に引数をとります。したがって、= と != は単項プライマリの引数として使用できません。

condition を記述するために以下の基本式が使用できます。

-a *file*	ファイル *file* が存在すれば真。(sh では使用不可)
-b *file*	ファイル *file* が存在し、ブロック型特殊ファイルであれば真。
-c *file*	ファイル *file* が存在し、文字型特殊ファイルであれば真。
-d *file*	ファイル *file* が存在し、ディレクトリであれば真。
-e *file*	ファイル *file* が存在すれば真。(sh では使用不可)
-f *file*	ファイル *file* が存在し、そのファイルが通常ファイルであれば真。あるいは、/usr/bin/sh ユーザーが自分の PATH 環境変数内で /usr/bin よりも前に /usr/ucb を指定しており、*file* が存在し、そのファイルがディレクトリでなければ、test は真を返します。csh の test と [ビルトインは常に後者の動作を行います。
-g *file*	ファイル *file* が存在し、セットグループ ID フラグが設定されていれば真。
-G *file*	ファイル *file* が存在し、ファイルのグループがプロセスの実効グループ ID と一致していれば真。(sh では使用不可)
-h *file*	ファイル *file* が存在し、シンボリックリンクであれば真。
-k *file*	ファイル *file* が存在し、スティッキビットが設定されていれば真。

`-L` *file*	ファイル *file* が存在し、シンボリックリンクであれば真。
`-n` *string*	*string* の長さがゼロでなければ真。(`sh` では使用不可)
`-o` *option*	オプション *option* がついていれば真。(`csh` または `sh` では使用不可)
`-O` *file*	ファイル *file* が存在し、プロセスの実効ユーザー ID がそのファイルを所有していれば真。(`sh` では使用不可)
`-p` *file*	ファイル *file* が名前付きパイプ (FIFO) であれば真。
`-r` *file*	ファイル *file* が存在し、読み取り可能であれば真。
`-s` *file*	ファイル *file* が存在し、サイズがゼロより大きければ真。
`-S` *file*	ファイル *file* が存在し、そのファイルがソケットであれば真。(`sh` では使用不可)
`-t` [*file_descriptor*]	ファイル記述子番号が *file_descriptor* であるファイルがオープンされていて、端末装置に対応していれば真。*file_descriptor* が指定されていない場合、デフォルト値として `1` が使用されます。
`-u` *file*	ファイル *file* が存在し、セットユーザー ID フラグが設定されていれば真。
`-w` *file*	ファイル *file* が存在し、書き込み可能であれば真。真であっても、書き込みフラグが設定されていることを表すだけです。ファイルシステムが読み取り専用であれば、この条件が真であっても *file* は書き込み不可能です。
`-x` *file*	ファイル *file* が存在し、実行可能であれば真。真であっても、実行フラグが設定されていることを表すだけです。*file* がディレクトリの場合には、真は *file* が検索可能なことを表します。
`-z` *string*	文字列 *string* の長さがゼロであれば真。
file1 `-nt` *file2*	ファイル *file1* が存在し、*file2* よりも新しい場合は真。(`sh` では使用不可)
file1 `-ot` *file2*	ファイル *file1* が存在し、*file2* よりも古い場合は真。(`sh` では使用不可)
file1 `-ef` *file2*	ファイル *file1* と *file2* が存在し、同じファイルを参照している場合は真。(`sh` では使用不可)
string	文字列 *string* が NULL の文字列でなければ真。
string1 `=` *string2*	文字列 *string1* と *string2* が等しければ真。
string1 `!=` *string2*	文字列 *string1* と *string2* が等しくなければ真。

n1 `-eq` *n2*　整数 *n1* と *n2* が代数的に等しければ真。

n1 `-ne` *n2*　整数 *n1* が代数的に整数 *n2* より大きければ真。

n1 `-gt` *n2*　整数 *n1* が代数的に整数 *n2* より大きければ真。

n1 `-ge` *n2*　整数 *n1* が代数的に整数 *n2* より大きいか等しければ真。

n1 `-lt` *n2*　整数 *n1* が代数的に整数 *n2* より小さければ真。

n1 `-le` *n2*　整数 *n1* が代数的に整数 *n2* より小さいか等しければ真。

condition1 `-a` *condition2*　*condition1* および *condition2* の両方が真であれば真。`-a` 2 項プライマリは左結合であり、`-o` 2 項プライマリよりも優先される。

condition1 `-o` *condition2*　*condition1* または *condition2* のどちらかが真であれば真。`-o` バイナリプライマリは左結合である。

これらの基本式は、以下の演算子と組み合わせて指定できます。

`!` *condition*　条件 *condition* が偽であれば真。

(*condition*)　condition が真であれば真。丸括弧 ( ) を使用すると、通常の優先順位や結合規則を変更できる。なお、丸括弧はシェルにとっても意味を持つので、引用する必要がある。

演算子の優先順位を決めるアルゴリズム、および生成される戻り値は、 `test`に渡す引数の数により異なります。ただし `[ . . . ]` 形式を使う場合、最終引数の右角括弧は、このアルゴリズムでは引数には含まれません。

以下のリストにおいて、`$1`、`$2`、`$3`、`$4` は、*condition*、*condition1*、または *condition2* として `test` に渡す引数を表します。

引数 0 個　偽 (1) で終了。

引数 1 個　`$1` が空でなければ真 (0) で終了。NULL なら偽。

引数 2 個

- `$1` が `!` のとき、 `$2` が空なら真、`$2` が空でなければ偽。
- `$1` が単項基本式のとき、単項テストの結果が真なら真、偽なら偽。
- その他の場合、結果は不定。

引数 3 個

- `$2` が 2 項基本式のとき、`$1` と `$3` を対象に 2 項テストを実行。
- `$1` が `!` のとき、`$2` と `$3` の 2 つの引数に対するテストを否定形に。
- その他の場合、結果は不定。

引数 4 個

- $1が ! のとき、$2、$3、$4 の 3 つの引数に対するテストを否定形に。
- その他の場合、結果は不定。

使用法

基本式と演算子が混在しているような、ユーザーが用意した入力データを処理する際、スクリプト側では十分な注意が必要です。スクリプトに対して入力データが生成されるケースをアプリケーション作成者がすべて理解していない限り、test "$1" -a "$2" というような記述は、test "$1" && test "$2" と変えた方がいいでしょう。ユーザーが $1 を ! に設定し、$2 を空の文字列に設定したような場合の問題を防ぐためです。つまり移植性の高さが問題の場合には、test expr1 -a expr2 というような記述は以下の記述に変更し、test expr1 && test expr2 また、test expr1 -o expr2 の記述は以下のように変更してください。test expr1 || test expr2 ただし、シェルでは && と || が同等の優先順位を持つのに対し、test では -a の方が -o よりも優先順位が高いことに注意してください。

シェルコマンド言語では、グループ分け用に小括弧と中括弧が使用できます。

sh を使うときは、小括弧はエスケープ付きで記述しなければなりません。次の例を見てください。

```
test \( expr1 -a expr2 \) -o expr3
```

このコマンドは、 XSI 準拠のシステム以外では、必ずしも移植可能ではありません。この形式の代わりに、以下のように記述できます。

```
( test expr1 && test expr2 ) || test expr3
```

次に、以下の 2 つのコマンドを見てください。

```
test "$1"
test ! "$1"
```

古いシステムの一部では、このコマンドの動作は信頼性を欠くことがあります。つまりこのような *string* 条件を使い $1 を !、(、または単項基本式に展開すると、その結果は保証できません。したがって、それぞれ以下のように記述する方がいいでしょう。

```
test -n "$1"
test -z "$1"
```

同様に、古いシステムでは以下のようなコマンドの信頼性は保証できません。

```
test "$response" = "expected string"
```

これは、次のどちらかの形式に変えるようにしてください。

```
test "X$response" = "Xexpected string"
test "expected string" = "$response"
```

なお 2 番目の形式は、expected string が単項基本式とまぎらわしくなることはない、というのが前提です。つまり expected string の先頭文字が –、(、!、= のいずれかのときは、1 番目の形式を使用してください。注記した事項を除き、前述の規則を使用すれば、上記 3 つの比較形式はどのような入力データに対しても信頼性の高いものです。なお、どの形式でも文字列は引用符で囲んで記述する点に注意してください。

引数の数が 4 個を超えている場合、文字列比較の 2 項基本式 = と != はどの単項基本式よりも優先順位が高いので、引数が正しく指定されていないと処理結果は保証できません。たとえば次の例を見てください。

```
test -d $1 -o -d $2
```

ここで $1 の評価結果がディレクトリ名 = となった場合、先頭の 3 つの引数は文字列比較を表すものと見なされます。したがって 2 つ目の -d を検出したときに構文エラーとなります。これを避けるには、次に示すどちらかの形式、できれば 2 番目の方を使用してください。

```
test \( -d "$1" \) -o \( -d "$2" \)
test -d "$1" || test -d "$2"
```

また引数が 4 個を超えているときに次のように指定したとします。

```
test "$1" = "bat" -a "$2" = "ball"
```

このとき $1 の評価結果が (または ! だと構文エラーとなります。これを避けるには、次に示すいずれかの形式、できれば 3 番目を使用してください。

```
test "X$1" = "Xbat" -a "X$2" = "Xball"
test "$1" = "bat" && test "$2" = "ball"
test "X$1" = "Xbat" && test "X$2" = "Xball"
```

使用例

if の例では次の 3 つの状態が調べられ、3 つのすべてが真、または正常終了とみなされた場合、その結果が画面に出力されます。

- 1 に設定されている変数の値が 0 より大きいか
- 2 に設定されている変数の値が 2 に等しいか
- "root" という語がテキストファイル /etc/passwd に含まれているか

/usr/bin/test

例 1 /usr/bin/test の使用

ディレクトリが存在しなければ mkdir を実行する。

```
test ! -d tempdir && mkdir tempdir
```

例 1 /usr/bin/test の使用　(続き)

読み取り不可になるまで少し待つ。

```
while test -r thefile
do
   sleep 30
done
echo'"thefile" is no longer readable'
```

引数が 3 つの文字列 (2 つは変形) のうちの 1 つであり、test コマンドの開く角括弧バージョン ([) を使用していればコマンドを実行する。

```
if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
then
    command
fi
case "$1" in
    pear|grape|apple) command;;
esac
```

test 組み込みコマンド

以下の例では、Bourne シェルの if に続いて test 組み込みコマンドの 2 つの形式を記します。

例 2 sh 組み込みコマンドの使用

```
ZERO=0 ONE=1 TWO=2 ROOT=root

if  [ $ONE -gt $ZERO ]

[ $TWO -eq 2 ]

grep $ROOT  /etc/passwd >&1 > /dev/null  #出力を破棄する

then

    echo "$ONE is greater than 0, $TWO equals 2, and $ROOT is" \
          "a user-name in the password file"

else

    echo "At least one of the three test conditions is false"
fi
```

例 3 test 組み込みコマンドの使用

test 組み込みコマンドの例

```
test `grep $ROOT /etc/passwd >&1 /dev/null`   #出力を破棄する

echo $?    #成功テスト
[ `grep nosuchname /etc/passwd >&1 /dev/null` ]

echo $?    #失敗テスト
```

例 **3** test 組み込みコマンドの使用　*(続き)*

csh

例 **4** csh 組み込みコマンドの使用

```
@ ZERO = 0; @ ONE = 1; @ TWO = 2;  set ROOT = root
grep $ROOT  /etc/passwd >&1 /dev/null  #出力を破棄する
    # grep 実行直後に$ status をテストする
if ( "$status" == "0" && $ONE > $ZERO && $TWO == 2 ) then
       echo "$ONE is greater than 0, $TWO equals 2, and $ROOT is" \
             "a user-name in the password file"
 endif
```

ksh

例 **5** ksh 組み込みコマンドの使用

```
ZERO=0 ONE=1 TWO=$((ONE+ONE)) ROOT=root
if  ((ONE > ZERO))             # 数式比較
 [[ $TWO = 2 ]]                 # 文字列比較
 [ `grep $ROOT  /etc/passwd >&1 /dev/null` ] #出力を破棄する
then
     echo "$ONE is greater than 0, $TWO equals 2, and $ROOT is" \
             "a user-name in the password file"

else
     echo "At least one of the three test conditions is false"
fi
```

環境

test の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　*condition* の評価結果は真だった。

1　　*condition* の評価結果は偽だった、または *condition* が指定されていなかった。

>1　　エラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
インタフェースの安定性	標準

関連項目

csh(1), ksh(1), sh(1), test(1B), attributes(5), environ(5), largefile(5)

注意事項

-f オプションの代替である not-a-directory は BSD アプリケーションの移行を補助するものであり、将来のリリースではサポートされなくなる可能性があります。

名前 | time – 単純コマンドの時間測定

形式 | **time** [-p] *utility* [*argument. ..*]

機能説明 | time ユーティリティは *utility* オペランドで指定されたユーティリティを *argument* とともに呼び出し、 *utility* の時間統計情報をまとめて標準エラー出力に書き出します。この情報には以下のものが含まれます。

- *utility* を呼び出してからその実行が終了するまでの経過 (実) 時間
- ユーザー CPU 時間。これは *utility* を実行したプロセスに対して times(2) 関数が返す *tms_utime* と *tms_cutime* の値の合計と同じ意味です。
- システム CPU 時間。これは *utility* を実行したプロセスに対して times() 関数が返す *tms_stime* と *tms_cstime* の値の合計と同じ意味です。

time をパイプラインの一部として使用した場合、 それがパイプライン内のグループ分けコマンド中の 唯一のコマンドである場合を除き、 報告される時間情報の内容は予測できません。 以下の例を見てください。左側のコマンドの結果は予測できませんが、 右側のコマンドは、上の例ではユーティリティ a について、下の例ではユーティリティ c について報告されます。

```
time a | b | c         { time a } | b | c
a | b | time c         a | b | (time c)
```

オプション | 以下のオプションを指定できます。

-p | 時間情報を以下の形式で標準エラー出力に書き出します。

```
real %f\nuser %f\nsys %f\n < real seconds>, <user seconds>,
<system seconds>
```

オペランド | 以下のオペランドを指定できます。

utility | 呼び出すユーティリティの名前

argument | *utility* を呼び出す際に引数として与えるべき文字列

使用法 | time ユーティリティは、エラーが発生すると終了ステータス 127 を返します。これによりアプリケーションは、 「ユーティリティが見つからなかった」エラーと 「呼び出したユーティリティがエラーで終了した」状態とを 見分けることができます。127 という値を選んだのは、通常この値は他の意味で使われることがないためです。ほとんどのユーティリティは、 小さい値を使って「一般的なエラー状態」を表し、128 より大きな値は、シグナル受信による終了の場合と 区別が困難になりかねないためです。 同じような観点から、ユーティリティは見つかったが実行できなかった、という場合に使う値として 126 が選ばれています。

使用例

例 1 time コマンドの使用

time は、パイプラインまたは一連のコマンドに対して使用すると 便利なことがよくあります。 まずパイプラインやコマンド群を 1 つのファイルに入れ、それをユーティリティとして呼び出します。 それにより time をファイル中のすべてに適用させることができます。

他の方法として、以下のような使い方でも time を複雑なコマンドに適用できます。

```
time sh -c 'complex-command-line'
```

例 2 C シェルに組み込まれている time の使用

以下の 2 つの例では、 csh に組み込まれている time と /usr/bin/time との違いを示しています。 現在使用中のシェルが csh であるとします。

```
example% time find / -name csh.1 -print
/usr/share/man/man1/csh.1
95.0u 692.0s 1:17:52 16% 0+0k 0+0io 0pf+0w
```

time の出力形式に関する詳細は、 csh(1) を参照してください。

```
example% /usr/bin/time find / -name csh.1 -print
/usr/share/man/man1/csh.1
real  1:23:31.5
user     1:33.2
sys     11:28.2
```

環境

time の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、LC_NUMERIC、NLSPATH、PATH についての詳細は、 environ(5) を参照してください。

終了ステータス

utility で指定したユーティリティが呼び出された場合、 そのユーティリティの終了ステータスが time の終了ステータスとなります。呼び出されなかった場合には、time は以下のいずれかの値を返して終了します。

1–125	time ユーティリティの中でエラーが発生した
126	*utility* は見つかったが呼び出すことができなかった
127	*utility* が見つからなかった。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), shell_builtins(1), timex(1), times(2), attributes(5), environ(5)

注意事項

time コマンドをマルチプロセッサ・マシン上で実行した場合、 出力された user と sys の値の合計が real 値を超える場合があります。これは、マルチプロセッサ・マシンでは、 処理を複数のプロセッサに分けて実行することが可能なためです。

時間測定対象のコマンド処理中に割り込みが発生すると、 出力される時間の値に誤差が生じることがあります。

使用上の留意点

経過時間の精度は秒単位までですが、 CPU 時間は 100 分の 1 秒まで計測されます。そのため CPU 時間の合計値は経過時間を最大 1 秒上回る可能性があります。

名前 | mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式 | **mailp** [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明 | `mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション | 次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D`	出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。
`-F`	メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。
`-h`	バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。
`-l`	横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。
`-P` *printer*	`-d` オプションを指定した場合と同じです。
`-s` *subject*	*subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename	読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0`	正常終了
`1`	エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

名前 | times – 現在のシェルの時間使用を報告するためのシェル組み込み関数

sh | **`times`**

ksh | **`times`**

sh | シェルから実行されたプロセスの、 ユーザー時間およびシステム時間の累計値を表示します。

ksh | シェルおよびシェルから実行されたプロセスの、 ユーザー時間およびシステム時間の累計値を表示します。

1 つまたは 2 つのアスタリスク（*）が先頭に付加されている `ksh`(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性 | 次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | `ksh`(1), `sh`(1), `time`(1), `attributes`(5)

名前 | mailp, digestp, filep, newsp, filofaxp, franklinp, timemanp, timesysp – テキストからプリンタ記述言語 (PDL) プリティプリントフィルタである mp へのフロントエンド

形式 | **mailp** [*options*] *filename...*

newsp [*options*] *filename...*

digestp [*options*] *filename...*

filep [*options*] *filename...*

filofaxp [*options*] *filename...*

franklinp [*options*] *filename...*

timemanp [*options*] *filename...*

timesysp [*options*] *filename...*

機能説明 | `mailp` ユーティリティは、`mp`(1) プログラムへのフロントエンドです。異なる名前を使用して、さまざまな `mp` オプションを提供します。

`mailp`	メールメッセージを印刷します。
`newsp`	USENET のニュース記事を印刷します。
`digestp`	USENET ダイジェストファイルを印刷します。
`filep`	通常の ASCII ファイルを印刷します。
`filofaxp`	Filofax のパーソナル・オーガナイザ・フォーマットで印刷します。
`franklinp`	Franklin のパーソナル・オーガナ イ ザ ・フォーマットで印刷します。
`timemanp`	Time Manager のパーソナル・オーガナイザ・フォーマットで印刷します。
`timesysp`	Time/System International のパーソナル・オー ガナイザ・フォーマットで印刷します。

`mailp` (およびそれに関連するプログラム) は、各 *filename* を順に読み取り、内容の清書バージョンを生成します。ファイル名引数が指定されていない場合、`mailp` は標準入力を読み取ります。

`mailp` は、次の 2 つの方法で動作します。`-D` オプションを指定した場合には、X 印刷サーバーのクライアントとして動作し、出力先プリンタの PDL を生成してスプールします。`-d` または `-P` オプションを指定した場合には、PostScript™ の出力を生成してスプールします。

オプション | 次のオプションを指定できます。

`-d` *printer*	指定したプリンタへ出力を送信します。それ以外の場合は、`PRINTER` 環境変数で指定されたプリンタへ出力を送信します。

`-D` 出力先プリンタ用に PDL を生成して、プリンタ用にスプールします。

`-F` メールの上部のヘッダーに、メールの送信先ではなく、送信元が印刷されます。自分専用のプリンタを持っているユーザー向けのオプションです。

`-h` バナーは印刷されません。通常バナーシートに表示される情報が、`mp` バナーでの出力です。

`-l` 横モードで出力します。紙 1 枚にテキストの 2 ページ分が印刷されます。

`-P` *printer* `-d` オプションを指定した場合と同じです。

`-s` *subject* *subject* を、印刷する新しいサブジェクトとして使います。コマンド行で指定した通常の ASCII ファイルを印刷する場合は、サブジェクトはデフォルトでその ASCII ファイル名となります。

オペランド

次のオペランドを指定できます。

filename 読み取るファイルの名前。

環境

`-d`、`-D`、`-P` オプションのいずれも指定されていない場合は、`mailp` は、`PRINTER` 環境変数を使用して `mp`(1) プログラムからの出力の送り先プリンタを決定します。この `PRINTER` 環境変数が見つからない場合、デフォルトで PostScript™ プリンタに送られます。

終了ステータス

次の終了ステータスが返されます。

`0` 正常終了

`1` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWmp

関連項目

`mp`(1), `attributes`(5)

注意事項

`-P` オプションは、 `mp`(1) で使用された場合は出力先プリンタに直接 `PDL` をスプールしますが、`mailp` で使用される場合には、下位互換性を保つために PostScript を生成します。

名前 | touch, settime – ファイルのアクセス日時および更新日時の変更

形式 | **touch** [-acm] [-r *ref_file* | -t *time*] *file...*

touch [-acm] [*date_time*] *file...*

settime [-f *ref_file*] [*date_time*] *file...*

機能説明 | touch ユーティリティは、各ファイルのアクセス日時または更新日時を設定します。指定した *file* が存在しない場合には、*file* が作成されます。

新たに設定する日時は、 -t *time*、 -r *ref_file* に指定したファイルの時刻フィールド、または *date_time* オペランドによって、指定することができます。 これらのオプションやオペランドを 1 つも指定しない場合には、 touch は、 time(2) 関数によって返される現在の日時を設定します。

-a オプションと -m オプションの両方が省略された場合、touch は更新日時とアクセス日時の両方を更新します。

ファイルへの書き込み権を持つが、そのファイルの所有者ではなく、スーパーユーザーでもないユーザーは、ファイルの更新日時とアクセス日時を現在の日時にだけ変更できます。touch で特定の日時に設定しようとするとエラーになります。

settime ユーティリティは、touch -c [*date_time*] *file* と同等です。

touch | touch には、以下のオプションを指定できます。

-a　*file* のアクセス日時だけを変更します。 -m オプションも同時に指定された場合を除き、 更新日時は変更しません。

-c　*file* が示す名前のファイルが存在しない場合、そのファイルを作成しません。 また、この状態についての診断メッセージを出力しません。

-m　*file* の更新日時だけを変更します。-a オプションも同時に指定しなければ、アクセス日時は更新しません。

-r *ref_file*　設定する日時として、現在の日時の代わりに *ref_file* が示すファイルの日時を使用します。

-t *time*　設定する日時として、現在の日時の代わりに *time* が示す値を使用します。 *time* は以下の形式の 10 進数です。

[[*CC*] *YY*] *MMDDhhmm* [.*SS*]

各 2 桁の値は次のような意味を持ちます。

MM　月 [01-12]

DD　日 [01-31]

hh　時 [00-23]

mm　分 [00-59]

CC　西暦年の上 2 桁

YY　西暦年の下 2 桁

SS　秒 [00-61]

CC と *YY* はともに省略可能です。両方とも省略すると、現在の年と見なされます。*YY* を指定して *CC* を省略すると、 *CC* は以下に示す値と見なされます。

YY の値	**CC** のデフォルト
69-99	19
00-38	20
39-68	エラー

結果として得られる日時の値は、環境変数 `TZ` の値によって影響を受けます。日時の値が 1970 年 1 月 1 日 0 時 0 分 0 秒 (グリニッジ標準時) 以前を示しているとき、`touch` はエラーとなりただちに実行を終了します。 許される日時の値の範囲は、1970 年 1 月 1 日 0 時 0 分 0 秒 (グリニッジ標準時) 以後から 2038 年 1 月 18 日までです。

SS の範囲は、 [00-59] ではなく [00-61] です。これはうるう秒を考慮しているためです。 *SS* が 60 または 61、`TZ` により加工された後の時刻の値がうるう秒を表していない場合、結果の時刻値は *SS* を 59 とした時刻の 1 秒後 または 2 秒後となります。 *SS* を省略すると 0 と見なされます。

settime　`settime` には、以下のオプションを指定できます。

`-f` *ref_file*　現在の日時の代わりに、*ref_file* に指定されたファイルの日時を使用します。

オペランド　`touch` および `settime` には、 以下のオペランドを指定できます。

file　日時を変更するファイルのパス名

date_time　現在の日時の代わりに *date_time* を使用します。*date_time* は以下の形式の 10 進数です。

MMDDhhmm [*YY*] 各 2 桁の値は次のような意味を持ちます。

file　日時を変更するファイルのパス名。

MM　月 [01-12]

DD　日 [01-31]

hh　時 [00-23]

mm　分 [00-59]

YY　西暦年の下 2 桁。

YY は省略可能です。省略すると、現在の年と見なされます。 *YY* を指定すると、 年は以下に示す値と見なされます。

YY	対応する年
69-99	1969-1999
00-38	2000-2038
39-68	エラー

`-r` オプションと `-t` オプションの両方を省略し、最低 2 つのオペランドが指定され、そのうちの先頭のオペランドの値が 8 または 10 桁の 10 進数であるとき、その先頭オペランドは *date_time* であると見なされます。それ以外の場合には、先頭オペランドは `file` であると見なされます。

使用法 ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `touch` の動作については、`largefile`(5) を参照してください。

環境 `touch` の実行に影響を与える環境変数 `LANG`、`LC_ALL`、`LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、 `environ`(5) を参照してください。

`TZ` *time* オプション引数 や *date_time* オペランド に適用するタイムゾーンを指定します。

終了ステータス 以下の終了ステータスが返されます。

`0` `touch` の実行が正常終了し、要求されたすべての変更が行われた

`>0` エラーが発生した。 `touch` は、日時の変更を実施できなかったファイルの数を返す

属性 次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 `time`(2), `attributes`(5), `environ`(5), `largefile`(5)

注意事項 BSD 環境に熟知しているユーザーは、 `-f` オプションが `touch` に 受け入れられたのに `-f` オプションは無視されることに気づくでしょう。 ファイル上のアクセス権にかかわりなく、 `touch` は ユーザーが所有するすべてのファイルに対して成功するので、 `-f` オプションは不要です。

名前 | tput – 端末の初期化または terminfo データベースの照会

形式 | **tput** [-T *type*] *capname* [*parm...*]

tput -S <<

機能説明 | tput は、 terminfo データベースを使って、 端末に依存した機能やシェルに渡される情報 (sh(1) を参照) の値を決定するか、 端末をクリア、初期化、またはリセットするか、 または要求された端末タイプの長い名前 (正式な名前) を返します。 *capname* 引数で指定された機能属性が文字列型であれば、 tput は文字列を出力します。 同様に、属性が整数型であれば整数を出力します。 属性がブール型であれば、 tput は単に終了ステータスを設定するだけで何も出力しません。 この場合の終了ステータスの値は、 端末がその機能を持っていれば真を示す 0 であり、持っていなければ偽を示す 1 となります。 標準出力に返された値を使用する前に、 必ず終了ステータス ($?、sh(1) を参照) の値が 0 であることを確認してください。 詳細については後述の「終了ステータス」 の項を参照してください。

オプション | 以下のオプションを指定できます。

-T *type*
: 端末の種類を *type* 引数で指定します。 通常は、デフォルトの種類が TERM 環境変数から得られるので、このオプションは必要ありません。-T オプションを指定すると、シェル変数の LINES と COLUMNS、およびレイヤーサイズは参照されません。

-S
: 1 回の tput の呼び出しで複数の機能指定を可能にするオプションです。 この場合、対象となる機能はコマンド行からではなく 標準入力から tput に渡すことになります。 「使用例」の項にある例を参照してください。 1 行には 1 つの *capname* しか指定できません。 なお、 -S オプションを指定した場合、 ブール型および文字列型の終了ステータス 0 と 1 の意味が変わります。 詳しくは「終了ステータス」の項を参照してください。

オペランド | 以下のオペランドを指定できます。

capname
: terminfo データベースから得られる、機能属性を表します。 機能とそれに対応する *capname* の一覧については、 terminfo(4) を参照してください。

  C ロケールを使ったシステムでは、以下の文字列が オペランドとしてサポートされています。

  clear
  : クリアスクリーンのシーケンスを表示します。

  init
  : terminfo データベースが存在し、 ユーザーの端末用のエントリが存在していれば (前述の -T*type* の説明を参照)、以下のすべての処理が実行されます。

    1. 端末の初期化文字列が存在すれば、それが出力される(is1、is2、is3、if、iprog)
    2. エントリ中の遅延指定 (たとえば復帰改行) が、 tty ドライバに設定される

3. エントリ中の指定に従い、 タブの展開指定を有効または無効にする
4. タブを展開しない場合、 標準タブ (8 文字ごと) が設定される。これら 4 つの処理を行うための情報がエントリ中に存在しなければ、 その該当処理はスキップされますが、 それを表すメッセージは出力されません。

`reset` 初期化文字列を出力する代わりに、 端末のリセット文字列が存在すればそれを出力します (`rs1`、`rs2`、`rs3`、`rf`)。リセット文字列が存在せず、初期化文字列が存在している場合には、 その初期化文字列が出力されます。 それ以外の場合には、 `reset` の動作は `init` と同一です。

`longname` `terminfo` データベースが存在し、 ユーザーの端末用のエントリが存在していれば (前述の `-Ttype` の説明を参照)、端末の長い名前が出力されます。 ここでいう長い名前とは、 `terminfo` データベース中の当該端末の記述における、 先頭行の最後に書かれている名前です (`term`(5) を参照)。

parm 上記 *capname* で指定した属性がパラメタを伴う文字列の場合、この *parm* で指定する値が文字列に代入されます。 数字で表された引数は、すべて数値として属性に渡されます。

使用例

例 **1** `tput` コマンドの使用

最初の例は、環境変数 `TERM` が示す端末タイプに従って端末を初期化する処理です。なお、 `profile`(4) マニュアルページで説明されているように、 このコマンドは (環境変数 `TERM` をエクスポートした後) どのユーザーの `.profile` にも入れておくべきコマンドです。

```
example% tput init
```

次の例は、 AT&T 5620 端末を初期化して、 環境変数 `TERM` が示す端末タイプを置き換えるコマンドです。

```
example% tput -T5620 reset
```

次の例は、画面上の行番号 `0` 列番号 `0` 、つまり左上角 (通常ホームポジションと呼ばれている位置) にカーソルを移動するための シーケンスを送るコマンドです。

```
example% tput cup 0 0
```

次の例は、使用している端末の画面をクリアするシーケンスを エコーさせるコマンドです。

```
example% tput clear
```

次の例は、使用している端末の列の数を表示するコマンドです。

例 1 tput コマンドの使用　(続き)

```
example% tput cols
```

次の例は、 450 型端末の列の数を印刷するコマンドです。

```
example% tput -T450 cols
```

次の例は、使用している端末を、シェル変数 bold を強調モード・シーケンスを開始する値に設定し、 offbold を強調モード・シーケンスを 終了する値に設定するコマンドです。 このコマンドを実行すると、 以下のプロンプトが出力されることがあります。

```
echo "${bold}Please type in your name: ${offbold}\c"
example% bold='tput smso'
example% offbold='tput rmso'
```

次の例は、使用している端末がハードコピー端末であるかどうかを 示す終了ステータスを設定するコマンドです。

```
example% tput hc
```

次の例は、カーソルを 23 行 4 列に移動させるシーケンスを送信するコマンドです。

```
example% tput cup 23 4
```

次の例は、環境変数 TERM が示す端末タイプに対応した長い名前を terminfo データベースから検索して表示するコマンドです。

```
example% tput longname
```

最後の例は、 1 回の呼び出しで複数の機能を処理する tput コマンドです。この例では、画面をクリアし、カーソルを 10 行 10 列の位置に移動させ、ボールド (超高輝度) モードを オンに設定するという一連の処理を実行しています。 処理の記述の終わりは、感嘆符 (!) だけの行で表します。

```
example% tput -S <<!
> clear
> cup 10 10
> bold
> !
```

環境

tput の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、 environ(5) を参照してください。

TERM　端末のタイプを決定します。 この変数が設定されていないかあるいは NULL に設定されていて、 -T オプションが省略された場合、デフォルトの端末タイプ (予測できない) が用いられます。

終了ステータス

以下の終了ステータスが返されます。

0

- *capname* がブール型で -S オプションが省略された場合、真を表す
- *capname* が文字列型で -S オプションが省略された場合、 *capname* が当該端末タイプに対して定義されていることを表す
- *capname* がブール型または文字列型で -S が指定された場合、すべての行が正常に処理されたことを表す
- *capname* が整数型であることを表す

- 要求された文字列が正しく書き出されたことを表す

1

- *capname* がブール型で `-S` オプションが省略された場合、偽を表す
- *capname* が文字列型で `-S` が省略された場合、 *capname* が当該端末タイプに対して定義されていないことを表す

2 使用方法のエラー

3 指定された端末タイプに関して、情報が何も存在しない

4 不正なオペランドが指定された

>4 エラーが発生した

-1 *capname* は `terminfo` データベースに指定された数値ではなかった。たとえば `tput -T450 lines` や `tput -T2621 xmc` などが指定された

ファイル

`/usr/include/curses.h`	`curses`(3CURSES) ヘッダー
`/usr/include/term.h`	`terminfo` ヘッダー
`/usr/lib/tabset/*`	ある端末用のタブ設定を、 その端末に出力する上で適切な形式 (マージンとタブを設定するエスケープシーケンス) で指定。 詳細については `terminfo`(4) の説明中の「タブと初期化」の項を参照
`/usr/share/lib/terminfo/?/*`	コンパイルされた端末記述データベース

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`clear`(1), `sh`(1), `stty`(1), `tabs`(1), `curses`(3CURSES), `profile`(4), `terminfo`(4), `attributes`(5), `environ`(5), `term`(5)

名前 | tr – 文字の変換

形式 | **/usr/bin/tr** [-cs] *string1 string2*

/usr/bin/tr -s -d [-c] *string1*

/usr/bin/tr -ds [-c] *string1 string2*

/usr/xpg4/bin/tr [-cs] *string1 string2*

/usr/xpg4/bin/tr -s -d [-c] *string1*

/usr/xpg4/bin/tr -ds [-c] *string1 string2*

機能説明 | tr ユーティリティは、選択した文字を置き換えるか削除して、 標準入力を標準出力へコピーします。指定されたオプションと *string1* と *string2* の両オペランドにより、文字や単一文字比較要素のコピー中に発生する変換を制御します。

