Guia do desenvolvedor de empacotamento de aplicativos

Número do Item: E38878
Janeiro de 2013

ORACLE

130224@25097

Conteúdo

Prefácio

O *Guia do desenvolvedor de empacotamento de aplicativos* oferece instruções detalhadas e informações relevantes para a criação, construção e verificação de pacotes. Este guia inclui também técnicas avançadas que podem ser úteis durante o processo de criação de pacotes.

Observação – Esta versão do Oracle Solaris oferece suporte a sistemas que usam as famílias SPARC e x86 de arquiteturas de processadores. Os sistemas compatíveis aparecem no *Oracle Solaris OS: Hardware Compatibility Lists*. Este documento cita todas as diferenças de implementação entre os tipos de plataformas.

Neste documento, esses termos relacionados ao x86 significam o seguinte:

- x86 refere-se à maior família de produtos compatíveis com x86 de 32 e 64 bits.
- x64 refere-se especificamente às CPUs compatíveis com x86 de 64 bits.
- "x86" de 32 bits indica informações específicas de 32 bits sobre sistemas baseados em x86.

Para saber mais sobre os sistemas suportados, consulte *Oracle Solaris OS: Hardware Compatibility Lists*.

Quem deve usar este manual

Este livro está destinado a desenvolvedores de aplicativos cujas responsabilidades incluem a criação e a construção de pacotes.

Embora a maior parte do livro esteja dedicada a desenvolvedores principiantes de pacotes, também contém informações úteis para os desenvolvedores mais experientes.

Como este livro é organizado

O quadro abaixo descreve os capítulos deste livro.

Nome do capítulo	Descrição do capítulo
Capítulo 1, “Criando um Pacote”	Descreve os componentes e o critério de criação do pacote. Também descreve os comandos, arquivos e scripts relacionados.
Capítulo 2, “Construindo um Pacote”	Descreve o processo e as tarefas necessárias para a construção de um pacote. Também oferece instruções detalhadas de cada tarefa.
Capítulo 3, “Melhorando a funcionalidade de um pacote (Tarefas)”	Oferece instruções detalhadas para a adição de recursos opcionais a um pacote.
Capítulo 4, “Verificando e transferindo um pacote”	Descreve como verificar a integridade de um pacote e como transferir um pacote a um meio de distribuição.
Capítulo 5, “Estudos de caso de criação de pacote”	Oferece estudos de caso da criação de pacotes.
Capítulo 6, “Técnicas avançadas para a criação de pacotes”	Descreve técnicas avançadas para a criação de pacotes.
Glossário	Define os termos usados neste livro.

Livros relacionados

A documentação seguinte, disponível em livrarias, pode proporcionar informações adicionais sobre a construção de pacotes de System V.

- *Interface Binária do Aplicativo System V*
- *Interface Binária do Aplicativo System V - Suplemento do Processador SPARC*
- *Interface Binária do Aplicativo System V - Suplemento do Processador Intel386*

Acesso ao suporte Oracle

Os clientes Oracle possuem acesso a suporte eletrônico por meio do My Oracle Support. Para obter informações, visite `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou visite `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` se você é portador de deficiência auditiva.

Convenções tipográficas

A tabela a seguir descreve as convenções tipográficas usadas neste livro.

TABELA P–1 Convenções tipográficas

Fonte	Descrição	Exemplo
`AaBbCc123`	Nomes de comandos, arquivos, diretórios e saídas do computador na tela	Edite seu arquivo `.login`. Use `ls -a` para listar todos os arquivos. `machine_name%, você tem e-mail.`
`AaBbCc123`	O que você digita, em comparação com a saída do computador na tela	`machine_name%` **`su`** `Senha:`
aabbcc123	Espaço reservado: substitua, aplicando um nome ou valor real	O comando para remover um arquivo é `rm` *filename* .
AaBbCc123	Títulos de manuais, termos novos e termos a serem enfatizados	Consulte o Capítulo 6 do *Guia do Usuário*. Um *cache* é uma cópia que é armazenada localmente. *Não* salve o arquivo. **Nota:** Alguns itens enfatizados aparecem on-line em negrito.

Prompts do shell em exemplos de comando

A tabela a seguir mostra o prompt de sistema UNIX padrão e o prompt do superusuário para shells, que estão incluídos no Oracle Solaris OS. Note que o prompt do sistema padrão que é exibido em exemplos de comando varia dependendo da versão do Oracle Solaris.

TABELA P–2 Prompts de shell

Shell	Prompt
Bash shell, Korn shell e Bourne shell	`$`
Bash shell, Korn shell e Bourne shell para o superusuário	`#`
Shell C	`nome_da_máquina%`
Shell C para superusuário	`nome_da_máquina#`

◆ ◆ ◆ CAPÍTULO 1

Criando um Pacote

Antes de construir um pacote, você precisa saber que arquivos precisam ser criados e que comandos precisam ser executados. Também é necessário levar em consideração os requisitos do software de aplicativos e as necessidades dos seus clientes. Seus clientes são os administradores que instalarão seu pacote. Este capítulo trata dos arquivos, comandos e critérios que você deveria conhecer e considerar antes de construir um pacote.

A lista abaixo traz as informações encontradas neste capítulo.

Onde encontrar tarefas de empacotamento

Use estes mapas de tarefas para obter as instruções detalhadas de construção e verificação de pacotes.

O que são pacotes?

O software de aplicativos é distribuído em unidades denominadas *pacotes*. Um pacote é um conjunto de arquivos e diretórios necessários para um produto de software. Um pacote é geralmente criado e construído por um desenvolvedor de aplicativos depois de terminar de desenvolver o código do aplicativo. Um produto de software precisa ser construído em um ou mais pacotes para que possa ser facilmente transferido para um meio de distribuição. Depois, o produto de software pode ser produzido em massa e instalado por administradores.

Um pacote é um conjunto de arquivos e diretórios em um formato definido. Este formato adapta-se à ABI (Application Binary Interface), que é um suplemento de System V Interface Definition.

Componentes do pacote

Os componentes de um pacote se dividem em duas categorias.

- Os *objetos do pacote* são arquivos de aplicativo que serão instalados.
- Os *arquivos de controle* controlam como, onde e se o pacote está instalado.

Os arquivos de controle também estão divididos em duas categorias: *arquivos de informação* e *scripts de instalação*. Alguns arquivos de controle são obrigatórios. Alguns arquivos de controle são opcionais.

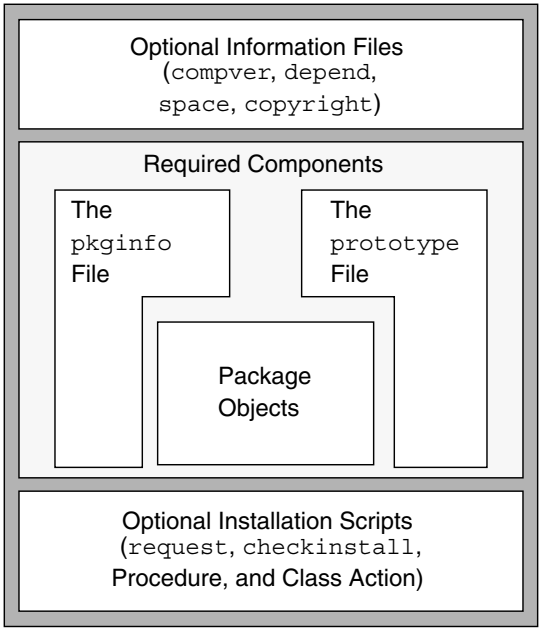
Para empacotar seus aplicativos, você deve primeiro criar os componentes obrigatórios e os componentes opcionais que farão parte do seu pacote. Você pode, então, construir o pacote usando o comando `pkgmk`.

Para construir um pacote, você deve fornecer:

- Objetos de pacote (arquivos e diretórios do software de aplicativo)
- Dois arquivos de informação (os arquivos `pkginfo` e `prototype`)
- Arquivos de informação opcionais
- Scripts de instalação opcionais

A ilustração seguinte descreve o conteúdo de um pacote.

FIGURA 1–1 O conteúdo de um pacote



Componentes de pacote necessários

Você deve criar os seguintes componentes antes de construir o pacote:

- Objetos de pacote

  Estes componentes fazem parte do aplicativo. Podem ser os seguintes itens:

  - Arquivos (arquivos executáveis ou arquivos de dados)
  - Diretórios
  - Pipes nomeados
  - Links
  - Dispositivos

- O arquivo `pkginfo`

  O `pkginfo` é um arquivo de informação do pacote necessário que define os valores de parâmetro. Os valores de parâmetro são a abreviação, o nome completo e a arquitetura do pacote. Para obter mais informações, consulte "Criando um arquivo `pkginfo`" na página 26 e a página do manual `pkginfo(4)`.

Observação – Há duas páginas do manual `pkginfo(1)` A primeira página do manual descreve um comando da seção 1 que exibe informações sobre os pacotes instalados. A segunda página do manual descreve um arquivo da seção 4 que descreve as características de um pacote. Ao acessar as páginas do manual, certifique-se de especificar a seção de página do manual aplicável. Por exemplo: `man -s 4 pkginfo`.

- O arquivo `prototype`

  O arquivo `prototype` é um arquivo de informação do pacote necessário que lista os componentes do pacote. Existe uma entrada para cada objeto de pacote, arquivo de informação e script de instalação. Uma entrada consiste em vários campos de informação que descrevem cada componente, incluindo o local, os atributos e o tipo de arquivo. Para obter mais informações, consulte "Criando um arquivo `prototype`" na página 31 e a página do manual `prototype(4)`.

Componentes de pacote opcionais

Arquivos de informação do pacote

Você pode incluir quatro arquivos de informação de pacote opcionais no seu pacote:

- O arquivo `compver`

  Define as versões anteriores do pacote compatíveis com esta versão do pacote.
- O arquivo `depend`

  Indica outros pacotes que possuem relação especial com o seu pacote.
- O arquivo `space`

  Define os requisitos de espaço em disco do ambiente de destino, além do que é necessário pelos objetos definidos no arquivo `prototype`. Por exemplo, poderia ser necessário espaço adicional para os arquivos criados dinamicamente no momento da instalação.
- O arquivo `copyright`

  Define o texto da mensagem de copyright exibida no momento da instalação do pacote.

Cada arquivo de informação do pacote deve ter uma entrada no arquivo `prototype`. Consulte "Criando arquivos de informação" na página 52 para obter mais informações sobre a criação desses arquivos.

Scripts de instalação do pacote

Os scripts de instalação não são necessários. No entanto, você pode fornecer scripts que realizem ações personalizadas durante a instalação do seu pacote. Um script de instalação tem as seguintes características:

- O script está composto por comandos do shell Bourne.
- As permissões do arquivo do script devem estar definidas como 0644.
- O script não precisa conter o identificador do shell (`#! /bin/sh`).

Os quatro tipos de script são os seguintes:

- O script `request`

  O script `request` solicita a entrada do administrador que estiver instalando o pacote.

- O script `checkinstall`

  O script `checkinstall` realiza verificação especial do sistema de arquivos.

  Observação – O script `checkinstall` está disponível somente no Solaris 2.5 e em versões compatíveis.

- Scripts de procedimento

  Os scripts de procedimento definem as ações que ocorrem em determinados pontos durante a instalação e remoção do pacote. Você pode criar quatro scripts de procedimento com esses nomes predefinidos: `preinstall`, `postinstall`, `preremove` e `postremove`.

- Scripts de ação de classe

  Os *scripts de ação de classe* definem um conjunto de ações que será realizado em um grupo de objetos.

Consulte "Criando scripts de instalação" na página 58 para obter mais informações sobre os scripts de instalação.

Considerações antes de construir um pacote

Antes de construir um pacote, é preciso decidir se o produto terá um ou mais pacotes. Observe que muitos pacotes pequenos demoram mais para serem instalados que um pacote grande. Embora criar um único pacote seja uma boa idéia, às vezes nem sempre é possível. Se decidir construir mais de um pacote, é preciso determinar como segmentar o código do aplicativo. Esta seção oferece uma lista de critérios para serem usados quando se está planejando construir um pacote.

