ChorusOS™ man pages section 1CC: Target Utilities

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part Number 806-3323
October 4, 1999

Contents

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the man pages and the information it references:

- Section 1CC: User Utilities; Target Utilites
- Section 1M: System Management Utilities
- Section 2DL: System Calls; Data Link Services
- Section 2K: System Calls; Kernel Services
- Section 2MON: System Calls; Monitoring Services
- Section 2POSIX: System Calls; POSIX System Calls
- Section 2SEG: System Calls; Virtual Memory Segment Services
- Section 3FTPD: Libraries; FTP Daemon
- Section 3M: Libraries; Mathematical Libraries
- Section 3POSIX: Libraries; POSIX Library Functions
- Section 3RPC: Libraries; RPC Services
- Section 3STDC: Libraries; Standard C Library Functions
- Section 3TELD: Libraries; Telnet Services
- Section 4CC: Files

- Section 5FEA: ChorusOS Features
- Section 7P: Protocols
- Section 7S: Services
- Section 9DDI: Device Driver Interfaces
- Section 9DKI: Driver to Kernel Interface
- Section 9DRV: Driver Implementations

ChorusOS Man pages are grouped in Reference Manuals, with one reference manual per section

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and **man**(1) for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ‘ "`filename` . . ." .

| Separator. Only one of the arguments separated by this character can be specified at time.

	{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.
PROTOCOL	This section occurs only in subsection 3R to indicate the protocol description file.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.
OPTIONS	This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output - standard output, standard error, or output files - generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each

condition is described in a separate paragraph under the error code.

USAGE

This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality:

Commands
Modifiers
Variables
Expressions
Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.

ENVIRONMENT VARIABLES

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

EXIT STATUS

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.

FILES

This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications.

DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

Target Utilities

NAME

Intro – introduction to user commands

DESCRIPTION

This section describes the user commands and daemons supplied with *ChorusOS*, including system production and configuration utilities.

LIST OF COMMANDS

Caution - This list still needs careful review for ChorusOS 4.0 Pre Release *** NEW PAGES ARE NOT CITED ***

Name:	**Description:**
`C_INIT`	initial ChorusOS actor and command interpreter
`PROF`	profiling daemon
`RDBC`	remote debugging daemon
`RDBK`	system remote debugging daemon
`chorusNS`	CHORUS site name servers
`cs`	display CHORUS Status
`console`	connect to remote target console
`date`	print and set the date
`inetNS`	INET name servers
`mkmerge`	create a merged tree
`profctl`	profiling control
`profrpg`	profiling report generator
`sysenv`	system environment description

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

Name	Description
ChorusOSMkMf(1CC)	Create a Makefile from an Imakefile for ChorusOS
PROF(1CC)	ChorusOS profiler server
catdist(1CC)	concatenate several binary components
chadmin(1CC)	ChorusOS DebugServer administration tool
chconsole(1CC)	ChorusOS Debug Console
chlog(1CC)	ChorusOS DebugServer logging tool
chls(1CC)	ChorusOS Debug List tool
chorusStat(1CC)	print information about ChorusOS resources
chserver(1CC)	ChorusOS DebugServer
conf(1CC)	ChorusOS tunable parameters
configurator(1CC)	ChorusOS configuration utility
configure(1CC)	prepare a build directory for ChorusOS
cp(1CC)	copy files
cs(1CC)	report the status of ChorusOS resources
date(1CC)	print and set the date
dd(1CC)	convert and copy a file
df(1CC)	display free disk space
domainname(1CC)	set or display the name of the current YP/NIS domain
ftp(1CC)	ARPANET file transfer program
hostname(1CC)	set or print name of current host system
ls(1CC)	list directory contents

mkdir(1CC)	create directories
mkfifo(1CC)	make fifos
mkmerge(1CC)	create a merged tree
mv(1)	move files
netboot(1CC)	load and execute standalone programs over the network
netstat(1CC)	show network status
nfsstat(1CC)	display NFS statistics
pax(1CC)	read and write file archives and copy directory hierarchies
profctl(1CC)	ChorusOS profiling control tool
profrpg(1CC)	ChorusOS profiling report generator
rdbc(1CC)	ChorusOS remote debugging daemon
rdbs(1CC)	ChorusOS system debug server for the Microtec XRAY debugger
rm(1)	remove directory entries
rmdir(1CC)	remove directories
touch(1CC)	change file access and modification times
uname(1CC)	display information about the system
ypcat(1CC)	print the values of all keys in a YP database
ypmatch(1CC)	print the values of one or more keys in a YP database
ypwhich(1CC)	return the name of the NIS server or map master

NAME

catdist – concatenate several binary components

SYNOPSIS

catdist *component1 component2... componentn*

DESCRIPTION

The `catdist` utility allows several binary components to be concatenated *component1*, *component2*, ... , *componentn* in order to build a target component which will appear as a unique binary delivery.

In order to do so, `catdist` only copies all original binary components to the same target component and concatenates all the original `Makefile.bin` in a unique `Makefile.bin` for the target component.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`configure`(1CC)

NAME

chadmin – ChorusOS DebugServer administration tool

SYNOPSIS

chadmin –shutdown

chadmin –stat

chadmin –save *file*

chadmin –add-serial-target *name* –device *dev-file* –layout-file *file*

chadmin {–remove-target *name* | –activate *name* | –deactivate *name*}

chadmin –set *name value*

DESCRIPTION

chadmin is the ChorusOS DebugServer administration tool. It allows the DebugServer to be stopped, to obtain statistical information about it, and to register new targets dynamically.

OPTIONS

The following options are supported:

–shutdown — Stops the DebugServer.

–stat — Prints certain statistics collected by the ChorusOS DebugServer.

–save *file* — Saves the DebugServer configuration into file *file*.

–add-serial-target *name* — Registers a new target with a serial line backend. *name* indicates the name of the target, as it will appear in the object tree of the DebugServer.

–device *dev-file* — Allows to specify the serial line device that the backend should use. This option is only relevant when a new target is being registered.

–layout-file *dev-file* — This option indicates the path of the ChorusOS XML layout file. The path should point to the layout.xml file generated during the creation of the target image.

–remove-target *name* — Removes the target *name*

–activate *name* — Activates the target *name*. When the target is activated, the DebugServer gets the control of the serial line and it is possible for debuggers and other tools to control or get access to the target.

–deactivate *name*	Deactivates the target *name*. When the target is deactivated, the DebugServer releases the control of the serial line. The target continues as if it had never been registered to the DebugServer.
–set *name value*	Sets the value of an attribute. Arguments that follow consist of name-value pairs. The name should be an absolute path. The value is converted into an integer depending on the attribute type.

EXTENDED DESCRIPTION

The chadmin tool is used in the following situations:

To register a new target:

The DebugServer needs to know some information about the target so that it can control it. In general, you will register your target only once. The information that you need to pass are the target name, the serial device name and the path of the XML image layout file generated by mkimage.

To update the information about a target:

Sometimes the information that you have specified during the registration changes. This is often the case for the serial line speed and the path of the XML image layout file. You will use the chadmin tool in that case to update the information.

To activate or deactivate a target:

Activation and deactivation of a target is useful when some targets are momentarily not available to the DebugServer, or, when you want that the DebugServer releases the control of the serial line without having to go to the remove and/or register process of the target.

EXAMPLES

The example below illustrates how you can register a new target:

```
chadmin -add-serial-target quintet -device /dev/ttyb\
-layout-file image/LILO/kts/layout.xml
```

If you find out that the serial line was not set correctly, or if you change the dbg.driver.baud tunable for example to increase the speed of the serial line, you can update your target configuration by typing the following commands:

```
chadmin -set targets/quintet/backends/log:bkpts:\
cache:serial/device /dev/ttya
chadmin -set targets/quintet/backends/log:bkpts:\
cache:serial/baud 38400
```

Assuming that `quintet` is the name of your target, `ttya` is the new device and `38400` is the baud rate that you have setup in the debug agent tunable. Sometimes, it happens that the path of the XML layout file is not correct, or is changed. This is the case if you are booting another archive. In that case, you can update the target configuration by typing the following command:

```
chadmin -set targets/quintet/layout_file image/LILO/chorus/layout.xml
```

where `image/LILO/chorus/layout.xml` is the new path.

ENVIRONMENT VARIABLES

The following environment variable is read by `chadmin`:

`CHSERVER_HOST` The host name where the DebugServer is running

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

chserver(1CC)

NAME

chconsole – ChorusOS Debug Console

SYNOPSIS

chconsole [–h] [–`target-host` *name*] [*target*]

DESCRIPTION

`chconsole` is the ChorusOS Debug Console. It prints the messages produced on the target with the *sysWrite*(2K) system call and it provides keyboard input for the *sysPoll*(2K) and *sysRead*(2K) system calls. The `chconsole` tool uses the Debug API and it connects to the `chserver` server. It is intended to be used when the `DEBUG_AGENT` feature is enabled. The `chconsole` tool can also be used for the console management of a boot or prom monitor.

The `chserver` server can control one or several targets. Each of these targets have a separate debug console. When the `chconsole` tool is started, you must specify the name of the target for which you want to have a Debug Console. Your target must be registered to the Debug Server. You can obtain the list of registered targets using the `chls` tool with the –`t` option and you can register a new target with the `chadmin` tool.

If there is only one target controlled by the Debug Server, then `chconsole` will select it automatically. Otherwise, it is necessary to specify the *target* name explicitly on the command line.

When the `chconsole` tool is connected to a target, it displays the message produced by the target (with the *sysWrite*(2K) system calls for example). The input that you type is sent to the target and feeds the *sysPoll*(2K), *sysRead*(2K) system calls or the prom monitor. While the console is running, it also monitors the state of the target and is able to report events such as *Stop*, *Resume*, *Reboot*, *Breakpoint*, *Single step* and more. The event monitoring is controlled by the `event_report` variable.

To exit the `chconsole` tool, type ^C or use the ~q command.

OPTIONS

The following options are supported:

–`target-host` *name*	Indicates the host name where the chserver server is running. You may also use the `CHSERVER_HOST` environment variable to avoid to pass this option. Refer to `chserver` manual page for the setting of this environment variable.
–h	Prints the help description and exit..
target	The list of registered targets is available using the `chls` tool.

EXTENDED DESCRIPTION

Command

A tilde (~) appearing as the first character of a line is an escape signal which directs `chconsole` to perform some special action. `chconsole` recognizes the following escape sequences:

~

~.

`ChorusOSq`	Exit the `chconsole`
`~c [name]`	Change directory to name (no argument implies change to your home directory)
`~!`	Escape to an interactive shell on the local machine (exiting the shell returns you to `chconsole`)
~h	
`~?`	Get a summary of the tilde escapes
`~speed [speed]`	Set the speed of the serial line connected to the DebugServer and the target
`~parity [parity]`	Set the parity of the serial line
`~set var value`	Set a variable (see discussion below)
`~reboot [hard]`	Reboots the target. The reboot command has two mode: a soft and a hard mode. The soft reboot mode assumes that the Debug Agent is running on the target (`DEBUG_AGENT` feature eanbled). In this mode, the reboot is executed by the target itself. In the hard reboot mode, an external tool is used to reset the target. Such external tool can operate directly on the reset line for example.
`~list`	Lists the variables with their value
`~info`	Prints some information about the target

Internal Variables

`chconsole` maintains a set of variables which are used in normal operation. Variables may be displayed with the `~list` command and they may be set with the `~s` escape.

ENVIRONMENT VARIABLES

The following environment variables are read by chconsole:

`CHSERVER_HOST` The host name where the DebugServer is running.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

chadmin(1CC), **chsls**(1CC), **chserver**(1CC), **sysPoll**(2K), **sysRead**(2K), **sysWrite**(2K)

NAME

chlog – ChorusOS DebugServer logging tool

SYNOPSIS

chlog –`level` *n component...*

chlog –`listlevel`

chlog –`report`

DESCRIPTION

The `chlog` is the ChorusOS DebugServer logging tool. The tool has two basic functionalities:

- First, it configures the logging mechanism of the DebugServer.
- Second, it reports the log messages produced by the DebugServer.

Each log message is characterized by the following identifiers:

1. A log level identifier *n*
2. A log component identifier *component*
3. A log message identifier. This is a unique number which identifies the log message. It is allocated for each log message.

OPTIONS

The following options are supported:

–`level` *n component...*	Configures the logging mechanism of the DebugServer. *n* indicates the level of log messages for the components that are listed. A log level of 0 turns off the production of log messages. High values produce more log messages. The highest log level value which is valid is 5. The list of components can be obtained by using the –`list-level` option.
–`list-level`	Prints the list of the DebugServer components and indicates the log levels for each of them.
–`report`	Causes the `chlog` tool to operate in a mode in which it continuously reports the log messages produced by the DebugServer. While log messages are being reported, the log levels may be changed in another window. Press ^C to stop the continuous report.

EXTENDED DESCRIPTION

Log Component Id

The log component identifier *component* is a unique identifier which represents the component (software part) that has produced the log message. This identifier is used to organize the log message and to provide a simple and efficient filter mechanism. The following components identifiers are available:

Component Id	Name	Description
1	rpc	This component represents the server RPC part of the DebugServer front-end. Activation of logs of the DebugServer
2	frontend	This component represents the DebugServer front-end.
3	backend	This component represents the DebugServer back-end.
4	core	This component represents the DebugServer core back-end.
5	serial	This component represents the DebugServer serial line back-end.
6	errors	This component represents all the errors raised by the DebugServer.
7	console	*** TO BE EDITED ***
8	serial-device	This component represents the serial device. Activating the messages of this component will display the data passed on the serial line.

Log Level Id

The log level identifier *n* is a number which indicates the degree of verbosity of the log message. It is used to filter messages which are produced very often from others. Several log levels are pre-defined (first is lowest, last is highest):

`0` Log messages are disabled.

`1` Log messages produced at initialization only.

`2` Log messages to indicate the basic activity. Such log message should only be produced to indicate that the component is entered (called) or left (exit).

`3` First level of debugging. These log messages provide more information about the behavior and the execution of the component.

4 Second level of debugging to produce more information. This log message gives hexadecimal dump of area of memory which is read or written.

5 Highly detailed messages.

Configuring the Log

Configuring the log means defining the log levels for the components of the DebugServer. The following command:

`chlog –level` *n component...*

allows to set the log level of a set of components. The log levels are described in the previous section. A value of `0` disables the log for the given components. High log levels produce more messages. The set of components is specified as a set of strings, each of them refering to a specific component.

The list of components and the current configuration of log levels can be obtained using the command:

`chlog -list-level`

The log levels can be changed while the DebugServer is running.

Displaying Log Messages

There are two ways for displaying log messages: in a file or using the `chlog` command. The following command:

`chlog -report`

continuously displays the log messages produced by the DebugServer. The log levels may be changed while the report is in progress. To stop the report, hit ^C or kill the `chlog` process.

Log Examples

Below are some example of log messages produced by the DebugServer:

```
...
L851:1:2:T57:start 'get_objects'
L852:1:2:T57:'get_objects' done
L853:1:2:T57:start 'list_objects'
L854:3:2:T57:Read 4 bytes memId ffffffff at 0x00000017
L855:3:3:T57:Read result 0
...
L1056:1:2:T57:'list_objects' done
L1063:1:2:T58:start 'list_objects'
L1064:6:2:T58:Error 2: Browsing operation 'scan_threads' not recognized
L1065:1:2:T58:'list_objects' raised an exception
L1066:1:3:T58:Code: 2 Msg: Browsing operation 'scan_threads' not recognized
```

Log Id	Component Id	Log Level	Thread	Message
L851	1	2	T57	start 'get_objects'
L852	1	2	T57	'get_objects' done
L853	1	2	T57	start 'list_objects'
L854	3	2	T57	Read 4 bytes memId...
...				

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

chserver(1CC)

NAME

chls – ChorusOS Debug List tool

SYNOPSIS

chls [–lA] [–la *actor*] [–lE] [–li] [–ll *nbline*] [–lM] [–lR] [–lt *thread*] *target...*

chls [–a] [–b] [–d] [–flat] [–format] [–h] [–l] [–p] [–r] [–t] [*path...*]

DESCRIPTION

chls is a utility tool based on the Debug API to list the objects of the target. The tool has two modes:

- A cs mode in which it has the same options as the cs tool and produces similar output.
- A alternative mode which allows to look at all the objects and attributes of the Debug Object Tree. The selection of the mode is based on the options that you use.

In the cs mode, you must specify a set of cs options followed by one or several target names. The targets that you specify must be registered to the Debug Server (see *chadmin*(1CC) and *chserver*). Refer to the *cs*(1CC) utility for a detailed explanation of the listing produced in this mode.