オプション | 以下のオプションを指定できます。

-c　　*string1* で指定された文字のセットを補完します。

-d　　*string1* で指定された文字を入力中からすべて削除します。

-s　　繰り返して現れた文字を 1 個の文字に置き換えます。

-d オプションを省略すると、以下の処理が行われます。

- *string1* で指定した配列中の文字が入力中で見つかった場合、 *string2* で指定した配列中の同じ位置にある文字に置き換えます。 *string2* が示す配列が *string1* が示す配列より短い場合、処理の結果は予測できません。
- -c オプションを指定すると、 *string1* で指定された文字の補完集合 (LC_CTYPE の現在の設定値で定義されている現文字セット内のすべての文字から、 *string1* オペランドで実際に指定された文字を除いたもの) が、 LC_COLLATE の現在の設定値で定義されている照合順序に従って、 昇順で配列に置かれます。
- なお文字クラス表現や同等クラス表現により指定される文字の順序は 定義されていないため、そのような表現を使用するのは、 複数の文字を同一の文字にマップする場合に限るべきです。 ただし前述のように、大文字と小文字との間の変換は例外です。

-d オプションを指定すると、以下の処理が行われます。

- *string1* で指定した配列中にある入力文字は削除されます。
- -c オプションも一緒に指定された場合、 *string1* で指定した文字を除くすべての文字が削除されます。 -s オプションが指定されていなければ、 *string2* の内容は無視されます。
- -d と -s の 2 つのオプションをともに指定する場合、削除用の *string1* と圧縮用の *string2* を両方とも指定しなければなりません。 この場合、 同じ文字列を指定することはできません。

-s オプションを指定すると、 何らかの削除または変換発生後に同じ文字が連続して出現し、 その文字が最終オペランドで指定した配列中に存在していれば、 一連の文字はその文字 1 個に置き換えられます。 最終オペランドとして、次の例に示すような文字クラスが指定されたとします。

```
tr -s '[:space:]'
```

この場合、最終オペランドの配列には、 その文字クラスのすべての文字が含まれることになります。 ただし、次のような大文字と小文字の変換が指定された場合には 最終オペランドの配列には、 toupper と tolower の対のうち 2 番目の文字として定義されている方の文字だけが含まれます (toupper(3C) と tolower(3C) を参照)。

```
tr -s '[:upper:]' '[:lower:]'
```

string1 または *string2* に空の文字列を指定した場合、処理の結果は予測できません。

オペランド

以下のオペランドを指定できます。

string1

string2

変換を制御する文字列です。 各文字列は、変換に使用される文字配列に置き換えられる、一群の文字を表します。

オペランド *string1* と *string2* により 2 個の文字配列を定義します。以下に示す要素を使って、文字や 単一文字照合要素を指定できます。これらの要素を使った結果、複数文字 照合要素が得られた場合には、 tr は配列からその複数文字要素を除きます。 このとき、診断メッセージは発行しません。

character

以下の説明文中に現れない文字は、その文字自体を表します。

\ *octal*

8 進数のシーケンスを使って、具体的なコード値を示す文字を表すことができます。 8 進数シーケンスは、バックスラッシュの後に最大 1 、 2 、または 3 桁の 8 進数 (01234567) を付加したものです。 ここで指定したシーケンスにより、 1 、 2 、または 3 桁の 8 進整数で表される文字が、配列中に配置されます。 複数バイト文字は、 この種のエスケープシーケンスがいくつか連続したものを必要とします。 そのとき、各バイトの先頭に \ が必要です。

\ *character*

バックスラッシュ付きのエスケープシーケンスとして、 \a、 \b、 \f、 \n、 \r、 \t、 \v がサポートされています。 バックスラッシュの後に、これ以外の文字で 8 進数字でもない文字を指定した場合、結果は予測できません。

/usr/xpg4/bin/tr

c-c

/usr/bin/tr

[*c-c*]

現在の LC_COLLATE ロケールカテゴリの設定による定義に従い、照合要素の範囲を指定します。 ここで指定する 2 つの文字も範囲に含まれます。最初の文字は、現在の照合順序において、 2 番目の文字の前に位置していなければなりません。 この範囲に含まれる文字または照合要素は、 昇順で配列中に置かれます。

[:*class*:] 現在の LC_CTYPE ロケールカテゴリの設定値に従い、 指定された文字クラスに属するすべての文字を表します。 以下の文字クラス名が *string1* として指定できます。

```
alnum  blank  digit  lower  punct  upper
alpha  cntrl  graph  print  space
xdigit
```

さらに、 [:*name*:] 形式の文字クラス式も指定できます。ただし、 *name* が LC_CTYPE カテゴリ内で charclass 定義を与えられているようなロケールにおいてのみです。

注*:* /usr/bin/tr が文字クラス式をサポートするのは、単一バイトのロケールにおいてのみです。その他のロケールで文字クラス式をサポートするには、/usr/xpg4/bin/tr を使用してください。

-d と -s の両オプションが指定されていれば、どんな文字クラス名でも *string2* に指定できます。両オプションが指定されなければ、 lower または upper だけが *string2* として指定できます。ただし、これに対応する文字クラス upper または lower が *string1* 中の同等の位置に指定された場合だけです。 このような指定は、大文字と小文字間の変換要求と見なされます。 [:lower:] が *string1* 中に現れ、 [:upper:] が *string2* 中に現れた場合、現在のロケールの LC_CTYPE カテゴリ中の toupper マッピングから得られる文字が、配列に含まれることになります。 [:upper:] が *string1* 中に現れ、 [:lower:] が *string2* 中に現れた場合、現在のロケールの LC_CTYPE カテゴリ中の tolower マッピングから得られる文字が、配列に含まれることになります。 各マッピングの対のうち、最初の文字が *string1* の配列に、 2 番目の文字が *string2* の配列に置かれます。配列内での位置は同じになります。

大文字と小文字間の変換の場合を除き、 文字クラス式で指定された文字を配列中に置く順序は決まっていません。

class に指定した文字が現在のロケール用の正しい文字クラスを表していない場合、 処理の結果は予測できません。

[=*equiv*=] 現在の LC_COLLATE ロケールカテゴリの設定による定義に従い、指定された文字クラスに属するすべての文字を表します。 *equiv* と同じ同等クラスに属するすべての文字または照合要素を表します。 同等クラス式は *strint1* に指定できます。また -d と -s の両オプションを指定した場合には、 *string2* にも指定できます。この同等クラスに属する文字を配列に置く順序は、 決まっていません。

[*x***n*] *x* が示す文字の *n* 回の連続発生を表します。この式は複数の文字を 1 個の文字にマップするために使用するので、 *string2* にだけ指定できます。 *n* を省略するか、または 0 を指定した場合、 *string2*

に基づくシーケンスを *string1* に基づくシーケンスに拡張するのに十分な大きさ、と見なされます。 *n* の先頭の数字が `0` のとき、この数値は 8 進数と見なされます。 `0` でなければ 10 進数と見なされます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `tr` の動作については、`largefile`(5) を参照してください。

使用例

例 **1** 文字リストを作成する

以下の例は、 *file1* 中にあるすべての単語を 1 行に 1 個の形式で *file2* に出力します。ここで言う単語とは、最大文字列を表します。

```
tr −cs "[:alpha:]" "[\n*]" <file1 >file2
```

例 **2** 文字を変換する

次の例は、 `file1` 中のすべての小文字を大文字に変換し、その結果を標準出力に 書き出します。

```
tr "[:lower:]" "[:upper:]" <file1
```

なお、対応する XPG3 での例に示されている警告は、 この場合有効ではありません。この大文字・小文字変換は、 `tolower` と `toupper` を使った特殊なケースで、ロケールが正しく定義されているとき、 マッピングが正しく行われることを確認するものです。

例 **3** 同じ文字を識別する

次の例は、同等クラスを使って `file1` 中でアクセント記号付きのベース文字 `e` を認識し、その記号を取り除いて `file2` に書き出します。

```
tr "[=e=]" e <file1 >file2
```

環境

`tr` の実行に影響を与える環境変数 `LC_COLLATE`、`LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、`environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 入力データはすべて正常に処理された

`>0` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

/usr/bin/tr

属性タイプ	属性値
使用条件	SUNWcsu
CSI	未対応

/usr/xpg4/bin/tr

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目 ed(1), sed(1), sh(1), tolower(3C), toupper(3C), ascii(5), attributes(5), environ(5), largefile(5), XPG4(5)

注意事項 以前のバージョンとは違って、 /usr/xpg4/bin/tr は入力ストリーム中の NUL 文字を正しく処理します。 NUL 文字は tr -d '\000' によって取り除くことができます。

名前 | trap, onintr – (ハードウェア) シグナルに応答するためのシェル組み込み関数

sh | **trap** [*argument n* [*n2*...]]

csh | **onintr** [-| *label*]

ksh | ***trap** [*arg sig* [*sig2*...]]

sh | *argument* が示す trap コマンドを、シェルが数値形式または 名前形式で指定されているシグナル（*n*） を受信した時に読み取り、実行します。なお *argument* は、トラップ設定時とトラップ取り出し時に 1 度ずつ検索されます。 トラップコマンドは、シグナル番号または 対応するシグナル名の順序で実行されます。 現在のシェルへの入力時に無視されたシグナルにトラップを 設定しようとしても無効となります。 シグナル 11（メモリフォルト） にトラップを指定しようとすると、 エラーになります。 *argument* を省略すると、 *n* のトラップはすべてその元の値に再設定されます。 *argument* が NULL 文字列の場合、 シェルおよびシェルが呼び出すコマンドは、 このシグナルを無視します。 *n* が 0 の場合、 *argument* が示すコマンドはシェル終了時に実行されます。 引数なしの trap コマンドは、コマンドの一覧を 各々が対応しているシグナル番号とともに表示します。

csh | onintr は割り込み時のシェルの動作を制御します。 引数を指定しないと、onintr は割り込み時にはデフォルトの動作を実行します （シェルはシェルスクリプトを終了して、 端末のコマンド入力レベルに戻ります）。- 引数を指定すると、シェルはすべての割り込みを無視します。*label* 引数を指定すると、割り込みを受信するか 割り込みのために子プロセスが終了したときに、シェルは goto *label* を実行します。

ksh | trap は *arg* を *sig* が示すシグナルをシェルが受信したときに読み取られ、 実行されるコマンドとして使用します。 なお *arg* は、 トラップ設定時と トラップ取り出し時に 1 度ずつ検索されます。各 *sig* は、 数値またはシグナルの名前です。trap コマンドは、 シグナル番号の順序で実行されます。 現在のシェルで無視されているシグナルにトラップを 設定しようとしても無効となります。*arg* を省略するか、 または - と指定する場合、各 *sig* 用のトラップはすべてその元の値に再設定されます。*arg* が NULL 文字列（"" などの空の文字列）の場合、 シェルおよびシェルが呼び出すコマンドは、このシグナルを無視します。 *sig* が ERR の場合、 コマンドが 0 以外の終了状態で終わると必ず *arg* が実行されます。*sig* が DEBUG の場合、 各コマンドの後に *arg* が実行されます。 *sig* が 0 または EXIT で、 トラップが関数の外側で設定されている場合、シェルの終了時に *arg* が示すコマンドが実行されます。 引数なしの trap コマンドは、 コマンドの一覧を 各々が対応しているシグナル番号とともに表示します。

1 つまたは 2 つのアスタリスク（*）が先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。つまり、 チルド置換は = 符号の後に実行され、ワード分割とファイル名生成は実行されません。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), exit(1), ksh(1), sh(1), attributes(5)

名前 | troff – ドキュメントのタイプセットと清書

形式 | **troff** [-a] [-f] [-F*dir*] [-i] [-m*name*] [-n*N*] [-o*list*] [-r*aN*] [-s*N*] [-T*dest*] [-u*N*] [-z] [*filename...*]

機能説明 | troff は、 *filename* 引数で指定されたファイル群中のテキストを、 タイプセット用またはレーザープリンタでの印刷用に 清書します。 troff への入力ファイルの中身は、 清書用の要求とマクロを含んだテキストです。*filename* 引数が省略された場合、 troff は標準入力から読み込みます。 複数のファイル名も指定可能で、そのうちの 1 つとして標準入力を指定したければ、該当する位置に ハイフン (–) を記述します。

troff の出力は、通常、 印刷可能な PostScript ファイルを作成するために dpost(1) を通されます (「使用例」の項を参照)。

オプション | 以下のオプションを指定できます。 任意の順序で指定できますが、 すべてのオプションは先頭の *filename* より前に指定してください。

-a
: ASCII の近似形式で清書した出力を 標準出力に書き出します。(最近はほとんど使用されない /usr/bin/ta コマンドを使うと、ASCII の未完版も通常端末上に出力されるので注意してください。)

-f
: 最終ページを出力したあと、トレーラページを印刷しないか、 またはポストプロセッサが装置の制御権を放棄しないようにします。

-F *dir*
: フォント幅や端末テーブルを検索するディレクトリとして、 システムのデフォルトディレクトリの代わりに *dir* を使用します。

-i
: 指定した入力ファイルをすべて処理したあと、 標準入力を読み込みます。

-m *name*
: 指定した入力ファイルのあとに、マクロファイル /usr/share/lib/tmac/*name* を追加します。 なお、マクロパッケージへの参照の多くは、 通常、 名前の先頭に *m* を含みます。たとえば man(5) マクロは、 /usr/share/lib/tmac/an 中に存在しています。 マクロディレクトリを変更したければ、 環境変数 TROFFMACS の値として希望するパスを設定してください。 なお、パス名の最後に必ずスラッシュ (/) を付加するよう注意してください。

-n *N*
: 出力される先頭ページのページ番号を *N* に設定します。

-o *list*
: 特定のページだけを印刷出力したい場合、 *list* 引数を使ってそのページ番号のリストを指定します。 リスト中では項目をコンマで区切って記述し、 各項目として単一のページ番号または番号の範囲を記述できます。 範囲指定には 3 つの形式があり、 *N–M* はページ番号 *N* から *M* までを表し、先頭項目として指定する *–N* は先頭ページからページ *N* までを表し、最終項目として指定する *N–* はページ *N* から最終ページまでを表します。

-q	nroff では静寂モードを示すオプションです。troff では無視されます。
-r *aN*	レジスタ *a* の値として *N* を設定します。*a* は常に 1 文字の名前です。
-s *N*	*N* ページを出力するごとに 写真植字機を停止させます。 機器の種類によっては、 troff がトレーラページを出力するのでカセットを 交換することができます。 機器のスタートボタンを押せば 印刷が再開できます。
-T *dest*	*dest* 引数で示す印刷機器用の出力を生成します。 *dest* には次の値を指定できます。
	post PostScript プリンタを表します。 これがデフォルト値です。 -T オプションで出力する場合、 完全な出力を行うために、 dpost(1) を通してから PostScript プリンタへ送る必要があります。
	aps Autologic APS–5 を表します。
-u *N*	ポジション 3 にマウントされているフォント用の ボールド（太文字）化ファクタ値を *N* に設定します。 *N* 引数を省略すると、ボールド化ファクタ値は 0 に設定されます。
-z	清書した出力結果を印刷しません。 つまり診断メッセージや .tm 要求を使って出力されたメッセージだけが印刷されます。

オペランド

以下にオペランドを示します。

filename	troff によって処理されるテキストを含むファイル

使用例

例 **1** troff の使用

以下に、清書要件とマクロでコーディングされた入力テキストファイル mytext の印刷方法の例を示します。 入力ファイルは式とテーブルを含んでおり、 tbl(1) と eqn(1) を通してから、 ms マクロを使用して troff によって清書し、 dpost(1) によって処理したあと、lp(1) で印刷する必要があります。

```
tbl mytext | eqn | troff -ms | dpost | lp
```

ファイル

/tmp/trtmp	一時ファイル
/usr/share/lib/tmac/*	標準マクロファイル
/usr/lib/font/*	マウントされている代替 troff フォント用のフォント幅テーブル
/usr/share/lib/nterm/*	nroff 用の端末駆動テーブル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc

関連項目 checknr(1), col(1), dpost(1), eqn(1), lp(1), man(1), nroff(1), tbl(1), attributes(5), man(5), me(5), ms(5)

注意事項 troff は 7 ビット ASCII に基づいて設計されているので、 8 ビットクリーンではありません。

以前のマニュアルに書かれていた、数値レジスタ yr が「現在の年号の末尾の 2 桁」を示す、という記述は正しくありません。yr は、実際には 1900 年以降の数値を対象にします。2099 年までの、現在の年号の末尾の 2 桁を正しく獲得するには、以下の文字列レジスタ yr の定義をドキュメントに含めると、 2 桁の年号を表示するために使用できます。yr の代わりに、別の 1 つまたは 2 つの文字をレジスタ名として使用できます。

```
.\" definition of new string register yy--last two digits of year
.\" use yr (# of years since 1900) if it is < 100
.ie \n(yr<100 .ds yy \n(yr
.el \{              .\" else, subtract 100 from yr, store in ny
.nr ny \n(yr-100
.ie \n(ny>9 \{      .\" use ny if it is two digits
.ds yy \n(ny
.\" remove temporary number register ny
.rr ny \}
.el \{.ds yy 0
.\" if ny is one digit, append it to 0
.as yy \n(ny
.rr ny \} \}
```

名前 | truss – システムコールとシグナルの追跡

形式 | **truss** [-fcaeildD] [- [tTvx] [!] *syscall* ,...] [- [sS] [!] *signal* ,...] [- [mM] [!] *fault* ,...] [- [rw] [!] *fd* ,...] [- [uU] [!] *lib* ,... : [:] [!] *func* ,...] [-o *outfile*] *command* | -p *pid*...

機能説明 | truss ユーティリティーは指定されたコマンドを実行し、それ自体が実行するシステムコール、受け取ったシグナル、および検出したマシン障害の追跡情報を生成します。追跡出力の各行には、障害またはシグナルの名前、あるいはシステムコール名とその引数および戻り値が示されます。システムコールの引数は、可能なかぎの、関連するシステムヘッダー内の定義に従って記号で表示されます (パス名へのポインタ引数の場合、ポイント先の文字列が表示される)。エラーが発生した場合の戻り値は、intro(3) のマニュアルページに説明されているエラーコード名を使用して報告されます。

truss は、-u オプションにより、追跡するプロセスによって実行されたユーザーレベルの関数呼び出しの開始および終了時の追跡情報も生成します。この場合、入れ子レベルを示すためにインデントが行われます。

オプション | 以下のオプションを指定できます。リスト引数を受け付けるオプションでは、リスト内のすべての可能なメンバーを指定する短縮形として名前 all を使用できます。リストが ! で始まるオプションは否定を意味します (たとえば、追跡せずに除外するなど)。同じオプションを複数回指定できます。リスト内の同じ名前に対しては、あとから指定されたオプションが先に指定されているオプション (リスト内で左側にあるもの) を無効にします。

-p
truss に指定される *command* 引数を、実行されるコマンドとしてではなく、既存のプロセス (ps(1) のマニュアルページを参照) のプロセス ID のリストとして解釈します。プロセスのユーザー ID とグループ ID が実行するユーザーの ID と一致するか、あるいはユーザーが特権ユーザーである場合、truss は各プロセスを制御し、それらの追跡を開始します。プロセスの指定は、/proc ディレクトリ内に名前を指定することによっても行えます (例: /proc/12345)。

-f
fork() または vfork() によって作成されたすべての子プロセスを追跡し、それらのシグナル、障害、およびシステムコールを追跡出力に含めます。通常は、最初のコマンドまたはプロセスだけが追跡されます。-f を指定すると、追跡出力の各行にプロセス ID が表示され、どのプロセスがシステムコールを実行したか、あるいはどのプロセスがシグナルを受信したかが示されます。

-c
追跡情報を 1 行ずつ表示するのではなく、追跡されたシステムコール、障害、およびシグナルをカウントします。追跡されたコマンドが終了するか、あるいは truss が割り込まれたときに、要約レポートが生成されます。-f も指定すると、子プロセスについて追跡されたシステムコール、障害、およびシグナルもすべてカウントに含められます。

-a
各 exec() システムコールに渡される引数文字列を表示します。

`-e`
各 `exec()` システムコールに渡される環境文字列を表示します。

`-i`
割り込み可能な休眠状態のシステムコールを表示しません。端末デバイスまたはパイプ上での `open()` や `read()` など、特定のシステムコールは不確定時間に休眠でき、割り込み可能です。一般に、システムコールが 1 秒を超えてそのような休眠状態になっている場合、`truss` はそれらを報告します。システムコールは完了時に再度、報告されます。`-i` オプションは、そのようなシステムコールを、完了時のみの 1 度だけ報告します。

`-l`
追跡出力の各行に対応する軽量プロセス (LWP) の ID を含めます。`-f` も指定すると、プロセス ID と *LWP* ID の両方を含めます。

`-d`
追跡出力の各行にタイムスタンプを含めます。タイムスタンプは、*seconds* . *fraction* という形式で行頭に示されます。これは、追跡の開始時からの経過秒数を示したものです。追跡出力の最初の行には、epoch (`time`(2) のマニュアルページを参照) 以後の経過秒数として、個々のタイムスタンプ測定の基点となるベースタイムおよび、開始時の日付が表示されます。報告される時間は、当該イベントが発生した時間です。どのシステムコールについても、イベントはシステムコールの終了時であり、システムコールの開始時ではありません。

`-D`
追跡出力の各行にデルタタイムを含めます。*seconds* . *fraction* という形式で示されるこの値は、LWP が呼び出したイベントの前回の報告以降にそのイベントが次に発生した時点までの経過時間を示します。システムコールの場合、これはシステムコール内で経過した時間ではありません。

`-t` [`!`]*syscall*`,...`
追跡または除外するシステムコールを指定します。コンマで区切ったリストに指定されたシステムコールの追跡が行われます。リストが `!` で始まる場合、指定したシステムコールが追跡出力から除外されます。デフォルトは `-tall` です。

`-T` [`!`]*syscall*`,...`
プロセスを停止するシステムコールを指定します。指定されたシステムコールが、`-t` によって指定されるセットに追加されます。指定されたシステムコールの 1 つが見つかると、`truss` はプロセスを停止したままにし、終了します。つまり、`truss` はプロセスを解放して実行を終了しますが、当該システムコールの完了時にそのプロセスを停止状態のままにします。これにより、停止したプロセスにデバッガなどのプロセス検査ツール (`proc`(1) のマニュアルページを参照) を適用できるようになります。追跡を継続するには、同じオプションまたは異なるオプションを指定して、停止されたプロセスに `truss` を適用し直します。デフォルトは `-T!all` です。

この方法で停止されたままになったプロセスは、アプリケーション `kill -CONT` によって再開することはできません。これは、停止シグナル (`signal`(3HEAD) のマニュアルページを参照) のデフォルトアクションによってではなく、`/proc` を介した イベント上でこのプロセスが停止されているためです。停止中のプロセスを再実行するように設定するには、`proc`(1) のマニュアルページで説明されている `prun`(1) コマンドを使用できます。

`-v` [`!`]*syscall*, . . .
詳細表示。指定されたシステムコールに対してアドレスで渡された任意の構造体の内容を表示します (`-t` による追跡が行われた場合)。入力した値とオペレーティングシステムによって返される値が示されます。入力と出力の両方に使用されるフィールドについては、出力値だけが示されます。デフォルトは `-v!all` です。

`-x` [`!`]*syscall*, . . .
指定されたシステムコールの引数を raw 形式で表示します (`-t` による追跡が行われた場合)。これは、通常、記号表示ではなく 16 進表示であり、raw ビットのままの方が良いと考えるハッカーのためのものです。

`-s` [`!`]*signal*, . . .
追跡または除外するシグナルを指定します。コンマで区切ったリストに指定したシグナルを追跡します。シグナルが無視される (ブロックされてない) 場合でも、追跡出力には、指定された各シグナルの受信が示されます (ブロックされているシグナルはブロックが解放されるまで受信されない)。シグナルは、名前または番号で指定できます (`<sys/signal.h>` を参照)。リストが `!` で始まる場合、指定されたシグナルは追跡出力から除外されます。デフォルトは `-sall` です。

`-S` [`!`]*signal*, . . .
プロセスを停止するシグナルを指定します。指定されたシグナルは、`-s` で指定されるセットに追加されます。指定されたシグナルのどれかが受信された場合、`truss` はプロセスを停止したままにし、終了します (`-T` オプションを参照)。デフォルトは `-S!all` です。

`-m` [`!`]*fault*, . . .
追跡または除外するマシン障害を指定します。コンマで区切ったリストに指定した障害が追跡されます。障害は、名前または番号で指定できます (`<sys/fault.h>` を参照)。リストが `!` で始まる場合、指定された障害は追跡出力から除外されます。デフォルトは `-mall -m!fltpage` です。

`-M` [`!`]*fault*, . . .
プロセスを停止するマシン障害を指定します。指定した障害は、`-m` で指定されるセットに追加されます。指定された障害の 1 つが見つかると、`truss` はプロセスを停止したままにし、終了します (`-T` オプションを参照)。デフォルトは `-M!all` です。

`-r` [`!`]*fd*, . . .
指定したファイル記述子の `read()` ごとに、入出力バッファの全内容を表示します。出力は行当たり 32 バイトに整形され、各バイトは ASCII 文字 (先頭に 1 個のブランクが入る)、または、水平タブ <\t> や復帰改行 <\n> などの制御文字のために 2 文字の C 言語エスケープシーケンスとして表示されます。ただし、ASCII 解釈

が不可能な場合は、2 文字の 16 進表現となります (-r が指定されない場合でも追跡された各 read() の入出力バッファの最初の 12 バイトは表示される)。デフォルトは -r!all です。

-w [!]*fd*, . . .
指定したファイル記述子の write() ごとに入出力バッファの内容を表示します (-r オプションを参照)。デフォルトは -w!all です。

-u [!]*lib*, . . . :[:][!]*func*, . . .
ユーザーレベルの関数呼び出しを追跡します。*lib*, . . は、動的なライブラリ名 (.so.*n* 接尾辞を除く) をコンマで区切ったリストです。*func*, . . は、関数名をコンマで区切ったリストです。どちらの場合でも、名前の表現にメタ文字 *、?、[] を使用できます。これらのメタ文字の指定は sh(1) における指定と同じ意味を持ちますが、ファイルに対してではなくライブラリ名または関数名に対して使用されることになります。ライブラリまたは関数のリストを空にすると、デフォルトで * が使用され、ライブラリ内のすべてのライブラリまたは関数が追跡されます。リストの先頭に ! を付けると、追跡から除外されるライブラリまたは関数の名前を指定したことになります。1 つのライブラリを除外すると、そのライブラリ内のすべての関数が除外されます。つまり、ライブラリ除外リストのあとに続く関数リストは無視されます。

関数リストとライブラリリストを分離する 1 つの : は、ライブラリの外部から、それらのライブラリに対する呼び出しは追跡しますが、ライブラリ内部の他の関数からの呼び出しは除外することを意味します。2 つの : : は、呼び出し元に関係なくすべての呼び出しを追跡することを意味します。

ライブラリのパターンには、正確な一致がないかぎり、実行可能ファイルと動的リンカーのいずれとも対応付けはしません (l* は ld.so.1 に対応付けられない)。これらのオブジェクトのどちらかに含まれる関数を追跡するには、 truss -u a.out -u ld .. のように名前を明確に指定する必要があります。a.out はこの目的で使用されるリテラル名であり、実行可能ファイルの名前を意味するわけではありません。a.out 関数呼び出しを追跡すると、すべての呼び出しが暗黙に追跡されます (デフォルトは : :)。

-u オプションは複数回指定することが可能で、この場合左から順に受け付けられます。プロセスが -lthread にリンクしている場合は、呼び出しの追跡出力に関数呼び出しを行ったスレッドの ID が含められます。truss は、関数名を見つけるために各ライブラリ内の動的シンボルテーブルを検索するとともに、ストリップされてなければ標準のシンボルテーブルも検索します。

-U [!]*lib*, . . . :[:][!]*func*, . . .
プロセスを停止するユーザーレベルの関数呼び出しを指定します。指定される関数は、-u で指定されるセットに追加されます。指定された関数の 1 つが呼び出されると、truss はプロセスを停止したままにし、終了します (-T オプションを参照)。

-o *outfile*
追跡出力に使用されるファイル。デフォルトでは、出力は標準エラー出力に送られます。

-t、-T、-v、および -x オプションが受け付けるシステムコール名については、『*man pages section 2: System Calls*』を参照してください。システムコール番号も指定できます。

指定したコマンドを起動及び追跡するのに truss を使用する場合、-o オプションを使用するか、あるいは標準エラー出力を端末以外のファイルにリダイレクトすると、truss は ハングアップ、割り込みシグナル、および終了シグナルを無視して動作します。これにより、端末からの割り込みシグナルと終了シグナルを受け取る対話型プログラムの追跡が容易になります。

追跡出力を端末に転送したままにした場合、あるいは、既存のプロセスを追跡する (-p オプション) 場合、truss は追跡したすべてのプロセスを解放して処理を終了するので、ハングアップ、割り込み、終了の各シグナルに応答します。これにより、ユーザーは過度の追跡出力を抑制でき、既存のプロセスを解放できるようになります。解放されたプロセスは、なんの影響も受けず、それまでどおりの通常の処理を継続します。

使用例

例 **1** コマンドを追跡する

この例は、端末上の find(1) コマンドの追跡情報を生成します。

```
example$ find, -print >find.out
```

例 **2** 一般的なシステムコールを追跡する

オープン、クローズ、読み取り、書き込みの各システムコールの追跡情報だけを表示するには、次のように指定します。

```
example$ truss -t open,close,read,write find  .  -print >find.out
```

例 **3** シェルスクリプトを追跡する

この例は、ファイル truss.out 上の spell(1) コマンドの追跡情報を生成します。

```
example$ truss -f -o truss.out spell document
```

spell はシェルスクリプトであるため、シェルだけでなくシェルによって生成されたプロセスも追跡するためには -f フラグが必要です (spell スクリプトは 8 つのプロセスのパイプラインを実行する)。

例 **4** 出力を簡潔にする

出力の 97% が lseek()、read()、および write() システムコールの追跡情報である冗長な例を示します。

```
example$ truss nroff -mm document >nroff.out
```

出力を簡潔にまとめるには次のように指定します。

```
example$ truss -t ! lseek,read,write nroff -mm document >nroff.out
```

例 **5** C ライブラリの外部からのライブラリ呼び出しを追跡する

この例は、C ライブラリの外部から C ライブラリ内の任意の関数に対して行われるユーザーレベルの呼び出しをすべて追跡します。

例 5 C ライブラリの外部からのライブラリ呼び出しを追跡する　(続き)

```
example$ truss -u libc  . . .
```

例 6 C ライブラリ内からのライブラリ呼び出しを追跡する

この例では、C ライブラリからその C ライブラリ自体の関数に対して行われる呼び出しが含められます。

```
example$ truss -u libc : :  . . .
```

例 7 C ライブラリ以外のライブラリ呼び出しを追跡する

この例は、C ライブラリ以外のすべてのライブラリに対して行われるユーザーレベルの呼び出しをすべて追跡します。

```
example$ truss -u '*' -u !libc  . . .
```

例 8 pritf および scanf 関数呼び出しを追跡する

この例は、C ライブラリに含まれる printf および scanf ファミリ内の関数に対するユーザーレベルの呼び出しをすべて追跡します。

```
example$ truss -u 'libc : *printf,*scanf'  . . .
```

例 9 ユーザーレベルの関数呼び出しをすべて追跡する

この例は、任意の場所から任意の場所に対して行われるユーザーレベルの関数呼び出しをすべて追跡します。

```
example$ truss -u a.out -u ld : : -u : :  . . .
```

例 10 システムコールの追跡結果を詳細に表示する

この例は、プロセス #1 の `init`(1M) システムコールのアクティビティを追跡し、詳細な追跡情報を表示します (特権ユーザーがこのコマンドを実行できます)。

```
example# truss -p -v all 1
```

`truss` に割り込みを行うと、`init` は通常の動作に戻ります。

ファイル

`/proc/*`　　プロセスファイル

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWtoo (32-bit)

属性タイプ	属性値
	SUNWtoox (64-bit)

関連項目

date(1), find(1), proc(1), ps(1), sh(1), spell(1), init (1M), intro(3), exec(2), fork(2), lseek(2), open(2), read(2), time(2), vfork(2), write(2), ctime(3C), threads(3THR), proc(4), attributes(5), signal(3HEAD)

man pages section 2: System Calls

注意事項

マニュアルページのセクションでのシステムコールで説明されているシステムコールの中には、実際のオペレーティングシステムインタフェースとは異なるものがあります。追跡情報が、このマニュアルページの説明と若干異なる場合もあります。

マシン障害 (ページフォルトは除く) が発生するたびに、障害の原因となった LWP にシグナルが送られます。シグナルがブロックされない場合、各マシン障害 (ページフォルトを除く) が通知された直後に、受信シグナルが通知されます。

オペレーティングシステムは、プロセスの追跡に一定のセキュリティ制限を課します。具体的には、ユーザーは、自身が読み取りできないオブジェクトファイル (a.out) を持つコマンドを追跡することはできません。set-uid と set-gid をもつコマンドは、特権ユーザー以外は追跡できません。特権ユーザーが実行する場合を除き、truss は set-id または読み取り不可能なオブジェクトファイルの exec() を実行するプロセスを制御できません。このようなプロセスは、truss からは独立して、exec() ポイントで通常どおり処理を継続します。

ほかの制御プロセスとの衝突を防ぐため、truss は、/proc インタフェースを介してほかのプロセスによって制御されているプロセスを追跡しません。このため、truss を proc(4) ベースのデバッガだけでなく、それ自身の別のインスタンスにも適用できます。

8 カラムごとに標準のタブストップが設定されていると仮定して、追跡出力にはタブ文字が含まれます。

複数のプロセスまたはマルチスレッドプロセス (複数の LWP) を含むプロセス) の追跡出力は、厳密に時間順には生成されません。たとえば、パイプにおける read() は、対応する write() よりも前に報告される場合があります。しかし、個々の LWP (従来型のプロセスには 1 つしか含まれない) について出力は、厳密に時間順に行なわれます。

複数のプロセスを追跡する場合、truss は追跡対象のプロセスごとに 1 つの制御プロセスとして動作します。前述の spell コマンドの例の場合、spell 自身が 9 つのプロセススロット (シェル用に 1 つ、8 メンバーを持つパイプライン用に 8 つ) を使用し、truss がさらに 9 つのプロセスを追加するため、プロセススロットは合計で 18 になります。

-v オプションでは、すべてのシステムコールで渡すことができるあらゆる構造体を表示できるわけではありません。

名前 | tty – ユーザーの端末名の応答

形式 | **tty** [-l] [-s]

機能説明 | tty ユーティリティは、標準入力としてオープンしている 端末名を標準出力に書き出します。 使用される名前は ttyname(3C) 関数によって返される文字列と同じです。

オプション | 以下にオプションを示します。

-l　ユーザー端末がアクティブな状態の同期回線につながっている場合は、ユーザー端末に接続されている同期回線番号を出力します。

-s　端末パス名は出力しませんが、終了ステータスをテストできるようにします。

環境環境 | tty の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　標準入力は端末である

1　標準入力は端末ではない

>1　エラーが発生した

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目 | isatty(3C), ttyname(3C), attributes(5), environ(5)

診断 | not on an active synchronous line
標準入力は同期端末ではなく、 -l が指定されています。

not a tty
標準入力は端末ではなく、 -s は指定されていません。

注意事項 | -s オプションは終了ステータスが必要な場合のみ 有効です。有効なパス名を表示することはできません。 移植性の必要なアプリケーションには test -t を使用してください。

名前 type – コマンドのタイプの検査

形式 **type** *name...*

機能説明 type ユーティリティは、オペランドに指定された各 *name* が、コマンドとして使用された場合どのように解釈されるかを チェックします。 具体的には、各々の *name* オペランドがシェルの組み込みコマンド、関数、別名、 ハッシュドコマンド、キーワードのいずれであるかを表示し、 さらに （該当するものについては） オペランドのパス名を表示します。

type というシェルの組み込みコマンドもあります。 機能は type ユーティリティと同様です。

オペランド 以下のオペランドが指定できます。

name 解釈される名前

環境 type の実行に影響を与える環境変数 LC_TYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

PATH *name* の位置を表す

終了ステータス 以下の終了ステータスが返されます。

0 正常終了

>0 エラーが発生した

属性 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 typeset(1), attributes(5), environ(5)

typeset(1)

名前 typeset, whence – シェル変数と関数の属性と値を設定または取得するためのシェル組み込み関数

形式 **typeset** [± HLRZfilrtux [*n*]] [*name* [= *value*]]...

whence [-pv] *name*...