Muitos dos critérios de empacotamento apresentam equiparações entre eles. Satisfazer todos os requisitos igualmente é muitas vezes difícil. Estes critérios são apresentados em ordem de importância. No entanto, esta seqüência serve apenas como um guia flexível dependendo das circunstâncias. Embora cada critério seja importante, depende de você otimizar estes requisitos para produzir um bom conjunto de pacotes.

Para obter mais idéias de criação, consulte o Capítulo 6, "Técnicas avançadas para a criação de pacotes".

Fazer com que os pacotes possam ser instalados remotamente

Todos os pacotes devem poder ser instalados *remotamente*. Ser remotamente instalável significa que o administrador que está instalando seu pacote pode instalá-lo em um sistema cliente, não necessariamente no sistema de arquivos raiz (`/`) onde o comando `pkgadd` está sendo executado.

Otimizar as configurações servidor-cliente

Leve em consideração os vários tipos de configurações de software do sistema (por exemplo, servidor e sistema independentes) ao dispor os pacotes. Um bom design de empacotamento divide os arquivos afetados para otimizar a instalação de cada tipo de configuração. Por exemplo, o conteúdo dos sistemas de arquivos raiz (`/`) e `/usr` deve ser segmentado para que as configurações do servidor possam ser facilmente suportadas.

Pacotes por limites funcionais

Os pacotes devem ser independentes e identificados claramente com um conjunto de funcionalidades. Por exemplo, um pacote que contém UFS deve conter todos os utilitários UFS e estar limitado somente a binários UFS.

Os pacotes devem estar organizados do ponto de vista do cliente dentro de unidades funcionais.

Pacotes com limites de royalty

Use códigos que necessitam pagamentos de royalty devido a acordos contratuais em um pacote dedicado ou grupo de pacotes. Não disperse o código em mais pacotes do que o necessário.

Pacotes por dependências de sistema

Mantenha os binários dependentes de sistemas em pacotes dedicados. Por exemplo, o código do kernel deve estar em um pacote dedicado, com cada arquitetura de implementação formada por uma instância diferente de pacote. Esta regra também se aplica a binários de arquiteturas diferentes. Por exemplo, os binários de um sistema SPARC devem estar em um pacote e os binários de um sistema x86 devem estar em outro pacote.

Eliminar sobreposições em pacotes

Ao construir pacotes, elimine os arquivos duplicados sempre que possível. A duplicação desnecessária de arquivos causa dificuldades de suporte e versão. Se o seu produto tiver vários pacotes, compare repetidamente o conteúdo desses pacotes para procurar arquivos duplicados.

Pacotes com limites de localização

Itens específicos de localização devem estar em um pacote próprio. Um modelo de empacotamento ideal deve ter as localizações de um produto distribuídas como um pacote por localidade. Infelizmente, em alguns casos os limites organizacionais entram em conflito com os critérios de limites funcionais e de produto.

Os padrões internacionais também podem ser distribuídos em um pacote. Este design isola os arquivos necessários nas alterações de localizações e padroniza o formato de distribuição dos pacotes de localização.

Comandos, arquivos e scripts do pacote

Esta seção descreve os comandos, arquivos e scripts que devem ser usados ao manipular pacotes. Eles são descritos nas páginas do manual e detalhadamente neste livro, de acordo com as tarefas específicas que eles realizam.

A tabela seguinte mostra os comandos que ajudam a construir, verificar, instalar e obter informações de um pacote.

TABELA 1–1 Comandos de pacotes

Tarefa	Comando/ página do manual	Descrição	Para Obter Mais Informações
Criar pacotes	`pkgproto`(1)	Gera um arquivo `prototype` para entrada do comando `pkgmk`	"Exemplo — Criando um arquivo `prototype` com o comando `pkgproto`" na página 37
`pkgmk`(1)	Cria um pacote instalável	"Construindo um Pacote" na página 45	
Instalar, remover e transferir pacotes	`pkgadd`(1M)	Instala um pacote de software em um sistema	"Instalando os pacotes de software" na página 88
`pkgask`(1M)	Armazena as respostas de um script `request`	"Regras de criação para scripts `request`" na página 63	
`pkgtrans`(1)	Copia pacotes em meio de distribuição	"Transferindo um pacote para um meio de distribuição" na página 97	

TABELA 1–1 Comandos de pacotes *(Continuação)*

Tarefa	Comando/ página do manual	Descrição	Para Obter Mais Informações
	`pkgrm`(1M)	Remove um pacote de um sistema	"Removendo um pacote" na página 97
Obter informações sobre pacotes	`pkgchk`(1M)	Verifica a integridade de um pacote de software	"Verificando a integridade de um pacote" na página 90
	`pkginfo`(1)	Exibe as informações de um pacote de software	"O comando `pkginfo`" na página 93
	`pkgparam`(1)	Exibe os valores de parâmetro do pacote	"O comando `pkgparam`" na página 92
Modificar os pacotes instalados	`installf`(1M)	Incorpora um novo objeto de pacote em um pacote instalado	"Regras de criação dos scripts de procedimento" na página 68 e Capítulo 5, "Estudos de caso de criação de pacote"
	`removef`(1M)	Remove um objeto de pacote de um pacote que já está instalado	"Regras de criação dos scripts de procedimento" na página 68

A tabela seguinte mostra os arquivos de informação que ajudam a construir um pacote.

TABELA 1–2 Arquivos de informação do pacote

Arquivo	Descrição	Para Obter Mais Informações
`admin`(4)	Arquivo de padrões de instalação de pacote	"O arquivo de padrões administrativos" na página 128
`compver`(4)	Arquivo de compatibilidade do pacote	"Definindo dependências do pacote" na página 52
`copyright`(4)	Arquivo de informação de copyright do pacote	"Escrevendo uma mensagem de copyright" na página 55
`depend`(4)	Arquivo de dependências do pacote	"Definindo dependências do pacote" na página 52
`pkginfo`(4)	Arquivo das características do pacote	"Criando um arquivo `pkginfo`" na página 26
`pkgmap`(4)	Arquivo de descrição do conteúdo do pacote	"O arquivo `pkgmap`" na página 45
`prototype`(4)	Arquivo de informações do pacote	"Criando um arquivo `prototype`" na página 31
`space`(4)	Arquivo de requisitos de espaço em disco do pacote	"Reservando espaço adicional em um sistema de destino" na página 57

A tabela seguinte descreve os scripts de instalação opcionais que você pode escrever e que afetam se e como um pacote é instalado.

TABELA 1–3 Scripts de instalação do pacote

Script	Descrição	Para Obter Mais Informações
`request`	Solicita informações do instalador	"Escrevendo um script `request`" na página 63
`checkinstall`	Coleta dados do sistema de arquivos	"Coletando dados do sistema de arquivos com o script `checkinstall`" na página 65
`preinstall`	Realiza qualquer requisito de instalação personalizada antes da instalação da classe	"Escrevendo scripts de procedimento" na página 68
`postinstall`	Realiza qualquer requisito de instalação personalizada depois que todos os volumes tiverem sido instalados	"Escrevendo scripts de procedimento" na página 68
`preremove`	Realiza qualquer requisito de remoção personalizada antes da remoção da classe	"Escrevendo scripts de procedimento" na página 68
`postremove`	Realiza qualquer requisito de remoção personalizada depois que todas as classes tiverem sido removidas	"Escrevendo scripts de procedimento" na página 68
Ação de classe	Realiza uma série de ações em um grupo de objetos específico	"Escrevendo scripts de ação de classe" na página 70

◆ ◆ ◆ CAPÍTULO 2

Construindo um Pacote

Este capítulo descreve o processo e as tarefas envolvidas na construção de um pacote. Algumas dessas tarefas são necessárias. Algumas dessas tarefas são opcionais. As tarefas necessárias são tratadas detalhadamente neste capítulo. Para obter mais informações sobre as tarefas opcionais, que permitem adicionar mais recursos ao pacote, consulte Capítulo 3, "Melhorando a funcionalidade de um pacote (Tarefas)" e Capítulo 6, "Técnicas avançadas para a criação de pacotes".

A lista abaixo traz as informações encontradas neste capítulo.

O processo de construção de um pacote (Mapa de tarefas)

Tabela 2–1 descreve um processo a ser seguido na construção de um pacote, especialmente se você não tiver experiência em construir pacotes. Embora não seja obrigatório realizar as primeiras quatro tarefas na ordem exata listada, a construção do pacote será mais fácil se a ordem for seguida. Depois que tiver adquirido experiência na criação de pacotes, você pode alterar a seqüência das tarefas de acordo com sua preferência.

Como um designer de pacotes com experiência, você pode automatizar o processo de construção de um pacote usando o comando `make` e os makefiles. Para obter mais informações, consulte a página do manual `make(1S)`.

TABELA 2–1 Mapa de tarefas do processo de construção de um pacote

Tarefa	Descrição	Instruções
1. Criar um arquivo `pkginfo`	Crie o arquivo `pkginfo` para descrever as características do pacote.	"Como criar um arquivo `pkginfo`" na página 29
2. Organizar o conteúdo do pacote	Ordene os componentes do pacote em uma estrutura de diretórios hierárquica.	"Organizando o conteúdo de um pacote" na página 30
3. (Opcional) Criar arquivos de informação	Defina as dependências do pacote, inclua uma mensagem de copyright e reserve espaço adicional no sistema de destino.	Capítulo 3, "Melhorando a funcionalidade de um pacote (Tarefas)"
4. (Opcional) Criar scripts de instalação	Personalize os processos de instalação e remoção do pacote.	Capítulo 3, "Melhorando a funcionalidade de um pacote (Tarefas)"
5. Criar um arquivo `prototype`	Descreva o objeto em seu pacote em um arquivo `prototype`.	"Criando um arquivo `prototype`" na página 31
6. Construir o pacote	Construa um pacote usando o comando `pkgmk`.	"Construindo um Pacote" na página 45
7. Verificar e transferir o pacote	Verifique a integridade do pacote antes de copiá-lo em um meio de distribuição.	Capítulo 4, "Verificando e transferindo um pacote"

Variáveis de ambiente do pacote

Você pode usar variáveis nos arquivos de informação necessários, `pkginfo` e `prototype`. Você também pode usar uma opção para o comando `pkgmk`, usada para construir um pacote. À medida que estes arquivos e comandos são tratados neste capítulo, mais informações sobre variáveis relacionadas ao contexto são proporcionadas. No entanto, antes de começar a construir o pacote, você deve conhecer os diferentes tipos de variáveis e como elas podem afetar a criação bem-sucedida de um pacote.

Há dois tipos de variáveis:

- Variáveis de construção

  As variáveis de construção começam com uma letra minúscula e são avaliadas no *tempo de construção*, conforme o pacote está sendo construído com o comando `pkgmk`.

- Variáveis de instalação

  As variáveis de instalação começam com uma letra maiúscula e são avaliadas no *tempo de instalação*, conforme o pacote está sendo instalado com o comando `pkgadd`.

Regras gerais sobre o uso das variáveis de ambiente

No arquivo `pkginfo`, a definição de uma variável tem a forma *PARAM=value*, na qual a primeira letra de *PARAM* é maiúscula. Estas vairáveis são avaliadas somente no momento da instalação. Se alguma dessas variáveis não puder ser avaliada, o comando `pkgadd` aborta com um erro.

No arquivo `prototype`, a definição de uma variável pode ter a forma `!`*PARAM=value* ou `$`*variable*. Tanto *PARAM* quanto *variable* podem começar com uma letra maiúscula ou minúscula. São avaliadas somente as variáveis cujos valores são conhecidos no momento da instalação. Se *PARAM* ou *variable* for uma variável de construção ou de instalação cujo valor não é conhecido no momento da construção, o comando `pkgmk` aborta com um erro.