OPTIONS

The following options are supported in the cs mode which they activate:

–lA	Prints the list of running actors
–la *actor*	Prints the kernel resources belonging to the actor whose local identifier is *actor*
–lE	Prints the kernel configuration environment (see *sysSetEnv*(K)), if any.
–li	Prints target image description
–ll *nbline*	Prints the last nbline lines of the kernel system log (see *sysLog*(2K)), if any
–lM	Prints the list of kernel modules
–lR	Prints the target ram description
–lt *thread*	Gives the status of the thread whose local identifier is *thread*

The following options are supported in the alternative mode:

–a	Lists all the attributes

`-b`	Lists only the attributes related to browsing information
`-d`	Lists the attributes related to debug information
`-flat`	Flat listing
`-format`	Flat listing
`-h`	Help option. The usage of the tool is printed and the tool exits with the status code.
`-l`	Lists the objects
`-p`	Lists the pre-defined attributes related to debug information
`-r`	Recursive listing of the objects
`-t`	Lists the targets registered in the DebugServe

ENVIRONMENT VARIABLES

The following environment variable is read by `chls`:

`CHSERVER_HOST`	The host name where the DebugServer is running.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`chserver`(1CC), **`cs`**(1CC), **`sysLog`**(2K)

NAME

ChorusOSMkMf – Create a Makefile from an Imakefile for ChorusOS

SYNOPSIS

ChorusOSMkMf `config-file` [–b *build-dir*] [–d *dist-dir*] [–s *source-dir*]

DESCRIPTION

Use `ChorusOSMkMf` to create a Makefile from an Imakefile for a ChorusOS project.

OPTIONS

The following options are supported:

`config-file` `config-file` is the configuration file containing all the paths of the different ChorusOS components. This file is generally found in the ChorusOS build directory after a `configure` command; its name is `Paths`. If `config-file` is a directory name, the `Paths` file within this directory will be used.

–b *build-dir* *build-dir* is the directory where the code is built. By default this is the directory from which `ChorusOSMkMf` is called.

–d *dist-dir* *dist-dir* is the directory where the binary delivery is stored. This directory is used if `imake` distribution rules specified in a `Package.rules` file are used.

–s *source-dir* *source-dir* is the directory containing the source code. By default, this is the directory from which `ChorusOSMkMf` is called. If the source directory contains a `Project.tmpl` file, the definitions it contains are applied recursively in the hierarchy.

EXTENDED DESCRIPTION

`ChorusOSMkMf` uses `imake(1)` to generate the Makefile from a project template, a set of cpp macros, and the application's Imakefile. In most cases, an Imakefile contains three things: a list of source files, a number of macros to specify the global compilation rules, and a number of macros which specify how to build the target. An Imakefile can generate several targets.

An Imakefile defines global compilation rules; the compilation rules use the VPATH make mechanism in order to allow multiple builds from the same set of source code.

The `ChorusOSMkMf` command will automatically use the `Project.tmpl` file located at the top-level of either the source directory or the build directory of the application. This `Project.tmpl` file is intended to contain the generic definitions for the whole application hierarchy. Typical information located in the `Project.tmpl` file are compilation flags.

Flag	Value	Default value
WARN	$(WARN_ON), $(WARN_OFF)	$(WARN_ON)
DEBUG	$(DEBUG_ON), $(DEBUG_OFF)	$(DEBUG_OFF)
PROF	$(PROF_ON), $(PROF_OFF)	$(PROF_OFF)
OPT	$(OPT_ON), $(OPT_OFF)	$(OPT_ON)

The `WARN`, `DEBUG`, `PROF`, `OPT` flags are used to set or unset the warning, debug, profiling and optimization options of the compiler.

The `INCLUDES` and `DEFINES` flags are used to specify additional includes directives or variables definitions for the compiler.

An Imakefile also defines a macro with which to build the dependencies, as follows:

```
Depend(srcs)
```

Packaging rules are also available to prepare a binary package for the application. They allow you to select a subset of files of either the original source files, or of the built files to be installed in the binary delivery application package. The various packaging rules are very primitive, they copy the files without using any sophisticated installation rules.

Make Targets

The content of the generated Makefile depends on the original Imakefile. Each Makefile contains two targets for imake management:

- `Makefile`: generates a Makefile in the current directory. Before rebuilding the Makefile, the current Makefile is renamed Makefile.bak.
- `Makefiles`: generates Makefile files recursively, in subdirectories.

Standard cleaning rules are provided:

- `clean` removes the .o objects files and temporary files.
- `veryclean` removes the objects files, executables and libraries.
- `distclean` remove the files from the binary delivery.

In most cases, the generated Makefile contains the following make targets:

- `all`, which constructs everything which must be built.

- `depend`, which constructs the source file dependencies that are used by `make` to maintain the generated files. This target uses the `makedepend(1)` tool.

Most targets are called recursively in all sub-directories.

By default, `ChorusOSMkMf` runs in *silent* mode, and the `make log` includes an outline trace of the operations performed. To disable silent mode and get a full make trace, type:

`make SILENT=''`

or

`gmake SILENT=''`

EXAMPLES

Assuming the `example` directory contains the source code hierarchy of the application and the Imakefiles, and that you want to build an application for a ChorusOS version previously built in the `ChorusOSdir-x86`, use the following sequence of commands to compile the project:

`cd example`

`ChorusOSMkMf ChorusOSDir-x86` (creates the top-level Makefile)

`make Makefiles` (recursively creates all Makefiles)

`make depend`

`make`

To compile your example for the PowerPC platform from the same set of source code, you may use the following sequence of commands:

`mkdir build_powerpc`

`cd build_powerpc`

`ChorusOSMkMf ChorusOSDir_ppc -s ../example`

`make Makefiles`

`make`

Following is a simple example of a `Project.tmpl` file which allows the build of the application in the source directory:

```
DEBUG=$(DEBUG_ON)
DEFINES=-Dvar1 -Dvar2
INCLUDES=-I.
```

In order to build the application in a different directory and to be able to use the packaging rules, the `Project.tmpl` must contain the following:

```
#include "Package.rules"

SRC_DIR         = SourceDir
BUILD_DIR       = BuildDir
DIST_DIR        = DistDir

VPATH           = $(SRC_DIR)$(REL_DIR)
```

Imake compatibility rules with ChorusOS 3.2 are accessible through the directive:

```
#include "Imake32.rules"
```

FILES

All imake description files are located in the `tool/imake` directory of the ChorusOS installation directory:

`Imake.rules`	ChorusOS build rules
`Imake32.rules`	ChorusOS 3.2 compatibility rules
`Package.rules`	Packaging rules
`Imake.tmpl`	Definition of variables

The ChorusOS specific information depending on the host/target configuration are located in the `tool/tgt-make` directory.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

imake(1), **configure**(1CC)

Introducing ChorusOS 4.0

NOTES

Caution - It is not possible to have a compilation in the source directory and another compilation of the same source code in another build directory. The user must choose either to have an unique build in the source directory or multiple builds in other directories.

NAME

chorusStat – print information about ChorusOS resources

SYNOPSIS

chorusStat

DESCRIPTION

`chorusStat` prints information about currently allocated ChorusOS resources such as actors, threads, and memory regions. It prints a heading with the target system physical memory, the memory version (`MEM_FLM`, `MEM_PRM` or `MEM_VM`), and the allocated and unallocated physical memory.

The column headings for the list of actors are described in the following list:

lid The local identifier for the actor, a decimal value.

priv The privilege of the actor (`SUP`, `SYSTEM` or `USER`).

#threads The number of threads in the actor, a decimal value.

text The memory allocated for the text area, a decimal value.

data The memory allocated for the data area, a decimal value.

dyn The memory allocated dynamically, a decimal value.

name The symbolic name of the actor.

DIAGNOSTICS

`chorusStat` is always available as a standalone command. For example:

host% **`rsh`** *target* **`arun chorusStat`**

Running `chorusStat` as a built-in command:

host% **`rsh`** *target* **`chorusStat`**

requires the `ADMIN_CHORUSSTAT` feature to be set to `true`.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`C_INIT`(1M), **`cs`**(1CC)

NAME

chserver – ChorusOS DebugServer

SYNOPSIS

chserver [–`config-dirs` *paths*] [–`log`] [–`log-level` *level*] [–`slot` *n*]

DESCRIPTION

The `chserver` is the ChorusOS DebugServer for debugging ChorusOS in system mode. This server must be started on the host which has a serial line connected to the target. It can manage several targets at the same time. These targets can be running different versions of ChorusOS, and they can be of different architectures (PowerPC or x86).

The `chserver` uses a set of XML files to configure itself for a given ChorusOS kernel and target architecture. It also uses the XML configuration files to find out the list of targets to be managed. Configuration files are usually located in the `etc/debug_server` directory.

OPTIONS

The following options are supported:

–`config-dirs` *paths* — Indicates the path list that the DebugServer must use to find the DebugServer XML configuration files. Paths are separated by a colon(:) on UNIX, and by a semicolon (;) on Windows NT. When no option is specified, the DebugServer uses the content of the `CONFIG_DIRS` environment variable or of the `PATH` environment variable.

–`log` — Turns on log generation on the DebugServer standard output. The log produced by the DebugServer can be turned on or off at run-time, by using `chlog`, or by changing the value of the *stdout_report* on the *log* object.

–`log-level` *level* — Sets the initial log level of all the components of the DebugServer to *level*. The log level of individual components may then be changed by using the `chlog` tool.

–`slot` *n* — Uses the RPC slot number *n* to register the DebugServer. This is only necessary when several DebugServer sessions are to be started on the same host.

ENVIRONMENT VARIABLES

The DebugServer, as well as all tools based on the Debug Library, uses an optional environment variable:

`CHSERVER_HOST` — This environment variable is optional. It can be used to avoid having to pass the –`target-host`

host option. The format of the string as well as this option is:

host:slot-id

When `:slot-id` is present, it indicates a slot number for the DebugServer RPC service. The environment variable is overriden by the `–target-host` *host* option. For example:

```
setenv CHSERVER_HOST jericho
setenv CHSERVER_HOST concerto:3
```

DebugServer Startup

You can specify an RPC slot number for the DebugServer. The slot number is specified using either the `–slot` option or with the environment variable. When no `–slot` is passed, the `CHSERVER_HOST` environment variable is checked, and the DebugServer uses the slot number specified there. If both the option and the environment variable are specified, the same slot number should be assigned, otherwise an error message is issued. When no slot number is specified, the default slot number `0` is used.

It is completely safe for anyone to set the `CHSERVER_HOST` to refer to the host name and slot number of their DebugServer. This is valid for the DebugServer and for the clients. This is useful for avoiding interactions with other DebugServer sessions if a separate environment is required.

The DebugServer is a Sun RPC server that is registered to the rpcbind server. When several DebugServers need to be started on the same host, associate a slot number to each of them so that you can identify the DebugServer you want to use.

If you start only one DebugServer on a given host, it is not necessary to allocate a slot number. The default slot number (`0`) will be used.

PROBLEMS AND SOLUTIONS

`Impossible to initialize the 'name' tool`

When this kind of message is printed by tools like `chls`, `chconsole` or `chadmin`, it means that the DebugServer is not running or could not be found. Check that the DebugServer is running or update your `CHSERVER_HOST` environment variable.

`Symbol table not loaded`

This message is reported by tools, and in particular `chls`, when they try to access some objects on the target and there was a problem while initializing the target information. The problem may be due to:

`The layout file cannot be read: ...`

The path of the XML image layout file is not correct or the file cannot be read. In that case, use the `chadmin` tool to update the path.

`The kernel image does not correspond to the XML layout description file`

A new image was built and the target is running some old image. You must reboot the target with the new kernel image.

`Cannot open serial line 'xxx'`

Check the main configuration file to see whether the serial line device is correct. Make sure the device exists, is known on the host, etc.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

chadmin(1CC), **chlog**(1CC), **chls**(1CC)

NAME

conf – ChorusOS tunable parameters

SYNOPSIS

This manual page describes the tunable parameters of a ChorusOS system,

not a user command.

DESCRIPTION

These variables can be set or modified using the *mktune*(1CC) utility at system build time.

This page describes the tunable parameters of *C_INIT(1M)* only, the default values are shown between parentheses:

cinit.dfl.gid Default user GID (0).

cinit.dfl.uid Default user UID (0).

cinit.rsh.startup Command file executed by C_INIT at system startup (/etc/rc.classix).

cinit.rsh.threadprio Priority of the C_INIT thread (30).

cinit.rsh.threadclass Scheduling class of the C_INIT thread (K_SCHED_FIFO).

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M), **mktune**(1CC), **security**(4CC), **ftpd**(1M), **telnetd**(1M)

NAME

configurator – ChorusOS configuration utility

SYNOPSIS

configurator –c *config-file* [–debug] [–verbose] *command...*

DESCRIPTION

The configurator tool allows static configuration of the ChorusOS kernel.

It allows three levels of configuration:

- Features
- Static tunables
- Environment variables

The `configurator` utility reads the XML configuration files in the `conf` directory and saves the user settings in the originating configuration files.

Setting an environment variable creates an environment file which is included in the ChorusOS boot archive.

After a configuration change, the system must be rebuilt with the **`make build`** command.

OPTIONS

The following options are supported:

–c *config-file* For ChorusOS, *config-file* is `conf/ChorusOS.xml`

–debug Instructs the configurator to apply *command* to all entities. This option gives a more complete status of the information. All XML declarations, whether or not their attributes are visible or configurable are displayed.

–verbose This mode is used for debugging.

OPERANDS

The following commands are supported:

`-help`

Displays the command line options.

`-p` *profile*

Get the list of commands from the *profile* file. Each line in the profile file contains a command, for example:

```
-set EVENT=true
.
```

Lines starting with # are comments.

`-check`

Checks both the XML syntax of the configuration files and the coherency of the configuration information. An error message is issued if an error is found in an XML file during parsing.

`-display` *output-file*

Writes a description of the currently selected configuration in HTML, storing the output in *output-file*. Use the –`debug` option if you need more information about the available configuration options. If you specify both –`debug` and –`display`, the HTML displayed describes the configuration options available, and not the current configuration settings.

`-list` **{***definition* **|** *feature* **|** *tunable* **|** *env***}** **[***pattern***]**

Lists configurable definitions, features, tunables or environment variables that match the *pattern*. If you do not specify a *pattern*, all the definitions, features, tunables, or environment variables are listed.

{`-set` *name=value...* **|** `-reset` *name...* **}**

Sets the named feature or tunable to *value*. Depending on the type of the *name* declaration, *value* may be a boolean (true, false), an integer or hexadecimal value, or a string.

Resets the value of *name* to its default value.

{`-setenv` *name=value...* **|** `-resetenv` *name...* **}**

Adds the variable *name* with value *value* to the ChorusOS environment.

Removes the variable *name* from the ChorusOS environment.

`-action` *name*

Executes the action described by *name*. For internal use only.

EXAMPLES

To add the EVENT feature to your default configuration, type:

```
configurator -c conf/ChorusOS.xml -set EVENT=true
```

.

FILES

The following files are used by this utility:

`conf/ChorusOS.xml` Is the top level configuration file. It contains references to all other configuration files:

```
conf/typedef.xml
conf/mkconfig/mkconfig.xml
conf/mkconfig/os.xml
conf/mkimage/mkimage.xml
```

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

sysenv(1M), **sysGetEnv**(2K), **sysSetEnv**(2K), **sysUnsetEnv**(2K)

Graphical Configuration Tool, In: *Introducing ChorusOS 4.0*

NAME

configure – prepare a build directory for ChorusOS

SYNOPSIS

configure [–f *profile-file*...] [–s *source-dir*...] [–b *bin-dir*...] [–d *var=value*...] [–t *build-dir*]

DESCRIPTION

The `configure` utility is a program used to prepare a build directory for ChorusOS by selecting a number of components either in source or in binary code.

`configure` prepares a directory where the different ChorusOS components will be built. The build actions may differ from one component to the other, typical build actions are compilation, build of a ChorusOS archive, configuration, build of a NFS root directory for ChorusOS targets.

OPTIONS

The following options are supported:

–f *profile-file*	This option is only used for source components to be built with the `mkmerge`, such as the ChorusOS nucleus. The *profile-file...* arguments are names for the profile files given as argument to mkmerge.
–s *source-dir*	The *source-dir...* arguments are names for the pathnames of the different source components.
–b *bin-dir*	The *bin-dir...* are arguments names for the pathnames of the different binary components.
–d *var=value*	This option allows to add definitions of variables in the `Paths` configuration file.
–t *build-dir*	This option allows to select a build directory. The default value is the current directory.

The component directories given as arguments to configure through the –s and –b options are read-only. All build files related to those components are created in the build directory.

Each component given as argument to configure must contain component Makefiles:

- `Makefile.bin` for binary components given through the –b option
- `Makefile.src` and `Makefile.bin` for source components given through the –s option

These component Makefiles contains all actions to be done to use the component in the ChorusOS environment.

`configure` creates two files in the build directory:

- `Makefile`
- `Paths`

`Makefile` `Makefile` is the main makefile, it includes all `Makefile.src` and `Makefile.bin` of all configured components and thus allows to do the actions of each of those components.

`Paths` The `Paths` file defines variables referencing the pathnames of all configured components. If the `–d` option is used in the `configure` command, it allows for example to set the cross compiler installation directory with the `CDS_DIR` variable.

There may be two variables defined per component, for example for a component named COMP:

`COMP`	The source directory for `COMP` components
`COMP_DIR`	The directory where `COMP` binaries are located

`COMP` is used by the `COMP` build rules in order to generate `COMP` from its source code (`Makefile.src`).

`COMP_DIR` may be used by other components that depend on `COMP`. Only `COMP_DIR` is relevant if `COMP` is provided as a binary component.