機能説明 typeset はシェル変数と関数の属性と値を設定します。 関数内で typeset を実行すると、*name* が示す変数の新しいインスタンスが生成されます。 関数が完了すると、その変数の値と型が復元されます。このコマンドには、 以下の属性を指定できます。

-H このフラグは UNIX 以外のマシン上で、 UNIX とホスト名ファイルとのマッピング情報を提供します。

-L 左詰めを行い、先行する空白文字を *value* から取り除きます。 *n* は、 0 以外であればフィールドの幅を定義します。 0 の場合、 フィールドの幅は最初に代入される値の幅で決定されます。 変数に値を代入したとき、 フィールド幅より短ければ右側に空白文字が詰められ、 長ければ切り捨てられます。 -Z フラグも指定されていれば、 先行する 0 を削除します。 -R フラグは無効になります。

-R 右詰めを行い、 先行する空白文字を挿入します。 *n* は、 0 以外であればフィールドの幅を定義します。 0 の場合、 フィールドの幅は最初に代入される値の幅で決定されます。 変数に値を代入したとき、 フィールド幅より短ければ左側に空白文字が詰められ、 長ければ終端が切り捨てられます。 -L フラグは無効になります。

-Z 最初の、空白文字でない文字が数字で、 かつ -L フラグが設定されていない場合、 右詰めを行い先頭に 0 を詰めます。 *n* は、 0 以外であればフィールドの幅を定義します。 0 の場合、 フィールドの幅は最初に代入される値の幅で決定されます。

-f 名前は、 変数名ではなく関数名を指します。 代入は行われません。 このフラグと共に指定できる他のフラグは、-t、-u、 -x だけです。 -t フラグは、 この関数の実行トレースを有効にします。 -u フラグは、 この関数に「未定義」を示すマークを付けます。 関数が参照されると、 関数定義を見つけるために FPATH 変数が検索されます。 -x フラグを指定すると、 名前で呼び出されるシェル手続き全体で 関数定義が有効になります。

-i パラメタを整数とします。 これにより算術演算が高速化されます。 *n* は、 0 以外であればその値を底として定義します。 0 の場合、 最初の代入で底が決定されます。

-l 大文字をすべて小文字に変換します。 大文字への変換を示す -u フラグを無効にします。

-r 指定された *name* を読み取り専用にします。 後の代入でこれらの名前を変更できないようにします。

-t 変数にタグを付けます。 タグはユーザーが定義可能で、 シェルに対して特別の意味を持ちません。

`-u` 小文字をすべて大文字に変換します。 小文字への変換を示す `-l` フラグを無効にします。

`-x` 指定された *name* に対し、 後で実行されるコマンドの「環境」へ 自動的にエクスポートされるようにマークを付けます。

`-i` 属性は、 `-R`、 `-L`、 `-Z`、 `-f` と同時に指定することはできません。

– の代わりに + を使用すると、 これらのフラグは無効になります。 *name* 引数をまったく指定せずにフラグを指定すると、 これらのフラグが設定されている変数の名前 （および選択により値も） が一覧表示されます。 具体的には – を付加すれば名前と値が、 + を付加すれば名前だけが表示されます。 *name* 引数とフラグを 1 つも指定しないと、 すべての変数の名前と属性が表示されます。

`whence` コマンドは、 *name* ごとに、 コマンド名として使用される場合に どのように解釈されるかを指示します。 `-v` フラグをつけると、より冗長に表示されます。 `-p` フラグをつけると、 コマンド名が別名、関数、 または予約語である場合でも *name* のパスが検索されます。

1 つまたは 2 つのアスタリスク（*）が先頭に付加されている `ksh`(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`ksh`(1), `set`(1), `sh`(1), `attributes`(5)

名前 | limit, ulimit, unlimit – 現在のシェルとそのシェルから起動されたプロセスで利用できるシステム資源の制限値を設定または取得

形式 | **/usr/bin/ulimit** [-f] [*blocks*]

sh | **ulimit** [- [HS] [a | cdfnstv]]

ulimit [- [HS] [c | d | f | n | s | t | v]] *limit*

csh | **limit** [-h] [*resource* [*limit*]]

unlimit [-h] [*resource*]

ksh | **ulimit** [-HSacdfnstv] [*limit*]

/usr/bin/ulimit | ulimit ユーティリティは、シェルとその子プロセスに課せられる 書き込み時のファイルサイズの制限値を、設定または報告します (ファイルサイズに関係なく 読み取りは可能です) 。適切な特権をもつプロセスのみ 制限値を上げることができます。

sh | Bourne シェルの組み込み関数 ulimit は、資源の強い制限値また弱い制限値を表示または設定します。これらの制限値については getrlimit(2) の説明を参照してください。

limit 引数を省略すると、ulimit は指定されている制限値を表示します。制限値は一度にいくつでも表示できます。-a オプションは制限値すべてを表示します。

limit 引数を指定すると、ulimit は指定されたフラグに対応する制限値をその引数の値に設定します。*limit* 引数の値として unlimited という文字列を指定すると、有効な最大値に設定されます。一度に資源 1 つについてだけ制限値を設定できます。ユーザーは誰でも、弱い制限値を強い制限値を超えない 任意の値に設定できます。また、強い制限値を下げることもできます。ただし、強い制限値を上げることができるのはスーパーユーザーだけです。詳細は su(1M) を参照してください。

-H オプションは強い制限値を表し、-S オプションは弱い制限値を表します。どちらのオプションも指定しない場合、ulimit は両方の制限値を設定し、弱い制限値を表示します。

以下のオプションは、制限値を表示または設定すべき資源を指定します。オプションをいっさい指定しないと、ファイルサイズ制限値を表示または設定します。

-c　最大コアファイルサイズ (512 バイトブロック単位)

-d　データセグメントまたはヒープの最大サイズ (K バイト単位)

-f　最大ファイルサイズ (512 バイトブロック単位)

-n　最大ファイル記述子に 1 を加えたもの

-s　スタックセグメントの最大サイズ (K バイト単位)

-t　最大 CPU 時間 (秒単位)

-v　仮想記憶の最大サイズ (K バイト単位)

csh C シェルの組み込み関数 `limit` は現在のプロセス、またはそれが生成したすべてのプロセスについて、各プロセスが指定された *resource* を *limit* 以上消費しないよう制限します。*limit* を省略すると、現在の制限値を表示します。*resource* を省略すると、すべての制限値を表示します。システムで利用可能な最大上限値を 調べるには `sysdef`(1M) コマンドを実行してください。表示される値は 16 進数ですが、`bc`(1) コマンドを使って 10 進数に変換できます。

`-h` 現在の制限値ではなく強い制限値を使用します。強い制限値は現在の制限値を制限します。強い制限値を上げることができるのは特権ユーザーだけです。

resource として指定できるものは次のとおりです。

`cputime`	プロセス当たりの最大 CPU 使用時間 (秒)
`filesize`	ファイルシステムのサイズに制限された最大の単一ファイル容量 (`df`(1M) 参照)
`datasize`	K バイト単位の プロセスのヒープの最大サイズ
`stacksize`	プロセスの最大スタックサイズ (`swap`(1M) 参照)
`coredumpsize`	最大コアダンプのファイルサイズ ファイルシステムのサイズに制限する
`descriptors`	ファイル記述子の最大数 (`sysdef()` を実行)
`memorysize`	仮想記憶の最大サイズ

limit は数値で、以下の単位を付加して指定することもできます。

n`h`	(`cputime` の) 時間
n`k`	*n*K バイト。これは `cputime` を除くすべての値のデフォルト単位
n`m`	*n*M バイトまたは (`cputime` の) 分
mm`:`*ss*	(`cputime` の) 分と秒

`unlimit` は *resource* に関する制限値を削除します。*resource* が指定されないと、すべての資源の制限値が削除されます。資源名の一覧については、前述の `limit` コマンドの説明を参照してください。

`-h` 対応する強い制限値を削除します。これは特権ユーザーだけが実行できます。

ksh Korn シェルの組み込み関数 `ulimit` は資源の制限を表示または設定します。使用可能な資源の制限は以下に説明します。システムによっては、以下に挙げたすべての 資源の制限を提供していないこともあります。*limit* 引数を指定すると、制限値が設定されます。*limit* の値は、各資源に対応した単位 (後述) の数値、または `unlimited` という文字列です。`-H` と `-S` の両フラグは、資源に対して 強い制限と弱い制限のどちらを設定するかを表します。強い制限値は、いったん設定したら あとで値を上げることはできません。弱い制限値は、強い制限値を超えない範囲で 値を上げることが可能です。`-H` も `-S` も省略すると、指定した制限値が強い制限と弱い制限の 両方に適用され

ます。*limit* 引数を省略すると、現在の資源制限値が表示されます。このとき、-H が指定された場合を除き、表示されるのは弱い制限値です。複数の資源を指定すると、値の前に制限する資源名と単位とが表示されます。

-a　現在の資源制限値をすべて表示します。

-c　コアダンプ時のコアファイルのサイズをブロック (512 バイト) 単位で表します。

-d　データ領域のサイズを K バイト単位で表します。

-f　子プロセスが書き込むファイルのサイズをブロック (512 バイト) 単位で表します。読み取るファイルにはサイズの制限はありません。

-n　最大ファイル記述子に 1 を加えた値を表します。

-s　スタック領域のサイズを K バイト単位で表します。

-t　各プロセスが使用する秒数 (CPU 時間) を表します。

-v　仮想記憶のサイズを K バイト単位で表します。

オプションをすべて省略すると、-f が指定されたものとみなします。

オプション

ulimit では以下のオプションが指定できます。

-f　ブロックごとにファイルサイズの制限を設定 (*blocks* を指定しない場合は報告) します。これはデフォルト値です。

オペランド

ulimit では以下のオペランドが指定できます。

blocks　新しくファイルサイズの制限として使用する 512 バイトごとのブロック数

/usr/bin/ulimit

例 **1** スタックサイズを制限する

スタックサイズを 512K バイトに 制限します。

```
% ulimit -s 512
% ulimit -a
% time(seconds)         unlimited
file(blocks)            100
data(kbytes)            523256
stack(kbytes)           512
coredump(blocks)        200
nofiles(descriptors)    64
memory(kbytes)          unlimited
```

sh/ksh

例 **2** ファイル記述子の数を 12 に制限する

ファイル記述子の数を 12 に制限します。

```
$ ulimit -n 12
$ ulimit -a
time(seconds)            unlimited
file(blocks)             41943
data(kbytes)             523256
```

例 **2** ファイル記述子の数を 12 に制限する　(続き)

```
stack(kbytes)            8192
coredump(blocks)         200
nofiles(descriptors)     12
vmemory(kbytes)          unlimited
```

csh

例 **3** コアダンプファイルのサイズを制限する

コアダンプファイルのサイズを 0K バイトに 制限します。

```
% limit coredumpsize 0
% limit
cputime                 unlimited
filesize                unlimited
datasize                523256 kbytes
stacksize               8192 kbytes
coredumpsize            0 kbytes
descriptors             64
memorysize              unlimited
```

例 **4** コアファイルサイズについての制限を削除する

上記の制限から コアファイルサイズについての制限を削除します。

```
% unlimit coredumpsize
% limit
cputime                 unlimited
filesize                unlimited
datasize                523256 kbytes
stacksize               8192 kbytes
coredumpsize            unlimited
descriptors             64
memorysize              unlimited
```

環境

`ulimit` の実行に影響を与える環境変数 `LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、`environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0`　　正常終了

`>0`　　要求した制限値が大きいため拒否された、またはエラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | bc(1), csh(1), ksh(1), sh(1), df(1M), su(1M), swap(1M), sysdef(1M), getrlimit(2), attributes(5), environ(5)

名前 | umask – ファイルモード生成マスクの獲得と設定

形式 | **/usr/bin/mask** [-S] [*mask*]

sh | **umask** [*ooo*]

csh | **umask** [*ooo*]

ksh | **umask** [-S] [*mask*]

機能説明 | umask ユーティリティは、現在のシェル実行環境のファイルモード生成マスクを *mask* オペランドが示す値に設定します。このマスクは、 以降生成されるファイルのファイル許可ビットの初期値に影響します。 以下のように、サブシェルや別のユーティリティ実行環境で umask が呼び出された場合は、 呼び出し側の環境のファイルモード生成マスクには影響しません。

```
(umask 002)
nohup umask . . .
find . -exec umask . . .
```

このため /usr/bin/umask ユーティリティは、 進行中のセッションにある umask の変更には使用できません。 この機能は、呼び出し側の umask の確認を限定するものです。 進行中のセッションにある umask を変更するには、 シェルの組み込みコマンドを使用する必要があります。

mask オペランドが省略されると、 umask ユーティリティは呼び出し側プロセスのファイルモード生成マスクの値を 標準出力に書き出します。

sh | ユーザーのファイル生成モードマスクを *ooo* で示す値に設定します。 *ooo* は 3 つの 8 進数で、左から順番に所有者、 グループ、その他のユーザー用の、読み取り /書き込み/実行権 を表します。詳しくは chmod(1) 、 chmod(2) 、 umask(2) の説明を参照してください。umask で指定したそれぞれの値は、 ファイル生成時にシステムが指定する値から減算されます (creat(2) を参照)。たとえば umask 022 と指定すると、グループとその他のユーザーに対する 書き込み権が取り除かれます。具体的には、モード 777 で生成されたファイルは 755 に、 666 で生成されたファイルは 644 にそれぞれモードが変更されます。

- *ooo* 引数を省略すると、マスクとして用いられている現在の値が出力されます。
- umask はシェルにより認識され実行されます。
- ユーザーの .profile に umask を登録しておくと (profile(4)) を参照)、ログイン時に実行され、生成されたファイルやディレクトリに対するユーザーのアクセス権モードが自動的に設定されます。

csh | 前述の、sh の umask 組み込みコマンドににについての説明を参照してください。

ksh | ユーザーファイルの作成時のマスクを *mask* 引数が示す値に設定します。 *mask* には、chmod(1) で説明する記号値または 8 進数を指定できます。 記号値を指定すると、 新しい umask 値は、 *mask* を直前の umask 値の補数に適用した結果の補数になります。 *mask* 引数を省略すると、 マスクの現在の値を表示します。

オプション | 以下のオプションを指定できます。

`-S` 記号出力を生成します。

デフォルトの出力形式は不定ですが、 同じシステム上の以降に `umask` が呼び出されたときには、以前のファイルモード生成マスクを復元する *mask* オペランドとして認識されます。

オペランド

以下のオペランドを指定できます。

mask 新しいファイルモード生成マスクを指定する文字列。この文字列は `chmod`(1) マニュアルページで説明されている *mode* オペランドと同じ方法で扱われます。

記号 値の場合、ファイルモード生成マスクの新しい値は、 記号 文字列が指定しているファイルモードの ファイル許可ビット部分の論理補数です。

記号 値の許可 *op* 文字 + と – は、現在のファイルモード生成マスクとの相対的な関係で解釈されます。 + は、示された許可をマスクからクリアします。 – は、示された許可ビットをマスクに設定します。

ファイル許可ビット以外のファイルモードビットを指定する *mode* 値の解釈は不定です。

ファイルモード生成マスクは、結果の数値に設定されます。

同じシステム上で以前にオペランドを指定せずに `umask` を実行した場合のデフォルト出力も、 *mask* オペランドとして認識されます。この方法で得たオペランドは、たとえ 8 進数であっても、無効ではありません。

出力

mask オペランドが省略されると、 `umask` ユーティリティは、後で `umask` の *mask* オペランドとして使用可能なメッセージを標準出力に書き出します。

`-S` が指定されると、メッセージは以下の形式になります。

```
"u=%s,g=%s,o=%s\n", owner permissions, group permissions, \
                                other permissions
```

3 つの値は、 {`r`, `w`, `x`} セットの文字の組み合わせです。 文字がある場合は、ファイルモード生成マスク内の対応するビットが 設定されていないことを示しています。

mask オペランドが指定されると、標準出力には何も出力されません。

使用例

例 1 `umask` コマンドの使用

以下のコマンドはどちらも、 以降生成されるファイルの `S_IWOTH` ビットがクリアされるように、モードマスクを設定します。

```
umask a=rx,ug+w
umask 002
```

上のコマンドでモードマスクを設定した後、 `umask` コマンドを使用してモードマスクの現在の値を書き出すことができます。

例 1 umask コマンドの使用　(続き)

```
$ umask
0002

$ umask -S
u=rwx,g=rwx,o=rx
```

この出力はどちらも、以降の umask 呼び出しで *mask* オペランドとして使用できます。

上記のようにモードマスクが設定されていると想定すると、 以下のコマンドは、 以降生成されるファイルの S_IWGRP ビットと S_IWOTH ビットがクリアされるように、モードマスクを設定します。

```
umask g-w
```

以下のコマンドは、 以降生成されるファイルの書き込みビットがクリアされるように、 モードマスクを設定します。

```
umask - -w
```

なお *mask* オペランドの r、w、x 、またはハイフン (–) で始まるものは、オプションと間違われないようにその前に - を付ける必要があります。

環境

umask の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　ファイルモード生成マスクが正常に変更された、または *mask* がなかった

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

chmod(1), csh(1), ksh(1), sh(1), chmod(2), creat(2), umask(2), profile(4), attributes(5), environ(5)

名前 | alias, unalias – コマンドまたはコマンド群の別名または省略形の生成と削除

形式 | **/usr/bin/alias** [*alias-name* [= *string*...]]

/usr/bin/unalias *alias-name*...

/usr/bin/unalias -a

csh | **alias** [*name* [*def*]]

unalias *pattern*

ksh | **alias** [-tx] [*name* [= *value*]...]

unalias *name*...

機能説明 | alias および unalias ユーティリティは、コマンドまたはコマンド群の別名あるいは省略形を作成または削除します。これらのユーティリティの動作は、C シェル環境と Korn シェル環境では異なります。

/usr/bin/alias | alias ユーティリティは、別名定義を作成または再定義するか、 あるいは既存の別名定義を標準出力に書き出します。 別名定義は、コマンド名を置き換える文字列を指定するものです。

別名定義は、現在のシェルの実行環境、およびそのシェルの全サブシェル の実行環境に影響を及ぼします。 このマニュアルに記述されているように用いれば、 別名定義は現在のシェルの親プロセスにも、 シェルが呼び出すユーティリティの環境にも、影響は与えません。

/usr/bin/unalias | unalias ユーティリティは、指定された別名の定義を削除します。 それにより、現在のシェルの実行環境から別名が削除されます。

csh | alias は別名 *name* に *def* で指定した別名の定義を割り当てます。 *def* はワードの並びで、エスケープされた ヒストリ置換のメタシンタックスを含んでいてもかまいません。 *name* に alias または unalias を使用することはできません。 *def* を省略すると、別名 *name* が現在の定義と共に表示されます。 *name* と *def* の両方を省略すると、現在あるすべての別名が表示されます。

実装状態の制限によって、別名の定義は、それが使われる前のコマンド行 で入力されていなければなりません。

unalias は *pattern* が示すファイル名置換パターン に一致する別名を破棄します。 'unalias *' と指定すると、すべての別名が破棄されます。

ksh | 引数なしの場合、このコマンドは標準出力上に *name=value* という形式の別名のリストを表示します。*value* が指定された名前に対しては別名を定義します。*value* の後方に空白があると、次のワードが別名置換指定かどうかをチェックします。 -t フラグは、検索済みの別名を設定または一覧表示します。検索済み別名の値は、指定した *name* に対応する完全パス名になります。PATH の値を再設定すると この値は未定義になりますが、 別名は検索済みのままです。-t フラグを省略すると、 *value* が指定されていない引数リスト内の各 *name* について、 別名の名前と値を表示します。 -x フラ

グは、 エクスポートされた別名を設定または表示します。 エクスポートされた別名は、 名前で起動されるスクリプト用に定義されます。 *name* が指定されているが、 *value* は指定されておらず、 *name* に対しての別名も定義されていない場合は、 終了状態は 0 以外になります。

unalias を指定して *name* が示す別名を別名リストから削除します。

オプション

unalias では、以下のオプションが指定できます。

-a 現在のシェルの実行環境から、すべての別名定義を削除します。

オペランド

以下のオペランドが指定できます。

alias *alias-name* 別名定義を標準出力に書き出します。

unalias *alias-name* 削除する別名を指定します。

alias-name=string *alias-name* で示す別名に、 *string* で示す文字列を割り当てます。

オペランドを 1 つも指定しないと、すべての別名定義が 標準出力に書き出されます。

出力

オペランドがすべて省略された場合、または *name* オペランドだけが指定された場合の、別名の表示形式は次のとおりです。

"%s=%s\n" *name, value*

文字列 *value* は、シェルへ再入力できるよう、適切な引用符を付加して出力されます。

使用例

例 1 コマンドの出力を変更する

次の例は、ls ユーティリティの出力を複数カラム形式の注釈付きに変更します。

```
example% alias ls="ls –CF"
```

例 2 コマンド履歴ファイルにある直前のエントリを繰り返す

次の例は、コマンドのヒストリファイルにある直前の入力を繰り返す単純な “redo” コマンドを生成します。

```
example% alias r='fc –s'
```

例 3 コマンドの出力オプションを指定する

次の例は、du ユーティリティがディスク出力を 1024 バイト単位にまとめるようにします。

```
example% alias du='du –k'
```

例 **4** 引数自身も別名であるような引数を処理する

次の例は、引数自身も別名であるような引数を処理できるよう、 nohup ユーティリティを設定します。

```
example% alias nohup="nohup "
```

環境

alias と unalias の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

alias

>0　*alias-name* オペランドで指定した名前の 1 つが別名定義を持っていなかった、もしくはエラーが発生した。

unalias

>0　*alias-name* オペランドで指定した名前の 1 つが正しい別名定義を表していなかった、もしくはエラーが発生した。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), shell_builtins(1), attributes(5), environ(5)

名前 | uname – 現在のシステム名の表示

形式 | **uname** [-aimnprsvX]

uname [-S *system_name*]

機能説明 | uname ユーティリティは、現在のシステムに関する情報を標準出力に出力します。オプションを指定した場合、1 つ以上のシステム特性を表すシンボルが 標準出力に書き出されます。オプションを指定しない場合、uname は、現在のオペレーティングシステムのシステム名を出力します。オプションを指定すると、uname(2) と sysinfo(2) (またはいずれか一方) が返した選択情報を出力します。

オプション | 次のオプションを指定できます。

-a　システムから現在、入手できる基本情報を出力します。

-i　実装されているハードウェア名 (プラットフォーム) を出力します。

-m　コンピュータのハードウェア名 (クラス) を出力します。このオプションを使用するのは望ましくないため、代わりに uname -p を指定してください。以下の「注意事項」の項を 参照してください。

-n　ノード名を出力します (ノード名とは、システムが通信ネットワークに認識されている名前です)。

-p　現在のホストの ISA またはプロセッサ型を出力します。

-r　オペレーティングシステムのリリースレベルを出力します。

-s　オペレーティングシステムのシステム名を出力します。これはデフォルトです。

-S *system_name*　システム名の引数を指定することによって、ノード名を変更できます。システム名の引数は、SYS_NMLN の文字に限られています。SYS_NMLN は、<sys/utsname.h> に定義された実装特有の値です。スーパーユーザーだけがこの機能を使用できます。このオプションを使用して変更した内容は、システムのリブート後には無効になります。システムをリブートしても変更内容が保持されるようにホスト名を変更するには、sys-unconfig(1M) を使用してください。

-v　オペレーティングシステムのバージョンを出力します。

-X　システムの拡張情報を出力します。SCO UNIX による情報と同じように、1 行に 1 つの情報が出力されます。表示される情報には次のものが含まれます。

- システム名、ノード、リリース、バージョン、マシン、CPU 番号
- バス形式、シリアル、ユーザー (Solaris では unknown に設定)
- OEM 番号 (0 に設定) とオリジナルの番号 (1 に設定)

使用例

例 **1** オペレーティングシステム名とリリースレベルを表示

次のコマンドを実行すると、オペレーティングシステム名とリリースレベルが 1 つの空白文字で区切られて出力されます。

```
example% uname -sr
```

環境

SYSV3

この環境変数は、デフォルトの uname の動作を無効にするために使用します。これは、一部の INTERACTIVE UNIX システムと SCO UNIX のプログラムやスクリプトの適切な動作を可能にするために必要です。多くのスクリプトは、ソフトウェアがその OS と互換性があることを確かめるために、uname を使用して SYSV3 タイプや OS バージョンを特定します。空文字列に SYSV3 を設定すると、uname は次のデフォルト値を印刷します。

```
nodename nodename 3.2 2 i386
```

次の形式で SYSV3 を設定して、uname が表示する個々の要素を修正することもできます。

os,sysname,node,rel,ver,mach

os	オペレーティングシステム (IUS または SCO)
sysname	システム名
node	-n オプションで表示されるノード名
rel	-r オプションで表示されるリリースレベル
ver	-v オプションで表示されるバージョン番号
mach	-m オプションで表示されるマシン名

要素間には空白を入れないでください。ある要素を指定しない場合、現在のシステムの値が使用されます。

uname の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | arch(1), isalist(1), sys-unconfig(1M), sysinfo(2), uname(2), nodename(4), attributes(5), environ(5)

注意事項 | 独立ソフトウェアベンダー (ISV) やその他のベンダーで、それぞれの提供するソフトウェアがインストールされているまたは実行されているプラットフォームの詳しい特性を知りたい場合は、uname コマンドを使用してください。

オペレーティングシステム名とオペレーティングシステムのリリースレベルを表示するには、uname -sr を使用してください。オペレーティングシステムのリリースレベルだけを表示するには、uname -r を使用してください。オペレーティングシステムのリリースレベルは *x.y* 形式 (たとえば 5.3、5.4、5.5 など) に限らず、*x.y.z* 形式 (たとえば 5.3.1、5.3.2、5.4.1 など) でも表示される場合があります。

SunOS 4.*x* リリースにおいて、uname コマンドと同様の情報を得るために arch(1) コマンドを使用し、arch(1) コマンドの sun4 という出力が SunOS SPARC システムを示すものとして誤って解釈されることがありました。ハードウェアプラットフォームの情報を得るには、uname -sp を使用してください。

arch -k コマンドと uname -m コマンドは 同じ値を返します。ただし、一般的に arch コマンドの使用が望ましくないのと同様に、この 2 つのコマンドをサードパーティのプログラム上で使用することは望ましくありません。マシンの Instruction Set Architecture (ISA またはプロセッサ型) を確認するには、uname -p を使用してください。

名前 | compress, uncompress, zcat – ファイルの圧縮、圧縮解除、または圧縮解除結果の表示

形式 | **compress** [-fv] [-b *bits*] [*file*...]

compress [-cfv] [-b *bits*] [*file*]

uncompress [-cfv] [*file*...]

zcat [*file*...]

compress | compress ユーティリティは、適応型レンペル・ジブ・コーディング法を利用して、指定されたファイルの圧縮を試みます。出力先が標準出力でないとき、各ファイルは、可能な限り、拡張子 .Z の付いたファイルに置き換えられます。所有モード、アクセス時刻、更新時刻は変わりません。なお .Z をファイルに付加したときパス名の長さが 1023 バイトを超えてしまう場合、コマンドの実行は失敗します。ファイル名が指定されていない場合には、標準入力が圧縮されて結果が標準出力に送られます。

どの程度圧縮されるかは、入力ファイルのサイズ、コードあたりのビット数、共通部分文字列の配置などによって異なります。通常、ソースコードや英語の文章などのテキストの場合、50% から 60% は圧縮されます。この圧縮処理は、ハフマンコーディング法 (pack(1) で使用されている) で行われる圧縮処理よりも優れており、計算時間も短くてすみます。このコマンド中に指定される *bits* パラメータの値は、圧縮結果のファイル内にコード化されます。さらにこのファイル内には、ランダムデータの圧縮解除および圧縮データの再圧縮を防止するためのマジックナンバーも書き込まれます。

uncompress | uncompress ユーティリティは、compress ユーティリティによって圧縮されたファイルを元の状態に復元します。ファイル名を省略すると、標準入力を圧縮解除して標準出力に書き出します。

このユーティリティは、compress が生成したどのようなファイルでも、圧縮解除します。ただし他のシステム上で compress が生成したファイルに関しては、9 および 16 ビット圧縮だけがサポートの対象となります。詳しくは -b の説明を参照してください。

zcat | zcat ユーティリティは、compress により圧縮されたファイルの圧縮解除した内容を、標準出力に出力します。これは uncompress -c と同じ結果となります。入力ファイルの内容は変わりません。

オプション | 以下のオプションを指定できます。

-c　結果を標準出力に書き出します。既存のファイルは変更されず、.Z ファイルも作られません。つまり uncompress -c という指定は zcat と同機能です。

-f　圧縮の場合、圧縮してもファイルが小さくならない場合や、対応する *file*.Z ファイルがすでに存在している場合でも、強制的に圧縮を行います。このオプションを指定しない場合は、プロセスがバックグラウンドで実行している場合を除き、既存の *file*.Z ファイルを上書きするかどうかの確認を求めるプロンプトが表示されます。圧縮解除の場合、このオプションは、上書きの確認用プロンプトを表示しないよう指定します。-f オプションを指定しない場合は、プロセスがバックグラウンドで実行してい

る場合を除き、既存のファイルを上書きするかどうかの確認を求めるプロンプトが表示されます。標準入力が端末ではないときに -f オプションが省略されると、標準エラー出力に診断メッセージを出力し、0 より大きな終了ステータスで終了します。

-v　冗長 (verbose) 。各ファイルが何パーセント圧縮または圧縮解除されたかを、標準エラー出力に書き出します。

-b *bits*　共通部分文字列コードの最大長 (ビット単位) を指定します。*bits* には、9 から 16 の値を指定します (デフォルトは 16 です)。この値を小さくすると、圧縮率が低くなり、結果ファイルは大きくなります。

オペランド

以下のオペランドを指定できます。

file　compress で圧縮または uncompress で圧縮解除するファイルのパス名、または圧縮解除された内容が zcat によって標準出力に書き込まれるファイルのパス名。– を指定するか、このオペランドを省略すると、標準入力とみなされます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の compress 、uncompress 、zcat の動作については、largefile(5) を参照してください。

環境

compress 、uncompress 、zcat の実行に影響を与える環境変数 LC_CTYPE 、LC_MESSAGES 、NLSPATH の詳細については、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了した

1　エラーが発生した

2　(-f オプションが指定されていないため) サイズが大きくなってしまい、いくつかのファイルが圧縮されなかった

>2　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

ln(1)、pack(1)、attributes(5)、environ(5)、largefile(5)

関連項目

Usage: compress [-fvc] [-b maxbits] [*file*. . .]
　コマンド行に無効なオプションが指定されました。

Missing maxbits
　-b の後に最大ビット数が指定されていません、または不正な値か数値でない値が指定されました。

file: not in compressed format
uncompress に指定されたファイルは 圧縮されたファイルではありません。

file: compressed with *xx*bits, can only handle *yy*bits
file は、このマシンの圧縮コードよりも大きいビット数を扱うプログラムによって、圧縮されています。これより小さいビット数でファイルを再圧縮する必要があります。

file: already has .Z suffix -- no change
このファイルは、すでに圧縮されたものと思われます。ファイル名を変更して、再試行してください。

file: already exists; do you wish to overwrite (y or n)?
出力ファイルで置き換える場合は y を、置き換えない場合は n と応答してください。

uncompress: corrupt input
SIGSEGV 違反が検出されました。通常、入力ファイルが破損していることを示します。

Compression: *xx.xx*%
入力ファイルが圧縮によって縮小された割合です。(-v オプションが指定された場合のみ表示します。)

– – not a regular file: unchanged
入力ファイルが通常ファイルではない場合 (たとえばディレクトリ)、内容は変更されません。

– – has *xx* other links: unchanged
入力ファイルにリンクがあります。内容は変更されません。詳細は、ln(1) を参照してください。

– – file unchanged
圧縮しても、縮小されません。入力ファイルの内容は 変更されません。

filename too long to tack on .Z
パス名が長すぎて、接尾辞 .z を付加できません。

注意事項

圧縮されたファイルは、大容量メモリーを持つマシン間では互換性がありますが、プロセスあたりのデータ領域が小さい (64K バイト以下) アーキテクチャへのファイル転送には -b 12 を指定してください。

拡張子 . z の付いたファイルが存在する場合の compress の処理は、もう少し柔軟であるべきです。

名前 | expand, unexpand – タブ文字を空白文字に展開する、またはその反対

形式 | **expand** [-t *tablist*] [*file*...]

expand [*–tabstop*] [*–tab1, tab2,. . ., tabn*] [*file*...]

unexpand [-a] [-t *tablist*] [*file*...]