É possível incluir a opção *PARAM=value* como uma opção do comando `pkgmk`. Esta opção funciona da mesma forma que no arquivo `prototype`, exceto que seu escopo é global e abrange todo o pacote. A definição de `!`*PARAM=value* em um arquivo `prototype` é local para tal arquivo e para a parte do pacote que ele define.

Se *PARAM* é uma variável de instalação e *variable* é uma variável de instalação ou de construção com um valor conhecido, o comando `pkgmk` insere a definição no arquivo `pkginfo` para que a definição esteja disponível no momento da instalação. No entanto, o comando `pkgmk` não avalia os *PARAM* que estão em quaisquer nomes de caminho especificados no arquivo `prototype`.

Resumo das variáveis de ambiente do pacote

A tabela seguinte resume o local, o escopo e os formatos de especificação das variáveis.

TABELA 2–2 Resumo das variáveis de ambiente do pacote

Onde a variável está definida	Formato de definição da variável	Tipo de variável que está sendo definida	Quando a variável é avaliada	Onde a variável é avaliada	Itens que podem substituir a variável
Arquivo `pkginfo`	*PARAM=value*	Construção	Ignorado no tempo de construção	N/D	Nenhum
Instalar	Tempo de instalação	No arquivo `pkgmap`	*owner*, *group*, *path* ou destino de link		
Arquivo `prototype`	`!`*PARAM=value*	Construção	Tempo de construção	No arquivo `prototype` e nos arquivos incluídos	*mode*, *owner*, *group* ou *path*
Instalar	Tempo de construção	No arquivo `prototype` e nos arquivos incluídos	Somente nos comandos `!search` e `!command`		
Linha de comando `pkgmk`	*PARAM=value*	Construção	Tempo de construção	No arquivo `prototype`	*mode*, *owner*, *group* ou *path*

TABELA 2–2 Resumo das variáveis de ambiente do pacote *(Continuação)*

Onde a variável está definida	Formato de definição da variável	Tipo de variável que está sendo definida	Quando a variável é avaliada	Onde a variável é avaliada	Itens que podem substituir a variável
Instalar	Tempo de construção	No arquivo `prototype`	Somente no `!search`		
Tempo de instalação	No arquivo `pkgmap`	*owner*, *group*, *path* ou destino de link			

Criando um arquivo `pkginfo`

O arquivo `pkginfo` é um arquivo ASCII que descreve as características de um pacote juntamente com informações que ajudam a controlar o fluxo de instalação.

Cada entrada do arquivo `pkginfo` é uma linha que estabelece o valor de um parâmetro usando o formato *PARAM=value*. O *PARAM* pode ser qualquer um dos parâmetros padrão descritos na página do manual `pkginfo(4)`. Não há uma ordem obrigatória na qual os parâmetros devem estar especificados.

Observação – Cada *value* pode estar entre aspas simples ou duplas (por exemplo, '*value*' ou "*value*"). Se *value* contiver caracteres considerados especiais para um ambiente shell, você deve usar aspas. Os exemplos e os estudos de caso deste livro não usam aspas. Consulte a página do manual `pkginfo(4)` para obter mais informações.

Você também pode criar seus próprios parâmetros de pacote atribuindo um valor a eles no arquivo `pkginfo`. Seus parâmetros devem começar com letra maiúscula seguida de letras maiúsculas ou minúsculas. Uma letra maiúscula indica que o parâmetro (variável) será avaliado no tempo de instalação (oposto ao tempo de construção). Para obter informações sobre a diferença entre as variáveis de instalação e de construção, consulte "Variáveis de ambiente do pacote" na página 24.

Observação – O espaço em branco final depois do valor do parâmetro é ignorado.

Você deve definir estes cinco parâmetros em um arquivo `pkginfo`: `PKG`, `NAME`, `ARCH`, `VERSION` e `CATEGORY`. Os parâmetros `PATH`, `PKGINST` e `INSTDATE` são inseridos automaticamente pelo software quando o pacote é construído. Não modifique estes oito parâmetros. Para obter informações sobre os parâmetros restantes, consulte a página do manual `pkginfo(4)`.

Definindo uma instância de pacote

O mesmo pacote pode apresentar diferentes versões, ser compatível com diferentes arquiteturas, ou ambos . Cada variação de um pacote é conhecida como uma *instância de pacote*. Uma instância de pacote é determinada pela combinação das definições dos parâmetros PKG, ARCH e VERSION no arquivo pkginfo.

O comando pkgadd atribui um *identificador de pacote* para cada instância de pacote no momento da instalação. O identificador de pacote é a abreviatura do pacote com um sufixo numérico, por exemplo, SUNWadm.2. Este identificador distingue uma instância de pacote de qualquer outro pacote, incluindo as instâncias do mesmo pacote.

Definindo uma abreviatura de pacote (PKG)

Uma *abreviatura de pacote* é um nome curto de um pacote definido pelo parâmetro PKG no arquivo pkginfo. A abreviatura de um pacote deve apresentar quatro características:

- A abreviatura deve estar formada por caracteres alfanuméricos. O primeiro caractere não pode ser um número.
- A abreviatura não pode ultrapassar 32 caracteres.
- A abreviatura não pode ser nenhuma das abreviaturas reservadas seguintes: install, new ou all.

Observação – Os primeiros quatro caracteres devem ser exclusivos da sua empresa. Por exemplo, os pacotes construídos pela Sun Microsystems apresentam "SUNW" com os quatro primeiros caracteres na abreviatura dos pacotes.

Um exemplo de entrada de abreviatura de pacote em um arquivo pkginfo é PKG=SUNWcadap.

Especificando uma arquitetura de pacote (ARCH)

O parâmetro ARCH no arquivo pkginfo identifica quais arquiteturas estão associadas ao pacote. O nome da arquitetura apresenta um máximo de 16 caracteres alfanuméricos. Se um pacote estiver associado a mais de uma arquitetura, especifique as arquiteturas em uma lista separa por vírgulas.

Abaixo se encontra um exemplo de uma especificação de arquitetura de pacote em um arquivo pkginfo:

```
ARCH=sparc
```

Especificando a arquitetura de um conjunto de instruções do pacote (SUNW_ISA)

O parâmetro SUNW_ISA em um arquivo pkginfo identifica qual arquitetura de conjunto de instruções está associada a um pacote da Sun Microsystems. Os valores são os seguintes:

- sparcv9, para um pacote que contém objetos de 64 bits
- sparc, para um pacote que contém objetos de 32 bits

Por exemplo, o valor SUNW_ISA em um arquivo pkginfo para um pacote que contém objetos de 64 bit pode ser:

```
SUNW_ISA=sparcv9
```

Se SUNW_ISA não estiver definido, a arquitetura de conjunto de instruções padrão do pacote está definida com o valor do parâmetro ARCH.

Especificando uma versão do pacote (VERSION)

O parâmetro VERSION no arquivo pkginfo identifica a versão do pacote. A versão tem um máximo de 256 caracteres ASCII e não pode começar com um parêntese esquerdo.

Abaixo se encontra um exemplo da versão de uma especificação em um arquivo pkginfo:

```
VERSION=release 1.0
```

Definindo o nome de um pacote (NAME)

Um *nome de pacote* é um nome completo do pacote, que é definido pelo parâmetro NAME no arquivo pkginfo.

É importante que os nomes de pacote sejam completos, claros e concisos porque os administradores de sistema usam geralmente os nomes de pacote para determinar se um pacote precisa ser instalado. Os nomes de pacote devem cumprir os seguintes critérios:

- Determinar quando o pacote é necessário (por exemplo, para fornecer determinados comandos e funcionalidades ou determinar se o pacote é necessário para um hardware específico).
- Determinar para que o pacote é usado (por exemplo, para o desenvolvimento de drivers de dispositivo).
- Incluir uma descrição do mnemônico da abreviatura do pacote, usando palavras-chave que indiquem que a abreviatura é uma forma abreviada da descrição. Por exemplo, o nome do pacote da abreviatura de pacote SUNWbnuu é “Basic Networking UUCP Utilities, (Usr)”.
- Nomear a partição na qual o pacote é instalado.
- Usar termos coerentes com significado no setor ao qual pertence.

- Aproveitar o limite de 256 caracteres.

Abaixo se encontra um exemplo de nome de pacote definido em um arquivo pkginfo:

```
NAME=Chip designers need CAD application software to design
abc chips.  Runs only on xyz hardware and is installed in the
usr partition.
```

Definindo a categoria de um pacote (CATEGORY)

O parâmetro CATEGORY no arquivo pkginfo especifica a que categorias pertence um pacote. Um pacote deve pertencer, como mínimo, à categoria system ou application. Os nomes da categoria estão formados por caracteres alfanuméricos. Os nomes da categoria têm um máximo de 16 caracteres e fazem distinção entre maiúsculas e minúsculas.

Se um pacote pertencer a mais de uma categoria, especifique as categorias em uma lista separada por vírgula.

Abaixo se encontra um exemplo da especificação CATEGORY em um arquivo pkginfo:

```
CATEGORY=system
```

▼ Como criar um arquivo pkginfo

1 **No seu editor de texto preferido, crie um arquivo nomeado pkginfo.**

Você pode criar este arquivo em qualquer local do seu sistema.

2 **Edite o arquivo e defina os cinco parâmetros necessários.**

Os cinco parâmetros necessários são: PKG, NAME, ARCH, VERSION e CATEGORY. Para obter mais informações sobre esses parâmetros, consulte "Criando um arquivo pkginfo" na página 26.

3 **Adicione quaisquer parâmetros opcionais ao arquivo.**

Crie seus próprios parâmetros ou consulte a página do manual pkginfo(4) para obter informações sobre os parâmetros padrão.

4 **Salve as alterações e saia do editor.**

Exemplo 2–1 Criando um arquivo pkginfo

Este exemplo mostra o conteúdo de um arquivo pkginfo válido, com os cinco parâmetros necessários, bem como o parâmetro BASEDIR. O parâmetro BASEDIR é tratado mais detalhadamente em "O campo *path*" na página 33.

```
PKG=SUNWcadap
NAME=Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.
ARCH=sparc
VERSION=release 1.0
CATEGORY=system
BASEDIR=/opt
```

Consulte também Consulte "Como organizar o conteúdo de um pacote" na página 30.

Organizando o conteúdo de um pacote

Organize os objetos do seu pacote em uma estrutura de diretório hierárquica que imita a estrutura que os objetos do pacote terão no sistema de destino após a instalação. Se realizar esta etapa antes de criar um arquivo `prototype`, você pode economizar tempo e energia ao criar tal arquivo.

▼ Como organizar o conteúdo de um pacote

1 **Determine quantos pacotes você precisa criar e quais objetos de pacote devem ser colocados em cada pacote.**

Para obter ajuda para realizar esta etapa, consulte "Considerações antes de construir um pacote" na página 17.

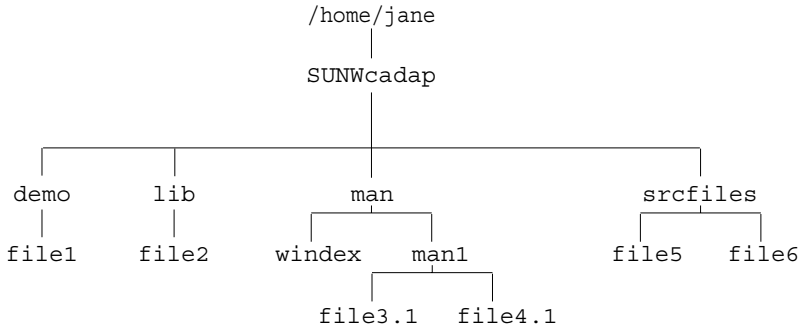
2 **Crie um diretório para cada pacote que você precisa construir.**

Você pode criar este diretório em qualquer local do sistema e nomeá-lo como quiser. Os exemplos deste capítulo pressupõem que o diretório de um pacote tem o mesmo nome da abreviatura do pacote.

```
$ cd /home/jane
$ mkdir SUNWcadap
```

3 **Organize os objetos de cada pacote em uma estrutura de diretório abaixo do diretório do pacote correspondente. A estrutura de diretório deve imitar a estrutura que os objetos de pacote terão no sistema de destino.**

Por exemplo, o pacote de aplicativo CAD, `SUNWcadap`, precisa da estrutura de diretório seguinte.