A concrete example is the generation of a board support package (BSP) for ChorusOS depends on the `NUCLEUS_DIR` directory containing the binary delivery providing the support for this target board. `NUCLEUS_DIR` is used in the `Makefile.src` and `Makefile.bin` makefiles of the BSP.

FILES `Makefile.src` in the `COMP` component defines how to build the binary component from the source component.

- The target `COMP.all` contains the build actions.

  If the component depends on another component, `COMP1` for example, it is necessary to express in the `Makefile.src` that `COMP.all` depends on `COMP1.all` so that build actions are done in the right order , that is `COMP1` builds before `COMP`.

One component cannot depend on the fact that the component it depends on, is in binary or in source form.

The build of the component `COMP` is done in the directory `build-COMP`.

In the examples, provided with ChorusOS, the source code is not copied into the `build-COMP` directory but accessed through the `VPATH` mechanism.

- The target `COMP.dist` allows to prepare a binary distribution of the component `COMP`. The convention is that this binary distribution is done in the directory `dist-COMP` of the build directory. Files to be included in the binary delivery are generally copied from the directory `build-COMP` into the directory `dist-COMP`. In particular, the binary distribution must at least contain a `Makefile.bin` file for the binary component.

`Makefile.src` files should be written in such a way that if the directory `build-COMP` is removed, `make` regenerates it.

`Makefile.bin` in the `COMP` component defines at least its name. If `COMP` allows some actions, those actions are present in `Makefile.bin`. Such typical actions are:

- ChorusOS archives generation
- Component configuration
- Creation of a root file system to be mounted on the target

All `Makefile.bin` must be written so that they can be concatenated and allow the installation of the different binary components in the same installation directory.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`mkmerge`(1CC), **`configurator`**(1CC), **`ChorusOSMkMf`**(1CC), **`catdist`**(1CC)

NAME

cp – copy files

SYNOPSIS

cp [–R][–f|–i]

cp [–R][–f|–i]

DESCRIPTION

In the first form in the synopsis, the utility copies the contents of the *source_file* to the *target_file*. In the second form of the synopsis, the contents of each named *source_file* are copied to the destination *directory*. The names of the files themselves are not changed. If `cp` detects an attempt to copy a file to itself, the copy will fail.

The following options are available:

`–R` If *source_file* designates a directory, `cp` copies the directory and the entire subtree connected at that point. This option allows `cp` to create special files rather than copying them as normal files. The directories created are identical to the corresponding source directory.

`–f` For each existing destination pathname, removes it and creates a new file, without prompting for confirmation, regardless of the permissions.

`–i` Causes `cp` to write a prompt to the standard error output before copying a file that would overwrite an existing file. If the response from the standard input begins with the character `'y'`, the file copy is attempted.

RESTRICTIONS FOR ChorusOS

In ChorusOS, `cp` will `NOT` preserve file attributes such as modification time, access time, file flags, file mode, user ID, and group ID. Therefore, the `-p` option is not present. Only file modes will be preserved (this may depend on the underlying file system).

In the second form shown in the synopsis, *target_directory* must exist, unless there is only one named *source_file* which is a directory and the `-R` flag is specified.

Appropriate permissions are required for file creation or overwriting.

DIAGNOSTICS

If successful, an exit status of 0 is returned, otherwise, a positive exit status is returned.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **dd**(1CC), **mv**(1CC)

NAME

cs – report the status of ChorusOS resources

SYNOPSIS

cs [–l[A]] [–lt<thread>] [–lp<port>] [–lpa] [–lpm] [–la<actor>] [–ls] [–lU] [–lu] [–lPA] [–lPa<actor>] [–ll<nblines>] [–lr] [–lE] [–lM]

OPTIONS

The options are the following:

–l[A]	prints the list of running actors (default option).
–lt *thread*	gives the status of the thread whose local identifier is *thread*.
–lp *port*	gives the status of the port whose local identifier is *port*.
–lpa	gives the status of all ports on the local site.
–lpm	gives the status of all ports with messages on the local site.
–la *actor*	prints the kernel resources belonging to the actor whose local identifier is *actor*.
–ls	prints the current scheduler status and run queues. The identity and priority of the thread currently running is provided. A list of thread ready to run is printed.
–lU	prints the complete list of Remote Unique Identifiers currently known on the local site.
–lu	prints the complete list of Local Unique Identifiers currently known on the local site.
–lPA	prints the list of persistent actors.
–lPa *actor*	prints the kernel resources belonging to the persistent actor whose local identifier is *actor*.
–ll *nbline*	prints the last *nbline* lines of the kernel system log (see *sysLog*(K)), if any.
–lr	prints kernel resources statistics.
–lE	prints the kernel configuration environment (see *sysSetEnv*(K)), if any.
–lM	prints the list of kernel modules.

DESCRIPTION

The `cs` utility prints information about currently allocated kernel resources such as actors, threads, ports, messages and memory regions. In each case, it prints a heading with the version number of the Kernel, the identifier of the siste where the Kernel is running and the time elapsed since the last reboot.

If no options are specified, *cs* prints the list of actors running actors on the site where it is executed.

The column headings and the column content for the list of actors are detailed below:

`ACTOR-UI` The actor's 64-bit Unique Identifier expressed as two hexadecimal numbers. The first one is the head, the second one is the tail.

`KEY` The actor's 64-bit modification key expressed as two hexadecimal numbers. The first one is the key head, the second one is the key tail.

`LID` The actor's local identifier, expressed as a decimal value.

`TYPE` The type of the actor: `SUP` if the actor is a supervisor actor, `USER` if the actor is a system actor or a user actor.

`STATUS` The actor's status: `STARTED` if the actor is active, `STOPPED` if it is stopped.

`TH#` The number of threads in the actor, expressed as a decimal number.

`NAME` The symbolic name of the actor.

For thread status information, the column headings and the column content are detailed below.

`THREAD-LI` The thread's local identifier, expressed as a decimal number.

`PRIORITY` The thread's priority, expressed as a decimal number. This thread priority is not printed according to the scheduling class, but is an absolute thread priority. Absolute thread priorities vary from 0 and 256, where 0 is the highest priority. 256 is reserved for "idle" threads which are internal threads which, on each processor, only run when no other thread is ready too run.

`TT` The total cpu thread time, expressed a a decimal number in milliseconds.

IT	The time the thread spent running in the actor in which it was originally created, expressed as a decimal number, in milliseconds.
CTX	A hexadecimal address for the thread's internal structure.
SC-MS-PN	Internal information about the thread, expressed as a set of 3 hexadecimal numbers. SC is the thread's suspend counter; MS is the thread's mask flags set; PN is the thread's pending flags set.
NAME	The thread name, if any, expressed as a string.

For port status information, the column headings and column content are detailed below.

PORT-UI	The port's 64-bit unique identifier expressed as two hexadecimal numbers. The first one is the head, the second one is the tail.
PORT-LI	The port's local identifier expressed as a decimal number.
ENABLED	Whether or not the port is enabled. If enabled, yes is printed, if not, no is printed.
CTX	The port context address, expressed as a hexadecimal number.
MSG#	The number of messages queued behind the port.
ACTOR-UI	The port's actor 64-bit unique identifier.

For memory status information, obtainable by using the -la option described above, the column headings and the column content are detailed below.

START	The virtual start address of the region, expressed as a hexadecimal address.
SIZE	The size of the region, expressed as a hexadecimal number of bytes.
OFFSET	The offset of the region in its segment, expressed as a hexadecimal number of bytes.
ALLOC	The amount of physical memory allocated to the region, expressed as a hexadecimal number of bytes.

OPTIONS The attributes of the region:

RW	Writable
EX	Executable
SU	Supervisor
NS	Non-Swappable
FZ	Fill Zero
ND	No Demand
NW	No Wait for Memory
AW	Anywhere
ST	Stack
IS	Inherit Share
IC	Inherit Copy

DIAGNOSTICS

error: invalid thread	The local identifier does not specify a valid thread.
error: invalid port	The local identifier does not specify an enabled port.
error: invalid message	The local identifier does not specify a message.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M)

NAME

date – print and set the date

SYNOPSIS

date [–u] [+format]

DESCRIPTION

The `date` command can be started using the arun C_INIT(1M) command.

If no argument is given or if the argument begins with "+", the current date is printed. Otherwise the current date is set (if the user is trusted).

The options are as follows:

`-u` — Print or set the date in UCT (universal) time.

+format — An operand with a leading plus ("+") sign signals a user-defined format string which specifies the format in which to display the date. The format string may contain any of the conversion specifications described in the *strftime(3STDC)* manual page, as well as any arbitrary text. The format string for the default display is: `"%a %b %e %H:%M:%S %Z"`

[yy[mm[dd[hh]]]]mm[.ss] — If an operand does not have a leading plus sign, it is interpreted as a value for setting the system's notion of the current date (everything except the minutes is optional). The canonical representation for setting the date and time is:

yy	Year in abbreviated form (for example, 89 for 1989).
mm	Numeric month. A number from 1 to 12.
`dd`	Day, a number from 1 to 31.
hh	Hour, a number from 0 to 23.
mm	Minutes, a number from 0 to 59.
.ss	Seconds, a number from 0 to 61 (59 plus a a maximum of two leap seconds).

Time changes for Daylight Saving and Standard time and leap seconds and years are handled automatically.

EXAMPLES

The command:

```
date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
```

will display:

```
DATE: 11/21/87 TIME: 13:36:16
```

ENVIRONMENT VARIABLES

The TZ environment variable affects the execution of `date`.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`univTime`(2K), **`strftime`**(3STDC), **`time`**(3STDC), **`tzset`**(3STDC)

NAME

dd – convert and copy a file

SYNOPSIS

dd [if=<inputfile>] [of=<outputfile>] [bs=<blocksizebothinput/output>]
[ibs=<inputblocksize>] [obs=<outputblocksize>]
[cbs=<conversionbuffersize>] [count=<copyonlynrecords>]
[files=<copyninputfiles>] [skip=<skipninputrecords>]
[seek=<skipnrecords>] [conv=<conversionfile>]

DESCRIPTION

The `dd` utility copies the standard input to the standard output. Input data is read and written in 512-byte blocks. If input reads are short, input from multiple reads are aggregated to form the output block. When finished, `dd` displays the number of complete and partial input and output blocks and truncated input records to the standard error output.

OPTIONS

The options are:

`bs=n` Set both input and output block size, superseding the *ibs* and *obs* operands. If no conversion values other than *noerror*, *notrunc* or `sync` are specified, each input block is copied to the output as a single block without any aggregation of short blocks.

`cbs=n` Set the conversion record size to *n* bytes. The conversion record size is required by the record oriented conversion values.

`count=n` Copy only *n* input blocks.

`files=n` Copy *n* input files before terminating. This operand is only applicable when the input device is a tape.

`ibs=n` Set the input block size to *n* bytes instead of the default of 512.

`if=file` Read input from `file` instead of the standard input.

`obs=n` Set the output block size to *n* bytes instead of the default of 512.

`of=file` Write output to `file` instead of the standard output. Any regular output file is truncated unless the *notrunc* conversion value is specified. If an initial portion of the output file is skipped (see the *seek* operand) the output file is truncated at that point.

`seek=n` Seek *n* blocks from the beginning of the output before copying. On non-tape devices, a *lseek 2* operation is used.

Otherwise, existing blocks are read and the data discarded. If the user does not have read permission for the tape, it is positioned using the tape *ioctl 2* function calls. If the seek operation is past the end of file, space from the current end of file to the specified offset is filled with blocks of NUL bytes.

`skip=n` Skip *n* blocks from the beginning of the input before copying. On input which supports seeks, a *lseek 2* operation is used. Otherwise, input data is read and discarded. For pipes, the correct number of bytes is read. For all other devices, the correct number of blocks is read without distinguishing between a partial or complete block.

`conv=value` Where *value* is one of the symbols from the following list:

ascii, oldascii The same as the *unblock* value except that characters are translated from ECBDIC to ASCII before the records are converted. (These values imply *unblock* if the operand *cbs* is also specified.) There are two conversion maps for ASCII. The value *ascii* specifies the recommended one which is compatible with System V. The value *oldascii* specifies the one used in historic AT&T and pre-4.3BSD-reno systems.

block Treats the input as a sequence of newline or end-of-file terminated variable length records independent of input and output block boundaries. Any trailing newline character is discarded. Each input record is converted to a fixed length output record where the length is specified by the *cbs* operand. Input records shorter than the conversion record size are padded with spaces. Input records longer than the conversion record size are truncated. The number of truncated input records, if any, are reported to the standard error output at the completion of the copy.

ebcdic, ibm, oldebcdic, oldibm The *block* $$$value except that characters are translated from ASCII to EBCDIC after the records are converted. (These values imply *block* if the operand *cbs* is also specified.) There are four

conversion maps for EBCDIC. The value *ebcdic* specifies the recommended one, which is compatible with System V. The value *ibm* is a slightly different mapping, which is compatible with the AT&T System V *ibm* value. The values *oldebcdic* and *oldibm* are maps used in historic AT&Tand pre-4.3BSD-reno systems.

lcase — Transform uppercase characters into lowercase characters.

noerror — Do not stop processing on an input error. When an input error occurs, a diagnostic message followed by the current input and output block counts will be written to the standard error output in the same format as the standard completion message. If the sync conversion is also specified, any missing input data will be replaced with *NUL* bytes (or with spaces if a block oriented conversion value was specified) and processed as a normal input buffer. If the sync conversion is not specified, the input block is omitted from the output. On input of files which are not tapes or pipes, the file offset will be positioned past the block in which the error occurred using *lseek 2*.

notrunc — Do not truncate the output file. This will preserve any blocks in the output file not explicitly written by dd. The *notrunc* value is not supported for tapes.

osync — Pad the final output block to the full output block size. If the input file is not a multiple of the output block size after conversion, this conversion forces the final output block to be the same size as preceding blocks for use on devices that require regularly sized blocks to be written. This option is incompatible with use of the *bs=n* block size specification.

swab	Swap every pair of input bytes. If an input buffer has an odd number of bytes, the last byte will be ignored during swapping.
sync	Pad every input block to the input buffer size. Spaces are used to pad bytes if a block oriented conversion value is specified, otherwise *NUL* bytes are used.
ucase	Transform lowercase characters into uppercase characters.
unblock	Treats the input as a sequence of fixed length records independent of input and output block boundaries. The length of the input records is specified by the *cbs* operand. Any trailing space characters are discarded and a newline character is appended.

Where sizes are specified, a decimal number of bytes is expected. If the number ends with a ''b'', ''k'', ''m'' or ''w'', the number is multiplied by 512, 1024 (1K), 1048576 (1M) or the number of bytes in an integer, respectively. Two or more numbers may be separated by an ''x'' to indicate a product.

When finished, dd displays the number of complete and partial input and output blocks, truncated input records and odd-length byte-swapping blocks to the standard error output. A partial input block is one where less than the input block size was read. A partial output block is one where less than the output block size was written. Partial output blocks to tape devices are considered fatal errors. Otherwise, the rest of the block will be written. Partial output blocks to character devices will produce a warning message. A truncated input block is one where a variable length record oriented conversion value was specified, and the input line was too long to fit in the conversion record, or was not newline terminated.

Normally, data resulting from input or conversion or both are aggregated into output blocks of the specified size. After the end of input is reached, any remaining output is written as a block. This means that the final output block may be shorter than the output block size.

DIAGNOSTICS

If successful, the dd utility returns 0, otherwise >0 if an error occurred.

The *files* operand and the *ascii*, *ebcdic*, *ibm*, *oldascii*, *oldebcdic*, and*oldibm* values are extensions to the POSIX standard.

RESTRICTIONS FOR ChorusOS

Tapes are not currently supported with this utility.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

df – display free disk space

SYNOPSIS

df [–t<type>][filesystem | file] [–i] [–n]

DESCRIPTION

This utility displays statistics about the amount of free disk space on the *filesystem* specified, or on the filesystem of which `file` is a part. Values are displayed in 512-byte per block block counts. If neither a file nor a filesystem operand is specified, statistics for all mounted filesystems are displayed (subject to the *-t* option below).

OPTIONS

The following options are available:

`-i` Include statistics on the number of free inodes.

`-n` Print out the previously obtained statistics from the filesystems. This option should be used if there is a possibility that one or more filesystems are in a state such that they will not be able to provide statistics without a long delay. When this option is specified, `df` will not request new statistics from the filesystems, but will respond with the (possibly) stale statistics that were previously obtained.

`-t` Only print out statistics for filesystems of the specified types. The recognized types are: *ufs*, *nfs*, *mfs*, *lfs*, *msdos*, *fdesc*, *portal*, *kernfs*, *procfs*, *afs* and *isofs*. The aggregates are: all (the default), local (ufs, mfs, lfs, msdos, isofs), and misc (fdesc, portal, kernfs, procfs). The string ‘‘no’’ may be prepended to a type to get its complement (for example, ‘‘nonfs’’ to get non-NFS filesystems). The first *-t* option overrides the default, additional such options will add to (or subtract from) the current set of types; for example, either *df -t ufs -t lfs* or *df -t local -t nomfs* will display statistics for UFS and LFS filesystems. On top of ChorusOS *ufs* and *nfs* are only supported for `df`.

ENVIRONMENT VARIABLES

`BLOCKSIZE` If the environment variable `BLOCKSIZE` is set, the block counts will be displayed in units of the size specified.