機能説明 | expand は 1 つ以上の *file* (または標準入力) のタブ文字を空白文字に展開して標準出力へコピーします。 バックスペース文字は出力中に保存され、タブのカラム幅の計算の際にカラム幅を 1 減算します。 expand はタブ文字を含む文字ファイルの前処理 (ソートをする前や、 特定のカラムを探す前など) を行う際に役に立ちます。

unexpand は 1 つ以上の *file* (または標準入力)を、 タブ文字を復活させて標準出力へコピーします。 オプションの指定がないときは、 行頭の空白文字とタブ文字だけがタブ列に置き換えられます。 -a オプションを指定するとこの指定は無効となります(「オプション」の項参照)。

オプション | expand のオプション

-t*tablist* タブの位置を指定します。引数 *tablist* は、 1 つまたは複数の 10 進整数からなります。複数個指定する場合には、昇順に並べて 空白文字またはコンマで区切らなければなりません。 1 つの整数だけを指定すると、そのカラム数ごとにタブが設定されます。 デフォルトでは 8 カラムおきです。 複数の整数を指定すると、それらのカラム位置にタブが設定されます。 指定する各カラム位置 (*N*) は、ゼロより大きい整数でなければなりません。 またカラム位置は昇順に指定する必要があります。 行を出力する際、カラム位置 *N* にタブを進めるということは、次の文字が *N*+1 カラムに出力されることになります。 複数のタブ位置が指定され、その最後のタブ位置を超えた地点で タブ文字の出力を処理する必要が生じた場合、 expand はそのタブを 1 つの空白文字に置き換えて出力します。

–tabstop 1 つの数を指定し、その個数分の空白文字ごとに、 タブを設定します。 省略時の値は 8 です。

–tab1, tab2,. . ., tabn 引数で指定された位置にタブ文字を設定します。

unexpand のオプション

-a 置き換えていく際に 2 つ以上空白文字が連続していたらタブ文字を挿入します。 より小さな出力ファイルを生成します。

-t*tablist* タブの位置を指定します。引数 *tablist* は、 1 つまたは複数の 10 進整数からなります。複数個指定する場合には、昇順に並べて 空白文字またはコンマで区切らなければなりません。 1 つの整数だけを指定すると、そのカラム数ごとにタブが設定されます。 デフォルトでは 8 カラムおきです。 複数の整数を指定すると、それらのカラム位置にタブが設定されます。 指定する各カラム位置 (*N*) は、ゼロより大きい整数でなければなりません。 またカラム位置は昇順に指定する必要があります。 行を出力する際、カラム位置 *N* にタブを進めるということは、次の文字が *N*+1 カラムに出力され

ることになります。 -t オプションを省略すると、デフォルトとして -t 8 を指定したことと同等となります。ただし、後述するように -a との関連については異なります。 複数のタブ位置が指定された場合、その最後のタブ位置を超えた地点では 空白文字からタブ文字への文字変換は発生しません。 -t オプションを指定すると、 -a オプションは意味を持たなくなり、タブ変換は 先行する空白文字だけに制限されることはありません。

オペランド

以下のオペランドを指定できます。

file 入力に用いるテキストファイルのパス名。

環境

expand と unexpand の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 正常終了

>0 エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

tabs(1), attributes(5), environ(5)

名前 | hash, rehash, unhash, hashstat – ディレクトリの内容の内部ハッシュテーブルの評価

形式 | **/usr/bin/hash** [*utility*]

/usr/bin/hash [-r]

sh | **hash** [-r] [*name*...]

csh | **rehash**

unhash

hashstat

ksh | **hash** [*name*...]

/usr/bin/hash | /usr/bin/hash ユーティリティは、見つかったユーティリティの位置を現在のシェル環境が どのように記憶するか、その記憶方法を変更します。 具体的には、指定された引数に従って、新たなユーティリティを ユーティリティ位置リストに追加したり、リストの内容を消去したりします。引数を指定しないと、リストの内容が報告されます。

シェルの組み込みユーティリティとして提供されているものについては、 hash は報告対象としません。

sh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。-r オプションを指定すると、シェルは記憶したすべての位置を破棄します。引数をまったく指定しないと、記憶されたコマンドに関する情報が表示されます。 この出力表示において、 *Hits* の列はシェルプロセスによってコマンドが呼び出された回数を表します。*Cost* は、検索パスのコマンドを見つけるのに必要な作業量です。 コマンドが検索パスの「相対」ディレクトリにある場合、 そのディレクトリの変更後にそのコマンドが格納された位置が再計算されます。 この再計算が行われる予定のコマンドに対しては、*Hits* 情報の隣にアスタリスク (*) が示されます。 *Cost* の値は、再計算が行われるたびに増加します。

csh | rehash は新しく追加されたコマンドに合わせて、 path 環境の変数に記憶しているディレクトリの内容の内部ハッシュテーブルを再計算します。

unhash は内部ハッシュテーブルを使用不能にします。

hashstat は内部ハッシュテーブルがコマンドの検索 (実行を伴わない) にどの程度有効に働いて いるかを示す統計情報を出力します。 ハッシュ関数がヒットの可能性を示しているパス の各構成要素と、' / ' で始まらない各構成要素について、実行しようとします。

ksh | シェルは、*name* に指定されたコマンドごとに、検索パス内での位置を決定し、記憶します。引数が与えられない場合、hash は記憶されたコマンドに関する情報を表示します。

オペランド | 以下のオペランドを指定できます。

utility 位置のリスト内で検索する、あるいはリストに 追加するユーティリティ名。

出力　引数を 1 つも指定しないと、hash の標準出力が使用されます。その形式は決まっていませんが、 現在のシェル環境用の位置リストにある各ユーティリティの パス名は含まれています。このリストは、以前に実行された hash 呼び出し中に指定されていたユーティリティをすべて含んでおり、さらに通常のコマンド検索プロセスで呼び出され見つけられた ユーティリティを含んでいることもあります。

環境　hash の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、 NLSPATH についての詳細は、environ(5) を参照してください。

PATH　*utility* の位置を指定します。

終了ステータス　以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性　次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目　csh(1), ksh(1), sh(1), attributes(5), environ(5)

名前 | uniq – ファイルの中の重複行の報告またはフィルタへの出力

形式 | **uniq** [-c | -d | -u] [-f *fields*] [-s *char*] [*input_file* [*output_file*]]

uniq [-c | -d | -u] [-n] [+ *m*] [*input_file* [*output_file*]]

機能説明 | uniq ユーティリティは、 隣接する行を比較しながら入力ファイルを読み取り、 各行の内容を出力します。 同じ内容の行が連続していれば、その重複行の内容を 1 回だけ出力します。

内容が同じでも連続していなければ、 重複しているとは認識されません。

オプション | 以下のオプションを指定できます。

-c　各行の先頭に、その行が入力中に連続して現れた回数を出力します。

-d　入力中で重複していなかった行は出力しません。

-f *fields*　内容比較を行う際に、各行の先頭の *fields* 個のフィールドは無視します。 *fields* は正の整数です。ここでいうフィールドとは、 以下の基本正規表現で一致する最大の文字列です。

[[:blank:]]*[^[:blank:]]*入力行のフィールドの総数より大きい値を *fields* に指定した場合、比較には NULL 文字列が使われます。

-s *chars*　内容比較を行う際に、各行の先頭の *chars* 個の文字は無視します。 *chars* は正の整数です。 -f オプションと一緒に指定した場合には、 先頭の *fields* 個のフィールドに続く *chars* 個の文字が無視されます。 入力行に残っている文字の総数より大きい値を *chars* に指定した場合、比較には NULL 文字列が使われます。

-u　入力中で重複していた行は出力しません。

-*n*　*fields* の値が *n* である -f *fields* 指定と同じ意味です。

+*m*　*chars* の値が *m* である -s *chars* 指定と同じ意味です。

オペランド | 以下のオペランドを指定できます。

input_file　入力ファイルのパス名。 - を指定するかまたはこのオペランドを省略すると、 標準入力が使用されます。

output_file　出力ファイルのパス名。 このオペランドを省略すると、標準出力が使用されます。 *input_file* と同じファイルを *output_file* で指定した場合、処理の結果は予測できません。

使用例 | 例 1 uniq コマンドの使用

次の例では uniq.test ファイルの内容をリストで示し、重複行を出力しています。

```
example% cat uniq.test
This is a test.
```

例 1 uniq コマンドの使用　(続き)

```
This is a test.
TEST.
Computer.
TEST.
TEST.
Software.

example% uniq -d uniq.test
This is a test.
TEST.
example%
```

次の例では uniq.test ファイルで重複していない行だけを出力しています。

```
example% uniq -u uniq.test
TEST.
Computer.
Software.
example%
```

最後の例では、ファイルの中でそれぞれの行が現われた回数を 各行の先頭に付けて出力しています。

```
example% uniq -c uniq.test
   2 This is a test.
   1 TEST.
   1 Computer.
   2 TEST.
   1 Software.
example%
```

環境

uniq の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0　　正常終了

>0　　エラーが発生した

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

comm(1), pack(1), pcat(1), sort(1), uncompress(1), attributes(5), environ(5)

名前 | limit, ulimit, unlimit – 現在のシェルとそのシェルから起動されたプロセスで利用できるシステム資源の制限値を設定または取得

形式 | **/usr/bin/ulimit** [-f] [*blocks*]

sh | **ulimit** [- [HS] [a | cdfnstv]]

ulimit [- [HS] [c | d | f | n | s | t | v]] *limit*

csh | **limit** [-h] [*resource* [*limit*]]

unlimit [-h] [*resource*]

ksh | **ulimit** [-HSacdfnstv] [*limit*]

/usr/bin/ulimit

ulimit ユーティリティは、シェルとその子プロセスに課せられる 書き込み時のファイルサイズの制限値を、設定または報告します (ファイルサイズに関係なく 読み取りは可能です) 。適切な特権をもつプロセスのみ 制限値を上げることができます。

sh

Bourne シェルの組み込み関数 ulimit は、資源の強い制限値また弱い制限値を表示または設定します。これらの制限値については getrlimit(2) の説明を参照してください。

limit 引数を省略すると、ulimit は指定されている制限値を表示します。制限値は一度にいくつでも表示できます。-a オプションは制限値すべてを表示します。

limit 引数を指定すると、ulimit は指定されたフラグに対応する制限値をその引数の値に設定します。*limit* 引数の値として unlimited という文字列を指定すると、有効な最大値に設定されます。一度に資源 1 つについてだけ制限値を設定できます。ユーザーは誰でも、弱い制限値を強い制限値を超えない 任意の値に設定できます。また、強い制限値を下げることもできます。ただし、強い制限値を上げることができるのはスーパーユーザーだけです。詳細は su(1M) を参照してください。

-H オプションは強い制限値を表し、-S オプションは弱い制限値を表します。どちらのオプションも指定しない場合、ulimit は両方の制限値を設定し、弱い制限値を表示します。

以下のオプションは、制限値を表示または設定すべき資源を指定します。オプションをいっさい指定しないと、ファイルサイズ制限値を表示または設定します。

-c　最大コアファイルサイズ (512 バイトブロック単位)

-d　データセグメントまたはヒープの最大サイズ (K バイト単位)

-f　最大ファイルサイズ (512 バイトブロック単位)

-n　最大ファイル記述子に 1 を加えたもの

-s　スタックセグメントの最大サイズ (K バイト単位)

-t　最大 CPU 時間 (秒単位)

-v　仮想記憶の最大サイズ (K バイト単位)

csh C シェルの組み込み関数 `limit` は現在のプロセス、またはそれが生成したすべてのプロセスについて、各プロセスが指定された *resource* を *limit* 以上消費しないよう制限します。*limit* を省略すると、現在の制限値を表示します。*resource* を省略すると、すべての制限値を表示します。システムで利用可能な最大上限値を 調べるには `sysdef`(1M) コマンドを実行してください。表示される値は 16 進数ですが、`bc`(1) コマンドを使って 10 進数に変換できます。

`-h` 現在の制限値ではなく強い制限値を使用します。強い制限値は現在の制限値を制限します。強い制限値を上げることができるのは特権ユーザーだけです。

resource として指定できるものは次のとおりです。

`cputime`	プロセス当たりの最大 CPU 使用時間 (秒)
`filesize`	ファイルシステムのサイズに制限された最大の単一ファイル容量 (`df`(1M) 参照)
`datasize`	K バイト単位の プロセスのヒープの最大サイズ
`stacksize`	プロセスの最大スタックサイズ (`swap`(1M) 参照)
`coredumpsize`	最大コアダンプのファイルサイズ ファイルシステムのサイズに制限する
`descriptors`	ファイル記述子の最大数 (`sysdef()` を実行)
`memorysize`	仮想記憶の最大サイズ

limit は数値で、以下の単位を付加して指定することもできます。

n`h`	(`cputime` の) 時間
n`k`	*n*K バイト。これは `cputime` を除くすべての値のデフォルト単位
n`m`	*n*M バイトまたは (`cputime` の) 分
mm`:`*ss*	(`cputime` の) 分と秒

`unlimit` は *resource* に関する制限値を削除します。*resource* が指定されないと、すべての資源の制限値が削除されます。資源名の一覧については、前述の `limit` コマンドの説明を参照してください。

`-h` 対応する強い制限値を削除します。これは特権ユーザーだけが実行できます。

ksh Korn シェルの組み込み関数 `ulimit` は資源の制限を表示または設定します。使用可能な資源の制限は以下に説明します。システムによっては、以下に挙げたすべての 資源の制限を提供していないこともあります。*limit* 引数を指定すると、制限値が設定されます。*limit* の値は、各資源に対応した単位 (後述) の数値、または `unlimited` という文字列です。`-H` と `-S` の両フラグは、資源に対して 強い制限と弱い制限のどちらを設定するかを表します。強い制限値は、いったん設定したら あとで値を上げることはできません。弱い制限値は、強い制限値を超えない範囲で 値を上げることが可能です。`-H` も `-S` も省略すると、指定した制限値が強い制限と弱い制限の 両方に適用され

ます。*limit* 引数を省略すると、現在の資源制限値が表示されます。このとき、`-H` が指定された場合を除き、表示されるのは弱い制限値です。複数の資源を指定すると、値の前に制限する資源名と単位とが表示されます。

`-a` 現在の資源制限値をすべて表示します。

`-c` コアダンプ時のコアファイルのサイズをブロック (512 バイト) 単位で表します。

`-d` データ領域のサイズを K バイト単位で表します。

`-f` 子プロセスが書き込むファイルのサイズをブロック (512 バイト) 単位で表します。読み取るファイルにはサイズの制限はありません。

`-n` 最大ファイル記述子に 1 を加えた値を表します。

`-s` スタック領域のサイズを K バイト単位で表します。

`-t` 各プロセスが使用する秒数 (CPU 時間) を表します。

`-v` 仮想記憶のサイズを K バイト単位で表します。

オプションをすべて省略すると、`-f` が指定されたものとみなします。

オプション `ulimit` では以下のオプションが指定できます。

`-f` ブロックごとにファイルサイズの制限を設定 (*blocks* を指定しない場合は報告) します。これはデフォルト値です。

オペランド `ulimit` では以下のオペランドが指定できます。

blocks 新しくファイルサイズの制限として使用する 512 バイトごとのブロック数

/usr/bin/ulimit

例 1 スタックサイズを制限する

スタックサイズを 512K バイトに 制限します。

```
% ulimit -s 512
% ulimit -a
% time(seconds)         unlimited
file(blocks)            100
data(kbytes)            523256
stack(kbytes)           512
coredump(blocks)        200
nofiles(descriptors)    64
memory(kbytes)          unlimited
```

sh/ksh

例 2 ファイル記述子の数を 12 に制限する

ファイル記述子の数を 12 に制限します。

```
$ ulimit -n 12
$ ulimit -a
time(seconds)            unlimited
file(blocks)             41943
data(kbytes)             523256
```

例 **2** ファイル記述子の数を 12 に制限する　(続き)

```
stack(kbytes)            8192
coredump(blocks)         200
nofiles(descriptors)     12
vmemory(kbytes)          unlimited
```

csh

例 **3** コアダンプファイルのサイズを制限する

コアダンプファイルのサイズを 0K バイトに 制限します。

```
% limit coredumpsize 0
% limit
cputime                 unlimited
filesize                unlimited
datasize                523256 kbytes
stacksize               8192 kbytes
coredumpsize            0 kbytes
descriptors             64
memorysize              unlimited
```

例 **4** コアファイルサイズについての制限を削除する

上記の制限から コアファイルサイズについての制限を削除します。

```
% unlimit coredumpsize
% limit
cputime                 unlimited
filesize                unlimited
datasize                523256 kbytes
stacksize               8192 kbytes
coredumpsize            unlimited
descriptors             64
memorysize              unlimited
```

環境

`ulimit` の実行に影響を与える環境変数 `LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、`environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0`　　正常終了

`>0`　　要求した制限値が大きいため拒否された、またはエラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | bc(1), csh(1), ksh(1), sh(1), df(1M), su(1M), swap(1M), sysdef(1M), getrlimit(2), attributes(5), environ(5)

名前 | pack, pcat, unpack – ファイルの圧縮および復元

形式 | **pack** [-f] [-] *file*...

pcat *file*...

unpack *file*...

pack | pack コマンドは、指定されたファイルを圧縮した形で保存します。 可能であれば (そして便利ならば)、 *file* という入力ファイルは、 *file* と同じアクセスモード、 同じアクセスの日付や変更した日付、同じ所有者を持つ *file*.z という圧縮されたファイルに置き換えられます。 pack の実行が正常に終了すると、 *file* は削除されます。

圧縮の総量は、入力ファイルのサイズおよび文字度数分布によります。 デコード用ツリーがそれぞれの .z ファイルの最初の部分を形成しているので、 3 ブロック未満のファイルを圧縮することは、あまり意味がありません。 ただし、プリンタプロットや図形の場合に起こるように、 文字度数分布が非常に偏っているものは例外です。

一般的にテキストファイルを元のサイズの 60% から 75% に圧縮します。大きい文字セットを使用し、 文字分布が一様なロードモジュールは、 元のサイズのおよそ 90% にしか圧縮されません。

pack は、圧縮できなかったファイルの数を示す値を返します。

次の場合には、圧縮は行われません。

- ファイルがすでに圧縮されている場合
- ファイル名が 14 – 2 バイトより長い場合
- ファイルにリンクがある場合
- ファイルがディレクトリの場合
- ファイルをオープンできない場合
- ファイルが空の場合
- 圧縮によってディスクブロックを減らせない場合
- *file*.z というファイルが既に存在している場合
- .z ファイルを作成することができない場合
- 処理中に、入出力エラーが発生した場合

ファイル名の最後のセグメントは、{NAME_MAX} – 2 文字以下でなければなりません。 この 2 文字は、追加される拡張子 .z 用です。ディレクトリは圧縮できません。

pcat | pcat コマンドは、 cat(1) が通常ファイルに対して行うことを、 圧縮したファイルに対して行います。 ただし pcat をフィルタとして使うことはできません。 指定したファイルは、復元され、標準出力に書き込まれます。

pcat は、復元できなかったファイルの数を返します。 以下の場合にはエラーとなります。

- ファイルがオープンできなかった
- ファイルが、 pack の出力ファイルと認識できなかった

unpack

unpack コマンドは、 pack で作成したファイルを復元します。 コマンドで指定したファイル *file* に対して、 *file*.z というファイル (あるいは *file* が .z で終わる場合は単に *file*)を検索します。 このファイルが圧縮されたファイルである場合は、 復元したファイルに置き換えます。新たなファイル名は .z 接頭辞が取り除かれ、アクセスモード、アクセス日付や変更日 付および所有者名は圧縮されたファイルと同じです。

unpack は、復元できなかったファイルの数を示す値を返します。 pcat で述べた理由のほかに、次のような場合には復元できません。

- "unpackされた"ときのファイル名が既に存在する場合
- 復元したファイルを作成できない場合
- ファイル名の長さ (拡張子 .z を除く) が 14 バイトを超えている場合

オプション

以下のオプションを指定できます。

-f　　*file* の強制圧縮。ディレクトリ全体を圧縮するのに有効です。 ただし、中には 圧縮しても小さくならないファイルがあります。 unpack または pcat は、圧縮したファイルを元の形式に復元できます。

オペランド

以下のオペランドを指定できます。

file　　pack、 unpack、または pcat するファイルのパス名。 *file* には拡張子 .z を指定しても省略してもかまいません。

–　　pack は、 1 バイトごとに Huffman (最小冗長度) コードを使用します。 – 引数を使用すると、それぞれのバイトの使用回数、 相対頻度およびバイトのコードを標準出力に出力するよ うに内部フラグが設定されます。 *file* の代わりに – を追加すると、内部フラグを設定し、リ セットします。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の pack 、 pcat 、 unpack の動作については、 largefile(5) を参照してください。

使用例

例 **1** 圧縮ファイルを表示する

file.z という名前の圧縮ファイルを見るには、次のようにします。

```
example% pcat file.z
```

あるいは、単に次のようにします。

```
example% pcat file
```

例 **2** 圧縮したファイルの復元コピーを作成する

file.z という名前の圧縮したファイルの復元コピー nnn を file.z を破壊せずに作成するには 、次のコマンドを使用します。

```
example% pcat file >nnn
```

環境

pack 、 pcat 、 unpack の実行に影響を与える環境変数 LC_CTYPE 、 LC_MESSAGES 、 NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	正常終了
>0	エラーが発生した。圧縮または復元できなかった ファイルの数が返される

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

cat(1), compress(1), zcat(1), attributes(5), environ(5), largefile(5)

名前 | set, unset, setenv, unsetenv, export – 現在のシェルおよびそこから起動されたプロセスでの環境変数の特性を決定するシェル組み込み関数

sh

set [--aefhkntuvx [*argument*]]...

unset [*name*...]

export [*name*...]

csh

set [*var* [= *value*]]

set *var* [*n*] = *word*

unset *pattern*

setenv [*VAR* [*word*]]

unsetenv *variable*

ksh

set [±aefhkmnopstuvx] [±o *option*]... [±A *name*] [*arg*...]

unset [-f] *name*...

****export** [*name* [=*value*]]...

sh

set 組み込みコマンドには次のようなオプションがあります。

-- どのフラグも変更しません。 $1 に - を設定する際に便利です。

-a 修正または作成された変数にエクスポートのマークを付けます。

-e コマンドが 0 以外の終了状態で終了した場合、直ちに終了します。

-f ファイル名を生成しないようにします。

-h 関数の定義時に、関数コマンドを検索しその位置を記憶します (通常、関数コマンドは関数実行時に検索されます)。

-k コマンド名の前にあるキーワード引数だけでなく、 すべてのキーワード引数がコマンド用の環境に置かれます。

-n コマンドを読み取るが、実行しません。

-t 1 つのコマンドを読み取り、実行したあと、終了します。

-u 未設定の変数を置換時にエラーとして扱います。

-v シェル入力行の読み取り時に、その内容を表示します。

-x コマンドの実行時に、コマンドと引数の内容を表示します。

- の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェルの起動時にも使用できます。 現在セットされているフラグは、 $- で見つけられます。 *argument* は定位置パラメタで、 $1、$2、... へ順に割り当てられます。 *argument* が指定されない場合、 すべての名前の値が出力されます。 unset は *name* ごとに、対応する変数または関数値を削除します。 変数 PATH 、PS1、 PS2、 MAILCHECK および IF は設定解除できません。

export 組み込みコマンドでは、 指定された *name* に対し、ひきつづき実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。 引数を省略すると、 現在のシェル実行中にエクスポートのマークが付けられた 変数名を一覧表示します。 関数名はエクスポートされません。

csh 引数を指定しないと set はすべてのシェル変数の値を表示します。 複数ワードからなる値は括弧でくくられて表示されます。 引数 *var* だけを指定すると、 set は空 (NULL) の値を *var* が示す変数に割り当てます。 引数を *var* = *value* の形式で指定すると、 set は、変数 *var* に値 *value* を割り当てます。 *value* は次のいずれかです。

word	単一ワード (もしくは引用符付きの文字列)
(*wordlist*)	空白で区切られた、括弧付きワードの並び

値は、割り当てられる前に、コマンドおよびファイル名展開されます。 set *var*[*n*]=*word* 形式は、複数ワードからなる値の *n* 番目のワードを *word* に置き換えます。

unset は *pattern* が示すファイル名置換パターンに一致する名の変数を削除します。 'unset *' と指定すると、すべての変数が削除されます。 ただしこれは、 csh の動作に悪影響をおよぼします。

引数を指定しないと setenv はすべての環境変数を表示します。引数 *VAR* を指定すると、 setenv は環境変数 *VAR* に空の値 (NULL) を設定します (慣習上、環境変数名は大文字で指定されるのが通常)。 *VAR* と *word* の両引数を指定すると、 setenv は、*VAR* に単一ワードまたは引用符付き文字列である値 *word* を設定します。 環境変数 PATH は、コロンで区切られた複数の *word* 引数を指定できます (後述の「使用例」を参照) 。 最もよく使用される環境変数 USER 、 TERM 、 PATH は、自動的に csh 変数 user、term、path から (へ) インポート (エクスポート) されます。 これらの変数を変更する場合には setenv を使用してください。 さらにシェルは、 csh 変数 cwd が変更されるたびに、その値を 環境変数 PWD へ設定します。

環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、LC_COLLATE、LC_NUMERIC、LC_MONETARY は、C シェル内で変更されると新しい値が即座に有効になります。 これらの環境変数の詳細については environ(5) を参照してください。

unsetenv は環境から *variable* が示す変数を削除します。 unset のようなパターンマッチングは行いません。

ksh set コマンドのフラグの意味は以下のとおりです。

-A	配列の代入。 *name* で示される変数の設定を解除し、 *arg* リストから順々に値を割り当てます。 +A は、 最初の変数設定を解除しません。
-a	定義される後続の変数すべてを自動的にエクスポートします。
-e	コマンドの終了状態が 0 でない場合、 ERR トラップ (設定されていれば) を実行し、 終了します。 このモードは、 プロファイル読み取り時は無効です。
-f	ファイル名の生成を無効にします。

-h　各コマンドは、 最初に検出された時点で、 検索済み別名になります。

-k　コマンド名に先行するものだけでなく、 すべての変数代入引数をコマンドの環境に格納します。

-m　バックグラウンドジョブを個別のプロセスグループで実行し、 完了時にメッセージを 1 行表示します。 バックグラウンドジョブの終了状態は 完了メッセージで報告されます。 ジョブ制御を備えたシステムでは、 このフラグは 対話型シェルに対して自動的に有効になります。

-n　コマンドを読み取り、 構文エラーがないかチェックします。 実行は行いません。 対話型シェルに対しては無視されます。

-o　このフラグの後に指定する引数は、 以下のオプション名のいずれかです。

allexport	-a と同じです。
errexit	-e と同じです。
bgnice	バックグラウンドジョブをすべて低い優先度で実行します。 これはデフォルトモードです。
emacs	コマンド入力用に、 emacs 形式のインラインエディタを起動します。
gmacs	コマンド入力用に、 gmacs 形式のインラインエディタを起動します。
ignoreeof	ファイルの終わりを検出してもシェルは終了しません。 終了させるには exit コマンドを使用する必要があります。
keyword	-k と同じです。
markdirs	ファイル名生成によって生成されるディレクトリ名には、 すべて最後に / を付加します。
monitor	-m と同じです。
noclobber	> によるリダイレクトが存在するファイルを切り捨てないようにします。 このオプションが有効なとき、 ファイルを切り捨てるには >| 指定が必要です。
noexec	-n と同じです。
noglob	-f と同じです。
nolog	履歴ファイルに関数定義を保存しません。
nounset	-u と同じです。
privileged	-p と同じです。
verbose	-v と同じです。
trackall	-h と同じです。

vi	vi 形式のインラインエディタの挿入モードになります。 033 というエスケープ文字を押すと、 挿入モードから制御モードに変わります。 Return で行を送信します。
viraw	各文字を vi モードで入力されたときと同様に処理します。
xtrace	-x と同じです。

オプション名を指定しない場合、 -o は現在のオプション設定を表示します。

-p	$HOME/.profile ファイルを処理しないようにし、 ENV ファイルの代わりに /etc/suid_profile ファイルを使用します。 このモードは、 実効ユーザー ID が実ユーザー ID と等しくないとき、また実効グループ ID が実グループ ID と等しくないときには 必ず有効になります。 このモードを無効にすると、 実効ユーザー ID が実ユーザー ID に、 実効グループ ID が実グループ ID にそれぞれ設定されます。
-s	定位置パラメタを辞書編集方式の順にソートします。
-t	コマンド 1 つを読み取って実行し、 終了します。
-u	置換を行う際に、設定されていないパラメタを エラーとして扱います。
-v	シェルへの入力行を読み取り時に表示します。
-x	コマンドとその引数を実行時に表示します。
–	-x フラグと -v フラグを無効にし、 フラグに対する引数の検査を停止します。
– –	どのフラグも変更しません。 このフラグは、 $1を – で始まる値に設定する際に便利です。 このフラグの後に引数がない場合、 定位置パラメタが設定解除されます。

– の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェル起動時にも使用できます。 現在設定されているフラグは、 $- で見ることができます。 -A を指定しないかぎり、 残りの引数は定位置パラメタとなり、 $1 $2 ... に順番に割り当てられます。 引数を 1 つも指定しない場合には、 すべての変数の名前と値を標準出力上に表示します。

name が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、 MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

unset を使用すると *name* が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、

MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

export 組み込みコマンドでは、 指定された *name* に対し、 後で実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

1 つまたは 2 つの (*) アスタリスクが先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

csh 次の例では、 /bin、/usr/bin、/usr/sbin、/usr/ucb/bin ディレクトリにあるファイルをその順番で検索するために、 環境変数 PATH を設定しています。

```
setenv PATH "/bin:/usr/bin:/usr/sbin:usr/ucb/bin"
```

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), read(1), sh(1), typeset(1), attributes(5), environ(5)

名前 | set, unset, setenv, unsetenv, export – 現在のシェルおよびそこから起動されたプロセスでの環境変数の特性を決定するシェル組み込み関数

sh | **set** [--aefhkntuvx [*argument*]]...

unset [*name*...]

export [*name*...]

csh | **set** [*var* [= *value*]]

set *var* [*n*] = *word*

unset *pattern*

setenv [*VAR* [*word*]]

unsetenv *variable*

ksh | **set** [±aefhkmnopstuvx] [±o *option*]... [±A *name*] [*arg*...]

unset [-f] *name*...