4 **Decida onde os arquivos de informação serão mantidos. Se apropriado, crie um diretório para manter os arquivos em um local.**

Este exemplo pressupõe que o arquivo `pkginfo` de "Como criar um arquivo `pkginfo`" na página 29 foi criado no diretório de início da Jane.

```
$ cd /home/jane
$ mkdir InfoFiles
$ mv pkginfo InfoFiles
```

Consulte também Consulte "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43.

Criando um arquivo `prototype`

O arquivo `prototype` é um arquivo ASCII usado para especificar informações sobre os objetos de um pacote. Cada entrada do arquivo `prototype` descreve um único objeto, tal como um arquivo de dados, diretório, arquivo de código-fonte ou objeto executável. As entradas de um arquivo `prototype` estão formadas por vários campos de informações separados por espaços em branco. Observe que os campos *devem* ser exibidos em uma ordem específica. As linhas de comentário começam com um símbolo de jogo da velha (#) e são ignoradas.

Você pode criar um arquivo `prototype` com o editor de texto ou usando o comando `pkgproto`. Quando você criar este arquivo pela primeira vez, é mais fácil fazê-lo com o comando `pkgproto`, porque ele cria o arquivo baseado na hierarquia do diretório criado anteriormente. Se seus arquivos não estiverem organizados conforme descrito em "Organizando o conteúdo de um pacote" na página 30, você tem a desagradável tarefa de criar o arquivo `prototype` do início com o editor de texto da sua preferência.

Formato do arquivo prototype

Abaixo se encontra o formato de cada linha no arquivo prototype:

partftypeclasspathmajorminormodeownergroup

part	É um campo numérico opcional que permite agrupar os objetos de pacote em partes. O valor padrão é 1 parte.
ftype	É um campo de um caractere que especifica o tipo do objeto. Consulte "O campo ftype" na página 32.
classe	É a classe de instalação à qual pertence o objeto. Consulte "O campo *class*" na página 33.
path	É o nome de caminho absoluto ou relativo que indica onde o objeto de pacote se localizará no sistema de destino. Consulte "O campo *path*" na página 33.
major	É o número principal de dispositivo dos dispositivos especiais de bloco ou de caracteres.
minor	É o número secundário de dispositivo dos dispositivos especiais de bloco ou de caracteres.
mode	É o modo octal do objeto (por exemplo, 0644). Consulte "O campo *mode*" na página 36.
owner	É o proprietário do objeto (por exemplo, bin ou root). Consulte "O campo *owner*" na página 36.
group	É o grupo ao qual pertence o objeto (por exemplo, bin ou sys). Consulte "O campo *group*" na página 37.

Em geral, somente os campos *ftype*, *class*, *path*, *mode*, *owner* e *group* são definidos. Estes campos estão descritos nas próximas seções. Consulte a página do manual prototype(4) para obter informações adicionais sobre estes campos.

O campo ftype

O campo *ftype*, ou file type, é um campo de um único caractere que especifica o tipo de arquivo de um objeto de pacote. Os tipos de arquivo válidos estão descritos na tabela abaixo

TABELA 2–3 Tipos de arquivo válidos no arquivo prototype

Valor do campo de tipo de arquivo	Descrição do tipo de arquivo
f	Arquivo executável ou de dados padrão

TABELA 2–3 Tipos de arquivo válidos no arquivo prototype *(Continuação)*

Valor do campo de tipo de arquivo	Descrição do tipo de arquivo
e	Arquivo a ser editado na instalação ou remoção (pode ser compartilhado por vários pacotes)
v	Arquivo volátil (cujo conteúdo se altera, tal como um arquivo log)
d	Diretório
x	Diretório exclusivo acessível somente por este pacote (pode conter informações de banco de dados e logs não registrados)
l	Arquivo vinculado
p	Pipe nomeado
c	Dispositivo especial de caracteres
b	Dispositivo especial de bloco
i	Arquivo de informação ou script de instalação
s	Link Simbólico

O campo *class*

class nomeia a classe à qual pertence um objeto. O uso de classes é um recurso opcional de criação de pacote. Este recurso é tratado detalhadamente em "Escrevendo scripts de ação de classe" na página 70.

Se você não usar classes, o objeto pertence à classe `none`. Quando o comando `pkgmk` é executado para construir o pacote, ele insere o parâmetro `CLASSES=none` no arquivo `pkginfo`. Os arquivos com tipo de arquivo `i` devem ter um campo *class* em branco.

O campo *path*

O campo *path* é usado para definir onde o objeto de pacote se localizará no sistema de destino. Você pode indicar o local tanto com um nome de caminho absoluto (por exemplo, `/usr/bin/mail`) quanto com um nome de caminho relativo (por exemplo, `bin/mail`). O uso de um nome de caminho absoluto significa que o local do objeto no sistema de destino é definido pelo pacote e não pode ser alterado. Os objetos de pacote com nomes de caminho relativo indicam que o objeto é *relocável*.

Um *objeto relocável* não precisa de um local de caminho absoluto no sistema de destino. Em vez disso, o local do objeto é determinado durante o processo de instalação.

Todos ou alguns objetos de pacote podem ser definidos como relocáveis. Antes de escrever os scripts de instalação ou criar o arquivo `prototype`, decida se os objetos de pacote terão locais fixos (tais como scripts de início em `/etc`) ou serão relocáveis.

Há dois tipos de objetos relocáveis, *relocável coletivamente* e *relocável individualmente*.

Objetos Relocáveis Coletivamente

Os objetos relocáveis coletivamente são colocados em relação à base de instalação comum chamada *diretório base*. O diretório base é definido no arquivo pkginfo, usando o parâmetro BASEDIR. Por exemplo, um objeto relocável no arquivo prototype denominado tests/generic requer que o arquivo pkginfo defina o parâmetro padrão BASEDIR. Por exemplo:

```
BASEDIR=/opt
```

Este exemplo significa que quando o objeto for instalado, ele será colocado no diretório /opt/tests/generic.

Observação – O diretório /opt é o único diretório em que o software que não é parte do software Oracle Solaris base pode ser especificado.

Use objetos relocáveis coletivamente sempre que possível. Em geral, a parte principal de um pacote pode ser relocável com poucos arquivos (tais como os arquivos em /etc ou /var) especificados como absoluto. No entanto, se um pacote contém várias relocações, leve em consideração dividir o pacote em vários pacotes com valores BASEDIR diferentes nos seus arquivos pkginfo.

Objetos Relocáveis Individualmente

Os objetos relocáveis individualmente não estão limitados ao mesmo local de diretório que os objetos relocáveis coletivamente. Para definir um objeto relocável individualmente, você precisa especificar uma variável de instalação no campo *path* do arquivo prototype. Depois de especificar a variável de instalação, crie um script request para solicitar ao instalador o diretório base relocável, ou um script checkinstall para determinar o nome do caminho dos dados do sistema de arquivos. Para obter mais informações sobre scripts request, consulte "Escrevendo um script request" na página 63 e sobre os scripts checkinstall, consulte "Como coletar dados do sistema de arquivos" na página 67.

Cuidado – Os objetos relocáveis indivualmente são difíceis de gerenciar. O uso de objetos relocáveis individualmente pode fazer com que os componentes de pacote fiquem muito dispersos e seja difícil isolá-los ao instalar várias versões ou arquiteturas do pacote. Use objetos relocáveis coletivamente sempre que possível.

Nomes de caminho paramétrico

Um *nome de caminho paramétrico* é um nome de caminho que inclui uma especificação de variável. Por exemplo, o *filename* /opt/$PKGINST/ é um nome de caminho paramétrico devido

à especificação de variável $PKGINST. O valor padrão de uma especificação de variável *deve* ser definido no arquivo pkginfo. O valor pode, então, ser alterado por um script request ou um script checkinstall.

Uma especificação de variável em um caminho deve começar ou terminar o nome de caminho ou estar limitada por barras (/). Os nomes de caminho paramétrico apresentam a seguinte forma:

```
$PARAM/tests
tests/$PARAM/generic
/tests/$PARAM
```

A especificação de variável, uma vez definida, pode fazer com que o caminho seja interpretado como absoluto ou relocável. No exemplo seguinte, o arquivo prototype contém esta entrada:

```
f none $DIRLOC/tests/generic
```

O arquivo pkginfo contém esta entrada:

```
DIRLOC=/myopt
```

O nome de caminho $DIRLOC/tests/generic é interpretado como o nome de caminho absoluto /myopt/tests/generic, apesar do parâmetro BASEDIR estar definido no arquivo pkginfo.

Neste exemplo, o arquivo prototype é igual ao do exemplo anterior e o arquivo pkginfo contém as seguintes entradas:

```
DIRLOC=firstcut
BASEDIR=/opt
```

O nome de caminho $DIRLOC/tests/generic será interpretado como o nome de caminho relocável /opt/firstcut/tests/generic.

Para obter mais informações sobre nomes de caminho paramétrico, consulte "Usando diretórios base paramétricos" na página 130.

Os locais de origem e de destino de um objeto em poucas palavras

O campo *path* no arquivo prototype define onde o objeto será colocado no sistema de destino. Especifique o local atual dos objetos de pacote no arquivo prototype se a estrutura de diretório não imitar a estrutura pretendida no sistema de destino. Consulte "Organizando o conteúdo de um pacote" na página 30 para obter mais informações sobre a estruturação de objetos em um pacote.

Se a sua área de desenvolvimento não tiver a mesma estrutura que deseja que o seu pacote tenha, você pode usar o formato *path1=path2* no arquivo *path*. Neste formato, *path1* é o local que o objeto deve ter no sistema de destino e *path2* é o local que o objeto tem no seu sistema.

Você também pode usar o formato de nome de caminho *path1=path2* com *path1* como nome de objeto relocável e *path2* como um nome de caminho completo de tal objeto no seu sistema.

Observação – *path1* não pode conter variáveis de construção não definidas, mas pode conter variáveis de instalação não definidas. *path2* não pode conter nenhuma variável não definida, embora ambas variáveis de construção e instalação possam ser usadas. Para obter informações sobre a diferença entre as variáveis de instalação e de construção, consulte "Variáveis de ambiente do pacote" na página 24.

Os links devem usar o formato *path1= path2* porque são criados pelo comando `pkgadd`. Como regra geral, o *path2* de um link nunca deve ser absoluto, mas, pelo contrário, deve ser relativo à parte do diretório de *path1*.

Uma opção para o uso do formato *path1=path2* é usar o comando `!search`. Para obter mais informações, consulte "Fornecendo um caminho de pesquisa para o comando `pkgmk`" na página 42.

O campo *mode*

O campo *mode* pode conter um número octal, um ponto de interrogação (`?`) ou uma especificação de variável. O número octal especifica o modo do objeto quando este é instalado no sistema de destino. O `?` significa que o modo não será alterado conforme o objeto é instalado, implicando que um objeto com o mesmo nome já existe no sistema de destino.

A especificação de variável da forma `$`*mode*, na qual a primeira letra da variável deve estar em minúscula, significa que este campo será definido conforme o pacote é construído. Observe que esta variável deve ser definida no tempo de construção no arquivo `prototype` ou como uma opção do comando `pkgmk`. Para obter informações sobre a diferença entre as variáveis de instalação e de construção, consulte "Variáveis de ambiente do pacote" na página 24.

Os arquivos com tipo de arquivo `i` (arquivo de informação), `l` (link de disco rígido) e `s` (link simbólico) devem deixar este campo em branco.

O campo *owner*

O campo *owner* pode conter um nome de usuário, um ponto de interrogação (`?`) ou uma especificação de variável. Um nome de usuário possui no máximo 14 caracteres e deve ser um nome que já exista no sistema de destino (tal como `bin` ou `root`). O `?` significa que o proprietário não será alterado conforme o objeto é instalado, implicando que um objeto com o mesmo nome já existe no sistema de destino.