BUGS

The *-n* and *-t* flags are ignored if a file or filesystem is specified.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **statfs**(2POSIX), **fstatfs**(2POSIX), **getfsstat**(2POSIX), **getmntinfo**(3POSIX)

NAME

domainname – set or display the name of the current YP/NIS domain

SYNOPSIS

domainname [*name-of-domain*]

DESCRIPTION

If no argument is passed, domainname displays the name of the current domain, which typically encompasses a group of hosts under the same administration.

The domain name can be set by supplying an argument.

NOTE

The YP/NIS (formerly ‘‘Yellow Pages’’ but renamed for legal reasons) domain name ID is not necessarily related to the Domain Name System domain name, although they are often set to equal for administrative convenience.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

inetNS(1M), **ypbind**(1M)

NAME

ftp – ARPANET file transfer program

SYNOPSIS

ftp [–dgintv] [*host* [*port*]]

DESCRIPTION

The `ftp` utility is the user interface to the *ARPANET* standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The following options may be specified at the command line, or to the command interpreter.

`–d` Enables debugging.

`–g` Disables file name globbing.

`–i` Turns off interactive prompting during multiple file transfers.

`–n` Restrains `ftp` from attempting *auto-login* upon initial connection. If auto-login is enabled, ftp will check the `.netrc` (see below) file in the user's home directory for an entry describing an account on the remote machine. On top of ChorusOS the user's home directory is replaced by the *root* directory. If no entry exists, ftp will prompt for the remote machine login name (see *security*(4CC)), and, if necessary, prompt for a password and an account with which to login (see RESTRICTIONS).

`–t` Enables packet tracing.

`–v` The verbose option forces `ftp` to show all responses from the remote server, as well as report on data transfer statistics.

The client host with which ftp is to communicate may be specified on the command line. If this is done, ftp will immediately attempt to establish a connection to an *FTP* server on that host; otherwise, ftp will enter its command interpreter and await instructions from the user. When ftp is awaiting commands, the prompt *ftp>* is provided to the user.

The following commands are recognized by ftp:

`macro-name [args]`

Execute the macro `macro-name` that was defined with the `macdef` command. Arguments are passed to the macro unglobbed.

`append local-file [remote-file]`

Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being

altered by any *ntrans* or *nmap* settings. File transfer uses the current settings for `type` , `format` , *mode* , and *structure*.

`ascii`

Set the file transfer `type` to network `ASCII`. This is the default type.

`bell`

Sound a bell after each file transfer command is completed.

`binary`

Set the file transfer `type` to support binary image transfer.

`bye`

Terminate the `FTP` session with the remote server and exit ftp . An end of file will also terminate the session and exit.

`case`

Toggle remote computer file name case mapping during *mget* commands. When `case` is on (default is off), remote computer file names with all letters in upper case are written to the local directory with the letters mapped to lower case.

`cd remote-directory`

Change the working directory on the remote machine to *remote-directory* .

`cdup`

Change the remote machine's working directory to the parent of the current remote machine's working directory.

`chmod mode file-name`

Change the permission modes of the file *file-name* on the remote sytem to *mode* .

`close`

Terminate the `FTP` session with the remote server, and return to the command interpreter. Any defined macros are erased.

`cr`

Toggle carriage return stripping during ascii type file retrieval. Records are denoted by a carriage return/linefeed sequence during ascii type file transfer. When *cr* is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ascii type transfer is made, these linefeeds may be distinguished from a record delimiter only when *cr* is off.

`delete remote-file`

Delete the file *remote-file* on the remote machine.

`debug debug-value`

Toggle debugging mode. If an optional *debug-value* is specified, it is used to set the debugging level. When debugging is on, ftp prints each command sent to the remote machine, preceded by the string '—>'.

`dir [remote-directory] [local-file]`

Print a listing of the directory contents in the directory, *remote-directory* , and, optionally, place the output in *local-file* . If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving *dir* output. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is – , the output is sent to the terminal.

`disconnect`

A synonym for *close* .

`form format`

Set the file transfer *form* to `format` . The default format is ''file'' .

`get remote-file [local-file]`

Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current `case` , *ntrans* , and *nmap* settings. The current settings for `type` , *form* , *mode* , and *structure* are used while transferring the file.

`glob`

Toggle filename expansion for *mdelete* , *mget* and mput . If globbing is turned off with `glob` , the file name arguments are taken literally and not expanded. Globbing for *mput* is done as in *csh(1UNIX)* . For *mdelete* and *mget* , each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name will probably be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by doing 'mls remote-files -' Note: *mget* and *mput* are not meant to transfer entire directory subtrees of files. That can be done by transferring a *tar(1UNIX)* archive of the subtree (in binary mode).

`hash`

Toggle hash-sign (''#'') printing for each data block transferred. The size of a data block is 1024 bytes.

`help [command]`

Print an informative message about the meaning of `command` . If no argument is given, ftp prints a list of the known commands.

`idle [seconds]`

Set the inactivity timer on the remote server to *seconds* seconds. If *seconds* is omitted, the current inactivity timer is printed.

`lcd [directory]`

Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

`ls [remote-directory] [local-file]`

Print a listing of the contents of a directory on the remote machine. The listing includes any system-dependent information that the server generated, for example, most UNIX systems will produce output from the command *'ls –l'* . (See also `nlist` .)If *remote-directory* is left unspecified, the current working directory is used. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving `ls` output. If no local file is specified, or if *local-file* is '–' , the output is sent to the terminal.

`macdefNs macro-name`

Define a macro. Subsequent lines are stored as the macro *macro-name* ; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a *close* command is executed. The macro processor interprets '$' and '\' as special characters. A '$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '$' followed by an 'i' signals to the macro processor that the executing macro is to be looped. On the first pass '$i' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '$'.

`mdelete [remote-files]`

Delete the *remote-files* on the remote machine.

`mdir remote-files local-file`

Similar to *dir* , except multiple remote files may be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving *mdir* output.

`mget remote-files`

Expand the *remote-files* on the remote machine and do a `get` for each file name thus produced. See `glob` for details on the filename expansion. Resulting file names will then be processed according to `case` , *ntrans* , and *nmap* settings. Files are transferred into the local working directory, which can be changed with '*lcd directory*' ; new local directories can be created with '*!mkdir directory*' .

`mkdir directory-name`

Make a directory on the remote machine.

`mls remote-files local-file`

Similar to *nlist* , except multiple remote files may be specified, and the *local-file* must be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving *mls* output.

`mode [mode-name]`

Set the file transfer *mode* to *mode-name* . The default mode is ''*stream*'' mode.

`modtime file-name`

Show the last modification time of the file on the remote machine.

`mput local-files`

Expand wild cards in the list of local files given as arguments and do a *put* for each file in the resulting list. See `glob` for details of filename expansion. Resulting file names will then be processed according to *ntrans* and *nmap* settings.

`newer file-name`

Get the file only if the modification time of the remote file is more recent than the file on the current system. If the file does not exist on the current system, the remote file is considered newer. Otherwise, this command is identical to `get` .

`nlist [remote-directory] [local-file]`

Print a list of the files in a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving *nlist* output. If no local file is specified, or if *local-file* is – , the output is sent to the terminal.

`nmap [inpattern] [outpattern]`

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during `mput` commands and `put` commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during `mget` commands and `get` commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern* . The *inpattern* parameter is a template for incoming filenames (which may have already been processed according to the *ntrans* and `case` settings). Variable templating is accomplished by including the sequences '$1', '$2', ..., '$9' in *inpattern* . Use '\' to prevent this special

treatment of the '$' character. All other characters are treated literally, and are used to determine the nmap *inpattern* variable values. For example, given *inpattern* $1.$2 and the remote file name "mydata.data", $1 would have the value "mydata", and $2 would have the value "data". The *outpattern* determines the resulting mapped filename. The sequences '$1', '$2', ..., '$9' are replaced by any value resulting from the *inpattern* template. The sequence '$0' is replaced by the original filename. Additionally, the sequence '*[seq1, seq2]*' is replaced by *[seq1]* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command *nmap $1.$2.$3 [$1,$2].[$2,file]* would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename ".myfile". Spaces may be included in *outpattern* , as in the example: 'nmap $1 sed "s/ *$/ /" > $1' . Use the '\' character to prevent special treatment of the '$','[','[', and ',' characters.

`ntrans [inchars] [outchars]`

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during `mput` commands and `put` commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during `mget` commands and `get` commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars* . If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

`open host [port]`

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, ftp will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), ftp will also attempt to automatically log the user in to the FTP server (see below).

`prompt`

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any *mget* or *mput* will transfer all files, and any `mdelete` will delete all files.

`proxy ftp-command`

Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first `proxy` command should be an `open`, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by `proxy` : `open` will not define new macros during the auto-login process, `close` will not erase existing macro definitions, `get` and `mget` transfer files from the host on the primary control connection to the host on the secondary control connection, and `put`, `mput`, and `append` transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

`put local-file [remote-file]`

Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any *ntrans* or *nmap* settings in naming the remote file. File transfer uses the current settings for `type`, `format`, *mode*, and *structure* .

`pwd`

Print the name of the current working directory on the remote machine.

`quit`

A synonym for `bye` .

`quote arg1 arg2 ...`

The arguments specified are sent, verbatim, to the remote FTP server.

`recv remote-file [local-file]`

A synonym for get.

`reget remote-file [local-file]`

Reget behaves like get, except that if *local-file* exists and is smaller than *remote-file*, *local-file* is presumed to be a partially transferred copy of

remote-file, and the transfer is continued from the apparent point of failure. This command is useful when transferring very large files over networks that are prone to dropping connections.

`rhelp [command-name]`

Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

`rstatus [file-name]`

With no arguments, show status of remote machine. If *file-name* is specified, show status of *file-name* on remote machine.

`rename [from] [to]`

Rename the file *from* on the remote machine, to the file *to*.

`reset`

Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.

`restart marker`

Restart immediately following `get` or `put` at the indicated *marker*. On UNIX systems, marker is usually a byte offset into the file.

`rmdir directory-name`

Delete a directory on the remote machine.

`runique`

Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a `get` or `mget` command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that `runique` will not affect local files generated from a shell command (see below). The default value is off.

`send local-file [remote-file]`

A synonym for put.

`sendport`

Toggle the use of PORT commands. By default, ftp will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, ftp will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which ignore PORT commands but incorrectly indicate that they have been accepted.

`site arg1 arg2 ...`

The arguments specified are sent, verbatim, to the remote FTP server as a SITE command.

`size file-name`

Return the size of *file-name* on the remote machine.

`status`

Show the current status of ftp.

`struct [struct-name]`

Set the file transfer *structure* to *struct-name*. By default the ''*stream*'' structure is used.

`sunique`

Toggle storing of files on the remote machine under unique file names. The remote ftp server must support the ftp protocol STOU command for successful completion. The remote server will report the unique name. The default value is off.

`system`

Show the type of operating system running on the remote machine.

`tenex`

Set the file transfer type to that needed to communicate with TENEX machines.

`trace`

Toggle packet tracing.

`type [type-name]`

Set the file transfer `type` to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

`umask [newmask]`

Set the default umask on the remote server to *newmask*. If *newmask* is not specified, the current umask is printed.

`user user-name [password] [account]`

Identify yourself to the remote FTP server. If the *password* is not specified and the server requires it, ftp will prompt the user for it without disabling local echo on top of ChorusOS. If an *account* field is not specified, and the FTP server requires it, the user will be prompted for it. If an *account* field is specified, an account command will be relayed to the remote server after the login sequence is completed (if the remote server did not require it for logging in). Unless ftp is invoked with ''*auto-login*'' disabled, this process is carried out automatically on initial connection to the FTP server.

`verbose`

Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

`? [command]`

A synonym for help. Command arguments which have embedded spaces may be quoted using double quotation (" ") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to ftp commands are processed according to the following rules.

1. For `mget` commands and `get` commands with unspecified local file names, the local filename is the remote filename, which may be altered by a `case`, *ntrans*, or *nmap* setting. The resulting filename may then be altered if *runique* is on.
2. For `mput` commands and `put` commands with unspecified remote file names, the remote filename is the local filename, which may be altered by an *ntrans* or *nmap* setting. The resulting filename may then be altered by the remote server if *sunique* is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The `type` may be one of ''*ascii*'', ''*image*'' (binary), ''*ebcdic*'', and ''*local byte size*'' (for `PDP-10's` and `PDP-20's` mainly). Ftp supports the ascii and image types of file transfer, plus local byte size 8 for *tenex* mode transfers.

Ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

CONFIGURATION FILE

The *.netrc* file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

`machine name`

Identify a remote machine *name*. The auto-login process searches the `.netrc` file for a *machine* token that matches the remote machine specified on the ftp command line or as an *open* command argument. Once a match is made, the subsequent `.netrc` tokens are processed, stopping when the end of file is reached or another *machine* or a *default* token is encountered.

`default`

This is the same as *machine name* except that *default* matches any name. There can be only one *default* token, and it must be after all *machine* tokens. This is normally used as:

```
fault login anonymous password user@site
```

thereby giving the user *automatic* anonymous ftp login to machines not specified in `.netrc` . This can be overridden by using the –n flag to disable auto-login.

`login name`

Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified *name* .

`password string`

Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if `.netrc` includes the `password` *string* for any user other than `anonymous`, and if `.netrc` is readable by any user other than the current user, `ftp` aborts the auto-login process.

`account string`

Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an `ACCT` command if it does not.

`macdef name`

Define a macro. This token functions in a similar way to the ftp *macdef* command. A macro is defined with the specified name; its contents begin with the next `.netrc` line and continue until a null line (consecutive new-line characters) is encountered. If a macro named `init` is defined, it is automatically executed as the last step in the auto-login process.

HISTORY

The ftp command appeared in 4.2BSD.

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD ascii-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the ascii type. Avoid this problem by using the binary image type.

RESTRICTIONS

`Password` is always displayed on the remote terminal. To avoid this restriction, use the `.netrc` file.

Once started, file transfers cannot be aborted by signals. No signal are supported by this release of ChorusOS.

It is not possible to invoke an interactive shell.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [–s] [*name_of_host*]

DESCRIPTION

`hostname` prints the name of the current host. The superuser can set the hostname by supplying an argument; this is usually done at boot time by including the `hostname` command in the network initialization script `/etc/rc.chorus.`*IP-address*.

OPTIONS

`hostname` accepts the following option:

`–s` Trim off any domain information from the printed name.

OPERANDS

The following operands are supported:

name_of_host Set the hostname to *name_of_host*.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`gethostname`(2POSIX)

NAME

ls – list directory contents

SYNOPSIS

ls [–ACFLRTacdfiloqrstu1] file...

DESCRIPTION

For each operand that names a `file` of a type other than a directory, `ls` displays its name as well as any associated information requested. For each operand that names a `file` of the type directory, `ls` displays the names of files contained within that directory, as well as any associated information requested.

If no operands are given, the contents of the current directory are displayed. If more than one operand is given, non-directory operands are displayed first; directory and non-directory operands are sorted separately and in lexicographical order.

The following options are available:

`–A` List all entries except for `.` and `..'.'` Always set for the super-user.

`–C` Force multi-column output; this is the default when output is to a terminal.

`–F` Display a slash (/) immediately after each pathname that is a directory, an asterisk (*) after each that is executable, and an at sign (@) after each symbolic link.

`–L` If the argument is a symbolic link, list the file or directory the link references rather than the link itself.

`–R` List the subdirectories encountered recursively.

`–T` Display complete time information for the file, including month, day, hour, minute, second, and year.

`–a` Include directory entries whose names begin with a dot (.).

`–c` Use the time when the file status was last changed for sorting or printing.

`–d` Directories are listed as plain files (not searched recursively) and symbolic links in the argument list are not followed through.

`–f` Output is not sorted.

`–i` For each file, print the file's file serial number (inode number).

`–k` If the 's' option is specified, print the file size allocation in kilobytes, not blocks.

`–l` (The lowercase letter ''ell.'') . List in long format. (See below.) If the output is to a terminal, a total sum for all the file sizes is output on a line before the long listing.

`–o` Include the file flags in a long `–l` output

`–q` Force printing of non-graphic characters in file names as the character '?'; this is the default when output is to a terminal.

`–r` Reverse the order of the sort to get reverse lexicographical order, or the oldest entries first.

`–s` Display the number of file system blocks actually used by each file, in units of 512 bytes, where partial units are rounded up to the next integer value. If the output is to a terminal, a total sum for all the file sizes is output on a line before the listing.

`–t` Sort by the time modified (most recently modified first) before sorting the operands in lexicographical order.

`–u` Use the time of last access, instead of last modification of the file for sorting `–t` or printing `–l`.

`–1` (The numeric digit ''one.'') Force output to be one entry per line. This is the default when output is not to a terminal.

The `–1`, `–C`, and `–l` options override each other; the last one specified determines the format used.

The `–c` and `–u` options override each other; the last one specified determines the file time used.

By default, `ls` lists one entry per line to standard output; the exceptions are to terminals or when the `–C` option is specified.

File information is displayed with one or more <blank>s separating the information associated with the `–i`, `–s`, and `–l` options.