****export** [*name* [=*value*]]...

sh | set 組み込みコマンドには次のようなオプションがあります。

`--` どのフラグも変更しません。 `$1` に - を設定する際に便利です。

`-a` 修正または作成された変数にエクスポートのマークを付けます。

`-e` コマンドが 0 以外の終了状態で終了した場合、直ちに終了します。

`-f` ファイル名を生成しないようにします。

`-h` 関数の定義時に、関数コマンドを検索しその位置を記憶します (通常、関数コマンドは関数実行時に検索されます)。

`-k` コマンド名の前にあるキーワード引数だけでなく、 すべてのキーワード引数がコマンド用の環境に置かれます。

`-n` コマンドを読み取るが、実行しません。

`-t` 1 つのコマンドを読み取り、実行したあと、終了します。

`-u` 未設定の変数を置換時にエラーとして扱います。

`-v` シェル入力行の読み取り時に、その内容を表示します。

`-x` コマンドの実行時に、コマンドと引数の内容を表示します。

- の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェルの起動時にも使用できます。 現在セットされているフラグは、 `$-` で見つけられます。 *argument* は定位置パラメタで、 `$1`、`$2`、... へ順に割り当てられます。 *argument* が指定されない場合、 すべての名前の値が出力されます。 unset は *name* ごとに、対応する変数または関数値を削除します。 変数 PATH 、PS1、 PS2、MAILCHECK および IF は設定解除できません。

export 組み込みコマンドでは、 指定された *name* に対し、ひきつづき実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。 引数を省略すると、 現在のシェル実行中にエクスポートのマークが付けられた 変数名を一覧表示します。 関数名はエクスポートされません。

csh 引数を指定しないと set はすべてのシェル変数の値を表示します。 複数ワードからなる値は括弧でくくられて表示されます。 引数 *var* だけを指定すると、 set は空 (NULL) の値を *var* が示す変数に割り当てます。 引数を *var* = *value* の形式で指定すると、 set は、変数 *var* に値 *value* を割り当てます。 *value* は次のいずれかです。

word 単一ワード (もしくは引用符付きの文字列)

(*wordlist*) 空白で区切られた、括弧付きワードの並び

値は、割り当てられる前に、コマンドおよびファイル名展開されます。 set *var*[*n*]=*word* 形式は、複数ワードからなる値の *n* 番目のワードを *word* に置き換えます。

unset は *pattern* が示すファイル名置換パターンに一致する名の変数を削除します。 'unset *' と指定すると、すべての変数が削除されます。 ただしこれは、 csh の動作に悪影響をおよぼします。

引数を指定しないと setenv はすべての環境変数を表示します。引数 *VAR* を指定すると、 setenv は環境変数 *VAR* に空の値 (NULL) を設定します (慣習上、環境変数名は大文字で指定されるのが通常)。 *VAR* と *word* の両引数を指定すると、 setenv は、*VAR* に単一ワードまたは引用符付き文字列である値 *word* を設定します。 環境変数 PATH は、コロンで区切られた複数の *word* 引数を指定できます (後述の「使用例」を参照) 。 最もよく使用される環境変数 USER 、 TERM 、 PATH は、自動的に csh 変数 user、term、path から (へ) インポート (エクスポート) されます。 これらの変数を変更する場合には setenv を使用してください。 さらにシェルは、 csh 変数 cwd が変更されるたびに、その値を 環境変数 PWD へ設定します。

環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、LC_COLLATE、LC_NUMERIC、LC_MONETARY は、C シェル内で変更されると新しい値が即座に有効になります。 これらの環境変数の詳細については environ(5) を参照してください。

unsetenv は環境から *variable* が示す変数を削除します。 unset のようなパターンマッチングは行いません。

ksh set コマンドのフラグの意味は以下のとおりです。

-A 配列の代入。 *name* で示される変数の設定を解除し、 *arg* リストから順々に値を割り当てます。 +A は、 最初の変数設定を解除しません。

-a 定義される後続の変数すべてを自動的にエクスポートします。

-e コマンドの終了状態が 0 でない場合、 ERR トラップ (設定されていれば) を実行し、 終了します。 このモードは、 プロファイル読み取り時は無効です。

-f ファイル名の生成を無効にします。

`-h` 各コマンドは、 最初に検出された時点で、 検索済み別名になります。

`-k` コマンド名に先行するものだけでなく、 すべての変数代入引数をコマンドの環境に格納します。

`-m` バックグラウンドジョブを個別のプロセスグループで実行し、 完了時にメッセージを 1 行表示します。 バックグラウンドジョブの終了状態は 完了メッセージで報告されます。 ジョブ制御を備えたシステムでは、 このフラグは 対話型シェルに対して自動的に有効になります。

`-n` コマンドを読み取り、 構文エラーがないかチェックします。 実行は行いません。 対話型シェルに対しては無視されます。

`-o` このフラグの後に指定する引数は、 以下のオプション名のいずれかです。

`allexport`	`-a` と同じです。
`errexit`	`-e` と同じです。
`bgnice`	バックグラウンドジョブをすべて低い優先度で実行します。 これはデフォルトモードです。
`emacs`	コマンド入力用に、 `emacs` 形式のインラインエディタを起動します。
`gmacs`	コマンド入力用に、 `gmacs` 形式のインラインエディタを起動します。
`ignoreeof`	ファイルの終わりを検出してもシェルは終了しません。 終了させるには `exit` コマンドを使用する必要があります。
`keyword`	`-k` と同じです。
`markdirs`	ファイル名生成によって生成されるディレクトリ名には、 すべて最後に `/` を付加します。
`monitor`	`-m` と同じです。
`noclobber`	`>` によるリダイレクトが存在するファイルを切り捨てないようにします。 このオプションが有効なとき、 ファイルを切り捨てるには `>|` 指定が必要です。
`noexec`	`-n` と同じです。
`noglob`	`-f` と同じです。
`nolog`	履歴ファイルに関数定義を保存しません。
`nounset`	`-u` と同じです。
`privileged`	`-p` と同じです。
`verbose`	`-v` と同じです。
`trackall`	`-h` と同じです。

vi　vi 形式のインラインエディタの挿入モードになります。 033 というエスケープ文字を押すと、 挿入モードから制御モードに変わります。 Return で行を送信します。

viraw　各文字を vi モードで入力されたときと同様に処理します。

xtrace　-x と同じです。

オプション名を指定しない場合、 -o は現在のオプション設定を表示します。

-p　$HOME/.profile ファイルを処理しないようにし、 ENV ファイルの代わりに /etc/suid_profile ファイルを使用します。 このモードは、 実効ユーザー ID が実ユーザー ID と等しくないとき、また実効グループ ID が実グループ ID と等しくないときには 必ず有効になります。 このモードを無効にすると、 実効ユーザー ID が実ユーザー ID に、 実効グループ ID が実グループ ID にそれぞれ設定されます。

-s　定位置パラメタを辞書編集方式の順にソートします。

-t　コマンド 1 つを読み取って実行し、 終了します。

-u　置換を行う際に、設定されていないパラメタを エラーとして扱います。

-v　シェルへの入力行を読み取り時に表示します。

-x　コマンドとその引数を実行時に表示します。

-　-x フラグと -v フラグを無効にし、 フラグに対する引数の検査を停止します。

- -　どのフラグも変更しません。 このフラグは、 $1を - で始まる値に設定する際に便利です。 このフラグの後に引数がない場合、 定位置パラメタが設定解除されます。

- の代わりに + を使用すると、 これらのフラグが無効になります。 これらのフラグはシェル起動時にも使用できます。 現在設定されているフラグは、 $- で見ることができます。 -A を指定しないかぎり、 残りの引数は定位置パラメタとなり、 $1 $2 ... に順番に割り当てられます。 引数を 1 つも指定しない場合には、 すべての変数の名前と値を標準出力上に表示します。

name が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、 MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

unset を使用すると *name* が示す変数の設定を解除します。 つまり、 それらの変数の値と属性を消去します。 読み取り専用の変数は設定を解除できません。 -f フラグが指定されていると、 *name* 引数は関数名を表します。 ERRNO、 LINENO、

MAILCHECK、 OPTARG、 OPTIND、 RANDOM、 SECONDS、 TMOUT、および _ の設定を解除すると、 これらの変数の特殊な意味が削除されます。 後でこれらの変数に値を代入しても、 特殊な意味はないままです。

export 組み込みコマンドでは、 指定された *name* に対し、 後で実行されるコマンドの環境へ 自動的にエクスポートされるようにマークを付けます。

1 つまたは 2 つの (*) アスタリスクが先頭に付加されている ksh(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

csh 次の例では、 /bin、/usr/bin、/usr/sbin、/usr/ucb/bin ディレクトリにあるファイルをその順番で検索するために、 環境変数 PATH を設定しています。

```
setenv PATH "/bin:/usr/bin:/usr/sbin:usr/ucb/bin"
```

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

csh(1), ksh(1), read(1), sh(1), typeset(1), attributes(5), environ(5)

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目 `intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | vacation – メールへの自動返信

形式 | **vacation** [-I]

vacation [-a *alias*] [-f *database_file*] [-j] [-m *message_file*] [-s *sender*] [-t*N*] *username*

機能説明 | vacation ユーティリティは、着信したメールに自動的に返信します。

インストール | インストールは、vacation の基本的な構成をセットアップする対話型プログラムで行います。

vacation をインストールするには、コマンド行で引数を指定しないで vacation と入力します。これで、.vacation.msg ファイルが生成されます。このファイルには、vacation を有効にしたときに、受信したメールの送信元に自動返信するメッセージが記述されています。.vacation.msg ファイルの生成語、メッセージの内容を編集するエディタが起動します (「使用法」の項を参照)。起動するエディタの種類は、環境変数 VISUAL または EDITOR で指定できます。この環境変数を設定していないと、vi(1) が選択されます。

.forward ファイルがホームディレクトリにない場合は、このファイルも生成されます。生成された .forward ファイルには 次のような形式の行が含まれます。

username, "|/usr/bin/vacation *username*"

この行の指定に従って、着信したメールのコピーが *username* に送られ、もう 1 つのコピーが vacation ユーティリティにパイプされます。

.forward ファイルがホームディレクトリにある場合は、このファイルを削除するかどうかを確認するメッセージが表示されます。ここでファイルを削除すると、vacation は無効になります。また、このツールのインストールも終了します。

vacation を有効にすると、不在時にメールを送信する宛先の一覧を記述する .vacation.pag ファイルと .vacation.dir が生成されます。

有効、無効の切り替え | vacation の有効、無効は、.forward ファイルの有無によって決定します。vacation を無効にするには、.forward ファイルを削除するか、別の名前に変更します。

初期化 | -I オプションは、以前の vacation セッションで使用した送信者の一覧を削除します。また、vacation のログファイル .vacation.pag と .vacation.dir の内容も消去します (「オプション」の項を参照)。

追加設定 | vacation には、構成オプション -a、-f、-j、-m、-s、-t が用意されています。これは、インストール時の構成には含まれません (「オプション」の項を参照)。

オプション | 次のオプションを指定できます。

-I　.vacation.pag ファイルと .vacation.dir ファイルを初期化し、vacation を有効にします。このフラグを指定しないで

username 引数を指定すると、標準入力から 1 行目 (コロンのない From 行) が読み取られます。指定しない場合、エラーメッセージが生成されます。

オプション -a、-f、-j、-m、-t、-s は、コマンド行ではなく、.forward ファイル中で使用する構成オプションです。たとえば、1 分おきに送信者に再返信を繰り返すように設定する場合は、次のようにします。

```
\username, "|/usr/bin/vacation -t1m username"
```

-a *alias*
: *alias* に、vacation を実行しているユーザーの有効な別名を指定します。これで、別名宛のメールにも、自動返信が行われます。

-f *file*
: データベースファイルのベース名として、.vacation の代わりに *file* を使用します。

-j
: vacation を実行しているユーザーの名前が、受信したメールの To: 行または Cc: 行に記述されているかどうかのチェックを行いません。このオプションを使用すると、メーリングリストなど、適切でない宛先にメールが返信されてしまうことがあります。どうしても必要な場合以外は、このオプションを使用しないでください。

-m *file*
: 返信用メッセージとして、.vacation.msg ファイルの代わりに *file* を使用します。

-s *sender*
: 受信したメッセージの From 行に示されている受信者ではなく、*sender* に返信します。

-t*N*
: 同一の送信元に再返信を繰り返す際の返信間隔を変更します。デフォルト値は 1 週間です。*N* の後に、時間の単位を表す文字を指定します。s は秒、m は分、h は時、d は日、w は週を表します。

ファイル

.vacation.msg ファイルにはヘッダーが必要です。また、このヘッダーには、必ず Subject: 行が含まれていなければなりません (From: 行と To: 行はヘッダーに含めない)。たとえば、次のように作成します。

```
Subject: I am on vacation
I am on vacation until July 22.  If you have something urgent,
please contact Joe Jones (jones@fB0).
       --John
```

.vacation.msg ファイル内の文字列 $SUBJECT は、メッセージの返信時に、受信した元のメッセージの表題に置き換えられます。次に、返信メッセージに受信したメッセージの表題を含む例を示します。

```
Subject: I am on vacation
I am on vacation until July 22.
Your mail regarding "$SUBJECT" will be read when I return.
If you have something urgent, please contact
Joe Jones (jones@fB0).
```

```
--John
```

vacation を使用しているユーザー名やその別名が、受信した元のメッセージの To: 行または Cc: 行に指定されていない場合、返信メッセージは送信されません。また、最初の From 行に文字列 -REQUEST@ が含まれている場合、ヘッダーに Precedence: bulk または Precedence: junk という記述が含まれている場合も、返信メッセージは送信されません。

vacation は、postmaster や Mailer-Daemon からのメールにも返信しません。

ファイル

~/.forward

~/.vacation.msg

送信者の一覧が、ユーザーのホームディレクトリの .vacation.pag と .vacation.dir に保存されます。これらのファイルは dbm 形式です。したがって、テキストエディタでは直接参照できません。

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

vi(1)、sendmail(1M)、dbm(3UCB)、getusershell(3C)、aliases(4)、shells(4)、attributes(5)

名前 | ckdate, errdate, helpdate, valdate – 日付の入力要求とその検証

形式 | **ckdate** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errdate [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helpdate [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valdate [-f *format*] *input*

機能説明 | ckdate ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに日付の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザーの応答は、date コマンドで定義されている書式に一致したものでなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckdate コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errdate (エラーメッセージを書式化して表示する)、helpdate (ヘルプメッセージを書式化して表示する)、および valdate (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errdate および helpdate の各モジュールに format が定義されている場合、メッセージには、指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は、どのような書式も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。指定できる書式とその定義は、次のとおりです。

- %b = 省略された月の名前 (jan、feb、mar)
- %B = 完全な月の名前 %d = 日 (01 ～ 31)
- %D = %m/%d/%y の形式の日付 (デフォルトの書式)
- %e = 日 (1 ～ 31、1 桁の数字の前にはブランクが挿入されます)

`%h` = 省略形の月の名前。`%b`%と同じ

`%m` = 月 (01 〜 12)

`%y` = 西暦中の年 (たとえば、89)

`%Y` = CCYY の形式の年 (たとえば、1989)

`-h` *help* `help` をヘルプメッセージとして定義します。

`-k` *pid* ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

`-p` *prompt* *prompt* をプロンプトメッセージとして定義します。

`-Q` 有効な応答として終了 (quit) を使用できないように指定します。

`-s` *signal* 終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル `signal` を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-W` *width* プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、`width` の行長に書式化します。

オペランド

次のオペランドを指定できます。

input 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0` 正常な終了

`1` 入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`3` ユーザー終了 (quit)

`4` 不正な書式引数

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckdate` のデフォルトのプロンプトは、次のとおりです。

```
Enter the date [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter a date.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter a date. Format is <format>.
```

終了 (quit) オプションを選択した場合 (かつ使用できる場合)、リターンコード 3 と共に q が返されます。valdate モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前 | ckgid, errgid, helpgid, valgid – グループ ID の入力要求とその検証

形式 | **ckgid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errgid [-W *width*] [-e *error*]

/usr/sadm/bin/helpgid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valgid *input*

機能説明 | ckgid は、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに既存のグループ ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckgid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errgid (エラーメッセージを書式化して表示する)、helpgid (ヘルプメッセージを書式化して表示する)、および valgid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような書式基準も満たす必要はありません。

-e *error* — *error* エラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するように指定します。

-m — ユーザーがヘルプを要求した場合、あるいはユーザーの入力が不正な場合は、グループ名の一覧を表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/group` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckgid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the name of an existing group [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

デフォルトのヘルプメッセージは、次のとおりです。

```
ERROR: Please enter one of the following group names: [List]
```

`ckgid` の `-m` オプションを使用すると、有効なグループのリストがここに表示されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valgid` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckint, errint, helpint, valint – 整数値の入力要求とその検証

形式 | **ckint** [-Q] [-W *width*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errint [-W *width*] [-b *base*] [-e *error*]

/usr/sadm/bin/helpint [-W *width*] [-b *base*] [-h *help*]

/usr/sadm/bin/valint [-b *base*] *input*

機能説明 | ckint ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーに整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーがキャリッジリターンで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義中の空白（復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckint コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errint (エラーメッセージを書式化して表示する)、helpint (ヘルプメッセージを書式化して表示する)、および valint (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errint と helpint の各モジュールに *base* が定義されている場合、メッセージには入力する整数の基数が含まれます。

オプション | 次のオプションを指定できます。

-b *base* — 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。

-d *default* — *default* をフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

`-s` *signal*　終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。

`-W` *width*　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　基数 *base* の基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　正常な終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは、使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckint` のデフォルトのプロンプトは、次のとおりです (基数は 10)。

```
Enter an integer [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
ERROR - Please enter an integer.
```

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
Please enter an integer.
```

基数を 10 以外の数に設定した場合は、上記のメッセージは "integer" から "base *base* integer" に変更されます。

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。`valint` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckpath, errpath, helppath, valpath – パス名の入力要求とその検証

形式 | **ckpath** [-Q] [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errpath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-e *error*]

/usr/sadm/bin/helppath [-W *width*] [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] [-h *help*]

/usr/sadm/bin/valpath [-a | l] [-b | c | f | y] [-n [o | z]] [-rtwx] *input*

機能説明 | ckpath ユーティリティは、ユーザーに入力を要求するプロンプトを出力し、ユーザーの応答を検証します。このユーティリティでは、ユーザーにパス名の入力を促すプロンプトメッセージ、ヘルプメッセージおよびエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

パス名は、最初のオプショングループで指定した基準に従わなければなりません。基準が定義されていない場合、パス名はまだ存在していない通常のファイルに対するものでなければなりません。-a (絶対) と -l (相対) のいずれも指定されていない場合は、どちらかが有効であるとみなされます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckpath コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errpath (標準出力上にエラーメッセージを書式化して表示する)、helppath (標準出力上にヘルプメッセージを書式化して表示する)、および valpath (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-a — パス名は絶対パスでなければなりません。

-b — パス名はブロック型特殊ファイルでなければなりません。

-c — パス名は文字型特殊ファイルでなければなりません。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

`-e` *error*	*error* をエラーメッセージとして定義します。
`-f`	パス名は通常ファイルでなければなりません。
`-h` *help*	*help* をヘルプメッセージとして定義します。
`-k` *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
`-l`	パス名は相対パスでなければなりません。
`-n`	パス名が存在していてはなりません (新規のものでなければなりません)。
`-o`	パス名が存在していなければなりません (既存のものでなければなりません)。
`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (`quit`) を使用できないようにします。
`-r`	パス名は読み取り可能でなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-t`	パス名は作成可能 (処理可能) でなければなりません。パス名が、まだ存在していない場合には作成されます。
`-w`	パス名は書き込み可能でなければなりません。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。
`-x`	パス名は実行可能でなければなりません。
`-y`	パス名はディレクトリでなければなりません。
`-z`	パス名には、`0` バイトを超えるサイズのファイルがなければなりません。

オペランド

次のオペランドを指定できます。

input	検証オプションと照合される入力

使用例

`ckpath` のデフォルトメッセージのテキストは、使用されている基準オプションによって異なります。

例 1 デフォルトのプロンプト

`ckpath` (`-a` オプションを使用) のデフォルトプロンプトの例は、次のとおりです。

```
example% ckpath -a
Enter an absolute pathname [?,q]
```

例 1 デフォルトのプロンプト　(続き)

例 2 デフォルトのエラーメッセージ

デフォルトのエラーメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/errpath -a
ERROR: A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 3 デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージ (-a オプションを使用) の例は、次のとおりです。

```
example% /usr/sadm/bin/helppath -a
A pathname is a filename, optionally preceded by parent directories.
The pathname you enter: - must begin with a slash (/)
```

例 4 終了オプション

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード 3 と共に q が返されます。終了を入力すると、後に復帰改行が続きます。

例 5 valpath モジュールの使用

valpath モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valpath
usage: valpath [-[a|l][b|c|f|y][n|[o|z]]rtwx] input
    .
    .
    .
```

終了ステータス

次の終了ステータスが返されます。

0	正常終了
1	入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった
2	同時に指定できないオプション
3	ユーザー終了 (quit)
4	同時に指定できないオプション

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 face(1), signal(3HEAD), attributes(5)

名前 | ckrange, errange, helprange, valrange – 整数の入力要求とその検証

形式 | **ckrange** [-Q] [-W *width*] [-l *lower*] [-u *upper*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errange [-W *width*] [-e *error*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/helprange [-W *width*] [-h *help*] [-l *lower*] [-u *upper*] [-b *base*]

/usr/sadm/bin/valrange [-l *lower*] [-u *upper*] [-b *base*] *input*

機能説明 | ckrange ユーティリティは、ユーザーに指定範囲内の整数の入力を要求して、ユーザーの応答が有効かどうかを検証します。このユーティリティでは、ユーザーに指定範囲内の整数の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

また、このコマンドは、有効な入力範囲も定義します。下限または上限のいずれかが定義されていない場合、範囲は一方の限界だけに制限されます。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckrange コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errrange (標準出力上にエラーメッセージを書式化して表示する)、helprange (標準出力上にヘルプメッセージを書式化して表示する)、および valrange (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

注：*input* 引数として負の値を指定すると、valrange 内の getopt で混乱が生じます。引数の前に - を付けると、getopt は処理を停止します。getopt のパラメータ処理については、getopt(1) および intro(1) のマニュアルページを参照してください。getopt は、位置指定パラメータを解析して、有効なオプションかどうかを検査するために使用されます。

オプション | 次のオプションを指定できます。

-b *base* | 入力する整数の基数を定義します。2 から 36 までの数字でなければなりません。デフォルト値は 10 です。基数を変換するには strtol(3C) を使用します。出力は常に、基数を 10 として実行されます。

-d *default*	*default* をデフォルト値として定義します。*default* は strtol(3C) を使用して指定した基数に変換できます。指定した基数に無効な文字があると、strtol はエラーを示すことなく変換を終了します。
-e *error*	*error* をエラーメッセージとして定義します。
-h *help*	*help* をヘルプメッセージとして定義します。
-k *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
-l *lower*	*lower* を範囲の下限として定義します。デフォルト値は、マシンの負の最大整数です。
-p *prompt*	*prompt* をプロンプトメッセージとして定義します。
-Q	有効な応答として終了 (quit) を使用できないようにします。
-s *signal*	終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。
-u *upper*	*upper* を範囲の上限として定義します。デフォルト値は、マシンの正の最大整数です。
-W *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定します。

input	上限、下限、および基数と照合される入力

使用例

例 1 デフォルトのプロンプト

ckrange のデフォルトのプロンプトは、次のとおりです (基数は 10)

```
example% ckrange Enter an integer between lower_bound and
upper_bound [lower_bound–upper_bound,?,q]:
```

例 2 デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/errange
ERROR: Please enter an integer between lower_bound \
     and upper_bound.
```

例 3 デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージは、次のとおりです (基数は 10)。

```
example% /usr/sadm/bin/helprange
Please enter an integer between lower_bound and upper_bound.
```

例 **3** デフォルトのヘルプメッセージ　*(続き)*

例 **4** 基数を 10 以外の数に変更した場合のメッセージ

基数を 10 以外の数に設定した場合、メッセージは "integer" から "base *base* integer" に変更されます。次に例を示します。

```
example% /usr/sadm/bin/helprange -b 36
```

例 **5** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **6** valrange モジュールの使用

`valrange` モジュールは、標準エラー出力に使用法に関するメッセージを作成します。正常終了したな場合は `0`、失敗した場合には `0` 以外の値を返します。

```
example% /usr/sadm/bin/valrange
usage: valrange [-l lower] [-u upper] [-b base] input
```

終了ステータス

次の終了ステータスが返されます。

`0`　正常終了

`1`　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった

`2`　使用法に誤りがあった

`3`　ユーザー終了 (quit)

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`intro`(1), `face`(1), `getopt`(1), `strtol`(3C), `attributes`(5), `signal`(3HEAD)

名前 | ckstr, errstr, helpstr, valstr – 文字列の入力要求とその検証

形式 |

ckstr [-Q] [-W *width*] [[-r *regexp*] [...]] [-l *length*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [- s *signal*]]

/usr/sadm/bin/errstr [-W *width*] [-e *error*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/helpstr [-W *width*] [-h *help*] [-l *length*] [[-r *regexp*] [...]]

/usr/sadm/bin/valstr [-l *length*] [[-r *regexp*] [...]] *input*

機能説明 | ckstr ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、文字列の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

このコマンドから返される応答は、定義済みの正規表現に一致していなければならず、指定された長さ以内でなければなりません。正規表現が指定されていない場合、有効な入力は、内部に空白を含まず、また先行や後続する空白がない、指定された長さ以内の文字列でなければなりません。長さが指定されていない場合、長さは検査されません。

メッセージの長さはすべて最大 79 文字に制限され、自動的に書式化されます。メッセージ定義内で 1 つの空白文字の後にあるタブと復帰改行は削除されますが、空白は削除されません。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「使用例」の項を参照) が表示されます。

ckstr コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errstr (標準出力上にエラーメッセージを書式化して表示する)、helpstr (標準出力上にヘルプメッセージを書式化して表示する)、および valstr (応答を検証する) です。これらのモジュールは、フレームドアクセスコマンド環境 (FACE) オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-l *length* — 入力の最大長を指定します。

`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (quit) を使用できないようにします。
`-r` *regexp*	入力を検証するときの基準となる正規表現、 *regexp* を指定します。空白を含めることができます。複数の式が定義されている場合、応答はその内のいずれかに一致しなければなりません。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを *width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input	書式の長さや正規表現の基準に対して検証される入力

使用例

例 **1** デフォルトのプロンプト

`ckstr` のデフォルトのプロンプトは、次のとおりです。

```
example% ckstrEnter an appropriate value [?,q]:
```

例 **2** デフォルトのエラーメッセージ

デフォルトのエラーメッセージは、適用される妥当性検査のタイプによって異なります。ユーザーには、長さまたはパターン照合のいずれかが失敗したことが通知されます。デフォルトのエラーメッセージは、次のとおりです。

```
example% /usr/sadm/bin/errstr
ERROR: Please enter a string which contains no embedded,
leading or trailing spaces or tabs.
```

例 **3** デフォルトのヘルプメッセージ

デフォルトのヘルプメッセージも、適用される妥当性検査のタイプによって異なります。正規表現が定義されている場合、メッセージは次のようになります。

```
example% /usr/sadm/bin/helpstr -r regexp
Please enter a string which matches the following pattern:
regexp
```

他のメッセージは、文字列の長さの要件と定義を指定します。

例 **4** 終了 (quit) オプションの使用

終了オプションを選択した場合 (かつ使用できる場合)、リターンコード `3` と共に `q` が返されます。終了を入力すると、後に復帰改行が続きます。

例 **5** valstr モジュールの使用

valstr モジュールは、標準エラー出力に使用法に関するメッセージを出力します。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

```
example% /usr/sadm/bin/valstr
usage: valstr [-l length] [[-r regexp] [ . . . ]] input
```

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは、使用法に誤りがあった

2 無効な正規表現

3 ユーザー終了 (quit)

属性

次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

face(1), signal(3HEAD), attributes(5)

名前 | cktime, errtime, helptime, valtime – 時刻の入力要求とその検証

形式 | **cktime** [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/errtime [-W *width*] [-e *error*] [-f *format*]

/usr/sadm/bin/helptime [-W *width*] [-h *help*] [-f *format*]

/usr/sadm/bin/valtime [-f *format*] *input*

機能説明 | cktime ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに時刻の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。ユーザー応答は、時刻について定義されている書式に一致しなければなりません。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

cktime コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、errtime (エラーメッセージを書式化して表示する) と helptime (ヘルプメッセージを書式化して表示する) と、valtime (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。errtime および helptime の各モジュールに format が定義されている場合、メッセージには指定した書式も含まれます。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-f *format* — 入力を検証するときの基準となる書式を指定します。次の書式と定義を指定できます。

```
%H  =  hour (00 - 23)
%I  =  hour (00 - 12)
%M  =  minute (00 - 59)
%p  =  ante meridian or post meridian
%r  =  time as %I:%M:%S %p
%R  =  time as %H:%M (the default format)
%S  =  seconds (00 - 59)
%T  =  time as %H:%M:%S
```

`-h` *help*	*help* をヘルプメッセージとして定義します。
`-k` *pid*	ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。
`-p` *prompt*	*prompt* をプロンプトメッセージとして定義します。
`-Q`	有効な応答として終了 (quit)を使用できないようにします。
`-s` *signal*	終了が選択された場合、`-k` オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、`SIGTERM` を送信します。
`-W` *width*	プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input 書式基準と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`	正常終了
`1`	入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、あるいは使用法に誤りがあった
`3`	ユーザー終了 (quit)
`4`	無効な書式引数

属性

次の属性については、`attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`cktime` のデフォルトのプロンプトは、次のとおりです。

```
Enter a time of day [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR: Please enter the time of day.  Format is <format>.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the time of day.  Format is <format>.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。`valtime` モジュールは、出力を生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckuid, erruid, helpuid, valuid – ユーザー ID の入力要求とその検証

形式 | **ckuid** [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erruid [-W *width*] [-e *error*]

/usr/sadm/bin/helpuid [-W *width*] [-m] [-h *help*]

/usr/sadm/bin/valuid *input*

機能説明 | ckuid ユーティリティは、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、ユーザーに既存のユーザー ID の入力を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義できます。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckuid コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erruid (エラーメッセージを書式化して表示する) と helpuid (ヘルプメッセージを書式化して表示する) と、valuid (応答を検証する) です。これらのモジュールは、FML オブジェクトと組み合わせて使用する必要があります。この場合、FML オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* をデフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-m — ヘルプが要求された場合、またはユーザーがエラーを犯した場合は、すべてのログインのリストを表示します。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスに、シグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

`-W` *width*　　プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input　　`/etc/passwd` と照合される入力

終了ステータス

次の終了ステータスが返されます。

`0`　　正常終了

`1`　　入力で `EOF` が検出された、`-W` オプションで負の行長が指定された、または使用法に誤りがあった

`2`　　使用法に誤りがあった

`3`　　ユーザー終了 (quit)

属性

次の属性については、 `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`attributes`(5)

注意事項

`ckuid` のデフォルトのプロンプトは、次のとおりです。

```
Enter the login name of an existing user [?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのエラーメッセージは、次のとおりです。

```
ERROR:  Please enter one of the following login names:  <List>
```

デフォルトのヘルプメッセージは、次のとおりです。

```
Please enter the login name of an existing user.
```

`-m` オプションを使用した場合のデフォルトのヘルプメッセージは、次のとおりです。

```
Please enter one of the following login names:  <List>
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード `3` と共に `q` が返されます。 `valuid` モジュールは出力を何も生成しません。正常終了した場合は `0`、失敗した場合には `0` 以外の値を返します。

名前 | ckyorn, erryorn, helpyorn, valyorn – yes/no の入力要求とその検証

形式 | **ckyorn** [-Q] [-W *width*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

/usr/sadm/bin/erryorn [-W *width*] [-e *error*]

/usr/sadm/bin/helpyorn [-W *width*] [-h *help*]

/usr/sadm/bin/valyorn *input*

機能説明 | ckyorn は、ユーザーに入力を要求してその応答を検証します。このユーティリティでは、「はい (yes)」または「いいえ (no)」の応答を促すプロンプトメッセージ、ヘルプメッセージとエラーメッセージ、およびデフォルト値 (ユーザーが RETURN キーで応答した場合に返される値) をオプションにより定義します。

メッセージの長さはすべて最大 70 文字に制限され、自動的に書式化されます。定義で使用される空白 (復帰改行を含む) はすべて削除されます。-W オプションは、自動書式化を取り消します。メッセージ定義の最初または最後にチルド文字がある場合は、そこにデフォルトテキストが挿入されて、指定したテキストとデフォルトテキストの両方を表示することができます。

プロンプトメッセージ、ヘルプメッセージ、またはエラーメッセージが定義されていない場合は、デフォルトメッセージ (「注意事項」の項を参照) が表示されます。

ckyorn コマンドには、3 つのビジュアルツールモジュールがリンクされています。これらのモジュールは、erryorn (エラーメッセージを書式化して表示する) と helpyorn (ヘルプメッセージを書式化して表示する) と、valyorn (応答を検証する) です。これらのモジュールは、FACE オブジェクトと組み合わせて使用する必要があります。この場合、FACE オブジェクトはプロンプトを定義します。

オプション | 次のオプションを指定できます。

-d *default* — *default* デフォルト値として定義します。デフォルト値は検証されないため、どのような基準も満たす必要はありません。

-e *error* — *error* をエラーメッセージとして定義します。

-h *help* — *help* をヘルプメッセージとして定義します。

-k *pid* — ユーザーが中断を選択した場合、プロセス ID *pid* のプロセスにシグナルを送信するようにします。

-p *prompt* — *prompt* をプロンプトメッセージとして定義します。

-Q — 有効な応答として終了 (quit) を使用できないようにします。

-s *signal* — 終了が選択された場合、-k オプションで定義されたプロセス ID *pid* のプロセスにシグナル *signal* を送信するようにします。シグナルを指定しないと、SIGTERM を送信します。

-W *width* — プロンプトメッセージ、ヘルプメッセージ、およびエラーメッセージを、*width* の行長に書式化します。

オペランド

次のオペランドを指定できます。

input y、yes、または n、no (大文字小文字の任意の組み合わせ) に対して検証される入力

終了ステータス

次の終了ステータスが返されます。

0 正常終了

1 入力で EOF が検出された、-W オプションで負の行長が指定された、あるいは使用法に誤りがあった

2 使用法に誤りがあった

3 ユーザー終了 (quit)

属性

次の属性については、 attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

attributes(5)

注意事項

ckyorn のデフォルトのプロンプトは、次のとおりです。

```
Yes or No [y,n,?,q]:
```

デフォルトのエラーメッセージは、次のとおりです。

```
ERROR - Please enter yes or no.
```

デフォルトのヘルプメッセージは、次のとおりです。

```
To respond in the affirmative, enter y, yes, Y, or YES.
To respond in the negative, enter n, no, N, or NO.
```

終了オプションを選択した場合 (かつ使用できる場合) は、リターンコード 3 と共に q が返されます。valyorn モジュールは、出力を生成しません。正常終了した場合は 0、失敗した場合には 0 以外の値を返します。

名前

vi, view, vedit – ex に基づいたスクリーン指向の (ビジュアル) ディスプレイエディタ

形式

/usr/bin/vi [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/bin/view [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/bin/vedit [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/vi [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/view [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/vedit [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

機能説明

vi (ビジュアル) は、ラインエディタ ex に基づいたディスプレイ指向のテキストエディタです。vi の中から ex のコマンドモードを使用することも、ex の中から vi のコマンドモードを使用することも可能です。ビジュアルコマンドは、本マニュアルページで説明しています。オプションの設定の仕方 (自動行番号付けや、改行を押したときの新しい出力行の開始など)、および ex ラインエディタのすべてのコマンドは、ex(1) で説明しています。

vi を使用すると、ユーザーがファイルへ行う変更は、ユーザーが見ている端末スクリーンに反映されます。スクリーン上のカーソルの位置は、ファイルの中の位置を示します。

view 呼び出しは、readonly (読み取り専用) フラグを設定することを除くと vi と同じです。

vedit 呼び出しは、初心者を想定したものです。基本的に vi と同じですが、フラグを 1 に設定し、showmode および novice フラグを設定し、magic をオフにします。このようなデフォルトに設定することによって、vi の使い方がより分かりやすくなります。

呼び出しオプション

以下の呼び出しオプションは、vi によって解釈されます (以前に文書化されていたオプションについては、以降の「注意事項」で説明しています)。

– | -s
: フィードバックのすべてを抑制します。編集スクリプト実行時に役に立ちます。

-C
: -x オプションと同様に、暗号オプションです。 ただし、vi は ex の c コマンドをシミュレートします。c コマンドは x コマンドに似ていますが、読み取られたテキストは、すべて暗号化されていると仮定します。

`-l`	LISP プログラムの編集のための設定を行います。
`-L`	エディタやシステムのクラッシュの結果として保存されたファイル名をすべてリストします。
`-r` *filename*	エディタやシステムのクラッシュ後、*filename* を編集します (クラッシュが発生したときにバッファの中にあった *filename* のバージョンを回復します)。
`-R`	`Readonly` (読み取り専用モード)。このフラグを設定すると、ファイルに誤って上書きすることを防止できます。
`-S`	このオプションは `-t` *tag* オプションと合わせて指定します。タグファイルがソートされていない場合や、バイナリ検索 (ソートされたタグファイルに依存) がタグの検索に失敗するときに遅延リニア検索も行うように、`vi` に伝えます。リニア検索は遅いため、大規模なタグファイルを処理する場合には、このフラグを使用するよりもソートされたタグファイルを使用するようにしてください。通常、タグファイルを作成すると、タグファイルはソートされて生成されます。タグファイルについての詳細は、`ctags`(1) を参照してください。
`-t` *tag*	*tag* を含むファイルを編集し、その定義に従ってエディタの編集位置を設定します。
`-v`	`vi` を使った編集画面を表示します。単に `vi` コマンドを入力しても同じように実行できます。
`-V`	冗長。`ex` コマンドが標準入力によって読み込まれる場合に、その入力を標準エラーに反映します。シェルスクリプト中の `ex` コマンドを実行する場合に役に立ちます。
`-w`*n*	デフォルトのウィンドウサイズを *n* に設定します。これは、低速の回線でエディタを使用するときに役に立ちます。
`-x`	暗号オプション。これを使用すると、`ex` の `X` コマンドをシミュレートし、ユーザーに対してプロンプトを出し、1 つのキーの入力を求めます。このキーを使って、`crypt` コマンドのアルゴリズムに従い、テキスト

を暗号化したり、暗号を解除したりし ます。x コマンドは、読み取ったテキストが暗号化されているかどうかを判別します。また、一時バッファファイルも -x オプションに対して入力したキーの変形バージョンを用いて暗号化されます。空の暗号化キーが入力された場合 (つまり、プロンプトに対してリターンを返した場合)、ファイルは暗号化されません。これは、Back Space キーや Undo キーなど、暗号化キーの入力ミスによって、誤って暗号化されてしまったファイルを解読するための回避策です。

+*command* | -c *command*　指定されたエディタコマンド *command* (通常は検索コマンドまたは位置設定コマンド) を実行し、編集を始めます。

/usr/xpg4/bin/vi

-t *tag* と -c *command* の両方のオプションを指定した場合、-t *tag* オプションがが始めに実行されます。つまり、*tag* のついたファイルを -t によって選んでから、コマンドが実行されます。