Uma especificação de variável pode ter a forma `$`*Owner* ou `$`*owner*, na qual a primeira letra da variável é maiúscula ou minúscula. Se a variável começar com uma letra minúscula, ela deve ser definida conforme o pacote é construído, tanto no arquivo `prototype` quanto como uma opção

do comando pkgmk. Se a variável começar com uma letra maiúscula, a especificação de variável será inserida no arquivo pkginfo como um valor padrão e pode ser redefinida no tempo de instalação por um script request. Para obter informações sobre a diferença entre as variáveis de instalação e de construção, consulte "Variáveis de ambiente do pacote" na página 24.

Os arquivos com tipo de arquivo i (arquivo de informação) e l b (link de disco rígido) devem deixar este campo em branco.

O campo *group*

O campo *group* pode conter um nome de grupo, um ponto de interrogação (?) ou uma especificação de variável. Um nome de grupo possui no máximo 14 caracteres e deve ser um nome que já exista no sistema de destino (tal como bin ou sys). O ? significa que o grupo não será alterado conforme o objeto é instalado, implicando que um objeto com o mesmo nome já existe no sistema de destino.

Uma especificação de variável pode ter a forma $*Group* ou $*group*, na qual a primeira letra da variável é maiúscula ou minúscula. Se a variável começar com uma letra minúscula, ela deve ser definida conforme o pacote é construído, tanto no arquivo prototype quanto como uma opção do comando pkgmk. Se a variável começar com uma letra maiúscula, a especificação de variável será inserida no arquivo pkginfo como um valor padrão e pode ser redefinida no tempo de instalação por um script request. Para obter informações sobre a diferença entre as variáveis de instalação e de construção, consulte "Variáveis de ambiente do pacote" na página 24.

Os arquivos com tipo de arquivo i (arquivo de informação) e l (link de disco rígido) devem deixar este campo em branco.

Criando um arquivo prototype desde o início

Se quiser criar um arquivo prototype desde o início, você pode fazê-lo com o editor de texto da sua preferência, adicionando uma entrada por objeto de pacote. Consulte "Formato do arquivo prototype" na página 32 e a página do manual prototype(4) para obter mais informações sobre o formato deste arquivo. No entanto, após definir cada objeto de pacote, você pode querer incluir alguns dos recursos descritos em "Adicionando funcionalidade a um arquivo prototype" na página 40.

Exemplo — Criando um arquivo prototype com o comando pkgproto

Você pode usar o comando pkgproto para construir um arquivo prototype básico, contanto que tenha organizado a esturtura de diretório do pacote conforme descrito em "Organizando o conteúdo de um pacote" na página 30. Por exemplo, usando a estrutura de diretório de amostra e o arquivo pkginfo descrito nas seções anteriores, os comandos para a criação do arquivo prototype são:

```
$ cd /home/jane
$ pkgproto ./SUNWcadap > InfoFiles/prototype
```

O arquivo prototype é semelhante a:

```
d none SUNWcadap 0755 jane staff
d none SUNWcadap/demo 0755 jane staff
f none SUNWcadap/demo/file1 0555 jane staff
d none SUNWcadap/srcfiles 0755 jane staff
f none SUNWcadap/srcfiles/file5 0555 jane staff
f none SUNWcadap/srcfiles/file6 0555 jane staff
d none SUNWcadap/lib 0755 jane staff
f none SUNWcadap/lib/file2 0644 jane staff
d none SUNWcadap/man 0755 jane staff
f none SUNWcadap/man/windex 0644 jane staff
d none SUNWcadap/man/man1 0755 jane staff
f none SUNWcadap/man/man1/file4.1 0444 jane staff
f none SUNWcadap/man/man1/file3.1 0444 jane staff
```

Observação – O proprietário e o grupo real da pessoa que está construindo o pacote são registrados pelo comando pkgproto. Uma boa técnica é usar os comandos chown -R e chgrp -R, definindo o proprietário e o grupo como desejado *antes* executando o comando pkgproto.

Este arquivo prototype de exemplo não está completo. Consulte a seção seguinte para obter informações sobre a conclusão deste arquivo.

Ajustando um arquivo prototype criado com o comando pkgproto

Embora o comando pkgproto seja útil na criação de um arquivo prototype inicial, ele não cria entradas para cada objeto de pacote que precisa ser definido. Este comando não cria entradas completas. O comando pkgproto não realiza nenhuma das seguintes ações:

- Criar entradas completas para objetos com tipos de arquivo v (arquivos voláteis), e (arquivos editáveis), x (diretórios exclusivos), ou i (arquivos de informação ou scripts de instalação)
- Oferecer suporte a várias classes com uma única chamada

Criando entradas de objeto com tipos de arquivo v, e, x e i

Por último, você precisa modificar o arquivo prototype para adicionar objetos com o tipo de arquivo i. Se você armazenou os arquivos de informação e os scripts de instalação no primeiro nível do diretório do seu pacote (por exemplo, /home/jane/SUNWcadap/pkginfo), a entrada de um arquivo prototype seria semelhante a:

```
i pkginfo
```

Se não armazenou os arquivos de informação e scripts de instalação no primeiro nível do diretório do seu pacote, então você precisa especificar o local de origem destes. Por exemplo:

```
i pkginfo=/home/jane/InfoFiles/pkginfo
```

Ou você pode usar o comando `!search` para especificar o local do comando `pkgmk` a ser pesquisado ao construir o pacote. Consulte "Fornecendo um caminho de pesquisa para o comando pkgmk" na página 42 para obter mais informações.

Para adicionar entradas dos objetos com tipos de arquivo `v`, `e` e `x`, siga o formato descrito em "Formato do arquivo prototype" na página 32 ou consulte a página do manual `prototype(4)`.

Observação – Não se esqueça de atribuir sempre uma classe aos arquivos com um tipo de arquivo `e` (editável) e tenha um script de ação de classe associado a tal classe. Do contrário, os arquivos serão removidos durante a remoção do pacote, mesmo que o nome de caminho seja compartilhado com outros pacotes.

Usando várias definições de classe

Se usar o comando `pkgproto` para criar o arquivo `prototype` básico, você pode atribuir todos os objetos de pacote à classe `none` ou a uma classe específica. Conforme mostrado no "Exemplo — Criando um arquivo prototype com o comando pkgproto" na página 37, o comando `pkgproto` básico atribui todos os objetos à classe `none`. Para atribuir todos os objetos a uma classe específica, você pode usar a opção `-c`. Por exemplo:

```
$ pkgproto -c classname /home/jane/SUNWcadap > /home/jane/InfoFiles/prototype
```

Se usar várias classes, você pode precisar editar manualmente o arquivo `prototype` e modificar o campo *class* de cada objeto. Se usar classes, você também precisa definir o parâmetro `CLASSES` no arquivo `pkginfo` e scripts de ação de classe. O uso de classe é um recurso opcional e é tratado detalhadamente em "Escrevendo scripts de ação de classe" na página 70.

Exemplo — Ajustando um arquivo `prototype` criado usando o comando `pkgproto`

Dado o arquivo `prototype` criado pelo comando `pkgproto` em "Exemplo — Criando um arquivo prototype com o comando pkgproto" na página 37, várias modificações precisam ser feitas.

- É necessária uma entrada para o arquivo `pkginfo`.
- Os campos *path* precisam ser alterados para o formato *path1=path2* porque o código-fonte do pacote está em `/home/jane`. Visto que o código-fonte do pacote está em um diretório hierárquico e que o comando `!search` não realiza pesquisa recursiva, pode ser mais fácil usar o formato *path1=path2*.

- Os campos *owner* e *group* devem conter os nomes de usuários e grupos existentes no sistema de destino. Isto é, o proprietário jane causará um erro porque este proprietário não faz parte do sistema operacional SunOS.

  O arquivo prototype modificado se assemelha a:

```
i pkginfo=/home/jane/InfoFiles/pkginfo
d none SUNWcadap=/home/jane/SUNWcadap 0755 root sys
d none SUNWcadap/demo=/home/jane/SUNWcadap/demo 0755 root bin
f none SUNWcadap/demo/file1=/home/jane/SUNWcadap/demo/file1 0555 root bin
d none SUNWcadap/srcfiles=/home/jane/SUNWcadap/srcfiles 0755 root bin
f none SUNWcadap/srcfiles/file5=/home/jane/SUNWcadap/srcfiles/file5 0555 root bin
f none SUNWcadap/srcfiles/file6=/home/jane/SUNWcadap/srcfiles/file6 0555 root bin
d none SUNWcadap/lib=/home/jane/SUNWcadap/lib 0755 root bin
f none SUNWcadap/lib/file2=/home/jane/SUNWcadap/lib/file2 0644 root bin
d none SUNWcadap/man=/home/jane/SUNWcadap/man 0755 bin bin
f none SUNWcadap/man/windex=/home/jane/SUNWcadap/man/windex 0644 root other
d none SUNWcadap/man/man1=/home/jane/SUNWcadap/man/man1 0755 bin bin
f none SUNWcadap/man/man1/file4.1=/home/jane/SUNWcadap/man/man1/file4.1 0444 bin bin
f none SUNWcadap/man/man1/file3.1=/home/jane/SUNWcadap/man/man1/file3.1 0444 bin bin
```

Adicionando funcionalidade a um arquivo prototype

Além de definir cada objeto de pacote no arquivo prototype, você também pode:

- Definir objetos adicionais a serem criados no tempo de instalação.
- Criar links no tempo de instalação.
- Distribuir pacotes em vários volumes.
- Aninhar arquivos prototype.
- Definir um valor padrão para os campos *mode*, *owner* e *group*.
- Fornecer um caminho de pesquisa para o comando pkgmk.
- Definir variáveis de ambiente.

Consulte as seções seguintes para obter informações sobre a realização destas alterações.

Definindo objetos adicionais a serem criados no tempo de instalação

Você pode usar o arquivo prototype para definir objetos que ainda não foram entregues ao meio de instalação. Durante a instalação, usando o comando pkgadd, estes objetos são criados com os tipos de arquivo requeridos, se ainda não existirem no momento da instalação.

Para especificar que um objeto seja criado no sistema de destino, adicione uma entrada para tal objeto no arquivo prototype com o tipo de arquivo apropriado.

Por exemplo, se você quiser que um diretório seja criado no sistema de destino, mas não quiser que seja entregue no meio de instalação, adicione a seguinte entrada para o diretório no arquivo prototype:

```
d none /directory 0644 root other
```

Se quiser criar um arquivo vazio no sistema de destino, a entrada do arquivo no arquivo `prototype` pode ser semelhante a:

```
f none filename=/dev/null 0644 bin bin
```

Os únicos objetos que *devem* ser entregues no meio de instalação são os arquivos regulares e scripts de edição (tipos de arquivo `e`, `v`, `f`), e os diretórios necessários para contê-los. Todos os objetos adicionais são criados sem referência a objetos, diretórios, pipes nomeados, dispositivos, links de disco rígido e links simbólicos entregues.

Criando links no tempo de instalação

Para criar links durante a instalação do pacote, defina o seguinte na entrada do arquivo `prototype` do objeto vinculado:

- Seu tipo de arquivo como `l` (um link) ou `s` (um link simbólico).
- O nome de caminho do objeto vinculado com o formato *path1=path2*, no qual *path1* é o arquivo de destino e *path2* é o arquivo de origem. Como regra geral, o *path2* de um link nunca deve ser absoluto, mas, pelo contrário, deve ser relativo à parte do diretório de *path1*. Por exemplo, a entrada de um arquivo `prototype` que define um link simbólico poderia ser semelhante a:

  ```
  s none etc/mount=../usr/etc/mount
  ```

Os links relativos seriam especificados desta forma se o pacote estivesse instalado como absoluto ou relocável.

Distribuindo pacotes em vários volumes

Ao construir o pacote com o comando `pkgmk`, este realiza os cálculos e as ações necessárias para organizar um pacote com vários volumes. Um pacote com vários volumes é chamado de *pacote segmentado*.