The Long Format

If the `–l` option is specified, the following information is displayed for each file: file mode, number of links, owner name, group name, number of bytes in the file, abbreviated month, day-of-month file was last modified, hour file was last modified, minute file was last modified, and the pathname. In addition,

for each directory whose contents are displayed, the total number of 512-byte blocks used by the files in the directory is displayed on a line by itself immediately before the information for the files in the directory.

If the owner or group names are not a known user or group name, the numeric ID's are displayed. This is always the case on ChorusOS.

If the file is a character special or block special file, the major and minor device numbers for the file are displayed in the size field. If the file is a symbolic link, the pathname of the file linked-to is preceded by –> .

The file mode printed under the -l option consists of the entry type, owner permissions, and group permissions. The entry type character describes the type of file, as follows:

`b` Block special file.

`c` Character special file.

`d` Directory.

`l` Symbolic link.

`s` Socket link.

`-` Regular file.

The next three fields are three characters each: owner permissions, group permissions, and other permissions. Each field has three character positions:

If `r`, the file is readable; if `-`, it is not readable.

If `w`, the file is writable; if `-`, it is not writable.

The first of the following that applies:

`S` If in the owner permissions, the file is not executable and set-user-ID mode is set. If in the group permissions, the file is not executable and set-group-ID mode is set.

`s` If in the owner permissions, the file is executable and set-user-ID mode is set. If in the group permissions, the file is executable and setgroup-ID mode is set.

`x` The file is executable or the directory is searchable.

`--` The file is neither readable, writeable, executable, nor set-user-ID nor set-group-ID mode, nor sticky. (See below.)

These next two apply only to the third character in the last group (other permissions):

`T` The sticky bit is set (mode `1000`), but ther is no execute or search permission.

`t` The sticky bit is set (mode `1000`), and is searchable or executable.

If successful, the `ls` utility returns 0, otherwise >0 if an error occurs.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of `ls`:

BLOCKSIZE If the environmental variable `BLOCKSIZE` is set, the block counts (see `-s`) will be displayed in units of that size of block.

COLUMNS If this variable contains a string representing a decimal integer, it is used as the column position width for displaying multiple-text-column output. The `ls` utility calculates how many pathname text columns to display based on the width provided. (See –`C`.)

TZ The timezone to use when displaying dates.

COMPATIBILITY

The group field is now automatically included in the long listing for files.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

mkdir – create directories

SYNOPSIS

mkdir [–p] [–m<mode>] *dir...*

OPTIONS

The options are:

–p	Create intermediate directories as required.
–m *mode*	Set the file permission bits of the created directory to the mode specified.

DESCRIPTION

The `mkdir` utility creates the directories named as operands, in the order specified using a default mode of rwxrwxrwx. Umask is not supported on ChorusOS and the file permission bit of the current process is therefore not recorded by a Process Manager.

The -m option sets the file permission bits of the created directory to the mode specified. The mode argument can be in any of the formats specified to the *chmod(1UNIX)* command.

In this version, symbolic links are not supported.

If the -p option is not specified, the full path prefix of each operand must already exist. Intermediate directories are created with permission bits of *rwxrwxrwx (0777)*. The current umask is not recorded.

The user must have write permission in the parent directory.

DIAGNOSTICS

If successful, 0 is returned, otherwise > 0 if an error occurred.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

(1CC)

RESTRICTIONS FOR ChorusOS

Currently, the `umask` variable is not supported; all directories are created in *rwxrwxrwx* mode by default.

NAME

mkfifo – make fifos

SYNOPSIS

mkfifo [fifo_name]

DESCRIPTION

The `mkfifo` utility creates the fifos requested, in the order specified, using mode `0777`.

The `mkfifo` utiltity requires write permission in the parent directory.

The `mkfifo` utility returns 0 if successful, and >0 if an error occurred.

STANDARDS

The `mkfifo` utility should be 1003.2 compliant.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`mkdir`(1CC), **`mkfifo`**(2POSIX), **`mknod`**(1M)

NAME

mkmerge – create a merged tree

SYNOPSIS

mkmerge [*options*]

mkmerge *configuration_file*

DESCRIPTION

The `mkmerge` utility is used to merge several sub-trees of a *spilt* source tree into a target *merged* tree.

Source directories within the split tree can be populated with special files, named *merge.rf* (for merge rule files), which provide a number of directives for `mkmerge`, such as when and where to merge these directories and their subdirectories.

The `mkmerge` utility can be viewed as a recursive combination of cp, ln, mv and rm.

The default option for the merge uses absolute symbolic links). It is also possible to use relative symbolic links, hard links or copies.

During a merge, directories are examined recursively and *merge.rf* files analyzed. If no problems are encountered, , the real merge begins: directories, links and files are created as specified in the various *merge.rf* files found.

A *merge.rf* file is composed of lines that are evaluated one at a time. A status is associated with this file. This status evolves as the file is analyzed. Depending on its value at the end of the *merge.rf* file, the current directory will be merged or pruned. A *merge.rf* file may contain commands to describe how the current directory should be merged, where it will go, and when it will be merged. The MKMERGE_IGNORE environment variable can be used to specify a list of file suffixes that should be ignored during the merge process.

OPTIONS

–a	Leave true files untouched. The default is to overwrite files in the target directory.
–c	Make copies instead of links. The default is to use symbolic links between the target directory and the source directories. This option causes `mkmerge` to copy files. Files are always exported (see the `export` command) with symbolic links.
–d *option*	Define a generation option. A generation option can take the form VAR or VAR=VALUE. These options can be tested in *merge.rf* files. The VALUE field may contain anything, but note that there are four values which hve specific meanings. These are:: "on" and "yes" mean that VAR is set

	but has no value attributed to tit. "off" and "no" mean that VAR is not set.
–D *option*	Same as the –d option
–f *configuration_file*	Read a master configuration file. Master configuration files contain option definitions, and target and source directories setting. Note that `mkmerge` is easier to use with a configuration file. Configuration files and command-line options accumulate: it is possible to use several configuration files in a command.
–h	Display a help message.
–H	Display a short description of the *merge.rf* language.
–k	Continue after conflict. A conflict occurs when several files are to be linked to a single target file. See the command priority for a description of conflict resolution. By default, when a conflict occurs, `mkmerge` reports it and continues.
–K	Stop after conflict. If this option is specified, `mkmerge` aborts in the case of conflicts.
–l	Make hard links instead of symbolic links. Files are always exported (see `export` command) with symbolic links.
–n	Do nothing. This flag is useful when testing *merge.rf* files.
–r	Make relative symbolic links instead of absolute ones. Use this option when the path to the split tree depends on the point of view (for example, ClearCase or NFS). By default, future revisions of `mkmerge` will use relative links.
–s *directory_name*	Specify a source directory.
–t *directory_name*	Specify a target directory.
–v[acdeghlmpsu]	Set verbosity flags (default: cehg):

a	Display argumentsQJ
c	Display conflicts
d	Display directories created
e	Execute echo commands
g	Display information about groups
h	Display information about hidden files
l	Display links created
m	Display *merge.rf* status
p	Display low priority files
s	Display statistics
u	Display unused options

`–q[acdehglmpsu]` Reset verbosity flags.

GENERATION OPTIONS

For `mkmerge`, a generation option is an entity that may have a value. Generation options are set using the -d option or within a configuration file, and tested in *merge.rf* files. Every generation option except *tree*can only have a single value. The values of *tree* are the base names of the source directories.

CONFIGURATION FILES

A master configuration file is composed of several sections. Each section begins with "[sectname]" where *sectname* is the name of the section.

The `mkmerge` utility only uses the content of the [profile] section. Tje [profile] section is the default section; it defines the generation options.

The [profile] section contains lines of the form *var=value* which are equivalent to the command line option "-d *var=value*". Three values of *var* have specific meanings: *include*, *merge_dir* and *tree*. include=*config_file* is equivalent to the option "-f *config_file*" merge_dir=*dir* is equivalent to the option "-t *dir*" tree=*dir* is equivalent to the option "-s *dir*"

COMMANDS

This section describes the commands which may be included within *merge.rf* files.

Each line beginning with a # is a comment. A command line has the following syntax:

command_name *parameter-1 ... parameter-n*

The status value can be one of three values: UNDEFINED (at the beginning of the file), TRUE or FALSE.

If the status is not FALSE (either UNDEFINED or TRUE) at the end of the file, the directory will be merged.

There is a concept of current source directory and current target directory. In general, files in the current source directory will be merged in the current target directory. The current source directory and the current target directory are modified when `mkmerge` enters a new sub-directory. The current target directory can also be modified using the "move" command.

Available commands are:

define, `echo`, *else*, *endif*, `error`, `exit`, `export`, *exportas*, *group*, *hide*, *if*, *ignore*, *import*, *module*, *move*, *option*, *priority*, *rename*, *run*, *subdirs* and *then*.

These commands are divided into two groups: commands that can modify the status whatever it value, and commands that are executed only if the status is UNDEFINED or TRUE.

The following commands can be used to modify the status whatever its value:

`if test1 test2 ...` This command allows the flags passed to `mkmerge` to be tested. If every result is true, the status will be set to TRUE. Otherwise, it will be set to FALSE if the status was UNDEFINED at the beginning of the file. After an “if” command, the status mechanism allows you to write logical *or* conditions using consecutive `if` statements. Tests can be defined as follows:

var	True if the flag *var* is set
var`=val`	True if the flag *var* is set and its value is `val`.
`!`*var*	True if the flag *var* is not set
`!`*var*`=val`	True if the flag *var* is not set or if its value is not `val`.

`else` If the status if FALSE, then it is changed to TRUE. If the status if TRUE, it is changed to FALSE. If the status is UNDEFINED, it remains unchanged.

`endif` Terminate an `if` block

The following commands are executed only if the status is UNDEFINED or TRUE.

`define` *flag* `...`	This command adds the line *flag* to the *defines.lst* file located at the root of the merged tree. These flags will be used to set the DEFLIST variable of all make files, thus defining the default preprocessor flags list used for the compilation of any source file.
`option` *flag*`=val ...`	This command adds the line *flag*`=val` to the *options.lst* file located at the root of the merged tree. These options are passed to all make files.
`echo` *message* `...`	This command displays a message. For best results, messages should be enclosed between quotes for best results.
`error` *error_message*	This command displays *error_message* and aborts the merge process.
`exit`	This command exits the current *merge.rf* and sets the status to FALSE; the current directory will not be merged and any sub-directories will be pruned.
`export` *path* `file ...`	This command will create a symbolic link from the file *path/file* (*path* is relative to the root of the merged tree) to the file named `file` in the current directory. The`file` does not need to be created prior to using this command, it may be a file generated during system production.
`exportas` *linkname* `file`	This command will create a symbolic link from the file */linkname* (*linkname* is relative to the root of the merged tree) to the file named `file` in the current directory. The `file` does not need to be created prior to using this command, it may be a file generated during system production.
`group` *groupname* `file ...`	This command adds `file` to the *groupname* group. A group is a set of files. Groups can be imported and exported. Groups are often used to import files whose names are not known into the current directory.
`hide`	This command will cause files in the current directory and its sub-directories not to be merged.

	Unlike `exit`,*hide* does not stop the analysis of the current *merge.rf*. Hidden files can be imported.
`ignore file ...`	This command tells `mkmerge` not to merge `file` into the current directory. If `file` is a directory, it will not be entered. This command is often used in conjunction with the rename command.
`import file`	This command will merge `file` into the current directory. The `file` must be either a relative path name or a group name.
`module` *modname rule*	This command will write information to the file *modules.lst*. If *rule* is empty, the current target directory is added to the list of directories which are considered to belong to the *modname* feature.

If *rule* is not empty, the status of *modname* will be set to `on` when *rules* is true. The *rule* command has the same syntax as the `if` command.

`move` *path*	This command modifies the location of the current target directory. The *path* variable points to the current target directory.
`priority [+`*n*`,-`*n*`,`*n*`]`	This command changes the priority of the current directory. Each file has a priority to be used when conflicts arise due to an attempt to merge several files at the same location on the merged tree. In such cases, the file with the higher priority will be merged. If two or more files have equal priority, `mkmerge` reports the conflict and doesn't merge the conflicting files. The priority command is used to increase the priority of the current directory and its sub-directories. If the n parameter is a signed value (+n or –n), the value is added to or subtracted from the current priority. Otherwise, the current priority is set to the value of the parameter. The priority of a file is calculated by taking the number of *merge.rf* files found from the root to the current directory, adding the priority of the current directory, and multiplying by 256.
`rename file` *newname*	This command tells `mkmerge` to merge a file named `file` to a file named *newname*.

`run command` — This command saves the current target directory. Once the merge has been performed, `mkmerge` will spawn a shell that will execute `command` from the saved directory.

`subdirs dirname ...` — This command adds `dirname` to the list of sub-directories that should be entered. By default, `mkmerge` enters all sub-directories; however, if the *subdirs* command has been used, only the sub-directories specified will be searched.

RESTRICTIONS

The *ignore* command does not work with groups.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

configurator(1CC)

NAME

mv – move files

SYNOPSIS

mv [–fi] source target

mv [–fi] source... directory

DESCRIPTION

In its first form, the mv utility renames the file named by the source operand to the destination path named by the *target* operand. This form is the default if the last operand does not name an existing directory.

In its second form, mv moves each file named by a source operand to a destination file in the existing directory named by the *directory* operand. The destination path for each operand is the pathname produced by the concatenation of the last operand, a slash, and the final pathname component of the named file.

The following options are available:

–f Do not prompt for confirmation before overwriting the destination path.

–i Causes mv to write a prompt to standard error before moving a file that would overwrite an existing file. If the response from the standard input begins with the character ‘‘y’’, the move is attempted.

These options overwrite each other, therefore the last *-f* or *-i* option is the one which affects the behavior of mv.

The source operand and the destination path must both specify either a file or a directory.

If the destination path cannot be written to, mv prompts the user for confirmation as specified for the *-i* OPTION.

If successful, the mv utility returns 0, otherwise >0 if an error occurs.

RESTRICTIONS

The first form of the mv command performs a copy if the source and the *target* files do not belong to the same file system. However, the owner and permissions of the source file are not propagated to the *target* file.

The second form of the mv command fails if the source operands and target operands do not belong to the same file system.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **rename**(2POSIX)

NAME

netboot – load and execute standalone programs over the network

DESCRIPTION

The netboot utility uses Internet protocols over an Ethernet line to load and boot a ChorusOS system bootable image.

As soon as netboot is loaded and starts running, it displays a status message similar to the following:

```
Netboot r2.x.y
```

This message shows the version number and system architecture for netboot. It is is followed by line(s) describing the ethernet boards which have been successfully detected, possibly asking you to enter the board you want to use.

Then the Ethernet address of the local system, which is determined by querying the locally installed network device(s) is displayed.

The local Internet address is then determined either by broadcasting an RARP request to the Ethernet and checking the response, or by reading the LOCAL_INADDR environment variable.

If RARP was used, netboot then attempts to download a configuration file from the systems which responded to the RARP request. Netboot uses TFTP for all file transfers and therefore assumes that each RARP server is running a TFTP daemon.

If netboot did not use RARP, the configuration file is downloaded from the TFTP server indicated by the CONFIGSERVER environment variable.

Netboot uses a configuration file to identify the name of the file to boot and the server on which the file resides. If specified in the file, netboot will display the address of the boot server and name of the boot file for the user to approve or correct, if required. If no configuration file is found, the user is prompted to enter the Internet address of the server and the file name to load.

Netboot determines whether the specified boot server is active and responding to network requests by sending a `ping` request. If the boot server does not respond, the user is prompted to enter the Internet address of a different server.

When an active boot server is found, netboot starts downloading the specified file. After the program is downloaded into local memory, netboot initializes the system as appropriate and starts executing the new program. There is no return to netboot once the downloaded program starts executing.

Netboot uses the decimal form of Internet addresses to name remote servers (for example, . `192.33.22.143`). A file name is assumed to be relative to where the remote TFTP server can reference it. The actual location of the boot file on the server depends upon how the TFTP daemon is configured. Under UNIX, a TFTP daemon can be configured to run in "secure" mode and so file

names refer to subdirectories of the TFTP "base" directory (for example, */tftpboot*). For more information, see *TFTPD*(8CSUNOS) and CONFIGURATION, below.

CONFIGURATION

Netboot works without any configuration, the only restriction being that the name of the boot file and address of the boot server must be specified by user input while netboot is running. In many cases, it is more convenient to configure netboot to load and execute a kernel automatically from a known location. This can be done by creating a configuration file on a machine which netboot recognizes as a boot server.

Netboot treats any system configured to run both RARP (not required, see below) and TFTP daemons as a boot server.. If RARP is used, netboot treats any of the systems that responded to the original RARP request as possible boot servers, containing an appropriate configuration file for the local system. The name netboot uses for its configuration file is the hex representation of the local Internet address. For example, a local machine with an IP address of *192.33.22.143* will try to access a configuration file named *C021168F*. Netboot uses only the hex IP address to access a configuration file, the path name is not qualified at all (a directory path is not specified). This means that the actual location on the server depends on how the TFTP daemon was configured to run.