オペランド

次のオペランドを指定できます。

filename　編集の対象となるファイル

vi のモード

コマンド　標準モードおよび初期モードです。他のモードは完了すると、このコマンドモードへ戻ります。コマンドの入力を中断するには ESC (エスケープ) を使用します。

入力

a A i I o O c C s S Rのいずれかを入力すると、任意のテキストを入力できます。入力モードの終了は、通常、ESC 文字で行います。ただし、割り込みでも異常終了できます。

最終行　: / ? または ! は読み取り入力です。キャリッジリターンで終了します。割り込みで終了をキャンセルします。

サンプルコマンド

説明では、CR はキャリッジリターンを表し、ESC はエスケープキーを表します。

←
→
下矢印
上矢印　矢印キーはカーソル移動

h j k l　矢印キーと同じ

i*text*ESC　*text* を挿入

cw*new*ESC　単語を *new* に変更

	easESC	単語を複数形に (単語の終わり; s の追加 ; 入力状態からエスケープ)
	x	文字の削除
	dw	単語の削除
	dd	行の削除
	3dd	3 行削除
	u	前回の変更を取り消し
	ZZ	変更を保存し、vi を終了
	:q!CR	変更を保存せず、vi を終了
	/*text*CR	*text* の検索
	^U ^D	スクロールアップとスクロールダウン
	:*cmd*CR	ex または ed のコマンド
vi コマンドの前の数	数を接頭辞としてコマンドの前に入力することができます。入力された数は、次のいずれかの方法で解釈されます。	
	行/カラム数	z G |
	スクロール量	^D ^U
	繰り返し	その他のほとんどのコマンド
割り込み、取り消し	ESC	挿入や途中のコマンドの終了
	DEL	(delete または rubout) 割り込み
ファイル操作	ZZ	ファイルが変更されていれば、書き込みと終了。それ以外、終了のみ
	:wCR	変更の書き込み
	:w!CR	アクセス権が有効でないときの強制書き込み
	:qCR	終了
	:q!CR	変更は無視して、終了
	:e *name*CR	ファイル *name* の編集
	:e!CR	変更は無視して、再編集
	:e + *name*CR	ファイルの最後から編集開始
	:e +*n*CR	ファイルの *n* 行目から編集開始
	:e #CR	代替ファイルの編集
	:e! #CR	変更は無視して、代替ファイルの編集
	:w *name*CR	ファイル *name* に書き込み

	:w! *name*CR	ファイル *name* に上書き
	:shCR	シェルの実行と復帰
	:!*cmd*CR	*cmd* の実行と復帰
	:nCR	引数リストの次のファイルを編集
	:n *args*CR	新しい引数リストを指定
	^G	現在のファイルと行の表示
	:ta *tag*CR	カーソルは *tag* の位置

ex または ed コマンド (*substitute* または *global* など) も入力することができます。このときは、前にコロン、後ろにキャッリッジリターンを入力します。

ファイルの中での位置決め	F	画面の先送り
	^B	画面の後戻し
	^D	半画面スクロールダウン
	^U	半画面スクロールアップ
	*n*G	指定した行の行頭に移動 (デフォルトは文末) *n* は行番号
	/*pat*	*pat* に一致する行 (前方)
	?*pat*	*pat* に一致する行 (後方)
	n	前回の / または ? コマンドの繰り返し
	N	前回の / または ? コマンドの逆転
	/*pat*/+*n*	*n* 回目に *pat* に一致する行 (前方)
	?*pat*?–*n*	*n* 回目に *pat* に一致する行 (後方)
	]]	次の節または関数
	[[	前の節または関数
	(	文の始まり
	)	文の終わり
	{	パラグラフの始まり
	}	パラグラフの終わり
	%	(に対応する)、または { に対応する } を検索
画面の調整	^L	ウィンドウのクリアと再表示
	^R	ウィンドウのクリアと再表示 (^L が → キーのとき)
	zCR	現在の行をウィンドウの一番上にして画面を再表示
	z–CR	現在の行をウィンドウの一番下にして画面を再表示

	z.CR	現在の行をウィンドウの真ん中にして画面を再表示
	/*pat*/z–CR	*pat* に一致する行はウィンドウの一番下
	z*n*.CR	ウィンドウは *n* 行
	^E	1 行スクロールダウン
	^Y	1 行スクロールアップ
マーク付けと復帰	“	前のコンテキストへカーソルを移動
	”	行の最初の空白以外の部分へカーソルを移動
	m*x*	現在の位置に ASCII 文字の *x* でマーク付け
	`*x*	*x* でマークを付けた行へカーソルを移動
	´*x*	*x* でマークを付けた行の最初の空白以外の部分へカーソルを移動
行の位置指定	H	画面の一番上
	L	画面の一番下
	M	画面の真ん中
	+	次行 (最初の空白以外の部分)
	–	前行 (最初の空白以外の部分)
	CR	改行。+ と同じ
	下矢印 または j	次行。同じカラム
	上矢印 または k	前行。同じカラム
文字の位置指定	^	最初の空白以外の文字
	0	行の始まり
	$	行の終わり
	l または →	正方向
	h または ←	逆方向
	^H	← と同じ (バックスペース)
	space	→ と同じ (スペースバー)
	f*x*	*x* の検索 (前方)
	F*x*	*x* の検索 (後方)
	t*x*	前方の *x* の直前の文字に移動
	T*x*	後方の *x* の直後の文字に移動
	;	前回の f、F、t、または T の繰り返し

	,	前回の f、F、t、または T の逆の繰り返し
	n |	*n* カラム移動
	%	(に対応する)、または { に対応する } を検索
単語、文、パラグラフ	w	前方の単語
	b	後方の単語
	e	単語の終わり
	)	次の文
	}	次のパラグラフ
	(	前の文
	{	前のパラグラフ
	W	空白文字で区切られた前方の単語
	B	空白文字で区切られた後方の単語
	E	空白文字で区切られた単語の終わり
挿入中の修正	^H	最後の文字の消去 (バックスペース)
	^W	最後の単語の消去
	erase	ユーザーの消去文字。^H (バックスペース)
	kill	ユーザーの抹消文字。入力行を消去
	\	ユーザーの消去文字および抹消文字をクォート
	ESC	挿入を終了。コマンドモードへ復帰
	CTRL–C	割り込み。挿入モードを中断
	^D	1 文字バックタブ。*autoindent* の左マージンの解除
	^^D	キャレット (^) と control-d (^D) 行頭までバックタブ *autoindent* の左マージンは解除しない
	0^D	行頭までバックタブ。*autoindent* の左マージンの解除
	^V	非出力文字のクォート
挿入および置換	a	カーソルの後に追加
	A	行の終わりに追加
	i	カーソルの前に挿入
	I	空白文字でない最初の文字の前に挿入
	o	下の行をオープン
	O	上の行をオープン

	r*x*	1 文字を *x* に置換
	R*text*ESC	複数文字を *text* に置換
演算子	演算子のあとにはカーソル動作が続き、カーソルが通過したテキストすべてに演算子が適用されます。たとえば、w は 1 単語移動しますが、dw とすると 1 単語削除します。演算子を重ねると、1 行に適用されます。たとえば dd とすると、1 行すべてを削除します。	
	d	削除
	c	変更
	y	行をバッファに退避 (yank)
	<	左シフト
	>	右シフト
	!	コマンドへのフィルタ
種々の操作	C	行の残りの部分を変更 (c$)
	D	行の残りの部分を削除 (d$)
	s	文字の置換 (cl)
	S	行の置換 (cc)
	J	行の結合
	x	文字の削除 (dl)
	X	カーソルの直前の文字を削除 (dh)
	Y	行の yank (yy)
Yank と **Put**	Put は削除したり退避したテキストのうち、最も新しいテキストを挿入します。しかし、バッファが指定された場合は (ASCII の小文字 a から z を使用) そのバッファのテキストを取り出して挿入します。	
	3yy	3 行退避 (yank)
	3yl	3 文字退避 (yank)
	p	カーソル後にテキストを取り出して挿入
	P	カーソルの前にテキストを取り出して挿入
	″*x*p	バッファ *x* から取り出して挿入
	“*x*y	バッファ *x* に退避 (yank)
	“*x*d	バッファ *x* に削除
undo、繰り返し、検索	u	前回の変更を取り消し (undo)
	U	現在の行に対する変更を取り消し

.	前回の変更を繰り返し
“*d*p	*d* 番目の最後の削除を検索

使用法　ファイルが 2G バイト(2^{31} バイト) 以上ある場合の vi と view の動作については、largefile(5) を参照してください。

環境　vi の実行に影響を与える環境変数 LC_CTYPE、LC_TIME、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

ファイル

/var/tmp	一時的な作業ファイルを置くデフォルトのディレクトリ。directory オプションを使用して、変更可能 (ex(1) の set コマンドを参照)
/usr/share/lib/terminfo/?/*	コンパイルされた端末記述のデータベース
/usr/lib/.COREterm/?/*	コンパイルされた端末記述のデータベースのサブセット

属性　次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/vedit

属性タイプ	属性値
使用条件	SUNWcsu
CSI	未対応

/usr/xpg4/bin/vedit

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目　intro(1), ctags(1), ed(1), edit(1), ex(1), attributes(5), environ(5), largefile(5), standards(5)

『*OpenWindows* ユーザーズガイド *(*上級編*)*』

著者　vi および ex は、カリフォルニア大学バークレイ校、コンピュータ科学学部、電子技術コンピュータ科学学科で開発されました。

注意事項　サポートを続けてはいますが、マニュアル上では 2 つのオプションがコマンド構文標準 (intro(1) を参照) に準拠したオプションに置き換えられています。オプション引数を取らない -r オプションは、-L オプションに置き換えられました。*+command* は -c *command* に置き換えられています。

ファイルが読み込まれたときに、メッセージ `file too large to recover with -r option` が表示されることがあります。これは、このファイルの編集および保存はできるが、万一、編集内容が失われた場合には `-r` オプションでの回復はできないことを意味します。

編集環境が構成オプションにデフォルトで設定されています。編集作業を開始するとき、`vi` は環境変数 `EXINIT` を読み込もうとします。変数が定義されていればエディタは `EXINIT` の値を使い、定義されていなければ `$HOME/.exrc` 中に設定された値を使います。`$HOME/.exrc` がなければ、デフォルト値を使います。

`$HOME` 以外の現在のディレクトリにある `.exrc` のコピーを使う場合は、`EXINIT` または `$HOME/.exrc` 中の *exrc* オプションを設定してください。*exrc* を `EXINIT` または `$HOME/.exrc` 中で設定すれば、`EXINIT` で設定されているオプションをローカルな `.exrc` で無効にすることができます。

`/usr/share/lib/terminfo/?/*` あるいは `/usr/share/lib/terminfo/?/*` のエントリを勝手に変更すると (たとえば、エントリを変えたり移したりする)、エントリが存在し、かつ正しいと想定されている `vi` などのプログラムに影響を及ぼすことがあります。特に、機能の低い端末のエントリを消去すると、不慮の問題を引き起こすことがあります。

`^T` を使用したソフトウェアタブは、 *autoindent* の直後にのみ動作します。

インテリジェント端末上の左右シフトは、削除や挿入などの端末上の文字操作機能を使用しません。

標準の Solaris で使用される `vi` は将来、POSIX.2 に適合するものに置き変わる予定です (`standards`(5) を参照)。アドレス指定や機能で `ex` ファミリを使用するスクリプトでは `/usr/xpg4/bin` バージョンのユーティリティを使用してください。

名前

vi, view, vedit – ex に基づいたスクリーン指向の (ビジュアル) ディスプレイエディタ

形式

/usr/bin/vi [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/bin/view [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/bin/vedit [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/vi [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/view [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/vedit [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

機能説明

vi (ビジュアル) は、ラインエディタ ex に基づいたディスプレイ指向のテキストエディタです。vi の中から ex のコマンドモードを使用することも、ex の中から vi のコマンドモードを使用することも可能です。ビジュアルコマンドは、本マニュアルページで説明しています。オプションの設定の仕方 (自動行番号付けや、改行を押したときの新しい出力行の開始など)、および ex ラインエディタのすべてのコマンドは、ex(1) で説明しています。

vi を使用すると、ユーザーがファイルへ行う変更は、ユーザーが見ている端末スクリーンに反映されます。スクリーン上のカーソルの位置は、ファイルの中の位置を示します。

view 呼び出しは、readonly (読み取り専用) フラグを設定することを除くと vi と同じです。

vedit 呼び出しは、初心者を想定したものです。基本的に vi と同じですが、フラグを 1 に設定し、showmode および novice フラグを設定し、magic をオフにします。このようなデフォルトに設定することによって、vi の使い方がより分かりやすくなります。

呼び出しオプション

以下の呼び出しオプションは、vi によって解釈されます (以前に文書化されていたオプションについては、以降の「注意事項」で説明しています)。

– | -s
: フィードバックのすべてを抑制します。編集スクリプト実行時に役に立ちます。

-C
: -x オプションと同様に、暗号オプションです。 ただし、vi は ex の C コマンドをシミュレートします。C コマンドは X コマンドに似ていますが、読み取られたテキストは、すべて暗号化されていると仮定します。

`-l`	LISP プログラムの編集のための設定を行います。
`-L`	エディタやシステムのクラッシュの結果として保存されたファイル名をすべてリストします。
`-r` *filename*	エディタやシステムのクラッシュ後、*filename* を編集します (クラッシュが発生したときにバッファの中にあった *filename* のバージョンを回復します)。
`-R`	`Readonly` (読み取り専用モード)。このフラグを設定すると、ファイルに誤って上書きすることを防止できます。
`-S`	このオプションは `-t` *tag* オプションと合わせて指定します。タグファイルがソートされていない場合や、バイナリ検索 (ソートされたタグファイルに依存) がタグの検索に失敗するときに遅延リニア検索も行うように、`vi` に伝えます。リニア検索は遅いため、大規模なタグファイルを処理する場合には、このフラグを使用するよりもソートされたタグファイルを使用するようにしてください。通常、タグファイルを作成すると、タグファイルはソートされて生成されます。タグファイルについての詳細は、`ctags`(1) を参照してください。
`-t` *tag*	*tag* を含むファイルを編集し、その定義に従ってエディタの編集位置を設定します。
`-v`	`vi` を使った編集画面を表示します。単に `vi` コマンドを入力しても同じように実行できます。
`-V`	冗長。`ex` コマンドが標準入力によって読み込まれる場合に、その入力を標準エラーに反映します。シェルスクリプト中の `ex` コマンドを実行する場合に役に立ちます。
`-w`*n*	デフォルトのウィンドウサイズを *n* に設定します。これは、低速の回線でエディタを使用するときに役に立ちます。
`-x`	暗号オプション。これを使用すると、`ex` の `X` コマンドをシミュレートし、ユーザーに対してプロンプトを出し、1 つのキーの入力を求めます。このキーを使って、`crypt` コマンドのアルゴリズムに従い、テキスト

を暗号化したり、暗号を解除したりし ます。x コマンドは、読み取ったテキストが暗号化されているかどうかを判別します。また、一時バッファファイルも -x オプションに対して入力したキーの変形バージョンを用いて暗号化されます。空の暗号化キーが入力された場合 (つまり、プロンプトに対してリターンを返した場合)、ファイルは暗号化されません。これは、Back Space キーや Undo キーなど、暗号化キーの入力ミスによって、誤って暗号化されてしまったファイルを解読するための回避策です。

+*command* | -c *command*　指定されたエディタコマンド *command* (通常は検索コマンドまたは位置設定コマンド) を実行し、編集を始めます。

/usr/xpg4/bin/vi

-t *tag* と -c *command* の両方のオプションを指定した場合、-t *tag* オプションがが始めに実行されます。つまり、*tag* のついたファイルを -t によって選んでから、コマンドが実行されます。

オペランド

次のオペランドを指定できます。

filename　編集の対象となるファイル

vi のモード

コマンド　標準モードおよび初期モードです。他のモードは完了すると、このコマンドモードへ戻ります。コマンドの入力を中断するには ESC (エスケープ) を使用します。

入力

a A i I o O c C s S Rのいずれかを入力すると、任意のテキストを入力できます。入力モードの終了は、通常、ESC 文字で行います。ただし、割り込みでも異常終了できます。

最終行　: / ? または ! は読み取り入力です。キャリッジリターンで終了します。割り込みで終了をキャンセルします。

サンプルコマンド

説明では、CR はキャリッジリターンを表し、ESC はエスケープキーを表します。

←
→
下矢印
上矢印　矢印キーはカーソル移動

h j k l　矢印キーと同じ

i*text*ESC　*text* を挿入

cw*new*ESC　単語を *new* に変更

	easESC	単語を複数形に (単語の終わり; s の追加 ; 入力状態からエスケープ)
	x	文字の削除
	dw	単語の削除
	dd	行の削除
	3dd	3 行削除
	u	前回の変更を取り消し
	ZZ	変更を保存し、vi を終了
	:q!CR	変更を保存せず、vi を終了
	/*text*CR	*text* の検索
	^U ^D	スクロールアップとスクロールダウン
	:*cmd*CR	ex または ed のコマンド
vi コマンドの前の数	数を接頭辞としてコマンドの前に入力することができます。入力された数は、次のいずれかの方法で解釈されます。	
	行/カラム数	z G |
	スクロール量	^D ^U
	繰り返し	その他のほとんどのコマンド
割り込み、取り消し	ESC	挿入や途中のコマンドの終了
	DEL	(delete または rubout) 割り込み
ファイル操作	ZZ	ファイルが変更されていれば、書き込みと終了。それ以外、終了のみ
	:wCR	変更の書き込み
	:w!CR	アクセス権が有効でないときの強制書き込み
	:qCR	終了
	:q!CR	変更は無視して、終了
	:e *name*CR	ファイル *name* の編集
	:e!CR	変更は無視して、再編集
	:e + *name*CR	ファイルの最後から編集開始
	:e +*n*CR	ファイルの *n* 行目から編集開始
	:e #CR	代替ファイルの編集
	:e! #CR	変更は無視して、代替ファイルの編集
	:w *name*CR	ファイル *name* に書き込み

	:w! *name*CR	ファイル *name* に上書き
	:shCR	シェルの実行と復帰
	:!*cmd*CR	*cmd* の実行と復帰
	:nCR	引数リストの次のファイルを編集
	:n *args*CR	新しい引数リストを指定
	^G	現在のファイルと行の表示
	:ta *tag*CR	カーソルは *tag* の位置

ex または ed コマンド (*substitute* または *global* など) も入力することができます。このときは、前にコロン、後ろにキャッリッジリターンを入力します。

ファイルの中での位置決め	F	画面の先送り
	^B	画面の後戻し
	^D	半画面スクロールダウン
	^U	半画面スクロールアップ
	*n*G	指定した行の行頭に移動 (デフォルトは文末) *n* は行番号
	/*pat*	*pat* に一致する行 (前方)
	?*pat*	*pat* に一致する行 (後方)
	n	前回の / または ? コマンドの繰り返し
	N	前回の / または ? コマンドの逆転
	/*pat*/+*n*	*n* 回目に *pat* に一致する行 (前方)
	?*pat*?–*n*	*n* 回目に *pat* に一致する行 (後方)
	]]	次の節または関数
	[[	前の節または関数
	(	文の始まり
	)	文の終わり
	{	パラグラフの始まり
	}	パラグラフの終わり
	%	(に対応する)、または { に対応する } を検索
画面の調整	^L	ウィンドウのクリアと再表示
	^R	ウィンドウのクリアと再表示 (^L が → キーのとき)
	zCR	現在の行をウィンドウの一番上にして画面を再表示
	z–CR	現在の行をウィンドウの一番下にして画面を再表示

	z.CR	現在の行をウィンドウの真ん中にして画面を再表示
	/*pat*/z–CR	*pat* に一致する行はウィンドウの一番下
	z*n*.CR	ウィンドウは *n* 行
	^E	1 行スクロールダウン
	^Y	1 行スクロールアップ
マーク付けと復帰	“	前のコンテキストへカーソルを移動
	”	行の最初の空白以外の部分へカーソルを移動
	m*x*	現在の位置に ASCII 文字の *x* でマーク付け
	`*x*	*x* でマークを付けた行へカーソルを移動
	´*x*	*x* でマークを付けた行の最初の空白以外の部分へカーソルを移動
行の位置指定	H	画面の一番上
	L	画面の一番下
	M	画面の真ん中
	+	次行 (最初の空白以外の部分)
	–	前行 (最初の空白以外の部分)
	CR	改行。+ と同じ
	下矢印 または j	次行。同じカラム
	上矢印 または k	前行。同じカラム
文字の位置指定	^	最初の空白以外の文字
	0	行の始まり
	$	行の終わり
	l または →	正方向
	h または ←	逆方向
	^H	← と同じ (バックスペース)
	space	→ と同じ (スペースバー)
	f*x*	*x* の検索 (前方)
	F*x*	*x* の検索 (後方)
	t*x*	前方の *x* の直前の文字に移動
	T*x*	後方の *x* の直後の文字に移動
	;	前回の f、F、t、または T の繰り返し

	,	前回の f、F、t、または T の逆の繰り返し
	n |	*n* カラム移動
	%	(に対応する)、または { に対応する } を検索
単語、文、パラグラフ	w	前方の単語
	b	後方の単語
	e	単語の終わり
	)	次の文
	}	次のパラグラフ
	(	前の文
	{	前のパラグラフ
	W	空白文字で区切られた前方の単語
	B	空白文字で区切られた後方の単語
	E	空白文字で区切られた単語の終わり
挿入中の修正	^H	最後の文字の消去 (バックスペース)
	^W	最後の単語の消去
	erase	ユーザーの消去文字。^H (バックスペース)
	kill	ユーザーの抹消文字。入力行を消去
	\	ユーザーの消去文字および抹消文字をクォート
	ESC	挿入を終了。コマンドモードへ復帰
	CTRL–C	割り込み。挿入モードを中断
	^D	1 文字バックタブ。*autoindent* の左マージンの解除
	^^D	キャレット (^) と control-d (^D) 行頭までバックタブ *autoindent* の左マージンは解除しない
	0^D	行頭までバックタブ。*autoindent* の左マージンの解除
	^V	非出力文字のクォート
挿入および置換	a	カーソルの後に追加
	A	行の終わりに追加
	i	カーソルの前に挿入
	I	空白文字でない最初の文字の前に挿入
	o	下の行をオープン
	O	上の行をオープン

	r*x*	1 文字を *x* に置換
	R*text*ESC	複数文字を *text* に置換
演算子	演算子のあとにはカーソル動作が続き、カーソルが通過したテキストすべてに演算子が適用されます。たとえば、`w` は 1 単語移動しますが、`dw` とすると 1 単語削除します。演算子を重ねると、1 行に適用されます。たとえば `dd` とすると、1 行すべてを削除します。	
	d	削除
	c	変更
	y	行をバッファに退避 (yank)
	<	左シフト
	>	右シフト
	!	コマンドへのフィルタ
種々の操作	C	行の残りの部分を変更 (`c$`)
	D	行の残りの部分を削除 (`d$`)
	s	文字の置換 (`cl`)
	S	行の置換 (`cc`)
	J	行の結合
	x	文字の削除 (`dl`)
	X	カーソルの直前の文字を削除 (`dh`)
	Y	行の yank (`yy`)
Yank と **Put**	Put は削除したり退避したテキストのうち、最も新しいテキストを挿入します。しかし、バッファが指定された場合は (ASCII の小文字 `a` から `z` を使用) そのバッファのテキストを取り出して挿入します。	
	3yy	3 行退避 (yank)
	3yl	3 文字退避 (yank)
	p	カーソル後にテキストを取り出して挿入
	P	カーソルの前にテキストを取り出して挿入
	″*x*p	バッファ *x* から取り出して挿入
	“*x*y	バッファ *x* に退避 (yank)
	“*x*d	バッファ *x* に削除
undo、繰り返し、検索	u	前回の変更を取り消し (undo)
	U	現在の行に対する変更を取り消し

.	前回の変更を繰り返し
“*d*p	*d* 番目の最後の削除を検索

使用法　ファイルが 2G バイト(2^{31} バイト) 以上ある場合の vi と view の動作については、largefile(5) を参照してください。

環境　vi の実行に影響を与える環境変数 LC_CTYPE、LC_TIME、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

ファイル

/var/tmp	一時的な作業ファイルを置くデフォルトのディレクトリ。directory オプションを使用して、変更可能 (ex(1) の set コマンドを参照)
/usr/share/lib/terminfo/?/*	コンパイルされた端末記述のデータベース
/usr/lib/.COREterm/?/*	コンパイルされた端末記述のデータベースのサブセット

属性　次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/vedit

属性タイプ	属性値
使用条件	SUNWcsu
CSI	未対応

/usr/xpg4/bin/vedit

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目　intro(1), ctags(1), ed(1), edit(1), ex(1), attributes(5), environ(5), largefile(5), standards(5)

『*OpenWindows* ユーザーズガイド *(*上級編*)*』

著者　vi および ex は、カリフォルニア大学バークレイ校、コンピュータ科学学部、電子技術コンピュータ科学学科で開発されました。

注意事項　サポートを続けてはいますが、マニュアル上では 2 つのオプションがコマンド構文標準 (intro(1) を参照) に準拠したオプションに置き換えられています。オプション引数を取らない -r オプションは、-L オプションに置き換えられました。*+command* は -c *command* に置き換えられています。

ファイルが読み込まれたときに、メッセージ `file too large to recover with -r option` が表示されることがあります。これは、このファイルの編集および保存はできるが、万一、編集内容が失われた場合には `-r` オプションでの回復はできないことを意味します。

編集環境が構成オプションにデフォルトで設定されています。編集作業を開始するとき、`vi` は環境変数 `EXINIT` を読み込もうとします。変数が定義されていればエディタは `EXINIT` の値を使い、定義されていなければ `$HOME/.exrc` 中に設定された値を使います。`$HOME/.exrc` がなければ、デフォルト値を使います。

`$HOME` 以外の現在のディレクトリにある `.exrc` のコピーを使う場合は、`EXINIT` または `$HOME/.exrc` 中の *exrc* オプションを設定してください。*exrc* を `EXINIT` または `$HOME/.exrc` 中で設定すれば、`EXINIT` で設定されているオプションをローカルな `.exrc` で無効にすることができます。

`/usr/share/lib/terminfo/?/*` あるいは `/usr/share/lib/terminfo/?/*` のエントリを勝手に変更すると (たとえば、エントリを変えたり移したりする)、エントリが存在し、かつ正しいと想定されている `vi` などのプログラムに影響を及ぼすことがあります。特に、機能の低い端末のエントリを消去すると、不慮の問題を引き起こすことがあります。

`^T` を使用したソフトウェアタブは、 *autoindent* の直後にのみ動作します。

インテリジェント端末上の左右シフトは、削除や挿入などの端末上の文字操作機能を使用しません。

標準の Solaris で使用される `vi` は将来、POSIX.2 に適合するものに置き変わる予定です (`standards`(5) を参照)。アドレス指定や機能で `ex` ファミリを使用するスクリプトでは `/usr/xpg4/bin` バージョンのユーティリティを使用してください。

名前

vi, view, vedit – ex に基づいたスクリーン指向の (ビジュアル) ディスプレイエディタ

形式

/usr/bin/vi [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/bin/view [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/bin/vedit [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/vi [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/view [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

/usr/xpg4/bin/vedit [-| -s] [-l] [-L] [-R] [-r [*filename*]] [-S] [-t *tag*] [-v] [-V] [-x] [-w*n*] [-C] [+*command* | -c *command*] *filename...*

機能説明

vi (ビジュアル) は、ラインエディタ ex に基づいたディスプレイ指向のテキストエディタです。vi の中から ex のコマンドモードを使用することも、ex の中から vi のコマンドモードを使用することも可能です。ビジュアルコマンドは、本マニュアルページで説明しています。オプションの設定の仕方 (自動行番号付けや、改行を押したときの新しい出力行の開始など)、および ex ラインエディタのすべてのコマンドは、ex(1) で説明しています。

vi を使用すると、ユーザーがファイルへ行う変更は、ユーザーが見ている端末スクリーンに反映されます。スクリーン上のカーソルの位置は、ファイルの中の位置を示します。

view 呼び出しは、readonly (読み取り専用) フラグを設定することを除くと vi と同じです。

vedit 呼び出しは、初心者を想定したものです。基本的に vi と同じですが、フラグを 1 に設定し、showmode および novice フラグを設定し、magic をオフにします。このようなデフォルトに設定することによって、vi の使い方がより分かりやすくなります。

呼び出しオプション

以下の呼び出しオプションは、vi によって解釈されます (以前に文書化されていたオプションについては、以降の「注意事項」で説明しています)。

– | -s
: フィードバックのすべてを抑制します。編集スクリプト実行時に役に立ちます。

-C
: -x オプションと同様に、暗号オプションです。 ただし、vi は ex の c コマンドをシミュレートします。c コマンドは x コマンドに似ていますが、読み取られたテキストは、すべて暗号化されていると仮定します。

`-l`	LISP プログラムの編集のための設定を行います。
`-L`	エディタやシステムのクラッシュの結果として保存されたファイル名をすべてリストします。
`-r` *filename*	エディタやシステムのクラッシュ後、*filename* を編集します (クラッシュが発生したときにバッファの中にあった *filename* のバージョンを回復します)。
`-R`	`Readonly` (読み取り専用モード)。このフラグを設定すると、ファイルに誤って上書きすることを防止できます。
`-S`	このオプションは `-t` *tag* オプションと合わせて指定します。タグファイルがソートされていない場合や、バイナリ検索 (ソートされたタグファイルに依存) がタグの検索に失敗するときに遅延リニア検索も行うように、`vi` に伝えます。リニア検索は遅いため、大規模なタグファイルを処理する場合には、このフラグを使用するよりもソートされたタグファイルを使用するようにしてください。通常、タグファイルを作成すると、タグファイルはソートされて生成されます。タグファイルについての詳細は、`ctags`(1) を参照してください。
`-t` *tag*	*tag* を含むファイルを編集し、その定義に従ってエディタの編集位置を設定します。
`-v`	`vi` を使った編集画面を表示します。単に `vi` コマンドを入力しても同じように実行できます。
`-V`	冗長。`ex` コマンドが標準入力によって読み込まれる場合に、その入力を標準エラーに反映します。シェルスクリプト中の `ex` コマンドを実行する場合に役に立ちます。
`-w`*n*	デフォルトのウィンドウサイズを *n* に設定します。これは、低速の回線でエディタを使用するときに役に立ちます。
`-x`	暗号オプション。これを使用すると、`ex` の `X` コマンドをシミュレートし、ユーザーに対してプロンプトを出し、1 つのキーの入力を求めます。このキーを使って、`crypt` コマンドのアルゴリズムに従い、テキスト

を暗号化したり、暗号を解除したりし ます。x コマンドは、読み取ったテキストが暗号化されているかどうかを判別します。また、一時バッファファイルも -x オプションに対して入力したキーの変形バージョンを用いて暗号化されます。空の暗号化キーが入力された場合 (つまり、プロンプトに対してリターンを返した場合)、ファイルは暗号化されません。これは、Back Space キーや Undo キーなど、暗号化キーの入力ミスによって、誤って暗号化されてしまったファイルを解読するための回避策です。

+*command* | -c *command*　指定されたエディタコマンド *command* (通常は検索コマンドまたは位置設定コマンド) を実行し、編集を始めます。

/usr/xpg4/bin/vi

-t *tag* と -c *command* の両方のオプションを指定した場合、-t *tag* オプションがが始めに実行されます。つまり、*tag* のついたファイルを -t によって選んでから、コマンドが実行されます。

オペランド

次のオペランドを指定できます。

filename　編集の対象となるファイル

vi のモード

コマンド　標準モードおよび初期モードです。他のモードは完了すると、このコマンドモードへ戻ります。コマンドの入力を中断するには ESC (エスケープ) を使用します。

入力

a A i I o O c C s S Rのいずれかを入力すると、任意のテキストを入力できます。入力モードの終了は、通常、ESC 文字で行います。ただし、割り込みでも異常終了できます。

最終行　: / ? または ! は読み取り入力です。キャリッジリターンで終了します。割り込みで終了をキャンセルします。

サンプルコマンド

説明では、CR はキャリッジリターンを表し、ESC はエスケープキーを表します。

←
→
下矢印
上矢印　矢印キーはカーソル移動

h j k l　矢印キーと同じ

i*text*ESC　*text* を挿入

cw*new*ESC　単語を *new* に変更

	easESC	単語を複数形に (単語の終わり; s の追加 ; 入力状態からエスケープ)
	x	文字の削除
	dw	単語の削除
	dd	行の削除
	3dd	3 行削除
	u	前回の変更を取り消し
	ZZ	変更を保存し、vi を終了
	:q!CR	変更を保存せず、vi を終了
	/*text*CR	*text* の検索
	^U ^D	スクロールアップとスクロールダウン
	:*cmd*CR	ex または ed のコマンド
vi コマンドの前の数	数を接頭辞としてコマンドの前に入力することができます。入力された数は、次のいずれかの方法で解釈されます。	
	行/カラム数	z G |
	スクロール量	^D ^U
	繰り返し	その他のほとんどのコマンド
割り込み、取り消し	ESC	挿入や途中のコマンドの終了
	DEL	(delete または rubout) 割り込み
ファイル操作	ZZ	ファイルが変更されていれば、書き込みと終了。それ以外、終了のみ
	:wCR	変更の書き込み
	:w!CR	アクセス権が有効でないときの強制書き込み
	:qCR	終了
	:q!CR	変更は無視して、終了
	:e *name*CR	ファイル *name* の編集
	:e!CR	変更は無視して、再編集
	:e + *name*CR	ファイルの最後から編集開始
	:e +*n*CR	ファイルの *n* 行目から編集開始
	:e #CR	代替ファイルの編集
	:e! #CR	変更は無視して、代替ファイルの編集
	:w *name*CR	ファイル *name* に書き込み

	:w! *name*CR	ファイル *name* に上書き
	:shCR	シェルの実行と復帰
	:!*cmd*CR	*cmd* の実行と復帰
	:nCR	引数リストの次のファイルを編集
	:n *args*CR	新しい引数リストを指定
	^G	現在のファイルと行の表示
	:ta *tag*CR	カーソルは *tag* の位置

ex または ed コマンド (*substitute* または *global* など) も入力することができます。このときは、前にコロン、後ろにキャッリッジリターンを入力します。