No entanto, você pode usar o campo *part* opcional no arquivo `prototype` para definir em que parte quer que o objeto seja colocado. Um número neste comando substitui o comando `pkgmk` e força a colocação do componente na parte dada no campo. Observe que há correspondência individual entre as partes e os volumes da mídia removível formatada como sistemas de arquivos. Se os volumes forem pré-atribuídos pelo desenvolvedor, o comando `pkgmk` publica um erro se não houver espaço suficiente em nenhum volume.

Aninhando arquivos `prototype`

Você pode criar vários arquivos `prototype` e, em seguida, incluí-los usando o comando `!include` no arquivo `prototype`. Você pode querer aninhar os arquivos para facilitar a manutenção.

No exemplo seguinte há três arquivos `prototype.` O arquivo principal (`prototype`) está sendo editado. Os outros dois arquivos (`proto2` e `proto3`) estão sendo incluídos.

```
!include /source-dir/proto2
!include /source-dir/proto3
```

Definindo valores padrão para os campos *mode*, *owner* e *group*

Para definir valores padrão para os campos *mode*, *owner* e *group* de objetos específicos, você pode inserir o comando `!default` no arquivo `prototype`. Por exemplo:

```
!default 0644 root other
```

Observação – O intervalo do comando `!padrão` começa onde ele foi inserido e vai até o final do arquivo. O intervalo do comando não se estende sobre os arquivos incluídos.

No entanto, nos diretórios (tipo de arquivo `d`) e arquivos editáveis (tipo de arquivo `e`) que você sabe que existem no sistema de destino (como `/usr` ou `/etc/vfstab`), certifique-se de que os campos *mode*, *owner* e *group* no arquivo `prototype` estejam definidos como pontos de interrogação (`?`). Desta forma você não destruirá as configurações existentes que podem ter sido modificadas por um administrador.

Fornecendo um caminho de pesquisa para o comando `pkgmk`

Se o local de origem dos objetos de pacote é diferente do local de destino e não quiser usar o formato *path1=path2* descrito em "Os locais de origem e de destino de um objeto em poucas palavras" na página 35, você pode usar o comando `!search` no arquivo `prototype`.

Por exemplo, se você tiver criado um diretório, `pkgfiles` , no diretório de início, e ele contiver todos os arquivos de informação e scripts de instalação, você pode especificar que este diretório seja pesquisado quando o pacote for construído com o comando `pkgmk`.

O comando no arquivo prototype pode ser semelhante a:

```
!search /home-dir/pkgfiles
```

Observação – As solicitações de pesquisa não se estendem aos arquivos incluídos. Além disso, a pesquisa se limita aos diretórios específicos listados e não realiza pesquisa recursiva.

Configurando variáveis de ambiente

Você também pode adicionar comandos ao arquivo `prototype` com a forma `!`*PARAM=value*. Os comandos com esta forma definem as variáveis no ambiente atual. Se você tem vários arquivos `prototype`, o escopo deste comando é o local do arquivo `prototype` no qual ele está definido.

A variável *PARAM* pode começar com letra minúscula ou maiúscula. Se o valor da variável *PARAM* não for conhecido no tempo de construção, o comando `pkgmk` aborta com um erro.

Para obter mais informações sobre a diferença entre as variáveis de instalação e de construção, consulte "Variáveis de ambiente do pacote" na página 24.

▼ Como criar um arquivo `prototype` usando o comando `pkgproto`

Observação – É mais fácil criar arquivos de informação e scripts de instalação antes de criar o arquivo `prototype`. No entanto, não é necessário seguir esta ordem. O arquivo `prototype` pode ser editado sempre depois que o conteúdo do pacote for alterado. Para obter mais informações sobre arquivos de informação e scripts de instalação, consulte Capítulo 3, "Melhorando a funcionalidade de um pacote (Tarefas)".

1 **Determine quais objetos de pacote serão absolutos e quais serão relocáveis, se ainda não tiver determinado.**

Para obter mais informações que o ajudem a concluir esta etapa, consulte "O campo *path*" na página 33.

2 **Organize os objetos do seu pacote a fim de imitar o local destes no sistema de destino.**

Se já tiver organizado seus pacotes conforme descrito em "Organizando o conteúdo de um pacote" na página 30, observe que pode ser necessário fazer algumas alterações baseadas nas decisões tomadas na Etapa 1. Se ainda não tiver organizado seu pacote, você deve fazê-lo agora. Se você não organizar seu pacote, não será possível usar o comando `pkgproto` para criar o arquivo `prototype` básico.

3 **Se o seu pacote tiver objetos relocáveis coletivamente, edite o arquivo `pkginfo` para definir o parâmetro `BASEDIR` com o valor apropriado.**

Por exemplo:

```
BASEDIR=/opt
```

Para obter informações sobre objetos relocáveis coletivamente, consulte "Objetos Relocáveis Coletivamente" na página 34.

4 **Se o seu pacote tiver objetos relocáveis individualmente, crie um script `request` para solicitar ao instalador o nome de caminho apropriado. Outra alternativa é criar um script `checkinstall` para determinar o nome de caminho apropriado dos dados do sistema de arquivos.**

A lista seguinte oferece os números de páginas para consulta em relação às tarefas comuns:

- Para criar um script `request`, consulte "Como escrever um script `request`" na página 64.
- Para criar um script `checkinstall`, consulte "Como coletar dados do sistema de arquivos" na página 67.

- Para obter mais informações sobre objetos relocáveis individualmente, consulte "Objetos Relocáveis Individualmente" na página 34.

5 Altere o proprietário e o grupo em todos os componentes do seu pacote que serão o proprietário e o grupo no sistema de destino.

Use os comandos `chown -R` e `chgrp -R` no diretório do seu pacote e no diretório de arquivos de informação.

6 Execute o comando `pkgproto` para criar um arquivo `prototype` básico.

O comando `pkgproto` escaneia os diretórios para criar um arquivo básico. Por exemplo:

```
$ cd package-directory
$ pkgproto ./package-directory > prototype
```

O arquivo `prototype` pode ser colocado em qualquer local do sistema. Simplifica o acesso e a manutenção manter os arquivos de informação e os scripts de instalação em um único local. Para obter informações adicionais sobre o comando `pkgproto`, consulte a página do manual `pkgproto(1)`.

7 Edite o arquivo `prototype` usando o editor de texto da sua preferência e adicione entradas para os arquivos de tipo `v`, `e`, `x` e `i`.

Para obter informações sobre alterações específicas que talvez precisem ser feitas, consulte "Ajustando um arquivo `prototype` criado com o comando `pkgproto`" na página 38.

8 (Opcional) Se estiver usando várias classes, edite os arquivos `prototype` e `pkginfo`. Use o editor de texto de sua preferência para fazer as alterações necessárias e crie os scripts de ação de classe correspondentes.

Para obter informações sobre alterações específicas que talvez precisem ser feitas, consulte "Ajustando um arquivo `prototype` criado com o comando `pkgproto`" na página 38 e "Escrevendo scripts de ação de classe" na página 70.

9 Edite o arquivo `prototype` usando o editor de texto de sua preferência para redefinir os nomes de caminho e alterar as configurações de outro campo.

Para obter mais informações, consulte "Ajustando um arquivo `prototype` criado com o comando `pkgproto`" na página 38.

10 (Opcional) Edite o arquivo `prototype` usando o editor de texto de sua preferência para adicionar funcionalidade ao seu arquivo `prototype`.

Para obter mais informações, consulte "Adicionando funcionalidade a um arquivo `prototype`" na página 40.

11 Salve as alterações e saia do editor.

Consulte também Se você já estiver preparado para ir para a próxima tarefa, consulte "Como construir um pacote" na página 46.

Construindo um Pacote

Use o comando `pkgmk` para construir seu pacote. O comando `pkgmk` realiza as seguintes tarefas:

- Coloca todos os objetos definidos do arquivo `prototype` no formato de diretório.
- Cria o arquivo `pkgmap`, que substitui o arquivo `prototype`.
- Cria um pacote instalável que é usado como entrada do comando `pkgadd`.

Usando o comando `pkgmk` mais simples

A forma mais simples deste comando é o comando `pkgmk` sem opções. Antes de usar o comando `pkgmk` sem opções, certifique-se de que o diretório de trabalho atual contém o arquivo `prototype` do pacote. A saída do comando, os arquivos e os diretórios estão gravados no diretório `/var/spool/pkg`.

O arquivo `pkgmap`

Ao construir um pacote com o comando `pkgmk`, ele cria um arquivo `pkgmap` que substitui o arquivo `prototype`. O arquivo `pkgmap` do exemplo anterior apresenta o conteúdo seguinte:

```
$ more pkgmap
: 1 3170
1 d none SUNWcadap 0755 root sys
1 d none SUNWcadap/demo 0755 root bin
1 f none SUNWcadap/demo/file1 0555 root bin 14868 45617 837527496
1 d none SUNWcadap/lib 0755 root bin
1 f none SUNWcadap/lib/file2 0644 root bin 1551792 62372 837527499
1 d none SUNWcadap/man 0755 bin bin
1 d none SUNWcadap/man/man1 0755 bin bin
1 f none SUNWcadap/man/man1/file3.1 0444 bin bin 3700 42989 837527500
1 f none SUNWcadap/man/man1/file4.1 0444 bin bin 1338 44010 837527499
1 f none SUNWcadap/man/windex 0644 root other 157 13275 837527499
1 d none SUNWcadap/srcfiles 0755 root bin
1 f none SUNWcadap/srcfiles/file5 0555 root bin 12208 20280 837527497
1 f none SUNWcadap/srcfiles/file6 0555 root bin 12256 63236 837527497
1 i pkginfo 140 10941 837531104
$
```

O formato deste arquivo é muito semelhante ao formato do arquivo `prototype`. No entanto, o arquivo `pkgmap` inclui a seguinte informação:

- A primeira linha indica o número de volumes que o pacote abrange, e o tamanho aproximado que o pacote terá quando estiver instalado.

Por exemplo, `: 1 3170` indica que o pacote abrange um volume e usará aproximadamente 3170 blocos de 512 bytes quando estiver instalado.

- Há três campos adicionais que definem o tamanho, a soma de verificação e tempo de modificação de cada objeto de pacote.
- Os objetos de pacote estão listados em ordem alfabética por classe e por nome de caminho para diminuir o tempo de instalação do pacote.

▼ Como construir um pacote

1 Crie um arquivo `pkginfo`, se ainda não tiver criado.

Para obter instruções detalhadas, consulte "Como criar um arquivo `pkginfo`" na página 29.

2 Crie um arquivo `prototype`, se ainda não tiver criado.

Para obter instruções detalhadas, consulte "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43.

3 Torne o diretório de trabalho atual o mesmo diretório que contém o arquivo `prototype` do pacote.

4 Construa o pacote.

```
$ pkgmk [-o] [-a arch] [-b base-src-dir] [-d device]
   [-f filename] [-l limit] [-p pstamp] [-r rootpath]
   [-v version] [PARAM=value] [pkginst]
```

`-o`	Substitui a versão existente do pacote.
`-a` *arch*	Ignora as informações da arquitetura do arquivo `pkginfo`.
`-b` *base-src-dir*	Solicita que *base-src-dir* seja adicionado ao início dos caminhos de nome relocáveis quando o comando `pkgmk` estiver procurando objetos no sistema de desenvolvimento.
`-d` *device*	Especifica que o pacote deve ser copiado em *device*, que pode ser um nome de caminho de diretório absoluto, um disquete ou um disco removível.
`-f` *filename*	Nomeia um arquivo, *filename*, que é usado como seu arquivo `prototype`. Os nomes padrão são `prototype` ou `Prototype`.
`-l` *limit*	Especifica o tamanho máximo, em blocos de 512 bytes, do dispositivo de saída.
`-p` *pstamp*	Ignora a definição do carimbo de produção no arquivo `pkginfo`.
`-r` *rootpath*	Solicita que o diretório raiz *rootpath* seja usado para situar os objetos no sistema de destino.