If the LOCAL_INADDR environment variable is set in Netboot's environment, and has a valid decimal IP format, Netboot will take the IP address for the machine from there. This allows Netboot to be used on networks which are not running an RARP server. Netboot then reads the CONFIGSERVER environment variable to obtain the IP address of the machine containing the configuration file. If CONFIGSERVER is not set, Netboot will prompt for the name of the TFTP boot host on the console. If the CONFIGFILE environment variable is also set, this is used as the path for the configuration file. If not, the configuration file name will be the hexadecimal form of the IP address of the local machine (the same as when RARP is used).

On UNIX systems, the TFTP daemon, `tftpd`, is commonly configured to run with a base directory specified; the server attempts to change to this directory before responding to any requests. A common base directory for `tftpd` is */tftpboot*; for example, a machine with the IP address mentioned above would try to access a configuration file on the remote server named */tftpboot/C021168F* (or */tftpboot/$CONFIGFILE* if RARP is not used and the CONFIGFILE variable has been set).

CONFIGURATION PARAMETERS

Options for the netboot program can be set or changed with keywords in the configuration file. Each line in the configuration file has the following format:

```
NETBOOT_PARAMETER=parametervalue
```

Each configuration parameter must begin on a new line. White space is ignored up to the first non-white space of the parameter value, after which all characters are taken as part of the value. A line containing a '#" in the first column is considered a comment.

The following keywords are recognized by netboot:

`BOOTSERVER=string` This is the Internet address of the remote system containing the boot file required. The Internet address should be of the form `ZZZ.ZZ.ZZ.ZZ` where `ZZ` is a decimal number.

`BOOTFILE=file name` This specifies the file name of the boot file required. If the remote TFTP daemon is running with a specified "base" directory, this name is referenced on the server relative to this directory. For example, `BOOTFILE=chorus` would correspond to */tftpboot/chorus* on a server running UNIX with the `tftpd` base directory set to */tftpboot*. If the TFTP daemon on the remote server is not running with a specified base directory, the file name must be fully specified and all directories leading to the file must be accessible to any user. For example, `BOOTFILE=/home/kernels/chorus` would correspond to */home/kernels/chorus* on the server running UNIX. Both */home*, and */home/kernels* would need public read and execute permissions, and the file */home/kernels/chorus* would need public read permission.

`INPUT_TIMEOUT=number` If netboot is waiting for input from the user and `AUTOBOOT` is not set, netboot will wait for `number` seconds before using a default value for the information needed. `INPUT_TIMEOUT` only applies if a default value was specified in a configuration file. The default timeout value is 10 seconds.

`AUTOBOOT=YES` Indicates whether netboot is to start loading the kernel immediately or should prompt the user for verification (or correction) of the default boot parameters specified in the configuration file. If no configuration file is found, `AUTOBOOT` cannot be set. If any of the default parameters is invalid, `AUTOBOOT` is overridden and user input is mandatory.

`TFTP_TIMEOUT=number` — `TFTP_TIMEOUT` specifies the amount of time netboot waits for a network response from a TFTP download request. As this parameter can only be changed via a configuration file, `TFTP_TIMEOUT` for configuration file requests is the default value, 10 seconds. Netboot attempts to initiate a download of a file 5 times before giving up. This means the actual time out for a download request is 5 * `TFTP_TIMEOUT`.

`ENV(envVar=string)` — For CHORUS kernels that support configuration environments, environment variable can be initialized and passed to the kernel environment at boot time by using the netboot macro `ENV`. The string "`envVar=string`" is passed to kernel, and is used to customize the Chorus configuration environment. CHORUS kernels which do not support configuration environments are not affected by environment strings or the `ENV` macro.

CUSTOMIZING THE BOOT PROCESS

To summarize, proceed as follows to configure netboot to boot a machine automatically using a known boot server and file name.

1. Decide which machines will be boot and RARP servers for the local network. At least one (if not all) of the RARP servers should be configured to provide TFTP services as well (see *inet.conf*(4SUNOS) or *inet.conf*(4SVR4)). If you do not have an RARP server, only the TFTP server needs to be specified..
2. Define the Internet address of the machine to be booted. If you are going to use RARP, both the Internet and Ethernet address of the machine to be booted must be in the local database of the local RARP server(s). Under UNIX, these are the *hosts* and *ethers* databases.
3. Create a netboot configuration file in the `tftpd` base directory on any of the local RARP servers running `tftpd`. If you do not use RARP, any TFTP server will be acceptable. For RARP users, the name of the file is the hexadecimal Internet address of the machine to be booted. For example, a machine with the IP address *192.33.22.143* will have a netboot configuration file named *C021168F*. Under UNIX, it is common for a TFTP daemon to be configured to run in "secure mode" with a specified base directory. In the example provided, the filename would be: */tftpboot/C021168F*.
4. If you do not want to use RARP, set the LOCAL_INADDR, CONFIGSERVER and CONFIGFILE in Netboot's environment, as previously explained.

5. Edit the file to contain references to the boot server and path name of the kernel to boot (initialize both the `BOOTFILE` and `BOOTSERVER`variables). For Netboot to boot automatically without requiring user input, include the line `"AUTOBOOT=YES"` in the configuration file.
6. Netboot can load a kernel from any machine providing TFTP service. Under UNIX, netboot can access any directory on a server provided:

   a) the TFTP daemon is not running with "secure" mode enabled, and

   b) no base directory has been specified.

   All directories leading to the file, as weel as the file itself, must have appropriate file permissions for public access.
7. On the machine to be booted, insert a netboot bootable diskette in the local floppy diskette drive.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

inetd.conf(4 SUNOS), inetd.conf(4 SVR4), UDP(7 SVR4), ARP(7 SVR4), fd(7 SVR4), tftpd(8C SUNOS)

RFC 783, RFC 903, RFC 906

DIAGNOSTICS

As Netboot loads and starts the execution of the kernel, it displays various messages. The following is a list of the Netboot messages and their meanings:

`Local Ethernet Address: XX:XX:XX:XX:XX:XX`

netboot displays the local Ethernet address configured in the local network hardware.

`Using a statically configured IP address: XX.XX.XX.XX`

netboot displays this message if the LOCAL_INADDR environment variable has been set and is valid.

`Local Internet Address: XX.XX.XX.XX`

netboot displays the local Internet address asdefined by the response(s) to an RARP request or by the LOCAL_INADDR environment variable.

`Unable to determine local Internet Address!`

Netboot has been configured to use RARP but there were no responses to the RARP request. This can mean several things:

- There are no RARP servers on the local network in which case Netboot does not look for TFTP servers.
- There is something wrong with the local network connection.
- There is a hardware compatibility problem with netboot (see below).

In each case, Netboot prints a fatal warning message and stops. Before rebooting and attempting to run Netboot again, check the installed hardware for compatibility, check that there is an RARP server on the local network, and check the network connections and cables. You can avoid using an RARP server by specifying the IP address to use in the LOCAL_INADDR environment variable in Netboot.

```
Variable CONFIGSERVER incorrectly set (...) in netboot's environment.
```

Either the variable is not set, or has an invalid syntax. Netboot will then display the following message and stop:
`Update netboot with mkenv(1) and reboot the system.`

```
Netboot parameter file <file> on server XX.XX.XX.XX: found!
```

A Netboot configuration file has been found on one of the RARP servers or on the server specified by the CONFIGSERVER environment variable. Netboot stops searching for configuration files.

```
Netboot parameter file <file> on server XX.XX.XX.XX: not found.
```

A Netboot configuration file was not found on the specified TFTP server. If RARP is used, netboot continues by trying to download the file from the next RARP server. If there are no other servers, Netboot stops looking for a configuration file and prompts the user for values.

```
Netboot parameter file not found on RARP servers.
```

Confirmation that no parameter files were found. The user must enter the server address and file name explicitly.

The following two messages are similar but will occur under different circumstances. The first message is displayed when a default Internet address for a boot server was specified. The second is printed if there is no default

address, or if the default value was found to be invalid (if the boot server didn't respond, for example).

`Enter Internet address of boot server->XX.XX.XX.XX`

There is a default address for a boot server but this can be overridden by backspacing and typing in a new value. Ctrl-W and Ctrl-U can be used to erase the previous word or the entire line, respectively. Pressing the Return key accepts the default value. The `INPUT_TIMEOUT` parameter applies here; if there is no input by `INPUT_TIMEOUT` seconds, the default is used. Any input disables the timeout feature.

`Enter Internet address of boot server->`

There is no default address for a boot server; a value must be typed in. The `INPUT_TIMEOUT` parameter does not apply.

`ZZZZZ: Incorrectly formatted Internet address.`

The address of the remote boot server was not typed correctly typed. Internet addresses must be specified in standard format, as 4 decimal numbers separated by a dot ('.'). The user is prompted to re-enter the address of the boot server.

`**** XX.XX.XX.XX is alive.`

The boot server specified by the user responded to an ICMP "ping" request. The boot file can be downloaded.

`**** No answer from XX.XX.XX.XX.`

The boot server specified by the user did not respond to an ICMP "ping" request. The user will be prompted to enter another address for the boot server (the original address is kept as the default).

The following two messages are similar but will occur under different circumstances. The first message is displayed when a default file name for a kernel was specified. The second is printed if there is no default file name, or if a default value was found to be invalid (for example, the file does not exist).

`Enter name of kernel to load-><file>`

There is a default name for a boot file but this can be overridden by backspacing and typing in a new value. Ctrl-W and Ctrl-U can be used to erase the previous word or the entire line respectively. Pressing the Return key accepts the default value. As with boot server addresses,

INPUT_TIMEOUT applies here; if there is no input by INPUT_TIMEOUT seconds, the default is used. Any input disables the timeout feature.

```
Enter name of kernel to load->
```

There is no default name for a boot file; a value must be typed in. The INPUT_TIMEOUT parameter does not apply.

```
Loading remote file XX.XX.XX.XX:<file>->
```

Beginning an attempt to download the remote file specified by the user.

```
File successfully downloaded!
```

File was downloaded and netboot continues.

```
Network timeout, retrying...
```

The network hasn't responded to a download request in TFTP_TIMEOUT seconds. The download request will be retried a maximum of 5 times before the user is prompted again for the address of a remote host. Check that the TFTP daemon is running on the remote host.

```
<file>: File not found on remote host
```

The file was not found, the user is prompted for another remote file name.

```
<file>: File not accessible via tftp
```

The file was found but was not accessible to the remote TFTP daemon. The user is prompted for another remote file name.

```
Protocol error while loading file.
```

There was an internal protocol error while downloading the file. The user is prompted for server and file.

```
Network timeout while loading boot file.
```

The network didn't respond after 5 timeouts at TFTP_TIMEOUT seconds. The network connections and remote server should be checked. The user is prompted for remote server and file.

```
Internal resource shortage while loading boot file.
```

An internal error occurred in netboot (usually caused by a overrun of data from the network). The user is prompted for remote server and file.

```
<file> not an x86 ELF or COFF binary.
```

The downloaded file was not in a format executable by Netboot. The user is prompted for remote server and file.

```
Booting downloaded file.
```

The downloaded file is of the correct format and booting continues.

COMPATIBILITY

Netboot works on PC/AT systems installed with the following list of Ethernet controllers:

ISA Adapters:

- SMC Ethercard Plus Elite 16 bits
- SMC Ethercard Ultra Elite 16 bits
- SMC EtherEZ
- Novell NE2000 plus

PCI Adapters:

- SMC EtherPowerII

To work with Netboot, the ISA SMC Ethernet adapters must be configured with the following settings:

Option:	**Value:**
Mode	Shared memory
Wait states	Disabled
ROM	Disabled
Shared Memory Base	User defined
IRQ	User defined
IO base	User defined

The default SMC Ethernet adapter configuration recognized by netboot is:

Shared Memory Base: `0xD0000`

IRQ:	`9`
IO Base:	`0x240`

Three environment variables may be defined using `mkenv(1)` to configure netboot to work with your SMC Ethernet adapter:

ND_WD_0_SHMBASE

ND_WD_0_ITLEVEL

ND_WD_0_BASEIO

The LOCAL_INADDR, CONFIGSERVER and CONFIGFILE environment variables can also be set using `mkenv(1)` to allow booting without an RARP server on the network. These variables are exported to the environment of the booted system.

To work with Netboot, the Novell Ethernet adapters must be configured with the following settings:

Option:	**Value:**
Mode	Programmed I/O
ROM	Disabled
IRQ	User defined
IO base	User defined

The default Novell Ethernet adapter configuration recognized by netboot is:

IRQ:	`3`
IO Base:	`0x300`

Two environment variables may be defined using `mkenv(1)` to configure netboot to work with your Novell Ethernet adapter:

- ND_NE_0_ITLEVEL
- ND_NE_0_BASEIO

No configuration is required for PCI Ethernet adapters.

Netboot can only execute files in x86 ELF or COFF formats.

Netboot supports systems with Pentium processors.

NAME

netstat – show network status

SYNOPSIS

netstat [–Aan] [–faddress_family] [–Mcore] [–Nsystem]

netstat [–dghimnrs] [–faddress_family] [–Mcore] [–Nsystem]

netstat [–dn] [–Iinterface] [–Mcore] [–Nsystem] [–wwait]

netstat [–pprotocol] [–Mcore] [–Nsystem]

DESCRIPTION

The `netstat` command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with a `wait` interval specified, `netstat` will continuously display the information regarding packet traffic on the configured network interfaces. The fourth form displays statistics about the named protocol.

The options have the following meanings:

`–A` Using the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

`–a` Using the default display, show the status of all sockets; sockets used by server processes are not usually shown.

`–d` Using either interface display (option *-i* or an interval, as described below), show the number of dropped packets. On top of ChorusOS this option is also used to show the list of symbols which have been found or not found by the IOM (an ioctl is performed on /dev/kmem).

`–f` *address_family* Limit statistics or address control block reports to those of the specified *address family*. The following address families are recognized: *inet* for `AF_INET`, *ns* for `AF_NS`, *iso* for `AF_ISO` and *unix* for `AF_UNIX`.

`–g` Show information related to multicast (group address) routing. By default, show the IP Multicast virtual-interface and routing tables. If the `-s` option is also present, show multicast routing statistics.

`–h` Show the state of the `IMP` host table (obsolete).

–I *interface*	Show information about the specified interface; this is used with a `wait` interval as described below.
–i	Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown). If the *-a* option is also present, multicast addresses currently in use are shown for each Ethernet interface and for each IP interface address. Multicast addresses are shown on separate lines following the interface address with which they are associated.
–M	Extract values associated with the name list from the specified core instead of the default `/dev/kmem`.
–m	Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).
–N	Extract the name list from the specified system instead of the default /vmunix. On top of ChorusOS symbols are always extracted from the IOM.
–n	Show network addresses as numbers (normally, `netstat` interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.
–p *protocol*	Show statistics about `protocol`, which is either a well-known name for a protocol or an alias for it. Some protocol names and aliases are listed in the file */etc/protocols*. A null response typically means that there are no significant numbers to report. The program will return an error message if *protocol* is unknown, or if there is no statistics routine for it.
–s	Show per-protocol statistics. If this option is repeated, counters with a value of zero are suppressed.
–r	Show the routing tables. When `-s` is also present, show routing statistics instead.
–w *wait*	Show network interface statistics at intervals of `wait` seconds.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form ‘‘host.port’’ or ‘‘network.port’’ if a socket’s address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the *-n* option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet ‘‘dot format,’’ refer to *inet(3STDC)*). Unspecified or “wildcard” addresses and ports appear as ‘‘*’’.

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit (‘‘mtu’’) are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows a collection of information about the route stored as binary choices. The individual flags are discussed in more detail in the *route(1M)* and *route(7P)* manual pages.

The mapping between letters and flags is:

```
1     RTF_PROTO2     Protocol specific routing flag #1
2     RTF_PROTO1     Protocol specific routing flag #2
B     RTF_BLACKHOLE  Simply discard pkts (during updates)
C     RTF_CLONING    Generate new routes on use
D     RTF_DYNAMIC    Created dynamically (by redirect)
G     RTF_GATEWAY    Destination requires forwarding by intermediary
H     RTF_HOST       Host entry (net otherwise)
L     RTF_LLINFO     Valid protocol to link address translation.
M     RTF_MODIFIED   Modified dynamically (by redirect)
R     RTF_REJECT     Host or net unreachable
S     RTF_STATIC     Manually added
U     RTF_UP         Route usable
X     RTF_XRESOLVE   External daemon translates proto to link address
```

Direct routes are created for each interface attached to the local host; the gateway field for these entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection, while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface used for the route.

When `netstat` is invoked with the *-w* option and a `wait` interval argument, it displays a running count of statistics related to network interfaces. An obsolete version of this option used a numeric parameter with no option, and

is currently supported for backward compatibility. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface using the –I option. The first line of each screen of information contains a summary of the status since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

nfsstat(1CC), **hosts**(4CC), **networks**(4CC), **protocols**(4CC), **services**(4CC)

BUGS

The notion of errors is ill-defined.

RESTRICTIONS

On top of ChorusOS symbols are always extracted from the running system, thus all the options used to access symbols from a file are not supported. Not all the symbols required by `netstat` are fully resolved, because ChorusOS doesn't support all protocol families and tables of statistics. The *-d* option shows the list of symbols which have been found by ChorusOS.

NAME

nfsstat – display NFS statistics

SYNOPSIS

nfsstat [–Mcore] [–Nsystem] [–wwait]

DESCRIPTION

`nfsstat` displays statistics kept about `NFS` client and server activity.