ファイルの中での位置決め	F	画面の先送り
	^B	画面の後戻し
	^D	半画面スクロールダウン
	^U	半画面スクロールアップ
	*n*G	指定した行の行頭に移動 (デフォルトは文末) *n* は行番号
	/*pat*	*pat* に一致する行 (前方)
	?*pat*	*pat* に一致する行 (後方)
	n	前回の / または ? コマンドの繰り返し
	N	前回の / または ? コマンドの逆転
	/*pat*/+*n*	*n* 回目に *pat* に一致する行 (前方)
	?*pat*?–*n*	*n* 回目に *pat* に一致する行 (後方)
	]]	次の節または関数
	[[	前の節または関数
	(	文の始まり
	)	文の終わり
	{	パラグラフの始まり
	}	パラグラフの終わり
	%	(に対応する)、または { に対応する } を検索
画面の調整	^L	ウィンドウのクリアと再表示
	^R	ウィンドウのクリアと再表示 (^L が → キーのとき)
	zCR	現在の行をウィンドウの一番上にして画面を再表示
	z–CR	現在の行をウィンドウの一番下にして画面を再表示

	z.CR	現在の行をウィンドウの真ん中にして画面を再表示
	/*pat*/z–CR	*pat* に一致する行はウィンドウの一番下
	z*n*.CR	ウィンドウは *n* 行
	^E	1 行スクロールダウン
	^Y	1 行スクロールアップ
マーク付けと復帰	“	前のコンテキストへカーソルを移動
	”	行の最初の空白以外の部分へカーソルを移動
	m*x*	現在の位置に ASCII 文字の *x* でマーク付け
	`*x*	*x* でマークを付けた行へカーソルを移動
	´*x*	*x* でマークを付けた行の最初の空白以外の部分へカーソルを移動
行の位置指定	H	画面の一番上
	L	画面の一番下
	M	画面の真ん中
	+	次行 (最初の空白以外の部分)
	–	前行 (最初の空白以外の部分)
	CR	改行。+ と同じ
	下矢印 または j	次行。同じカラム
	上矢印 または k	前行。同じカラム
文字の位置指定	^	最初の空白以外の文字
	0	行の始まり
	$	行の終わり
	l または →	正方向
	h または ←	逆方向
	^H	← と同じ (バックスペース)
	space	→ と同じ (スペースバー)
	f*x*	*x* の検索 (前方)
	F*x*	*x* の検索 (後方)
	t*x*	前方の *x* の直前の文字に移動
	T*x*	後方の *x* の直後の文字に移動
	;	前回の f、F、t、または T の繰り返し

	,	前回の f、F、t、または T の逆の繰り返し
	n |	*n* カラム移動
	%	(に対応する)、または { に対応する } を検索
単語、文、パラグラフ	w	前方の単語
	b	後方の単語
	e	単語の終わり
	)	次の文
	}	次のパラグラフ
	(	前の文
	{	前のパラグラフ
	W	空白文字で区切られた前方の単語
	B	空白文字で区切られた後方の単語
	E	空白文字で区切られた単語の終わり
挿入中の修正	^H	最後の文字の消去 (バックスペース)
	^W	最後の単語の消去
	erase	ユーザーの消去文字。^H (バックスペース)
	kill	ユーザーの抹消文字。入力行を消去
	\	ユーザーの消去文字および抹消文字をクォート
	ESC	挿入を終了。コマンドモードへ復帰
	CTRL–C	割り込み。挿入モードを中断
	^D	1 文字バックタブ。*autoindent* の左マージンの解除
	^^D	キャレット (^) と control-d (^D) 行頭までバックタブ *autoindent* の左マージンは解除しない
	0^D	行頭までバックタブ。*autoindent* の左マージンの解除
	^V	非出力文字のクォート
挿入および置換	a	カーソルの後に追加
	A	行の終わりに追加
	i	カーソルの前に挿入
	I	空白文字でない最初の文字の前に挿入
	o	下の行をオープン
	O	上の行をオープン

	r*x*	1 文字を *x* に置換
	R*text*ESC	複数文字を *text* に置換
演算子	演算子のあとにはカーソル動作が続き、カーソルが通過したテキストすべてに演算子が適用されます。たとえば、w は 1 単語移動しますが、dw とすると 1 単語削除します。演算子を重ねると、1 行に適用されます。たとえば dd とすると、1 行すべてを削除します。	
	d	削除
	c	変更
	y	行をバッファに退避 (yank)
	<	左シフト
	>	右シフト
	!	コマンドへのフィルタ
種々の操作	C	行の残りの部分を変更 (c$)
	D	行の残りの部分を削除 (d$)
	s	文字の置換 (cl)
	S	行の置換 (cc)
	J	行の結合
	x	文字の削除 (dl)
	X	カーソルの直前の文字を削除 (dh)
	Y	行の yank (yy)
Yank と **Put**	Put は削除したり退避したテキストのうち、最も新しいテキストを挿入します。しかし、バッファが指定された場合は (ASCII の小文字 a から z を使用) そのバッファのテキストを取り出して挿入します。	
	3yy	3 行退避 (yank)
	3yl	3 文字退避 (yank)
	p	カーソル後にテキストを取り出して挿入
	P	カーソルの前にテキストを取り出して挿入
	″*x*p	バッファ *x* から取り出して挿入
	“*x*y	バッファ *x* に退避 (yank)
	“*x*d	バッファ *x* に削除
undo、繰り返し、検索	u	前回の変更を取り消し (undo)
	U	現在の行に対する変更を取り消し

.	前回の変更を繰り返し
“*d*p	*d* 番目の最後の削除を検索

使用法　ファイルが 2G バイト(2^{31} バイト) 以上ある場合の vi と view の動作については、largefile(5) を参照してください。

環境　vi の実行に影響を与える環境変数 LC_CTYPE、LC_TIME、LC_MESSAGES、NLSPATH についての詳細は、environ(5) を参照してください。

ファイル

/var/tmp	一時的な作業ファイルを置くデフォルトのディレクトリ。directory オプションを使用して、変更可能 (ex(1) の set コマンドを参照)
/usr/share/lib/terminfo/?/*	コンパイルされた端末記述のデータベース
/usr/lib/.COREterm/?/*	コンパイルされた端末記述のデータベースのサブセット

属性　次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/vedit

属性タイプ	属性値
使用条件	SUNWcsu
CSI	未対応

/usr/xpg4/bin/vedit

属性タイプ	属性値
使用条件	SUNWxcu4
CSI	対応済み

関連項目　intro(1), ctags(1), ed(1), edit(1), ex(1), attributes(5), environ(5), largefile(5), standards(5)

『*OpenWindows* ユーザーズガイド *(*上級編*)*』

著者　vi および ex は、カリフォルニア大学バークレイ校、コンピュータ科学学部、電子技術コンピュータ科学学科で開発されました。

注意事項　サポートを続けてはいますが、マニュアル上では 2 つのオプションがコマンド構文標準 (intro(1) を参照) に準拠したオプションに置き換えられています。オプション引数を取らない -r オプションは、-L オプションに置き換えられました。*+command* は -c *command* に置き換えられています。

ファイルが読み込まれたときに、メッセージ `file too large to recover with -r option` が表示されることがあります。これは、このファイルの編集および保存はできるが、万一、編集内容が失われた場合には `-r` オプションでの回復はできないことを意味します。

編集環境が構成オプションにデフォルトで設定されています。編集作業を開始するとき、`vi` は環境変数 `EXINIT` を読み込もうとします。変数が定義されていればエディタは `EXINIT` の値を使い、定義されていなければ `$HOME/.exrc` 中に設定された値を使います。`$HOME/.exrc` がなければ、デフォルト値を使います。

`$HOME` 以外の現在のディレクトリにある `.exrc` のコピーを使う場合は、`EXINIT` または `$HOME/.exrc` 中の *exrc* オプションを設定してください。*exrc* を `EXINIT` または `$HOME/.exrc` 中で設定すれば、`EXINIT` で設定されているオプションをローカルな `.exrc` で無効にすることができます。

`/usr/share/lib/terminfo/?/*` あるいは `/usr/share/lib/terminfo/?/*` のエントリを勝手に変更すると (たとえば、エントリを変えたり移したりする)、エントリが存在し、かつ正しいと想定されている `vi` などのプログラムに影響を及ぼすことがあります。特に、機能の低い端末のエントリを消去すると、不慮の問題を引き起こすことがあります。

`^T` を使用したソフトウェアタブは、 *autoindent* の直後にのみ動作します。

インテリジェント端末上の左右シフトは、削除や挿入などの端末上の文字操作機能を使用しません。

標準の Solaris で使用される `vi` は将来、POSIX.2 に適合するものに置き変わる予定です (`standards`(5) を参照)。アドレス指定や機能で `ex` ファミリを使用するスクリプトでは `/usr/xpg4/bin` バージョンのユーティリティを使用してください。

名前 volcheck – ドライブ内の媒体の検査と、デフォルトでのすべてのフロッピー媒体の検査

形式 **volcheck** [-v] [-i *secs*] [-t *secs*] *pathname*

機能説明 volcheck ユーティリティーは、各 dev/*pathname* を順に調べ、ドライブに新しい媒体が挿入されていないかを検査するようにボリューム管理に指示します。

デフォルトでは、ボリューム管理によって管理されている、検査が可能なすべての媒体が volcheck の対象となります。

オプション 次のオプションを指定できます。

-i *secs* デバイス検査の頻度を *secs* 秒に設定します。デフォルトは 2 秒です。最小の頻度は 1 秒です。

-t *secs* *secs* に指定される秒数の間、指定されたデバイスを検査します。指定できる最大の秒数は 28800 (8 時間) です。検査の頻度は -i で指定します。この合計時間にデフォルト値はありません。

-v 詳細表示

オペランド 次のオペランドを指定できます。

pathname 媒体デバイスのパス名

使用例 例 **1** volcheck コマンドの例

次の例は、フロッピーディスクドライブに新しい媒体が挿入されていないか検査するようにボリューム管理に指示します。

```
example% volcheck –v /dev/diskette
/dev/diskette has media
```

次の例は、2 秒おきに 600 秒 (10 分) 間、フロッピーディスクドライブにフロッピーが挿入されていないか調べるようにボリューム管理に指示します。

```
example% volcheck –i 2 –t 600 /dev/diskette1 &
```

ファイル /dev/volctl ボリューム管理の制御ポート

属性 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWvolu

関連項目 eject(1), volcancel(1), volmissing(1) rmmount(1M), vold(1M), rmmount.conf(4), vold.conf(4), attributes(5), volfs(7FS)

警告 | 多くのフロッピーディスクドライブにはハードウェア上の制限があるため、媒体の検査では、フロッピーディスクドライブ内で機械的な動作が発生します。したがって、フロッピーディスクドライブを継続的にポーリングすると、ドライブの磨耗を引き起こします。ドライブのポーリングは、ドライブの使用頻度が高い時にだけ実施するようにお勧めします。

名前 | wait – 他のプロセスの終了を待つ

/bin/sh | **wait** [*pid*...]

/bin/jsh
/bin/ksh
/usr/xpg4/bin/sh | **wait** [% *jobid*...]

/bin/csh | **wait**

機能説明 | シェルは、新しいプロセスを作成せずに、シェル自身で wait を実行します。エラーメッセージ cannot fork, too many processes が表示された場合、wait コマンドを使用して、バックグラウンドプロセスをすべてクリアしてください。これを行なっても問題が解決しない場合には、システムのプロセステーブルが一杯になっていること、または実行中のフォアグラウンドプロセスの数が多すぎることが考えられます。ユーザーのログインに対するプロセス ID の数、およびシステムが把握できるプロセスの数には制限があります。

段階以上のパイプラインのプロセスは、必ずしもすべてがシェルの子プロセスであるわけではなく、そのようなプロセスは待つことができません。

/bin/sh, /bin/jsh | プロセス ID が *pid* であるバックグラウンドプロセスを待ち、そのプロセスの終了ステータスを報告します。*pid* を省略すると、ユーザーのシェルで現在実行中のバックグラウンドプロセスをすべて待ち、戻り値は 0 になります。ジョブ制御が有効 (jsh) なとき、wait はジョブ識別子の指定を受け付けます。引数のは、先頭にパーセント記号 (%) を付加して指定します。

pid が実行中のプロセス ID でない場合には、wait ユーティリティはすぐに戻り、戻り値は 0 になります。

csh | バックグラウンドプロセスを待ちます。

ksh | シェルが非同期リストを開始したとき、非同期リストの各要素内の最後のコマンドのプロセス ID が、現在のシェル実行環境に知らされます。

wait ユーティリティがオペランドなしで呼び出されると、呼び出し側シェルが認識しているすべてのプロセス ID が終了するまで待ち、終了ステータス 0 で終了します。

認識しているプロセス ID (またはジョブ ID) を示すいくつかの *pid* または *jobid*オペランドが指定された場合、wait は認識しているプロセス ID (またはジョブ ID) が終了ステータス 127 で終了したのと同じように扱います。wait ユーティリティが戻した終了ステータスは、最後の *pid* または *jobid* オペランドで要求したプロセスの終了ステータスです。

認識されたプロセス ID は、現在のシェル実行環境内の wait 呼び出しに対してだけ適用可能です。

オペランド | 以下のオペランドを指定できます。

以下のうちのいずれかを指定します。

pid ユーティリティが停止を待つ対象であるコマンドの符号なし 10 進整数プロセス ID

jobid 待つ対象のバックグラウンドプロセスグループを識別するジョブ制御ジョブ ID。ジョブ制御ジョブ ID 表記は、現在のシェル実行環境内の `wait` 呼び出しに対してだけ適用可能です。なお *pid* のジョブ制御ジョブ ID の形式は、ジョブ制御オプションをサポートしているシステム上でだけ使用可能です。

使用法

ほとんどのシステムでは、`wait` はシェルに組み込まれています。サブシェルや別のユーティリティ実行環境で、次のように `wait` が呼び出されたとします。

```
(wait)
nohup wait . . .
find . -exec wait . . . \;
```

これらの環境では待つ対象の認識されたプロセス ID がないため、ただちに戻ります。

使用例

例 1 終端シグナルを識別するスクリプトを使用する

プロセスがシグナルによって停止されたときに使用する正確な値は不定ですが、シグナルがプロセスを停止したことがわかっていれば、以下のようにすると、どのシグナルが `kill` を使用しているかをかなり正確に特定できます (`/bin/ksh` および `/usr/xpg4/bin/sh`)。

```
sleep 1000&
pid=$!
kill -kill $pid
wait $pid
echo $pid was terminated by a SIG$(kill -l  $(($?-128))) signal.
```

例 2 プロセスの終了ステータスを戻す

以下のコマンドが 31 秒未満で実行されたとします (`/bin/ksh` および `/usr/xpg4/bin/sh`):。

```
sleep 257 | sleep 31 &

jobs -l  %%
```

その場合、次のコマンドのどちらかが 2 番目の `sleep` の終了ステータスを戻します。

```
wait <pid of sleep 31>
wait %%
```

環境

`wait` の実行に影響を与える環境変数 `LC_CTYPE`、`LC_MESSAGES`、`NLSPATH` についての詳細は、`environ`(5) を参照してください。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 csh(1), jobs(1), ksh(1), sh(1), attributes(5), environ(5)

名前 | wc – ファイルの中の行数、単語数および文字数の表示

形式 | **wc** [-c | -m | -C] [-lw] [*file...*]

機能説明 | wc ユーティリティはいくつかの入力ファイルを読み込み、 デフォルトでは各ファイル中の復帰改行の数、単語数、バイト数を 標準出力に書き出します。

入力ファイルが複数指定された場合は、すべてのファイルの合計値も出力します。

wc の定義では、「単語」とは空白 (たとえば空白文字やタブ文字) により区切られた、長さがゼロでない文字列です。 iswspace(3C) や isspace(3C) の説明を参照してください。

オプション | 以下のオプション を指定できます。

-c　バイト数を数えます。

-m　文字数を数えます。

-C　-m と同じ意味です。

-l　行数を数えます。

-w　空白文字または復帰改行で区切られた単語数を数えます。 区切り文字は、iswspace() によって定義された任意のコードセット中の拡張 Unix コード (EUC) 文字です。

オプションが指定されていない場合、 デフォルトは -lwc です (行数、単語数、バイト数の表示) 。

オペランド | 以下のオペランドを指定できます。

file　入力ファイルのパス名。 このオペランドを省略すると、標準入力が使用されます。

使用法 | ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の wc の動作については、largefile(5) を参照してください。

環境 | wc の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス | 以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

属性タイプ	属性値
CSI	対応済み

関連項目 cksum(1), isspace(3C), iswalpha(3C), iswspace(3C), setlocale(3C), attributes(5), environ(5), largefile(5)

名前 | whatis – キーワードに関する要約の表示

形式 | **whatis** *command...*

機能説明 | whatis は command 引数で指定されたマニュアルページを参照し、そのヘッダー行を 表示します。ヘッダーを見て、そのコマンドについての詳細を知りたければ、man(1) を実行することによりマニュアルページ全体を得ることができます。 その際、ヘッダー行が 'name(*section*) ...' で始まっていれば、 'man -s *section* name' とセクション番号を指定することによりマニュアルページが得られます。試しに 'whatis ed' を実行し、その後 'man -s 1 ed' を実行してみてください。 ed(1) のマニュアルページが得られます。

whatis は、機能的には -f オプション付きの man(1) コマンドと同じです。

whatis は /usr/share/man/windex データベースを使用します。このデータベースは catman(1M) コマンドにより作成されます。このデータベースが存在しないと whatis コマンドは実行できません。

ファイル | /usr/share/man/windex　　目次とキーワードのデータベース

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWdoc
CSI	対応済み

関連項目 | apropos(1), man(1), catman(1M), attributes(5)

名前 typeset, whence – シェル変数と関数の属性と値を設定または取得するためのシェル組み込み関数

形式 **typeset** [± HLRZfilrtux [*n*]] [*name* [= *value*]]...

whence [-pv] *name*...

機能説明 typeset はシェル変数と関数の属性と値を設定します。 関数内で typeset を実行すると、*name* が示す変数の新しいインスタンスが生成されます。 関数が完了すると、その変数の値と型が復元されます。このコマンドには、 以下の属性を指定できます。

-H このフラグは UNIX 以外のマシン上で、 UNIX とホスト名ファイルとのマッピング情報を提供します。

-L 左詰めを行い、先行する空白文字を *value* から取り除きます。 *n* は、 0 以外であればフィールドの幅を定義します。 0 の場合、 フィールドの幅は最初に代入される値の幅で決定されます。 変数に値を代入したとき、 フィールド幅より短ければ右側に空白文字が詰められ、 長ければ切り捨てられます。 -Z フラグも指定されていれば、 先行する 0 を削除します。 -R フラグは無効になります。

-R 右詰めを行い、 先行する空白文字を挿入します。 *n* は、 0 以外であればフィールドの幅を定義します。 0 の場合、 フィールドの幅は最初に代入される値の幅で決定されます。 変数に値を代入したとき、 フィールド幅より短ければ左側に空白文字が詰められ、 長ければ終端が切り捨てられます。 -L フラグは無効になります。

-Z 最初の、空白文字でない文字が数字で、 かつ -L フラグが設定されていない場合、 右詰めを行い先頭に 0 を詰めます。 *n* は、 0 以外であればフィールドの幅を定義します。 0 の場合、 フィールドの幅は最初に代入される値の幅で決定されます。

-f 名前は、 変数名ではなく関数名を指します。 代入は行われません。 このフラグと共に指定できる他のフラグは、-t、-u、 -x だけです。 -t フラグは、 この関数の実行トレースを有効にします。 -u フラグは、 この関数に「未定義」を示すマークを付けます。 関数が参照されると、 関数定義を見つけるために FPATH 変数が検索されます。 -x フラグを指定すると、 名前で呼び出されるシェル手続き全体で 関数定義が有効になります。

-i パラメタを整数とします。 これにより算術演算が高速化されます。 *n* は、 0 以外であればその値を底として定義します。 0 の場合、 最初の代入で底が決定されます。

-l 大文字をすべて小文字に変換します。 大文字への変換を示す -u フラグを無効にします。

-r 指定された *name* を読み取り専用にします。 後の代入でこれらの名前を変更できないようにします。

-t 変数にタグを付けます。 タグはユーザーが定義可能で、 シェルに対して特別の意味を持ちません。

`-u` 小文字をすべて大文字に変換します。 小文字への変換を示す `-l` フラグを無効にします。

`-x` 指定された *name* に対し、 後で実行されるコマンドの「環境」へ 自動的にエクスポートされるようにマークを付けます。

`-i` 属性は、 `-R`、 `-L`、 `-Z`、 `-f` と同時に指定することはできません。

– の代わりに + を使用すると、 これらのフラグは無効になります。 *name* 引数をまったく指定せずにフラグを指定すると、 これらのフラグが設定されている変数の名前 （および選択により値も） が一覧表示されます。 具体的には – を付加すれば名前と値が、 + を付加すれば名前だけが表示されます。 *name* 引数とフラグを 1 つも指定しないと、 すべての変数の名前と属性が表示されます。

`whence` コマンドは、 *name* ごとに、 コマンド名として使用される場合に どのように解釈されるかを指示します。 `-v` フラグをつけると、より冗長に表示されます。 `-p` フラグをつけると、 コマンド名が別名、関数、 または予約語である場合でも *name* のパスが検索されます。

1 つまたは 2 つのアスタリスク（*）が先頭に付加されている `ksh`(1) コマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目

`ksh`(1), `set`(1), `sh`(1), `attributes`(5)

名前 | shell_builtins, case, for, foreach, function, if, repeat, select, switch, until, while – シェル組み込み関数

機能説明 | シェルインタプリタである csh(1)、ksh(1)、および sh(1) には、特殊の組み込みコマンドが用意されています。シェルは、case, for、foreach、function、if、repeat、select、switch、until、および while をコマンドとして解釈します。これらのコマンドの詳細は、各シェルのマニュアルページの「コマンド」の項に記載されています。以下に示すコマンドは、効率の改善とコマンド間でのデータ共有を可能にするため、シェルに組み込まれています。詳細については、各コマンドのマニュアルページを参照してください。

コマンド	組み込み対象シェル
alias	csh, ksh
bg	csh, ksh, sh
break	csh, ksh, sh
case	csh, ksh, sh
cd	csh, ksh, sh
chdir	csh, sh
continue	csh, ksh, sh
dirs	csh
echo	csh, ksh, sh
eval	csh, ksh, sh
exec	csh, ksh, sh
exit	csh, ksh, sh
export	ksh, sh
fc	ksh
fg	csh, ksh, sh
for	ksh, sh
foreach	csh
function	ksh
getopts	ksh, sh
glob	csh
goto	csh
hash	ksh, sh

コマンド	組み込み対象シェル
hashstat	csh
history	csh
if	csh, ksh, sh
jobs	csh, ksh, sh
kill	csh, ksh, sh
let	ksh
limit	csh
login	csh, ksh, sh
logout	csh, ksh, sh
nice	csh
newgrp	ksh, sh
notify	csh
onintr	csh
popd	csh
print	ksh
pushd	csh
pwd	ksh, sh
read	ksh, sh
readonly	ksh, sh
rehash	csh
repeat	csh
return	ksh, sh
select	ksh
set	csh, ksh, sh
setenv	csh
shift	csh, ksh, sh
source	csh
stop	csh, ksh, sh
suspend	csh, ksh, sh

コマンド	組み込み対象シェル
switch	csh
test	ksh, sh
time	csh
times	ksh, sh
trap	ksh, sh
type	ksh, sh
typeset	ksh
ulimit	ksh, sh
umask	csh, ksh, sh
unalias	csh, ksh
unhash	csh
unlimit	csh
unset	csh, ksh, sh
unsetenv	csh
until	ksh, sh
wait	csh, ksh, sh
whence	ksh
while	sh, ksh, sh

Bourne シェル (sh) の特殊コマンド

上記のコマンドに対しては、 入出力のリダイレクトが可能です。 ファイル記述子 1 は、デフォルトの出力位置です。 ジョブ制御が有効なときには、さらにいくつかの特殊コマンド がシェル環境に追加されます。

組み込みの予約コマンド以外に、 sh では以下のコマンドが使用されます。

: コマンドは何もせず、影響は何もありません。 終了状態 0 が返されます。

. *filename* *filename* で示すファイルからコマンドを読み取り、それを実行し、戻ります。 PATH で指定された検索パスを使用して、 *filename* が存在しているディレクトリを見つけます。

C シェルの特殊コマンド

組み込みコマンドは C シェル内で実行されます。 組み込みコマンドが、パイプラインの いずれかの構成要素 (最後の要素を除く) として現われると、 サブシェル内で実行されます。 組み込みの予約コマンド以外に、 csh では以下のコマンドが使用されます。

: NULL コマンド。このコマンドは解釈されますが、実行はされません。

Korn シェル (ksh) の特殊コマンド

入出力のリダイレクトが可能です。 特に断わりのない限り、 出力はファイル記述子 1 上に書き込まれ、 構文エラーがなければ終了状態は 0 です。

1 つまたは 2 つのアスタリスク (*) が先頭に付加されているコマンドは、 以下のような特殊な処理を受けます。

1. コマンドが完了しても、 コマンドの直前の 変数代入リストは依然として有効です。
2. 入出力のリダイレクトは変数代入後に行われます。
3. エラーが発生すると、 それを含むスクリプトは中止されます。
4. 変数代入形式で、 ** から始まるコマンドに続くワードは、 変数代入と同一の規則で展開されます。 つまり、 チルド置換は = 符号の後に実行され、 ワード分割とファイル名生成は実行されません。

組み込みの予約コマンド以外に、 `ksh` では以下のコマンドが使用されます。

* : [*arg* . . .] パラメタの展開だけを行います。

* .*file* [*arg* . . .] *file* 全体を読み取ってからコマンドを実行します。 コマンドは現在のシェル環境において実行されます。 `PATH` で指定された検索パスを使用して、*file* が存在しているディレクトリを見つけます。 引数の *arg* は (指定されていれば) 定位置パラメタになります。 引数を指定しないと定位置パラメタは変更されません。 終了状態は、最後に実行されたコマンドの終了状態です。

関連項目 `intro`(1), `alias`(1), `break`(1), `cd`(1), `chmod`(1), `csh`(1), `echo`(1), `exec`(1), `exit`(1), `find`(1), `getoptcvt`(1), `getopts`(1), `glob`(1), `hash`(1), `history`(1), `jobs`(1), `kill`(1), `ksh`(1), `let`(1), `limit`(1), `login`(1), `logout`(1), `newgrp`(1), `nice`(1), `nohup`(1), `print`(1), `pwd`(1), `read`(1), `readonly`(1), `set`(1), `sh`(1), `shift`(1), `suspend`(1), `test`(1B), `time`(1), `times`(1), `trap`(1), `typeset`(1), `umask`(1), `wait`(1), `chdir`(2), `chmod`(2), `creat`(2), `umask`(2), `getopt`(3C), `profile`(4), `environ`(5)

名前 | who – システムにログインしているユーザーの表示

形式 | **/usr/bin/who** [-abdHlmpqrstTu] [*file*]

/usr/bin/who -q [-n *x*] [*file*]

/usr/bin/who am i

/usr/bin/who am I

/usr/xpg4/bin/who [-abdHlmpqrtTu] [*file*]

/usr/xpg4/bin/who -q [-n *x*] [*file*]

/usr/xpg4/bin/who -s [-bdHlmpqrtu] [*file*]

/usr/xpg4/bin/who am i

/usr/xpg4/bin/who am I

機能説明 | who ユーティリティは、現在の UNIX システムを使用しているそれぞれのユーザーについて、 ユーザー名、端末回線名、ログイン時刻、 回線を稼働し始めてからの経過時間、コマンドインタプリタ (シェル) のプロセス ID を表示することができます。 who は、 /var/adm/utmp ファイルを検索して、その情報を入手します。 *file* を指定した場合、そのファイルを検索します (ファイルは、 utmp(4) フォーマットでなければなりません) 。通常、 file は /bar/adm/wtmp で、これはファイルが最後に作成されてからの すべてのログイン記録を格納しています。

出力のための一般形式は次のようになります。

name [*state*] *line time* [*idle*] [*pid*] [*comment*] [*exit*]

出力形式の説明

name	ユーザーのログイン名
state	端末への書き込み権
line	/dev に存在する回線名
time	ユーザーがログインしてからの時間
idle	ユーザーが最後に実行してからの経過時間
pid	ユーザーのプロセス ID
comment	inittab(4) の注釈欄
exit	活動していないプロセスの終了ステータス

オプション | 以下にオプションを示します。

-a　/var/adm/utmp または指定された *file* を、-b、-d、-l、-p、-r、-t、-T、-u オプションをつけた場合と同様に 処理します。

`-b` 直前のリブートの時間と日付を示します。

`-d` 時間切れになっていて、 `init` によって再生成されていないすべてのプロセスを表示します。`exit` フィールドは、活動していないプロセスについて出力され、 最終プロセスの最終値および終了値 (`wait`(3UCB) が戻す値) を表示します。 これは、プロセス終了の原因を究明するのに役立ちます。

`-H` 標準出力の上のカラムヘッダーを出力します。

`-l` 誰かがログインするのをシステムが待っている 回線のみを表示します。そのような場合、 *name* フィールドは、`LOGIN` です。*state* フィールドが存在しないということを除けば、 その他のフィールドは、ユーザーエントリと同じです。

`-m` 使用している端末についての情報だけを出力します。

`-n` *x* ユーザーの数を行ごとにディスプレイへ指定する数値引数、 *x* が付きます。*x* は、1 以上でなければなりません。`-n` オプションは、`-q` と共に使用してください。

`-p` 現在アクティブで、 `init` によって以前に生成された他のプロセスを表示します。*name* フィールドは、 `/sbin/inittab` 内の `init` によって実行されたプログラム名です。 *state* 、`line`、および *idle* フィールドは、意味がありません。*comment* フィールドは、このプロセスを生成した `/sbin/inittab` からの回線の `id` フィールドを示します。 `inittab`(4) を参照してください。

`-q` (`who` の簡略版) 現在ログオンしているユーザーの名前とユーザー番号しか表示しません。このオプションを使用するときには、その他のオプションはすべて無視されます。

`-r` このオプションは、 `init` プロセスの現在の実行レベル (*run-level*) を示します。

`-s` (デフォルト) *name* 、 `line` 、および `time` のフィールドのみを表示します。

/usr/bin/who

`-T` `-s` オプションと同じです。ただし `-T` では、*state*、*idle* 、*pid*、および *comment* フィールドが表示されます。*state* は次に示す文字のどれか 1 つです。

- `+` 端末に他のユーザーの書き込みが可能である
- `–` 端末に他のユーザーの書き込みが可能でない
- `?` 端末の書き込みアクセス状態が判定できない

/usr/xpg4/bin/who

`-T` `-s` オプションと同じです。ただし `-T` では、 *state* フィールドが表示されます。 *state* はこのオプションの `/usr/bin/who` バージョンに表示されている文字のどれか 1 つです。 `-u` オプションが `-T` と一緒に使用されている場合、 アイドル時間は前のフォーマットの 最後に追加されます。

`-t` root による システムクロックの最後の変更 (date ユーティリティを使用) を示します。 su(1M) および date(1) を参照してください。

`-u` 現在ログイン中のユーザーのみを表示します。*name* は、ユーザーのログイン名です。*line* は回線名です。 /dev ディレクトリにあります。 *time* は、ユーザーがログインした時間です。 *idle* カラムは、ある特定の回線を最後に稼働し始めてからの 経過時間を意味します。ドット (.) は、端末がごく最近起動したことを認識しており、それが 「現在の入力」であることを示します。 24 時間以上が経過したり、 ブート時間から回線が使用されていない場合は、エントリに old とマークされます。 人が端末で操作しているかどうかを判別しようとするとき、 このフィールドが役に立ちます。 *pid* は、ユーザーのシェルのプロセス ID です。 *comment* は、この回線に関連する注釈欄です。 /sbin/inittab (inittab(4) を参照) にあります。 注釈欄には、どこに端末があるか、データセットの電話番号、 直結の場合は端末の型などについての情報を盛り込むことができます。

オペランド

以下にオペランドを示します。

am i

am I C ロケールでは、 起動したユーザーの出力に限られます。 これは -m オプションと同じです。am と、i または I 引数は空白で区切らなければなりません。

file who がデフォルトで使用する、ログインしたユーザーの データベースの代わりに使用するファイルのパス名を 指定します。

環境

who の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、LC_TIME、NLSPATH についての詳細は、environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 入力ファイルはすべて、正常に出力された

>0 エラーが発生した

ファイル

/sbin/inittab	init のスクリプト
/var/adm/utmpx	現在のユーザーとアカウンティングの情報
/var/adm/wtmpx	記録されているユーザーとアカウンティングの情報

属性

次の属性については attributes(5) のマニュアルページを参照してください。

/usr/bin/who

属性タイプ	属性値
使用条件	SUNWcsu

/usr/xpg4/bin/who

属性タイプ	属性値
使用条件	SUNWxcu4

関連項目 date(1), login(1), mesg(1), init(1M), su(1M), wait(3UCB), inittab(4), utmp(4), attributes(5), environ(5), XPG4(5)

注意事項 スーパーユーザーへの注意 : シャットダウン後のシングルユーザー状態において、who は、プロンプトを返します。理由は、/var/adm/utmp がログイン時に更新され、シングルユーザー状態のログインがないと、who がこの状態を正確に報告できないからです。しかし、who am i は、正しい情報を返します。

名前 | whois – インターネットのユーザー名ディレクトリサービス

形式 | **whois** [-h *host*] *identifier*

機能説明 | whois は、名前 (Smith など) と ハンドル (SRI–NIC など) のどちらかの識別子 *identifier* について、インターネット・ディレクトリ・エントリを検索します。名前だけの検索にする場合には、名前の前にピリオドを付けてください。 ハンドルだけの検索にする場合には、 ハンドルの前に感嘆符を付けてください。

グループまたは組織のエントリを検索する場合には、 引数の前にアスタリスク (*) を付けてください。 これにより、 グループの全メンバーのリストが、 レコードとともに表示されます。

感嘆符とアスタリスク、または、ピリオドとアスタリスクを 同時に使用することもできます。

使用例 | 例 1 whois コマンドの使用

次のコマンドは、 SMITH という名前またはハンドルを検索します。

```
example% whois Smith
```

次のコマンドは、 SRI-NIC というハンドルだけを検索します。

```
example% whois !SRI-NIC
```

次のコマンドは、 JOHN SMITH という名前だけを検索します。

```
example% whois .Smith, John
```

名前またはハンドルの引数に . . . を付けると、 その名前またはハンドルの引数で始まる ものがすべて一致します。 つまり、 ZU . . . は、 ZUL、 ZUM などと一致します。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu

関連項目 | attributes(5)

名前 | write – 他のユーザーへのメッセージの書き込み

形式 | **write** *user* [*terminal*]

機能説明 | write ユーティリティは、ユーザーの標準入力から行を読み込み、 他のユーザーの端末へ書き出します。 最初の呼び出しのときに、次のメッセージを *user* で指定されたユーザーの端末へ書き出します。

```
Message from sender-login-id (sending-terminal) [date]...
```