`-v` *version*	Ignora as informações da versão do arquivo `pkginfo`.
PARAM=value	Define as variáveis de ambiente globais. As variáveis que começam com letra minúscula são resolvidas no tempo de construção. As variáveis que começam com letra maiúscula são colocadas no arquivo `pkginfo` para serem usadas no tempo de instalação.
pkginst	Especifica um pacote por sua abreviatura ou uma instância específica (por exemplo, `SUNWcadap.4`).

Para obter mais informações, consulte a página do manual `pkgmk(1)`.

5 Verifique o conteúdo do pacote.

```
$ pkgchk -d device-name pkg-abbrev
Checking uninstalled directory format package pkg-abbrev
from device-name
## Checking control scripts.
## Checking package objects.
## Checking is complete.
$
```

`-d` *device-name*	Especifique o local do pacote. Observe que *device-name* pode ser o nome completo do caminho de um diretório ou os identificadores de uma fita ou um disco removível.
pkg-abbrev	É o nome de um ou mais pacotes (separado por espaços) que serão verificados. Se for omitido, o comando `pkgchk` verifica todos os pacotes disponíveis.

O comando `pkgchk` imprime que aspectos do pacote estão sendo verificados e exibe avisos e erros, conforme apropriado. Para obter mais informações sobre o comando `pkgchk`, consulte "Verificando a integridade de um pacote" na página 90.

Cuidado – Os erros devem ser considerados muito seriamente. Um erro pode significar que um script precisa ser corrigido. Verifique todos os erros e faça alterações se discordar da saída do comando `pkgchk`.

Exemplo 2–2 Construindo um Pacote

Este exemplo usa o arquivo `prototype` criado em "Ajustando um arquivo `prototype` criado com o comando `pkgproto`" na página 38.

```
$ cd /home/jane/InfoFiles
$ pkgmk
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system990708093144"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
```

```
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
$
```

Exemplo 2–3 Especificando o diretório de origem dos arquivos relocáveis

Se o seu pacote contiver arquivos relocáveis, você pode usar a opção `-b` *base-src-dir* no comando `pkgmk` para especificar um nome de caminho a ser adicionado ao início dos nomes de caminho relocáveis durante a criação do pacote. Esta opção é útil se você não tiver usado o formato *path1=path2* em arquivos relocáveis ou não tiver sido especificado um caminho de pesquisa com o comando `!search` no arquivo `prototype`.

O comando seguinte constrói um pacote com as seguintes características:

- O pacote é construído usando o arquivo `prototype` de amostra criado pelo comando `pkgproto`. Consulte "Exemplo — Criando um arquivo `prototype` com o comando `pkgproto`" na página 37 para obter mais informações.
- O pacote é construído sem que os campos *path* sejam modificados.
- O pacote adiciona uma entrada para o arquivo `pkginfo`.

```
$ cd /home/jane/InfoFiles
$ pkgmk -o -b /home/jane
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter  set to "system960716102636"
WARNING: parameter  set to "none"
## Attempting to volumize 13 entries in pkgmap.
part  1 -- 3170 blocks, 17 entries
## Packaging one part.
/var/spool/pkg/SUNWcadap/pkgmap
/var/spool/pkg/SUNWcadap/pkginfo
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/demo/file1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/lib/file2
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file3.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/man1/file4.1
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/man/windex
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file5
/var/spool/pkg/SUNWcadap/reloc/SUNWcadap/srcfiles/file6
## Validating control scripts.
## Packaging complete.
```

Nesse exemplo, o pacote é construído no diretório padrão, `/var/spool/pkg` , especificando a opção `-o`. Esta opção substitui o pacote criado no Exemplo 2–2.

Exemplo 2–4 Especificando diretórios de origem diferentes para arquivos de informação e objetos de pacote

Se colocar os arquivos de informação (como `pkginfo` e `prototype`) e os objetos de pacote em dois diretórios diferentes, você pode criar seu pacote usando as opções `-b` *base-src-dir* e `-r` *rootpath* no comando `pkgmk`. Se tiver os objetos de pacote em um diretório chamado `/product/pkgbin` e os arquivos de informação do pacote em um diretório chamado `/product/pkgsrc`, você pode usar o comando seguinte para colocar o pacote no diretório `/var/spool/pkg`:

```
$ pkgmk -b /product/pkgbin -r /product/pkgsrc -f /product/pkgsrc/prototype
```

Opcionalmente, você pode usar estes comandos para atingir o mesmo resultado:

```
$ cd /product/pkgsrc
$ pkgmk -o -b /product/pkgbin
```

Nesse exemplo, o comando `pkgmk` usa o diretório de trabalho atual para encontrar as partes restantes do pacote (como os arquivos de informação `prototype` e `pkginfo`).

Consulte também Se quiser adicionar arquivos de informação e scripts de instalação opcionais ao seu pacote, consulte Capítulo 3, “Melhorando a funcionalidade de um pacote (Tarefas)”. Do contrário, depois de construir o pacote, você deve verificar a sua integridade. O Capítulo 4, “Verificando e transferindo um pacote” explica como fazê-lo e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

◆ ◆ ◆ CAPÍTULO 3

Melhorando a funcionalidade de um pacote (Tarefas)

Este capítulo descreve como criar arquivos de informação opcionais e scripts de instalação em um pacote. Enquanto o Capítulo 2, "Construindo um Pacote" aborda os requisitos mínimos para criar um pacote, este capítulo aborda a funcionalidade adicional que você pode construir em um pacote. Esta funcionalidade adicional está baseada no critério que você considerou ao planificar como criar o pacote. Para obter mais informações, consulte "Considerações antes de construir um pacote" na página 17.

A lista abaixo traz as informações gerais encontradas neste capítulo.

Criando arquivos de informação e scripts de instalação (Mapa de tarefas)

O mapa de tarefas seguinte descreve os recursos opcionais que você pode construir em um pacote.

TABELA 3–1 Criando arquivos de informação e scripts de instalação (Mapa de tarefas)

Tarefa	Descrição	Instruções
1. Criar arquivos de informação	*Define as dependências do pacote* Uma definição de dependências do pacote permite especificar se o seu pacote é compatível com versões anteriores, é dependente de outro pacote ou se outros pacotes são dependentes do seu pacote.	"Como definir as dependências do pacote" na página 53

TABELA 3–1 Criando arquivos de informação e scripts de instalação (Mapa de tarefas) *(Continuação)*

Tarefa	Descrição	Instruções
	Escreve uma mensagem de copyright Um arquivo `copyright` oferece proteção legal para seu aplicativo de software.	"Como escrever uma mensagem de copyright" na página 56
	Reserva espaço adicional no sistema de destino. Um arquivo `space` define blocos à parte no sistema de destino, o que permite criar, durante a instalação, arquivos que estão definidos no arquivo `pkgmap`.	"Como reservar espaço adicional em um sistema de destino" na página 57
2. Criar scripts de instalação	*Obtém informações do instalador* Um script `request` permite obter informações da pessoa que está instalando o pacote.	"Como escrever um script `request`" na página 64
	Coleta dados do sistema de arquivos necessários para a instalação Um script `checkinstall` permite realizar uma análise do sistema de destino e configurar o ambiente correto ou interromper perfeitamente a instalação.	"Como coletar dados do sistema de arquivos" na página 67
	Escreve scripts de procedimento Os scripts de procedimento permitem fornecer instruções de instalação personalizadas durante fases específicas do processo de instalação ou de remoção.	"Como escrever scripts de procedimento" na página 69
	Escreve scripts de ação de classe Os scripts de ação de classe permitem especificar um conjunto de instruções a serem executadas durante a instalação do pacote e a remoção de objetos de pacote em grupos específicos.	"Como escrever scripts de ação de classe" na página 77

Criando arquivos de informação

Esta seção trata dos arquivos de informação opcionais do pacote. Com estes arquivos você pode definir as dependências do pacote, fornecer uma mensagem de copyright ou reservar espaço em um sistema de destino.

Definindo dependências do pacote

Você precisa determinar se o seu pacote tem dependências em outros pacotes e se outros pacotes dependem do seu pacote. As dependências e incompatibilidades podem ser definidas com dois arquivos de informação opcionais do pacote, `compver` e `depend`.

Entregar um arquivo `compver` permite nomear as versões anteriores do pacote compatíveis com o pacote que está sendo instalado.

Entregar um arquivo `depend` permite definir três tipos dependências associadas ao pacote. Os tipos de dependências são:

- Um *pacote de pré-requisito* – Seu pacote depende da existência de outro pacote
- Uma *dependência inversa* – Outro pacote depende da existência do seu pacote

---

Observação – Use o tipo de dependência inversa somente quando um pacote que não puder entregar um arquivo `depend` dependa do seu pacote.

- Um *pacote incompatível* – Seu pacote é incompatível com o pacote nomeado

O arquivo `depend` resolve somente dependências muito básicas. Se seu pacote depende de um arquivo específico, do seu conteúdo ou do seu comportamento, o arquivo `depend` não oferece precisão adequada. Neste caso, um script `request` ou o script `checkinstall` deve ser usado para a verificação de dependência detalhada. O script `checkinstall` é também o único script capaz de interromper perfeitamente o processo de instalação do pacote.

Observação – Tenha certeza de que os arquivos `depend` e `compver` tenham entradas no arquivo `prototype`. O tipo de arquivo deve ser `i` (para arquivo de informação do pacote).

Consulte as páginas do manual `depend(4)` e `compver(4)` para obter mais informações.

▼ Como definir as dependências do pacote

1 **Torne o diretório que contém os arquivos de informação o diretório de trabalho atual.**

2 **Se houver versões anteriores do seu pacote e você precisar especificar que o novo pacote é compatível com tais versões, crie um arquivo nomeado `compver` com o seu editor de texto preferido.**

Liste as versões com as quais o pacote é compatível. Use este formato:

string string . . .

O valor de *string* é idêntico ao valor atribuído ao parâmetro `VERSION` no arquivo `pkginfo` para cada pacote compatível.

3 **Salve as alterações e saia do editor.**

4 Se o seu pacote depende da existência de outros pacotes, outros pacotes dependem do seu pacote ou seu pacote é incompatível com outros pacotes, crie um arquivo nomeado `depend` com o seu editor de texto preferido.

Adicione uma entrada para cada dependência. Use este formato:

type pkg-abbrev pkg-name
(*arch*) *version*
(*arch*) *version* . . .

type	Define o tipo de dependência. Deve ser um dos caracteres seguintes: `P` (pacote de pré-requisito), `I` (pacote incompatível) ou `R` (dependência inversa).
pkg-abbrev	Especifica a abreviatura do pacote, tal como `SUNWcadap`.
pkg-name	Especifica o nome completo do pacote, tal como `Chip designers need CAD application software to design abc chips. Runs only on xyz hardware and is installed in the usr partition.`
(*arch*)	Opcional. Especifica o tipo de hardware no qual o pacote é executado. Por exemplo, `sparc` ou `x86`. Se especificar uma arquitetura, você deve usar parênteses como delimitadores.
version	Opcional. Especifica o valor atribuído ao parâmetro `VERSION` no arquivo `pkginfo`.

Para obter mais informações, consulte `depend(4)`.

5 Salve as alterações e saia do editor.

6 Realize *uma* das seguintes tarefas:

- Se você quiser criar arquivos de informação adicionais e scripts de instalação, vá para a próxima tarefa, "Como escrever uma mensagem de copyright" na página 56.
- Se você não tiver criado o arquivo `prototype`, realize o procedimento "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43. Vá para a Etapa 7.
- Se você já criou o arquivo `prototype`, edite-o e adicione uma entrada para cada arquivo recém-criado.

7 Construa o pacote.

Consulte "Como construir um pacote" na página 46, se necessário.