The options are as follows:

`–>M` Extract values associated with the name list from the specified core instead of the default */dev/kmem* (see RESTRICTIONS).

`–N` Extract the name list from the specified system instead of the default */vmunix* (see RESTRICTIONS).

`–>w` Display a shorter summary of `NFS` activity for both the client and server at `wait` second intervals.

FILES

/dev/kmem

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`netstat`(1CC)

RESTRICTIONS

On top of ChorusOS, symbols are always read from the running system, thus the *-M* and *-N* options are not supported.

NAME

pax – read and write file archives and copy directory hierarchies

SYNOPSIS

pax [–cdnv] [–farchive] [–sreplstr] [–T[from_date][,to_date]] [pattern...]

pax –r [–cdknuvDYZ] [–farchive] [–ooptions] [–pstring] [–sreplstr] [–T[from_date][,to_date]]

pax –w [–dtuvHLPX] [–bblocksize] [[–a][–farchive]] [–xformat] [–Bbytes] [–sreplstr] [–ooptions] [–T[from_date][,to_date][/[c][m]]]

pax –r –w [–dklntuvDHLPXYZ] [–pstring] [–sreplstr] [–T[from_date],to_date] [/] [file...] *directory*

DESCRIPTION

pax will read, write, and list the members of an archive file, and will copy directory hierarchies. Operation is independent of the specific archive format, and therefore supports a wide variety of different archive formats. A list of the archive formats supported can be found under the description of the *x* option.

The presence of the *r* and the *w* options specify which of the following functional modes pax will operate under: *list* , *read* , *write* , and *copy.*

<none> *List.*

pax will write to *standard output* a table of contents of the members of the archive file read from *standard input*, whose pathnames match the specified *patterns.* The table of contents contains one filename per line and is written using single line buffering.

–r *Read.*

pax extracts the members of the archive file read from the *standard input*, with pathnames matching the specified *patterns.* The archive format and blocking is automatically determined on input. When an extracted file is a directory, the entire file hierarchy rooted at that directory is extracted. All extracted files are created relative to the current file hierarchy. The setting of ownership, access and modification times, and file mode of the extracted files are discussed in more detail under the p option.

–w *Write.*

pax writes an archive containing the file operands to *standard output* using the specified archive format. When no file operands are specified, a list of files to copy (one per line) is read from *standard input.* When a file operand is also a directory, the entire file hierarchy rooted at that directory will be included.

`-r -w` *Copy.*

`pax` copies the `file` operands to the destination *directory*. When no `file` operands are specified, a list of files to copy with one per line is read from the *standard input*. When a `file` operand is also a directory the entire file hierarchy rooted at that directory will be included. The effect of *copy* is as if the copied files were written to an archive file and then subsequently extracted, except that there may be hard links between the original and the copied files (see the `l` option below).

Warning: The destination *directory* must not be one of the `file` operands or a member of a file hierarchy rooted at one of the `file` operands. A *copy* under these conditions can lead to unpredictable results.

When processing a damaged archive during a *read* or *list* operation, `pax` will attempt to recover from media defects, and will search through the archive to locate and process the largest number of archive members possible (see the `E` option for more details on error handling).

OPERANDS

The *directory* operand specifies a destination directory pathname. If the *directory* operand does not exist, or it is not writable by the user, or it is not a directory, `pax` will exit with a non-zero exit status.

The *pattern* operand is used to select one or more pathnames of archive members. When the *pattern* operand is not supplied, all members of the archive will be selected. When a *pattern* matches a directory, the entire file hierarchy rooted at that directory will be selected. When a *pattern* operand does not select at least one archive member, `pax` will write these *pattern* operands in a diagnostic message to *standard error* and then exit with a non-zero exit status.

The `file` operand specifies the pathname of a file to be copied or archived. When a file operand does not select at least one archive member, `pax` will write these `file` operand pathnames in a diagnostic message to *standard error* and then exit with a non-zero exit status.

OPTIONS

The following options are supported:

`-r` Read an archive file from *standard input* and extract the specified *files*. If any intermediate directories are needed in order to extract an archive member, these directories will be created as if `mkdir`(2PSX) had beencalled with the bitwise inclusive OR of `S_IRWXU , S_IRWXG , ` and `S_IRWXO` as the mode argument. When the selected archive format supports the specification of linked files and these files cannot be linked while the archive is being extracted, `pax` will write a diagnostic message to *standard error* and exit with a non-zero exit status at the completion of operation.

–w Write files to the *standard output* in the specified archive format. When no `file` operands are specified, *standard input* is read for a list of pathnames with one per line without any leading or trailing blanks.

–a Append *files* to the end of an archive that was previously written. If an archive format is not specified using an –x option, the format currently being used in the archive will be selected. Any attempt to append to an archive in a format different to the format already used in the archive will cause `pax` to exit immediately with a non-zero exit status. The blocking size used in the archive volume where writing starts will continue to be used for the remainder of that archive volume.

Warning: Many storage devices are not able to support the operations necessary to perform an append operation. Any attempt to append to an archive stored on this type of device may damage the archive or have other unpredictable results. Tape drives in particular are not likely to support an append operation. An archive stored in a regular file system file or on a disk device will usually support an append operation.

–b *blocksize* When *writing* an archive, block the output at a positive decimal integer number of bytes per write to the archive file. The *blocksize* must be a multiple of 512 bytes with a maximum of 32256 bytes. A *blocksize* can end with k or b to specify multiplication by 1024 (1K) or 512, respectively. A pair of *blocksizes* can be separated by x to indicate a product. A specific archive device may impose additional restrictions on the size of blocking it will support. When blocking is not specified, the default *blocksize* is dependent on the specific archive format being used (see the –x option).

–c Match all file or archive members *except* those specified by the *pattern* and *file* operands.

–d Make directory files being copied, archived or extracted, match only the directory file or archive member and not the file hierarchy rooted at the directory.

–f *archive* Specify *archive* as the pathname of the input or output archive, overriding the default *standard input* (for *list* and *read*)or *standard output* (for *write*). A single archive may span multiple files and different archive devices. When required, `pax` will prompt for the pathname of the file or device of the next volume in the archive.

–k Do not overwrite existing files.

`-l` Link files. In *copy* mode (`rw`), hard links are made between the source and destination file hierarchies whenever possible.

`-n` Select the first archive member that matches each *pattern* operand. No more than one archive member is matched for each *pattern*. When members of the type directory are matched, the file hierarchy rooted at that directory is also matched (unless `d` is also specified).

`-o` *options* Information to modify the algorithm for extracting or writing archive files which is specific to the archive format specified by `x`. In general, *options* takes the form: `name=value`

`-p` *string* Specify one or more file characteristic options (privileges). The *string* option-argument is a string specifying file characteristics to be retained or discarded on extraction. The string consists of the specification characters `a, e, m,` and `p.` Multiple characteristics can be concatenated within the same string, and multiple `p` options can be specified. The meaning of the specification characters are as follows:

- `a` Do not store file access times. By default, file access times are stored whenever possible.
- `e` Store everything , the user ID, group ID, file mode bits, file access time, and file modification time. This is intended to be used by *root* (someone with all the appropriate privileges) in order to preserve all aspects of the files as they are recorded in the archive. The `e` flag is the sum of the `o` and `p` flags.
- `m` Do not store file modification times. By default, file modification times are stored whenever possible.
- `o` Store the user ID and group ID.
- `p` Store the file mode bits. This is intended to be used by a *user* with normal privileges who wants to retain all aspects of the file other than the ownership. The file times are preserved by default, but two other flags are available to disable this and use the time of extraction instead.

In the preceding list, preserve means that an attribute stored in the archive is given to the extracted file, subject to the permissions of the invoking process. Otherwise, the attribute of the extracted file is determined as part of the

normal file creation process. If neither the e nor the o specification character is used, or the user ID and group ID are not preserved for any reason, pax will not set the S_ISUID, (setuid) and S_ISGID (setgid) bits of the file mode. If the preservation of any of these items fails for any reason, pax will write a diagnostic message to *standard error*. Failure to preserve these items will affect the final exit status, but will not cause the extracted file to be deleted. If the file characteristic letters in any of the string option-arguments are duplicated or conflict with each other, the one(s) given last will take precedence. For example, if p eme is specified, file modification times are still preserved.

–s *replstr* Modify the file or archive member names specified by the *pattern* or file operands according to the substitution expression *replstr*, using the syntax of the ed(1UNIX) utility's regular expressions. The format of these regular expressions are: /old/new/[gp]. As in ed(1UNIX), old is a basic regular expression and new can contain an ampersand (&), \n (where n is a digit) back-references, or subexpression matching. The old string may also contain <*newline*> characters. Any non-null character can be used as a delimiter (/ is shown here). Multiple s expressions can be specified. The expressions are applied in the order they are specified on the command line, terminating with the first successful substitution. The optional trailing g continues to apply the substitution expression to the pathname substring which starts with the first character following the end of the last successful substitution. The first unsuccessful substitution stops the operation of the g option. The optional trailing p will cause the final result of a successful substitution to be written to *standard error* in the following format: <original pathname> >> <new pathname> File or archive member names that substitute to the empty string are not selected and will be skipped.

–t Reset the access times of any file or directory read or accessed by pax to be the same as they were before being read or accessed by pax.

–u Ignore files that are older (having a less recent file modification time) than a pre-existing file or archive member with the same name. During *read*, an archive member with the same name as a file in the file system will be extracted if the archive member is newer than the file. During *write,* a file system member with the same name as an archive member will be written to the archive if it is newer than the archive member. During *copy*, the file in the destination hierarchy is replaced by the file in the source hierarchy or by

a link to the file in the source hierarchy, if the file in the source hierarchy is newer.

–v

During a *list* operation, produce a verbose table of contents using the format of the `ls(1)` utility with the `-l` option. For pathnames representing a hard link to a previous member of the archive, the output has the format:
`<ls -l listing> == <link name>` For pathnames representing a symbolic link, the output has the format:
`<ls -l listing> => <link name>` Where
`<ls -l listing>` is the output format specified by the `ls(1)` utility when used with the `-l` option. Otherwise for all the other operational modes (*read* , *write* , and *copy*), pathnames are written and flushed to *standard error* without a trailing *<newline>* as soon as processing begins on that file or archive member. The trailing *<newline>* , is not buffered, and is written only after the file has been read or written.

–x `format`

Specify the output archive format, with the default format being *ustar*. The `pax` utility currently supports the following formats:

`cpio` — The extended cpio interchange format specified in the `-p1003.2` standard. The default blocksize for this format is 5120 bytes. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format are detected by `pax` and repaired.

`bcpio` — The old binary cpio format. The default blocksize for this format is 5120 bytes. This format is not very portable and should not be used if other formats are available. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format are detected by `pax` and repaired.

`sv4cpio` — The System V release 4 cpio. The default blocksize for this format is 5120 bytes. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format are detected by `pax` and repaired.

`sv4crc` The System V release 4 cpio with file crc checksums. The default blocksize for this format is 5120 bytes. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format are detected by `pax` and repaired.

`tar` The old BSD tar format as found in BSD4.3. The default blocksize for this format is 10240 bytes. Pathnames stored by this format must be 100 characters or less in length. Only *regular* files, *hard links* , *soft links* , and *directories* will be archived (other file system types are not supported). For backwards compatibility with even older tar formats, an `o` option can be used when writing an archive to omit the storage of directories. This option takes the form: `write_opt=nodir`.

`ustar` The extended tar interchange format specified in the `-p1003.2` standard. The default blocksize for this format is 10240 bytes. Pathnames stored by this format must be 250 characters or less in length.

The `pax` utility will detect and report any file that it is unable to store or extract as the result of any specific archive format restrictions. The individual archive formats may impose additional restrictions on use. Typical archive format restrictions include (but are not limited to): file pathname length, file size, link pathname length and the type of the file.

`-B` *bytes* Limit the number of bytes written to a single archive volume to `bytes`. The `bytes` limit can end with `m`, `k`, or `b` to specify multiplication by 1048576 (1M), 1024 (1K) or 512, respectively. A pair of `bytes` limits can be separated by `x` to indicate a product.

Warning: Only use this option when writing an archive to a device which supports an end of file read condition based on last (or largest) write offset (such as a regular file or a tape drive). The use of this option with a floppy or hard disk is not recommended.

`–D` This option is the same as the `u` option, except that the file inode change time is checked instead of the file modification time. The file inode change time can be used to select files whose inode information

(for exaqmple, uid and gid) is newer than a copy of the file in the destination *directory*.

–Elimit Limit the number of consecutive read faults while trying to read a flawed archive to *limit*. With a positive *limit*, pax will attempt to recover from an archive read error and will continue processing starting with the next file stored in the archive. A *limit* of 0 will cause pax to stop operation after the first read error is detected on an archive volume. A *limit* of NONE will cause pax to attempt to recover from read errors forever. The default *limit* is a small positive number of retries.

Warning: Using this option with NONE should be used with extreme caution as pax could get stuck in an infinite loop on a very badly flawed archive.

–G *group* Select a file based on its *group* name, or when starting with a # , a numeric gid. A '\' can be used to escape the # . Multiple G options may be supplied. Checking stops with the first match.

–H Follow only command line symbolic links while performing a physical file system traversal.

–L Follow all symbolic links to perform a logical file system traversal.

–P Do not follow symbolic links, perform a physical file system traversal. This is the default mode.

–T *[from_date][,to_date][/[c][m]]* Allow files to be selected based on a file modification or inode change time falling within a specified time range of *from_date* to *to_date* (the dates are inclusive). If only a *from_date* is supplied, all files with a modification or inode change time equal to or younger are selected. If only a *to_date* is supplied, all files with a modification or inode change time equal to or older will be selected. When the *from_date* is equal to the *to_date*, only files with a

modification or inode change time of exactly that time will be selected.

When pax is in the *write* or *copy* mode, the optional trailing field *[c][m]* can be used to determine which file time (inode change, file modification or both) are used in the comparison. If neither is specified, the default is to use file modification time only. The *m* specifies the comparison of file modification time (the time when the file was last written). The *c* specifies the comparison of inode change time (the time when the file inode was last changed; for example, a change of owner, group or mode). When *c* and *m* are both specified, the modification and inode change times are both compared. The inode change time comparison is useful in selecting files whose attributes were recently changed, or selecting files which were recently created and had their modification time reset to an older time (this happens when a file is extracted from an archive and the modification time is preserved). Time comparisons using both file times is useful when pax is used to create a time—based incremental archive (only files that were changed during a specified time range will be archived).

A time range is made up of six different fields and each field must contain two digits. The format is: [yy[mm[dd[hh]]]]mm[.ss] Where yy is the last two digits of the year, the first mm is the month (from 01 to 12), dd is the day of the month (from 01 to 31), hh is the hour of the day (from 00 to 23), the second mm is the minute (from 00 to 59), and ss is the seconds (from 00 to 59). The minute field mm is required, the other fields are optional but must be added in the following order: hh, dd, mm, yy. The ss field may be added independently of the other fields. Time ranges are relative to the current time, so T 1234/cm would select all files with a modification or inode change time of 12:34 PM today or later. Multiple T time ranges can be supplied. Checking stops with the first match.

-U *user* Select a file based on its *user* name, or when starting with a #, a numeric uid. A '\' can be used to escape the #. Multiple U options may be supplied. Checking stops with the first match.

-X When traversing the file hierarchy specified by a pathname, do not descend into directories that have a different device ID. See the st_dev field as described in stat(2) for more information about device ID's.

-Y This option is the same as the -D option, except that the inode change time is checked using the pathname created after all the file name modifications have completed.

`-Z` This option is the same as the `u` option, except that the modification time is checked using the pathname created after all the file name modifications have completed.

The options that operate on the names of files or archive members (`c , i , n , s , u , v , D , G , T , U , Y ,` and `Z`) interact as follows.

When extracting files during a *read* operation, archive members are *selected*, based only on the user—specified pattern operands as modified by the `c , n , u , D , G, T , U` options. Any `s` and `i` options will modify in that order, the names of the selected files. The `Y` and `Z` options will then be applied based on the final pathname. Finally, the `v` option will write the names resulting from these modifications.

When archiving files during a *write* operation, or copying files during a *copy* operation, archive members are *selected*, based only on the user specified pathnames as modified by the `n , u , D , G , T ,` and `U` options (the `D` option only applies during a copy operation). Any `s` options will modify the names of the selected files in that order. During a *copy* operation the `Y` and the `Z` options will then be applied based on the final pathname. Finally, the `v` option will write the names resulting from these modifications.

When one or both of the `u` or `D` options are specified along with the `n` option, a file is not considered selected unless it is newer than the file to which it is compared.

EXAMPLES

The command:

```
pax -w -f /dev/rst0
```

copies the contents of the current directory to the device `/dev/rst0`

The command:

```
pax -r -v -f filename
```

gives the verbose table of contents for an archive stored in `filename`.

The following commands:

```
mkdir newdir cd olddir pax -rw . newdir
```

will copy the entire `olddir` directory hierarchy to `newdir`.

The command:

```
pax -r -s ',^//*usr//*,,' -f a.pax
```

reads the archive `a.pax`, with all files rooted in ‘‘/usr’’ into the archive extracted relative to the current directory.

The command:

pax -rw -i . dest_dir

can be used to interactively select the files to copy from the current directory to `dest_dir`.