接続が正常に行われると、送り側ユーザーの端末にベルを 2 回送って、入力中の内容が宛先端末に送信されつつあることを示します。

受け取り側は、返答を送りたい場合、最初のメッセージの 受信時に以下のように入力します。

```
write sender-login-id [sending-terminal]
```

標準入力モードでは、NL、EOF、または EOL 特殊文字で区切られた 1 行分の入力データが累積されるたびに、その 1 行分が宛先ユーザーに送信されます。文字は次のように処理されます。

- 警告 (アラート) 文字を入力すると、警告文字が宛先に送られます。
- 行消去文字または文字消去文字を入力すると、termios(3C) インタフェースの規定に従って、送り側の端末が影響を受けます。
- 割り込みまたは EOF 文字を入力すると、 write は対応するメッセージ (C ロケールでは EOT\n) を宛先端末に送って処理を終了します。
- LC_CTYPE における print または space に属する文字を入力すると、それらの文字が宛先端末に送られます。
- stty iexten ローカルモードが有効な場合に限り、 特殊制御文字および複数バイト文字やシングルバイト文字は、 対応するワイド文字が印刷可能であれば、 印刷可能として処理されます。
- その他の印刷不可能な文字を入力した場合は、 以下に述べるような方法で宛先端末に送られます。 制御文字は '^' 文字のあとに適切な ASCII 文字が表示されます。 最上位ビットに 1 がたっている文字は「メタ」表記法で表示されます。 たとえば、 '\003' は '^C' と表示され、 '\372' は 'M-z' と表示されます。

複数ログインしているユーザーへメッセージを送りたいときには、 接続する端末を指定するために *terminal* 引数を使用できます。 それ以外の場合には、 /usr/adm/utmpx 中で見つかった、対象ユーザーの最初の出力可能な端末が宛先端末となります。 このとき、どの端末が選ばれたかを示すために以下の通知メッセージが 送り側の標準出力に書き出されます。

```
user is logged on more than one place.
You are connected to terminal.
Other locations are:terminal
```

write メッセージの受け取り側になると、mesg ユーティリティを使って拒否したり承諾したりすることが可能となります。ただしユーザーがどのような特権を持っているかにより、 他のどのユーザーの端末にアクセスできるかが制限されます。つまり要求した動作を行う上で必要な特権を持っていないと、write ユーティリティの実行は成功しません。

! 文字が行の先頭にあると、write は、シェルを呼び出して行の残りの部分をコマンドとして実行します。

write は、他のユーザー端末への書き込み権を得るために、グループ ID tty への setgid() (setuid(2) を参照) を実行します。

write を使用するために次の手順をお薦めします。ユーザーが相手のユーザーに最初の write を実行したら、メッセージを送り始める前に write を返してくるのを待ちます。 相手がいつ返事をすればよいかがわかるように、 それぞれのユーザーは終了を示すシグナル ((o) は "over (応答どうぞ)" を表す。) を入れてメッセージを終了します。会話を終了するときは、シグナル (oo) ("over and out (通信終わり)" を表す) をお薦めします。

オペランド

以下のオペランドを指定できます。

user メッセージの送信先となる人のユーザー (ログイン) 名。指定形式は、 who(1) ユーティリティが返す形式と同じでなければなりません。

terminal 端末の ID で、 who ユーティリティが返す形式と同一形式。

環境

write の実行に影響を与える環境変数 LC_CTYPE、LC_MESSAGES、NLSPATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0 正常終了

>0 指定されたユーザーはログインしていなかった、 または指定されたユーザーが書き込みを拒否した

ファイル

/var/adm/utmp write 用のユーザーおよびアカウント情報

/usr/bin/sh Bourne シェルの実行可能ファイル

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

mail(1), mesg(1), pr(1), sh(1), talk(1), who(1), setuid(2), termios(3C), attributes(5), environ(5)

診断

`user is not logged on`
ユーザーが `write` を試みている相手はログインしていません。

`Permission denied`
ユーザーが `write` を試みている相手は、書き込みを `mesg` を使って拒否しています。

`Warning: cannot respond, set mesg -y`
ユーザーの端末は `mesg n` に設定され、相手ユーザーが応答できません。

`Can no longer write to user`
ユーザーが書き込みを始めた後で、相手ユーザーが、書き込みを拒否 (`mesg n`) しました。

名前 | xargs – 引数リストを構築してコマンドを実行

形式 | **xargs** [-t] [-p] [-e [*eofstr*]] [-E *eofstr*] [-I *replstr*] [-i [*replstr*]] [-L *number*] [-l [*number*]] [-n *number* [-x]] [-s *size*] [*utility* [*argument. ..*]]

機能説明 | xargs ユーティリティは、オペランドで指定された *utility* と *argument* からなり、その後にいくつかの引数を伴ったコマンド行を生成します。 付加する引数は、標準入力から読み込んだものがその順序どおりに 与えられます。 ただしその長さや数は、オプションで指定した値を超えることはありません。 コマンド行を生成したら、xargs ユーティリティはそのコマンドを実行して、完了を待ちます。 この一連の処理は、標準入力上でファイルの終わりが検出されるまで、 あるいは実行したコマンドが終了ステータス 255 で終了するまで繰り返されます。

標準入力から読み込まれる引数は、引用符で囲まれていない空白文字、 エスケープされていない空白文字、もしくは復帰改行文字で 区切られていなければなりません。 0 個以上の、 二重引用符 (") および復帰改行以外の文字からなる文字列は、 二重引用符で囲むことができます。 0 個以上の、 アポストロフィ (') および復帰改行以外の文字からなる文字列は、 アポストロフィで囲むことができます。 また引用符でかこまれていない文字は、いずれもバックスラッシュ (\) を先頭に付加することによりエスケープできます。 *utility* で指定したユーティリティは、 ファイルの終わりが検出されるまで、繰り返して実行されます。 そのユーティリティが標準入力からデータを読み込もうとした場合、 処理の結果は予測できません。

生成したコマンド行の長さは、 ユーティリティ名のバイト長と文字列として認識された各引数のバイト長 の合計となります。文字列の終端用の NULL 文字も含まれます。 xargs ユーティリティは、生成したコマンドを呼び出すときに、引数と環境 リストをまとめた長さが {ARG_MAX}-2048 バイトを超えないよう、コマンド行の長さを制限します。 この制限範囲の中で、 -n と -s の両オプションがともに省略された場合、 デフォルトのコマンド行の長さの最小値は {LINE_MAX} となります。

オプション | 以下のオプションを指定できます。

-e [*eofstr*] — 論理的なファイルの終わり (EOF) を表す文字列を *eofstr* で指定します。 この -e オプションを省略し、さらに -E も省略すると、デフォルトとして下線 (_) が論理的 EOF 文字列となります。 また -e だけを記述して *eofstr* を省略すると、論理的 EOF 文字列機能は無効となり、下線はそのまま引数の値の一部とみなされます。 xargs の標準入力からの読み込みは、 ファイルの終わりに達するか、 *eofstr* が示す文字列が現れた時点で終了します。

-E *eofstr* — 論理的 EOF 文字列として、デフォルトの下線の代わりに *eofstr* で示す文字列を使用します。 xargs の標準入力からの読み込みは、ファイルの終わりに達するか、 *eofstr* が示す文字列が現れた時点で終了します。

-I *replstr* — 挿入モードを表します。 標準入力から読み込んだ各行に対して *utility* で指定したユーティリティが実行されます。 このとき、読み込んだ行全体を 1 つの引数とみなし、 *replstr* が示す文字列が *argument* 中に含まれていればその位置に読み込んだ行の内容を挿

入します。*argument* で指定する引数のうち、最大 5 つの引数に *replstr* を 1 つまたは複数個含めることができます。 標準入力からの行の先頭が空白文字の場合、それらは無視されます。 挿入によって生成される引数の合計長は、 255 バイトを超えることはできません。 この `-I` オプションを指定した場合は、 `-x` オプションが強制的に有効となります。 `-I` と `-i` は相互に排他的なオプションで、両方が指定された場合には 最後に現れた方が有効です。

`-i` [*replstr*]
: このオプションは `-I` *replstr* と同じ意味を持ちます。引数の *replstr* を省略すると、デフォルトの文字列として { } が用いられます。

`-L` *number*
: *number* で示す数の引数行 (空の行を除く) を標準入力から読み込むたびに、*utility* を実行します。 最後の呼び出し時、標準入力に残っている行数が *number* より少なくても、その引数で *utility* を実行します。各行は、その行で最初に現れた復帰改行をもって終わりと見なされます。ただし行の最後の文字が空白文字の場合、 次の空ではない行に継続することになります。`-L`、`-l`、`-n` の 3 つのオプションは相互に排他的で、 2 つ以上が指定された場合には最後に現れたものが有効です。

`-l` [*number*]
: このオプション (小文字のエル) は、`-L` *number* と同じ意味を持ちます。 *number* を省略すると、`1` とみなされます。この `-l` オプションを指定した場合は、 `-x` オプションも強制的に有効となります。

`-n` *number*
: できるだけ多くの、ただし最大 *number* 個の引数を標準入力から読み込んで *utility* を呼び出します。 *number* は正の 10 進整数です。以下の場合には、引数の個数は少なくなります。

  - コマンド行の累積長が、`-s` オプションで指定された長さ (`-s` オプション省略時には `LINE_MAX` が示す長さ) を超えた場合。
  - 繰り返し実行の最終回に、残っている引数の数が *number* 未満でゼロではない場合

`-p`
: プロンプトモードを表します。*utility* を呼び出すたびに、 実行してもよいかどうかユーザーに問い合わせます。トレースモード (`-t`) が自動的に有効となってその回のコマンド内容を表示し、 その後に プロンプトメッセージが表示されます。 この表示は標準エラー出力に現れます。`/dev/tty` から肯定の応答 (ユーザーのロケールに固有) を読み取るとコマンドを実行し、肯定の応答でない場合はその回の *utility* の実行は行われません。

`-s` *size*
: コマンド行の長さが *size* で示すバイト数を超えない範囲で、できるだけ多くの引数を標準入力から読み込んで *utility* を呼び出します。*size* は正の 10 進整数です。以下の場合には、引数の個数は少なくなります。

  - 引数の合計個数が `-n` オプションで指定した数を超えた場合
  - 行数が `-L` オプションで指定した値を超えた場合

- 行の長さが *size* バイトに到達する前にファイルの終わりを検出した場合

「機能説明」の項で記述されている 制限が満たされている場合は、*size* の値としては、少なくとも {LINE_MAX} バイトまでの値がサポートされます。 その制限値を超えたり、システムがサポートしている最大値を超えたりした 場合でも、エラーとは見なされません。そのような場合 xargs は、制限の範囲内で、サポートできる最大値を使用します。

`-t` トレースモードを有効にします。生成されたコマンド行は、 実行する直前に標準エラー出力に書き出されます。

`-x` 引数の数を表す *number* 引数 (`-n` オプションを参照) あるいは 行数を表す *number* 引数 (`-L` オプションを参照) を含むコマンド行が、 明に指定された長さ (`-s` オプションを参照) あるいは暗に指定された長さを超えた場合、実行を中止します。

オペランド

以下のオペランドを指定できます。

utility 実行するユーティリティの名前。環境変数 PATH を使った検索パスにより得られます (environ(5) を参照) 。 *utility* を省略すると、デフォルトとして echo(1) ユーティリティが実行されます。shell_builtins(1) に定義されている特殊な組み込みユーティリティの名前を *utility* で指定した場合の処理結果は定義されていません。

argument *utility* を実行するときの最初のオプションまたはオペランド。

使用法

xargs から実行されるユーティリティは、 再度呼び出されても現データストリームを使う限り実行は失敗する、という ことがわかっている場合、 終了ステータス 255 を使用して xargs に対し実行を終了するよう通知することができます。 したがって *utility* は、偶然 255 を返すことのないよう、適切な終了コード値を明に指定して 実行を終了することが必要です。

入力データは行の集まりとして解析されます。 引数は空白文字により区切られます。xargs を使って find *dir* -print や ls などのコマンドの出力を、実行対象コマンドの入力とする場合、 ファイル名に空白文字や復帰改行文字が含まれていると、 処理の結果は予測できません。これを防ぐには、 見つかった各ファイル名を引用符つきの文字列に変換するスクリプトを find を使って呼び出し、そのスクリプトを xargs にパイプでつなげるようにしてください。 なお xargs が使う引用符の規則は、シェルの規則とは異なります。 同じ規則を採用しないのは、既存のアプリケーションが現状の規則に 依存しているのに対し、シェルの構文規則は それと互換性を持たないためです。 文字列を xargs が正しく解釈できる形式に変換する簡単な方法は、 各文字の前にバックスラッシュ (\) を付加することです。

{ARG_MAX} 値が大きいシステムにおいては、xargs は {LINE_MAX} より長いコマンド行を生成することがあります。 これはユーティリティを呼び出すうえでは問題ではありません。 xargs を使ってテキストファイルを作成する場合、ユーザーは -s オプションを使ってコマンド行の長さを明に指定しなければなりません。

xargs ユーティリティは、エラーが発生すると終了ステータス 127 を返します。これによりアプリケーションは、 「ユーティリティが見つからなかった」エラーと「実行したユーティリティがエラーで終了した」状態とを 見分けることができます。127 という値を選んだのは、通常この値は他の意味で使われることがないためです。ほとんどのユーティリティは、 小さい値を使って「一般的なエラー状態」を表し、128 より大きな値は、シグナル受信による終了の場合と 区別が困難になりかねないためです。 同じような観点から、ユーティリティは見つかったが実行できなかった、という場合に使う値として 126 が選ばれています。

使用例

例 1 xargs コマンドの使用

次の例では、ディレクトリ $1 中の全ファイルをディレクトリ $2 へ mv コマンドで移動することと、 各実行の直前にそのコマンドの内容を表示することを指定しています。

```
ls $1 | xargs -I {} -t mv $1/{} $2/{}
```

次の例では、 カッコで囲まれたコマンドの実行結果を 1 行にまとめ、 それを log というファイルの終わりに追加出力することを指定しています。

```
(logname; date; printf "%s\n" "$0 $*") | xargs >>log
```

次のコマンドは、元来コマンド行引数として入力された いくつかの対の引数を使って diff を実行します。なお元の引数リストの各要素には 空白文字が埋め込まれていない、と仮定しています。

```
printf "%s\n" "$*" | xargs -n 2 -x diff
```

次の 2 つの例では、 現在のディレクトリ中のどのファイルをアーカイブするかをユーザーに問い合わせ、 指定されたファイルを一度に 1 つずつ (以下の a.)、 または複数個ずつ (以下の b.) arch にアーカイブします。

```
ls | xargs -p -L 1 ar -r arch
ls | xargs -p -L 1 | xargs ar -r arch
```

次のコマンドは、元来コマンド行引数として入力された いくつかの対の引数を使って diff コマンドを実行します。

```
echo $* | xargs -n 2 diff
```

環境

xargs の実行に影響を与える環境変数 LC_COLLATE、LC_CTYPE、LC_MESSAGES、NLSPATH 、PATH についての詳細は、 environ(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

0	*utility* で指定したユーティリティのすべての実行の終了ステータスは 0 だった
1–125	指定された要求に合うコマンド行が生成できなかった、または *utility* で指定されたユーティリティのうちいくつかの実行で 0 でない終了ステータスが返された。 または他の何らかのエラーが発生した

126 *utility* で指定されたユーティリティが見つかったが実行できなかった

127 *utility* で指定されたユーティリティが見つからなかった

指定された要求に合うコマンド行が生成できない、 またはユーティリティが実行できない、または ユーティリティの実行がシグナルにより中断した、 またはユーティリティの実行が終了ステータス 255 で終了した場合、xargs ユーティリティは診断メッセージを出力し、残りの入力データを 処理することなく終了します。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWcsu
CSI	対応済み

関連項目

echo(1), shell_builtins(1), attributes(5), environ(5)

名前 | yacc – 構文解析プログラムの生成

形式 | **/usr/ccs/bin/yacc** [-dltVv] [-b *file_prefix*] [-Q [y | n]] [-P *parser*] [-p *sym_prefix*] *file*

機能説明 | yacc コマンドは、文脈自由文法を、LALR(1) 構文解析アルゴリズムを実行する簡易オートマトン用のテーブル群に 変換します。文法はあいまいであっても構いません。指定された優先順位規則に従ってあいまいな記述を処理します。

出力ファイル y.tab.c は、yyparse() 関数を生成するために C コンパイラを使ってコンパイルする必要があります。 この関数は、字句解析プログラム yylex()、main()、およびエラー処理ルーチン yyerror() とともにロードすることが必要です。 これらのルーチンはユーザーが用意しなければなりませんが、yacc が使用できる字句解析プログラムを生成するには、lex(1) コマンドを使うと便利です。

オプション | 以下のオプションを指定できます。

-b *file_prefix* — すべての出力ファイルに対して、接頭辞として y の代わりに *file_prefix* を使用します。コードファイルの y.tab.c、ヘッダーファイルの y.tab.h (-d が指定されたとき)、記述ファイルの y.output (-v が指定されたとき) は、それぞれ *file_prefix*.tab.c, *file_prefix*.tab.h、 *file_prefix*.output に変更されます。

-d — yacc またはユーザーが割り当てたトークン番号を、 ユーザーが宣言したトークン名に対応させる #define 文を含んだ y.tab.h ファイルを生成します。この対応付けにより、 y.tab.c 以外のソースファイルから トークン番号を参照することが可能となります。

-l — y.tab.c 中に生成するコードには #line 構造を含めないことを指定します。 このオプションは、プログラムの文法および動作が完全にデバッグされるまでは使用すべきではありません。

-P *parser* — /usr/ccs/bin/yaccpar の代わりに 使用する構文解析プログラムの指定をすることができます。 たとえば、以下のように指定できます。

```
example% yacc -P ~/myparser parser.y
```

-p *sym_prefix* — yacc が生成するすべての外部名の接頭辞として、 yy の代わりに *sym_prefix* を使用します。 影響を受ける名前としては、関数 yyparse()、yylex()、yyerror()、および変数 *yylval*、*yychar*、*yydebug* が挙げられます。このマニュアルページでは、これらの 6 つのシンボルを便宜上デフォルト名を使って表しています。 局所的な名前も -p オプションの影響を受けますが、 -p オプションは yacc が生成する #define シンボルには影響を及ぼしません。

-Q[y|n] — -Qy オプションは、バージョン情報を y.tab.c 中に書き出すことを表します。この情報により、どのバージョンの yacc に

よってファイルが生成されたかが分かります。-Qn オプションは、このバージョン情報を書き出さないことを表すもので、 これがデフォルトとなります。

-t　実行時デバッグコードをコンパイルすることをデフォルトとします。 このコードは条件付きコンパイル制御行とともに、常に y.tab.c ファイル中に置かれます。デフォルトでは、 この部分のコードを含まないよう y.tab.c はコンパイルされます。 -t オプションを指定したか否かに関わらず、実行時デバッグコードは、プリプロセッサシンボルである YYDEBUG により制御されます。YYDEBUG の値がゼロ以外のとき、実行時デバッグコードが含まれるようにコンパイルされます。 値がゼロのときは含まれません。このコードなしで生成されたプログラムは、サイズが小さくなり、 実行の速度も少し速くなります。

-V　yacc 用のバージョン情報を標準エラー出力に書き出します。

-v　構文解析用テーブル群、および文法上のあいまいさにより発生した 矛盾点に関するレポートを記述した y.output ファイルを生成します。

オペランド

以下のオペランドを指定できます。

file　構文解析プログラム生成の対象とする命令を 含んでいるファイルのパス名

使用例

例 1 yacc コマンドの使用

cc コマンドのライブラリ検索オペランドに指定することにより、 yacc ライブラリを使用することが可能です。 main という yacc ライブラリを使用するには、以下のコマンドを実行します。

```
example% cc y.tab.c -ly
```

main が lex と yacc の両方のライブラリに存在しているとき、yacc ライブラリの main を使うには以下のコマンドを実行します。

```
example% cc y.tab.c lex.yy.c -ly -ll
```

これにより、最初に yacc ライブラリが検索されるので、 yacc ライブラリ中の main が用いられます。

yacc ライブラリには、通常はアプリケーションプログラマが作成する 2 つの簡単な関数が定義されています。これらの関数は 以下のようなコードです。

```
#include <locale.h>
int main(void)
{
        extern int yyparse();

        setlocale(LC_ALL, "");

        /* If the following parser is one created by lex, the
           application must be careful to ensure that LC_CTYPE
           and LC_COLLATE are set to the POSIX locale.  */
        (void) yyparse();
```

例 1 yacc コマンドの使用　(続き)

```
        return (0);
}

#include <stdio.h>

int yyerror(const char *msg)
{
        (void) fprintf(stderr, "%s\n", msg);
        return (0);
}
```

環境

yacc の実行に影響を与える環境変数 LC_CTYPE、 LC_MESSAGES、 NLSPATH についての詳細は、 environ(5) を参照してください。

yacc は EUC の主および補助コードセット中の文字を 1 文字トークン記号として扱うことができます。 この場合は、 引用符によって囲まれた 1 文字の終端記号でなければなりません。 yacc は、 yylex() がこれらの 1 文字トークン記号に対して 1 つのワイド文字 (wchar_t) を返すものと想定しています。

終了ステータス

以下の終了ステータスが返されます。

0　正常終了

>0　エラーが発生した

ファイル

y.output	生成した構文解析プログラムの状態遷移
y.tab.c	生成した構文解析プログラムのソースコード
y.tab.h	生成した構文解析プログラムのヘッダファイル
yacc.acts	一時ファイル
yacc.debug	一時ファイル
yacc.tmp	一時ファイル
yaccpar	C プログラム用の構文解析プログラムのプロトタイプ

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWbtool

関連項目

cc(1B), lex(1), attributes(5), environ(5)

診断

還元 - 還元衝突 および シフト - 還元衝突の矛盾の数が標準エラー出力に報告されます。詳細な情報は y.output ファイルに出力されます。また開始記号からたどることのできない 規則があれば、これについても報告されます。

注意事項

出力するファイル名が固定されているので、1 つのディレクトリ内で同時に複数の yacc プロセスを動作させることはできません。

識別名にはドル記号 ($) を使用しないようにしてください。

名前 | ypcat – NIS データベース内の値の出力

形式 | **ypcat** [-kx] [-d *ypdomain*] *mname*

機能説明 | ypcat コマンドは、*mname* で指定された NIS ネームサービスマップ内の値を出力します。mname は、マップ名かマップニックネームのいずれかです。ypcat は NIS ネームサービスを使用するため、NIS サーバーは指定されません。

NIS ネットワークサービスの概要については、ypfiles(4) を参照してください。

オプション |

-k　値がヌルであったり、キーが値の一部ではないマップについて、キーを表示します。/etc 内に ASCII バージョンがあるファイルから取り出されたマップは、このクラスに当てはまりません。

-d *ypdomain*　デフォルトドメイン以外のドメインを指定します。

-x　マップのニックネームを表示します。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWnisu

関連項目 | ypmatch(1), ypfiles(4), attributes(5)

名前 | ypmatch – NIS マップ内の 1 つまたは複数のキーの値の出力

形式 | **ypmatch** [-k] [-t] [-d *domain*] *key* [*key*...] *mname*

ypmatch -x

機能説明 | ypmatch は、*mname* で指定された NIS のネームサービスマップ内の 1 つまたは複数のキーに対応付けられた値を出力します。mname は、マップ名またはマップのニックネームです。

複数のキーを指定でき、すべてのキーが同じマップ内で検索されます。これらのキーは、大文字と小文字を区別し同じ長さにしなければなりません。またパターン照合は使用できません。キーが一致しない場合、診断メッセージが生成されます。

オプション | 次のオプションを指定できます。

-k　キーの値を表示する前に、キー自体とコロン (:) を付けて出力します。

-t　マップのニックネームの使用を禁止します。

-d *domain*　デフォルトのドメインではなく特定のドメインを指定します。

-x　マップのニックネームテーブルを表示します。このテーブルには、コマンドが認識しているニックネーム、および各ニックネームに対応付けられているマップネームが示されます。

オペランド | 次のオペランドを指定できます。

mname　NIS's のネームサービスマップ

終了ステータス | 次の終了ステータスが返されます。

0　正常終了

1　エラーが発生した

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWnisu

関連項目 | ypcat(1), ypfiles(4), attributes(5)

注意事項 | 十分な数のファイル記述子が使用できない場合、ypmatch は yp 操作に失敗し、RPC エラーメッセージを出力します。このような場合は、ファイル記述子の数を増やす必要があります。

名前 | yppasswd – NIS データベース内のネットワークパスワードの変更

形式 | **yppasswd** [*username*]

機能説明 | yppasswd ユーティリティーは、NIS (Network Information Service) データベース内のユーザー *username* に対応付けられたネットワークパスワードを変更します。ユーザーが、keylogin(1) を実行しており、NIS の publickey.byname マップ内にそのユーザー用の公開鍵と秘密鍵のペアが存在する場合には、yppasswd は新しいパスワードを使用して秘密鍵を再度、暗号化します。NIS パスワードは、ユーザーのマシンで設定されているローカルなパスワードとは異なる場合もあります。

yppasswd は、古い NIS パスワードの入力を求めたあとで、新しいパスワードの入力を求めます。変更を有効にするためには、古いパスワードを正しく入力する必要があります。新しいパスワードは、誤った入力を防ぐために 2 回入力する必要があります。

新しいパスワードの長さは、大文字や小文字、記号を組み合わせる場合は 4 文字以上、大文字または小文字だけのアルファベットの場合には 6 文字以上でなければなりません。しかし、ユーザーが執拗に要求する場合には、これらの規則は緩和されます。パスワードを変更できるのは、名前の所有者とスーパーユーザーだけです。ルートマスター上のスーパーユーザーは、古いパスワードの入力を求められず、パスワード作成の要件に従う必要もありません。

新しいパスワードを有効にするためには、NIS パスワードデーモン rpc.yppasswdd が NIS サーバー上で実行していなければなりません。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWnisu

関連項目 | keylogin(1), login(1), nis+(1), nispasswd(1), passwd(1), getpwnam(3C), getspnam(3C), secure_rpc(3NSL), nsswitch.conf(4), attributes(5)

警告 | このコマンドを使用してユーザーがパスワードを正常に変更したあとでも、新しいパスワードによる login(1) が行なえるのは、ユーザーのパスワード情報とシャドー情報が NIS から取得される場合だけです。getpwnam(3C)、getspnam(3C)、および nsswitch.conf(4) のマニュアルページを参照してください。

注意事項 | 現在、yppasswd は passwd(1) コマンドのラッパーでしかないため、yppasswd の使用はお勧めできません。代わりに passwd(1) を使用してください。-r nis オプションを指定して passwd(1) (nis+(1) のマニュアルページを参照) を使用すると、同じ結果が得られます。この方法は、現在利用できる種々のネームサービスすべてに渡って一貫性のあるものです。

使用上の留意点 | 更新プロトコルは、1 つの RPC 呼び出しですべての情報をサーバーに渡します。この場合、情報が確認されることがないため、古いパスワードを間違って入力しても、新しいパスワードの入力を終えるまで通知されることはありません。

名前 | ypwhich – NIS サーバーまたはマップマスターの名前の表示

形式 | **ypwhich** [-d *domain*] [[-t] -m [*mname*] | [-Vn] *hostname*]

ypwhich -x

機能説明 | ypwhich は、NIS クライアントに NIS ネームサービスを提供している NIS サーバーの名前、またはマップのマスターである NIS サーバーの名前を返します。引数なしで呼び出した場合、このコマンドはローカルマシンの NIS サーバーを表示します。hostname を指定すると、そのマシンの照会が行われ、マシンで使用されている NIS マスターが表示されます。

NIS ネームサービスの概要については、ypfiles(4) のマニュアルページを参照してください。

オプション | -d *domain* — デフォルトドメインの代わりに *domain* を使用します。

-t — マップのニックネームの使用を禁じます。

-m *mname* — マップのマスター NIS サーバーを見つけます。-m を使用してホスト名を指定することはできません。*mname* には、マップ名、またはマップのニックネームを指定できます。*mname* を省略すると、使用可能なマップの一覧が生成されます。

-x — マップのニックネームの対応テーブルを表示します。

–Vn — ypbind のバージョン。デフォルトは V3 です。

属性 | 次の属性については、attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWnisu

関連項目 | ypfiles(4), attributes(5)

名前 | compress, uncompress, zcat – ファイルの圧縮、圧縮解除、または圧縮解除結果の表示

形式 | **compress** [-fv] [-b *bits*] [*file*...]

compress [-cfv] [-b *bits*] [*file*]

uncompress [-cfv] [*file*...]

zcat [*file*...]

compress | compress ユーティリティは、適応型レンペル・ジブ・コーディング法を利用して、指定されたファイルの圧縮を試みます。出力先が標準出力でないとき、各ファイルは、可能な限り、拡張子 .Z の付いたファイルに置き換えられます。所有モード、アクセス時刻、更新時刻は変わりません。なお .Z をファイルに付加したときパス名の長さが 1023 バイトを超えてしまう場合、コマンドの実行は失敗します。ファイル名が指定されていない場合には、標準入力が圧縮されて結果が標準出力に送られます。

どの程度圧縮されるかは、入力ファイルのサイズ、コードあたりのビット数、共通部分文字列の配置などによって異なります。通常、ソースコードや英語の文章などのテキストの場合、50% から 60% は圧縮されます。この圧縮処理は、ハフマンコーディング法 (pack(1) で使用されている) で行われる圧縮処理よりも優れており、計算時間も短くてすみます。このコマンド中に指定される *bits* パラメータの値は、圧縮結果のファイル内にコード化されます。さらにこのファイル内には、ランダムデータの圧縮解除および圧縮データの再圧縮を防止するためのマジックナンバーも書き込まれます。

uncompress | uncompress ユーティリティは、compress ユーティリティによって圧縮されたファイルを元の状態に復元します。ファイル名を省略すると、標準入力を圧縮解除して標準出力に書き出します。

このユーティリティは、compress が生成したどのようなファイルでも、圧縮解除します。ただし他のシステム上で compress が生成したファイルに関しては、9 および 16 ビット圧縮だけがサポートの対象となります。詳しくは -b の説明を参照してください。

zcat | zcat ユーティリティは、compress により圧縮されたファイルの圧縮解除した内容を、標準出力に出力します。これは uncompress -c と同じ結果となります。入力ファイルの内容は変わりません。

オプション | 以下のオプションを指定できます。

-c | 結果を標準出力に書き出します。既存のファイルは変更されず、.Z ファイルも作られません。つまり uncompress -c という指定は zcat と同機能です。

-f | 圧縮の場合、圧縮してもファイルが小さくならない場合や、対応する *file*.Z ファイルがすでに存在している場合でも、強制的に圧縮を行います。このオプションを指定しない場合は、プロセスがバックグラウンドで実行している場合を除き、既存の *file*.Z ファイルを上書きするかどうかの確認を求めるプロンプトが表示されます。圧縮解除の場合、このオプションは、上書きの確認用プロンプトを表示しないよう指定します。-f オプションを指定しない場合は、プロセスがバックグラウンドで実行してい

る場合を除き、既存のファイルを上書きするかどうかの確認を求めるプロンプトが表示されます。標準入力が端末ではないときに `-f` オプションが省略されると、標準エラー出力に診断メッセージを出力し、`0` より大きな終了ステータスで終了します。

`-v` 冗長 (verbose) 。各ファイルが何パーセント圧縮または圧縮解除されたかを、標準エラー出力に書き出します。

`-b` *bits* 共通部分文字列コードの最大長 (ビット単位) を指定します。*bits* には、9 から 16 の値を指定します (デフォルトは 16 です)。この値を小さくすると、圧縮率が低くなり、結果ファイルは大きくなります。

オペランド

以下のオペランドを指定できます。

file `compress` で圧縮または `uncompress` で圧縮解除するファイルのパス名、または圧縮解除された内容が `zcat` によって標準出力に書き込まれるファイルのパス名。– を指定するか、このオペランドを省略すると、標準入力とみなされます。

使用法

ファイルが 2 ギガバイト (2^{31} バイト) 以上ある場合の `compress` 、`uncompress` 、`zcat` の動作については、`largefile`(5) を参照してください。

環境

`compress` 、`uncompress` 、`zcat` の実行に影響を与える環境変数 `LC_CTYPE` 、`LC_MESSAGES` 、`NLSPATH` の詳細については、`environ`(5) を参照してください。

終了ステータス

以下の終了ステータスが返されます。

`0` 正常終了した

`1` エラーが発生した

`2` (`-f` オプションが指定されていないため) サイズが大きくなってしまい、いくつかのファイルが圧縮されなかった

`>2` エラーが発生した

属性

次の属性については `attributes`(5) のマニュアルページを参照してください。

属性タイプ	属性値
使用条件	SUNWesu
CSI	対応済み

関連項目

`ln`(1)、`pack`(1)、`attributes`(5)、`environ`(5)、`largefile`(5)

関連項目

Usage: compress [`-fvc`] [`-b` maxbits] [*file*. . .]
コマンド行に無効なオプションが指定されました。

Missing maxbits
`-b` の後に最大ビット数が指定されていません、または不正な値か数値でない値が指定されました。

file: not in compressed format
uncompress に指定されたファイルは 圧縮されたファイルではありません。

file: compressed with *xx*bits, can only handle *yy*bits
file は、このマシンの圧縮コードよりも大きいビット数を扱うプログラムによって、圧縮されています。これより小さいビット数でファイルを再圧縮する必要があります。

file: already has .Z suffix -- no change
このファイルは、すでに圧縮されたものと思われます。ファイル名を変更して、再試行してください。

file: already exists; do you wish to overwrite (y or n)?
出力ファイルで置き換える場合は y を、置き換えない場合は n と応答してください。

uncompress: corrupt input
SIGSEGV 違反が検出されました。通常、入力ファイルが破損していることを示します。

Compression: *xx.xx*%
入力ファイルが圧縮によって縮小された割合です。(-v オプションが指定された場合のみ表示します。)

– – not a regular file: unchanged
入力ファイルが通常ファイルではない場合 (たとえばディレクトリ)、内容は変更されません。

– – has *xx* other links: unchanged
入力ファイルにリンクがあります。内容は変更されません。詳細は、ln(1) を参照してください。

– – file unchanged
圧縮しても、縮小されません。入力ファイルの内容は 変更されません。

filename too long to tack on .Z
パス名が長すぎて、接尾辞 .z を付加できません。

注意事項

圧縮されたファイルは、大容量メモリーを持つマシン間では互換性がありますが、プロセスあたりのデータ領域が小さい (64K バイト以下) アーキテクチャへのファイル転送には -b 12 を指定してください。

拡張子 . z の付いたファイルが存在する場合の compress の処理は、もう少し柔軟であるべきです。