The command:

pax -r -pe -U root -G bin -f a.pax

will extract all files from the archive `a.pax` which are owned by *root* with the group *bin* and will preserve all file permissions.

The command:

pax -r -w -v -Y -Z home /backup

will update (and list) only those files in the destination directory `/backup` which are older (less recent inode change or file modification times) than files with the same name found in the source file tree `home`.

STANDARDS

The `pax` utility is a superset of the `-p1003.2` standard. The options `B , D , E , G , H , L , P , T , U , Y , Z ,` the archive formats *bcpio, sv4cpio, sv4crc, tar*, and the flawed archive handling during *list* and *read* operations are extensions of the POSIX standard.

AUTHOR

Keith Muller at the University of California, San Diego

ERRORS

The `pax` utility will exit with one of the following values:

`0` All files were processed successfully.

`1` An error occurred.

Whenever `pax` cannot create a file or a link when reading an archive, or cannot find a file when writing to an archive, or cannot preserve the user ID, group ID, or file mode when the `p` option is specified, a diagnostic message is written to *standard error* and a non-zero exit status will be returned, but processing will continue. In the case where pax cannot create a link to a file, `pax` will not create a second copy of the file.

If the extraction of a file from an archive is prematurely terminated by a signal or error, `pax` may have only partially extracted a file the user wanted. Additionally, the file modes of extracted files and directories may have incorrect file bits, and the modification and access times may be wrong.

If the creation of an archive is prematurely terminated by a signal or error, `pax` may have only partially created the archive which may violate the specific archive format specification.

If, while doing a `copy`, `pax` detects that a file is about to overwrite itself, the file is not copied, a diagnostic message is written to *standard error* and when `pax` completes it will exit with a non-zero exit status.

RESTRICTIONS FOR ChorusOS

Options relative to user ID and group ID are not implemented in this release.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

NAME

PROF – ChorusOS profiler server

SYNOPSIS

PROF

DESCRIPTION

PROF is a daemon which answers profiling requests issued by *profctl*(1CC). *PROF* is a supervisor actor that can locate and modify static data into the memory context of the profiled supervisor actors. It does this using the system symbol tables (via the N-sym actor). The profiler also dynamically creates and deletes the memory regions used to construct the call graph and count the profiling ticks.

PROF must be running on any target where supervisor profiling is needed, and can only profile CHORUS kernel or supervisor actors. To tally the number of calls to a routine, the modules that make up the program must be compiled using the *cc -p* option. To be profiled, a dynamically loaded actor must be run using the *-k* option of the arun *C_INIT*(1M) command.

To enable profile, *PROF* should be launched before any profiling session as follows:

```
rsh -n $REMOTE_TARGET arun PROF &
```

The *PROF* daemon can be killed using the aps and akill *C_INIT*(1M) commands.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M), rsh(1UNIX), **profctl**(1CC)

NAME

profctl – ChorusOS profiling control tool

SYNOPSIS

profctl –`start` [–`oneshot`] [–`raten`] [–`c`cact_namecact_id...]
[–b[bact_name]...]

profctl –`stop`

DESCRIPTION

The *profctl* utility is a ChorusOS tool that controls the profiling of the CHORUS kernel and supervisor actors. It must be run as a `Supervisor` c_actor (see *-S* option of the arun *C_INIT*(1M) command).

The *PROF*(1CC) supervisor actor must be active on each site where profiling is needed.

The symbol table supervisor actor N_sym must be loaded and provide access to the symbol table of the specified components (the profiled components must either be using with the *-k* option of the arun *C_INIT*(1M) command, or they must be boot actors).

The options are:

–`start` Start profiling. The options *-start* and *-stop* are mutually exclusive.

–`c` Gives the list of profiled c_actors. The c_actor id of the c_actor is called *cact_id*. The c_actor is called *cact_name*

–`b` Gives the list of profiled boot actors. The boot actor is called *bact_name*. The special keyword *kern* may be used to profile the CHORUS kernel.

At a minimum, the CHORUS kernel or one supervisor actor must be specified in the *-start* command.

–`rate n` Sets the sampling period in microseconds. The profiler selects the closest sampling period permitted by the hardware. The selected sampling will be shownin the profiling report. The default sampling period is 1000 microseconds.

–`oneshot` Sets the one shot sampling mode. If tise option is omitted, the default is the square mode.

–`stop` Stop profiling. For each component currently being profiled, *profctl* creates a file of raw data results. The name of the file is the name of the profiled actor with a *.prof* suffix added. The raw file is created in the target's *tmp* directory. The raw

data results file is a binary file that is designed to be processed by the profile report generator *profrpg*(1CC).

RESTRICTIONS

Results are unpredictable when several users create simultaneous profiling sessions on a CHORUS site.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M), **PROF**(1CC), **profrpg**(1CC)

NAME

profrpg – ChorusOS profiling report generator

SYNOPSIS

profrpg [–lftrs] raw_report

DESCRIPTION

The profrpg utility program is used to produce reports about the profiling of the CHORUS kernel or supervisor actors. It may be run on the host system after a target profiling session.

The *actor* argument is mandatory. The argument is the name of the profiled component for which a report is requested. Theprofrpg utility takes raw profiling data from the *raw_report.prof* file.

The profrpg utility writes the profiling report to the standard output.

If several components were involved in the profiling, a separate report must be generated for each component.

The options are:

–l Sort by increasing symbol address.

–f Sort by decreasing flat time.

–t Sort by decreasing total time. This is the default option.

The *l*, *f* and *t* options are mutually exclusive.

–r When specified, the report displays any recursive loops detected in addition to profiling data. This option is only effective if the full report form can be produced for the component (components generated using the compiler profile option).

–s This option produces a simplified form of the output. The output includes, for each function, the symbolic function name and its component percentage cost. This option is very useful when the simple report form only is available for the component.

EXAMPLES

A full profiling session of a sample actor named *foo*,whose id is *foo_id* would contain the following sequence:

```
rsh -n $REMOTE_TARGET arun PROF &
rsh $REMOTE_TARGET arun -k foo &
                    # returns foo_id
rsh $REMOTE_TARGET arun profctl -start -c foo foo_id
                    # wait for foo to complete
rsh $REMOTE_TARGET arun profctl -stop
bin/profrpg $EXPORT_DIR/foo > foo.rpg
```

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M), **PROF**(1CC), **profctl**(1CC)

NAME

rdbc – ChorusOS remote debugging daemon

SYNOPSIS

rdbc [*udp-port udp-back-port*]

DESCRIPTION

`rdbc` provides debugging support for ChorusOS supervisor and user extended actors.

It is a daemon which processes remote debugging requests issued by the Microtec XRAY for ChorusOS debugger. It is accessed from a host by using the Sun RPC over UDP/IP. `rdbc` must run on a ChorusOS system which includes at least the ACTOR_EXTENDED_MNGT and POSIX_SOCKETS features (minimal Actor and I/O managers).

To enable remote actor debugging (if `rdbc` is not yet running on the target), launch `rdbc` before any remote debugging session, as follows:

```
rsh -n $REMOTE_TARGET arun rdbc &
```

The `rdbc` daemon can be killed using the aps and akill *C_INIT*(1M) commands.

It is possible to run several copies of `rdbc` in parallel; different *udp-port* and *udp-back-port* values must be used for each copy. The default values are 2072 and 2074. The selected value of *udp-port* must be re-used to create a new board entry in XRAY's `chorusos.brd` file.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

C_INIT(1M), rsh(1UNIX)

RESTRICTIONS

When the debugged actor has been started from XRAY, the standard input/output of the debugged application is that is inherited from the `rdbc` daemon. If this is not what you require, start the application using `rsh`, passing the `–d –D` options to `arun`, then attach from XRAY. An alternative method would be to use a separate copy of `rdbc` to debug the application.

NAME

rdbs – ChorusOS system debug server for the Microtec XRAY debugger

SYNOPSIS

rdbs [–echo-console|–fast-start|–force-slot|–slot *number*|–target-host *hostname*[*:slot*]|–target-port *portnumber*] [*targetname*]

DESCRIPTION

`rdbs` is a server which allows the Microtec XRAY debugger to debug a ChorusOS target at the system level. That means debugging threads, interrupt handlers, and the boot code.

`rdbs` is very similar to `rdbc`, the application debug server. However, `rdbc` runs on the target, while `rdbs` runs on a host.

`rdbs` uses the DebugServer server process also running on the host, and adapts its API to XRAY requirements.

`rdbs` is specific to a given target processor. It displays the target processor type on startup.

The setup and usage of `rdbs` are described in the *ChorusOS System Debug User's Guide*.

OPTIONS

The following options are supported:

–echo-console — Makes `rdbs` to echo the console output of the target machine.

–fast-start — Does not check for another `rdbs` already using the selected incoming slot. This significantly accelerates the startup.

–force-slot — If another `rdbs` is already using a given slot, steals it from it.

–slot *number* — Creates incoming slot *number* instead of the default 0. It is possible to run several `rdbs` servers on the same host at the condition that each one uses a different incoming slot number. The slot number must be copied as "port number" into the `chorusos.brd` file used by XRAY.

–target-host *hostname[:slot]* Defines the *hostname* of the host, and optionally the *slot* on which the DebugServer runs. Several DebugServers can run on a single host, each one using a different slot.

–`target-port` *portnumber* Defines the *portnumber* of DebugServer on the host where it runs. This option is reserved for future usage.

targetname Preferred target on DebugServer. By default, `rdbs` will use the first available target on DebugServer which is of the same processor type as `rdbs`.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

ChorusOS System Debug User's Guide

NOTES

On startup, `rdbs` will print the processor type for which it has been compiled (PowerPC, x86 or SPARC) and the name of the target if the target was found automatically.

NAME

rm – remove directory entries

SYNOPSIS

rm [– [[f]|[i]] dPRr] file...

DESCRIPTION

The `rm` utility attempts to remove the non-directory type files specified on the command line. If the permissions of the file do not permit writing, and the standard input device is a terminal, the user is prompted (on the standard error output) for confirmation.

The options are as follows:

`–d` Attempt to remove directories as well as other types of files.

`–f` Attempt to remove the files without prompting for confirmation, regardless of the file's permissions. If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error. The `–f` option overrides any previous `–i` options.

`–i` Request confirmation before attempting to remove each file, regardless of the file's permissions, or whether or not the standard input device is a terminal. The `–i` option overrides any previous `–f` options.

`–P` Overwrite regular files before deleting them. Files are overwritten three times, first with the byte pattern 0xff, then 0x00, and then 0xff again, before they are deleted.

`–R` Attempt to remove the file hierarchy rooted in each file argument. The `–R` option implies the `–d` option. If the `–i` option is specified, the user is prompted for confirmation before each directory's contents are processed (as well as before the attempt is made to remove the directory). If the user does not respond affirmatively, the file hierarchy rooted in that directory is skipped.

`–r` Equivalent to `–R`.

The `rm` utility removes symbolic links, but not the files referenced by the links.

It is an error to attempt to remove the files ''.'' and ''..''.

The `rm` utility exits 0 if all of the named files or file hierarchies were removed, or if the `–f` option was specified and all of the existing files or file hierarchies were removed. If an error occurs, `rm` exits with a value of >0.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO **rmdir**(1CC), **unlink**(2POSIX)

BUGS The –P option assumes that the underlying file system is a fixed-block file system. UFS is a fixed-block file system, LFS is not. In addition, only regular files are overwritten, other types of files are not.

COMPATIBILITY The rm utility differs from historical implementations in that the –f option only masks attempts to remove non-existent files instead of masking a large variety of errors.

Historical "implementations" are prompted on the standard output, not the standard error output.

NAME

rmdir – remove directories

SYNOPSIS

rmdir

DESCRIPTION

The `rmdir` utility removes the directory entry specified by each directory argument, provided it is empty.

Arguments are processed in the order given. In order to remove both a parent directory and a subdirectory of that parent, the subdirectory must be specified first so that the parent directory is empty when `rmdir` tries to remove it.

DIAGNOSTICS

This utility exits with 0 if each directory entry specified by a dir operand referred to an empty directory and was removed successfully, otherwise a value of > 0 is returned.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`mkdir`(1CC)

NAME

touch – change file access and modification times

SYNOPSIS

touch [–acfm] [–r *file*] [–t [[*CC*]*YY*]*MMDDhhmm*[.*SS*]] {*file*...}

DESCRIPTION

touch sets the modification and access times of files. If a file specified does not exist, it is created with default permissions.

OPTIONS

touch accepts the following options:

–a
: Change the access time of the file. The modification time of the file is not changed unless the –m option is also specified.

–c
: Do not create the file if it does not exist. touch does not treat this as an error. No error messages are displayed and the exit value is not affected.

–f
: Attempt to force the update, even if the file permissions do not currently permit it.

–m
: Change the modification time of the file. The access time of the file is not changed unless the –a option is also specified.

–r
: Use the access and modification times from the specified file instead of the current time of day.

–t
: Change the access and modification times to the specified time. The argument must be of the form
[[[*CC*]*YY*]*MMDDhhmm*[.*SS*]]
where each pair of letters represents the following:

  CC
  : First two digits of the year that represent the century.

  YY
  : Last two digits of the year. If *YY* is specified and *CC* is not: values of *YY* between 69 and 99 result in a value of 19 for *CC*; other values of *YY* result

	in a value of `20` for *CC*.
MM	The month of the year, from `01` to `12`.
DD	The day of the month, from `01` to `31`.
hh	The hour of the day, from `01` to `24`.
mm	The minute of the hour, from `01` to `59`.
SS	The second of the minute, from `01` to `59`.

OPERANDS

The following operands are supported:

file Change access and modification times of *file*.

EXIT STATUS

`touch` exits with `0` if successful, and a number greater than `0` if an error occurs.

ATTRIBUTES

See **`attributes`**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`utimes`(2POSIX)

NOTES

`touch` supports the obsolescent form of the same command, for which a time format is specified as the first argument. When no `–r` or `–t` option is specified, there are at least two arguments, the first argument being a string of characters either eight or ten characters in length, interpreteted as a time specification of the form

[*MMDDhhmm*[*YY*]]

where values of *YY* between `39` and `99` imply the twentieth century and values between `00` and `38` imply the twenty-first century.

NAME

uname – display information about the system

SYNOPSIS

uname [–amnprsv]

DESCRIPTION

uname writes the name of the operating system implementation to standard output. When options are specified, strings representing one or more system characteristics are written to standard output.

OPTIONS

uname supports the following options:

–a Behave as though the –m, –n, –r, –s and –v options are specified.

–m Write the type of the current hardware platform to standard output.

–n Write the name of the system to standard output.

–p Write the same value as –m. This option is provided for backward compatibility with SVR4.

–r Write the current release level of the operating system to standard output.

–s Write the name of the operating system implementation to standard output.

–v Write the version level of this release of the operating system to standard output.

If the –a option is used, or if multiple options are used, all output is written on a single line and is separated by spaces.

EXIT STATUS

uname exits with 0 if successful, and a number greater than 0 if an error occurs.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

sysctl(3POSIX), **sysctl**(1M)

NAME

ypcat – print the values of all keys in a YP database

SYNOPSIS

ypcat [–k] [–ddomainname] [–t] mapname

ypcat [–x]

DESCRIPTION

The ypcat command prints out the values in a Network Information Service (NIS) map specified by mapname, which may be either a map name or a map nickname . As ypcat uses the NIS service, no NIS server is specified.

For example, to look at the network-wide password database, passwd.byname, (with the nickname passwd), type in:

```
ypcat passwd
```

OPTIONS

–d *domainame*	Specify a domain other than the default domain.
–k	Display map keys. This option is useful with maps in which the values are null or the key is not part of the value.
–t	Inhibit translation of map nicknames to their corresponding map names.
–x	Display the map nickname table.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

ypmatch(1CC)

NAME

ypmatch – print the values of one or more keys in a YP database

SYNOPSIS

ypmatch [–d *domainname*] [–k] [–t] *key... mapname*

ypmatch [–x]

DESCRIPTION

The ypmatch command prints out the values of one or more keys from the database specified by mapname, which may be a map name or a map nickname.

OPTIONS

–d *domainame*	Specify a domain other than the default domain.
–k	Display map keys. This option is useful with maps in which the values are null or the key is not part of the value.
–t	Inhibit translation of map nicknames to their corresponding map names.
–x	Display the map nickname table.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

ypcat(1CC)

NAME

ypwhich – return the name of the NIS server or map master

SYNOPSIS

ypwhich [–ddomain][[–t]|–m|[mname]|host-name]

ypwhich –x

DESCRIPTION

The ypwhich command returns the name of the NIS server which supplies the NIS name services to an NIS client, or one which is the master for a map. If invoked without arguments, it returns the NIS server for the local machine. If hostname is specified, that machine is queried to find out which NIS master it is using.

OPTIONS

–d *domain*	Use this domain (instead of the default domain).
–t	This option inhibits map nickname translation.
–m *mname*	Find the master NIS server for a map. No hostname can be specified when using -m. The mname parameter can be a mapname or a nickname for a map. When mname is omitted, a list of all available maps is produced.
–x	Display the map nickname translation table.

ATTRIBUTES

See **attributes**(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

inetNS(1M), **ypbind**(1M)

Index

R

T

U

Y