Exemplo 3–1 Arquivo `compver`

Neste exemplo, há quatro versões de um pacote: 1.0, 1.1, 2.0 e o novo pacote, 3.0. O novo pacote é compatível com as três versões anteriores. O arquivo `compver` na versão mais recente deve ser semelhante a:

```
release 3.0
release 2.0
version 1.1
1.0
```

As entradas não têm que estar em ordem seqüencial. No entanto, devem corresponder exatamente à definição do parâmetro `VERSION` em cada arquivo `pkginfo` do pacote. Neste exemplo, os criadores de pacotes usaram diferentes formatos nas três primeiras versões.

Exemplo 3–2 Arquivo depend

Este exemplo pressupõe que o pacote de amostra, `SUNWcadap`, requer que os pacotes `SUNWcsr` e `SUNWcsu` já estejam instalados em um sistema de destino. O arquivo `depend` de `SUNWcadap` se assemelha a:

```
P SUNWcsr Core Solaris, (Root)
P SUNWcsu Core Solaris, (Usr)
```

Consulte também Depois de construir o pacote, instale-o para confirmar que ele é instalado corretamente e verificar sua integridade. O Capítulo 4, "Verificando e transferindo um pacote" explica estas tarefas e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

Escrevendo uma mensagem de copyright

Você precisa decidir se o pacote deve exibir uma mensagem de copyright enquanto estiver sendo instalado. Se for o caso, crie o arquivo `copyright`.

Observação – Você deve incluir um arquivo `copyright` para oferecer proteção legal ao seu aplicativo de software. Confirme com o departamento jurídico da sua empresa qual o texto exato da mensagem.

Para entregar uma mensagem de copyright, você deve criar um arquivo nomeado `copyright`. Durante a instalação, a mensagem é exibida exatamente como aparece no arquivo (sem formato). Consulte a página do manual `copyright(4)` para obter mais informações.

Observação – Tenha certeza de que o arquivo `copyright` tem uma entrada no arquivo `prototype`. O tipo de arquivo deve ser `i` (para arquivo de informação do pacote).

▼ Como escrever uma mensagem de copyright

1 **Torne o diretório que contém os arquivos de informação no diretório de trabalho atual.**

2 **Crie um arquivo nomeado `copyright` com o seu editor de texto preferido.**

Digite o texto da mensagem de copyright exatamente como deseja que ele apareça quando seu pacote for instalado.

3 **Salve as alterações e saia do editor.**

4 **Realize *uma* das seguintes tarefas:**

- Se você quiser criar arquivos de informação adicionais e scripts de instalação, vá para a próxima tarefa, "Como reservar espaço adicional em um sistema de destino" na página 57.
- Se você *não* tiver criado o arquivo `prototype`, realize o procedimento "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43. Vá para a Etapa 5.
- Se você já criou o arquivo `prototype`, edite-o e adicione uma entrada para o arquivo de informação recém-criado.

5 **Construa o pacote.**

Consulte "Como construir um pacote" na página 46, se necessário.

Exemplo 3–3 Arquivo `copyright`

Por exemplo, uma mensagem de copyright parcial pode ter a seguinte aparência:

```
Copyright (c) 2003 Company Name
All Rights Reserved

This product is protected by copyright and distributed under
licenses restricting copying, distribution, and decompilation.
```

Consulte também Depois de construir o pacote, instale-o para confirmar que ele é instalado corretamente e verificar sua integridade. O Capítulo 4, "Verificando e transferindo um pacote" explica estas tarefas e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

Reservando espaço adicional em um sistema de destino

Você precisa determinar se o seu pacote precisa espaço de disco adicional no sistema de destino. Este espaço é adicionado ao espaço requerido pelos objetos de pacote. Se for o caso, crie o arquivo de informação `space`. Esta tarefa é diferente da criação de arquivos e diretórios vazios no tempo de instalação, conforme tratado em "Definindo objetos adicionais a serem criados no tempo de instalação" na página 40.

O comando `pkgadd` garante que haja espaço em disco suficiente para instalar o pacote com base nas definições do objeto no arquivo `pkgmap`. No entanto, um pacote pode precisar de espaço adicional em disco além daquele necessário para os objetos definidos no arquivo `pkgmap`. Por exemplo, seu pacote pode criar um arquivo depois da instalação, o qual pode conter um banco de dados, arquivos log ou algum outro arquivo que consuma espaço em disco. Para ter certeza de que há espaço reservado para tal arquivo, você deve incluir um arquivo `space` que especifica os requisitos de espaço em disco. O comando `pkgadd` verifica o espaço adicional especificado em um arquivo `space`. Consulte a página do manual `space(4)` para obter mais informações.

Observação – Tenha certeza de que o arquivo `space` tem uma entrada no arquivo `prototype`. O tipo de arquivo deve ser `i` (para arquivo de informação do pacote).

▼ Como reservar espaço adicional em um sistema de destino

1 Torne o diretório que contém os arquivos de informação no diretório de trabalho atual.

2 Crie um arquivo nomeado `space` com o seu editor de texto preferido.

Especifique os requisitos de espaço em disco necessários para seu pacote. Use este formato:

pathname blocks inodes

pathname	Especifica o nome de um diretório, que pode ou não ser o ponto de montagem de um sistema de arquivos.
blocks	Especifica o número de blocos de 512 bytes que você deseja reservar.
inodes	Especifica a quantidade de inodes requerida.

Para obter mais informações, consulte a página do manual `space(4)`.

3 Salve as alterações e saia do editor.

4 Realize uma das seguintes tarefas.

- Se quiser criar scripts de instalação, vá para a próxima etapa, "Como escrever um script `request`" na página 64.
- Se você não tiver criado o arquivo `prototype`, realize o procedimento "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43. Vá para a Etapa 5.
- Se você já criou o arquivo `prototype`, edite-o e adicione uma entrada para o arquivo de informação recém-criado.

5 Construa o pacote.

Consulte "Como construir um pacote" na página 46, se necessário.

Exemplo 3–4 Arquivo `space`

Este arquivo `space` de exemplo especifica que 1000 blocos de 512 bytes e 1 inode sejam reservados no diretório `/opt` no sistema de destino.

```
/opt   1000   1
```

Consulte também Depois de construir o pacote, instale-o para confirmar que ele é instalado corretamente e verificar sua integridade. O Capítulo 4, "Verificando e transferindo um pacote" explica estas tarefas e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

Criando scripts de instalação

Esta seção trata dos scripts de instalação opcionais do pacote. O comando `pkgadd` realiza automaticamente todas as ações necessárias para instalar um pacote usando os arquivos de informação como entrada. *Não* é necessário fornecer nenhum script de instalação de pacote. No entanto, se você quiser criar procedimentos de instalação personalizados para o pacote, é possível fazê-lo com os scripts de instalação. Os scripts de instalação:

- Devem ser executáveis pelo shell Bourne (`sh`)
- Devem conter texto e comandos de shell Bourne
- Não precisam conter identificador de shell `#!/bin/sh`
- Não precisam ser um arquivo executável

Há quatro tipos de scripts de instalação com os quais é possível realizar ações personalizadas:

- O script `request`

  O script `request` solicita dados do administrador que estiver instalando um pacote para atribuição ou redefinição das variáveis de ambiente.
- O script `checkinstall`

O script `checkinstall` examina o sistema de destino a procura dos dados necessários, pode definir ou modificar as variáveis de ambiente do pacote e determina se a instalação deve continuar.

Observação – O script `checkinstall` está disponível a partir do Solaris 2.5 e versões compatíveis.

- Scripts de procedimento

  Os scripts de procedimento identificam um procedimento a ser invocado antes ou depois da instalação ou remoção de um pacote. Os quatro scripts de procedimento são `preinstall`, `postinstall`, `preremove` e `postremove`.

- Scripts de ação de classe

  Os scripts de ação de classe definem uma ação ou um conjunto de ações que deve ser aplicado a uma classe de arquivos durante a instalação ou remoção. Você pode definir suas próprias classes. Outra alternativa é usar uma das quatro classes padrão (`sed`, `awk`, `build` e `preserve`).

Processamento de script durante a instalação do pacote

O tipo de scripts que você usa depende de quando a ação do script é necessária durante o processo de instalação. Conforme um pacote é instalado, o comando `pkgadd` realiza as seguintes etapas:

1. Executa o script `request`.

   Esta etapa é o único ponto no qual seu pacote pode solicitar entrada do administrador que estiver instalando o pacote.

2. Executa o script `checkinstall`.

   O script `checkinstall` coleta os dados do sistema de arquivos e pode criar ou alterar as definições da variável de ambiente para controlar a instalação subseqüente. Para obter mais informações sobre as variáveis de ambiente do pacote, consulte "Variáveis de ambiente do pacote" na página 24.

3. Executa o script `preinstall`.

4. Instala objetos de pacote em cada classe a ser instalada.

   A instalação desses arquivos ocorre classe por classe e os scripts de ação de classe são executados do modo devido. A lista de classes operadas e a ordem na qual devem ser instaladas são definidas inicialmente com o parâmetro `CLASSES` no arquivo `pkginfo`. No entanto, o script `request` ou o script `checkinstall` podem alterar o valor do parâmetro

`CLASSES`. Para obter mais informações sobre como as classes são processadas durante a instalação, consulte "Como as classes são processadas durante a instalação do pacote" na página 70.

a. Cria links simbólicos, dispositivos, pipes nomeados e diretórios necessários.
b. Instala os arquivos regulares (tipos de arquivos `e`, `v`, `f`), baseados em suas classes

   O script de ação de classe é passado somente a arquivos regulares para instalar. Todos os objetos de pacote são criados automaticamente das informações do arquivo `pkgmap`.
c. Cria todos os links de disco rígido.

5. Executa o script `postinstall`.

Processamento de script durante a remoção do pacote

Quando está sendo removido, o comando `pkgrm` realiza as seguintes etapas:

1. Executa o script `preremove`.
2. Remove os objetos de pacote de cada classe

   A remoção também ocorre classe por classe. Os scripts de remoção são processados na ordem inversa da instalação, com base na seqüência definida no parâmetro `CLASSES`. Para obter mais informações sobre como as classes são processadas durante a instalação, consulte "Como as classes são processadas durante a instalação do pacote" na página 70.

   a. Remove os links de disco rígido.
   b. Remove os arquivos regulares.
   c. Remove os links simbólicos, os dispositivos e os pipes nomeados.
3. Executa o script `postremove`.

O script `request` não é processado no momento da remoção do pacote. No entanto, a saída do script é retida no pacote instalado e é disponibilizada para os scripts de remoção. A saída do script `request` é uma lista de variáveis de ambiente.

Variáveis de ambiente de pacote disponíveis para os scripts

Os seguintes grupos de variáveis de ambiente estão disponíveis para todos os scripts de instalação. Algumas das variáveis de ambiente podem ser modificadas por um script `request` ou um script `checkinstall`.

- O script `request` ou o script `checkinstall` podem definir ou modificar qualquer um dos parâmetros padrão no arquivo `pkginfo`, exceto os parâmetros necessários. Os parâmetros de instalação padrão estão descritos detalhadamente na página do manual `pkginfo(4)`.

Observação – O parâmetro BASEDIR pode ser modificado somente a partir do Solaris 2.5 e versões compatíveis.

- Você pode definir suas próprias variáveis de ambiente de instalação atribuindo valores a elas no arquivo `pkginfo`. Tais variáveis de ambiente devem ser alfanuméricas com letras maiúsculas iniciais. Qualquer uma dessas variáveis de ambiente pode ser modificada por um script `request` ou um script `checkinstall`.
- Tanto um script `request` quanto `checkinstall` pode definir novas variáveis de ambiente atribuindo valores a elas e colocando-as no ambiente de instalação.
- A tabela seguinte lista as variáveis de ambiente disponíveis a todos os scripts de instalação no ambiente. Nenhuma dessas variáveis de ambiente pode ser modificada por um script.

Variável de Ambiente	Descrição
`CLIENT_BASEDIR`	O diretório base com respeito ao sistema de destino. Enquanto que `BASEDIR` é a variável a ser usada ao fazer referência a um objeto de pacote específico do sistema de instalação (provavelmente um servidor), `CLIENT_BASEDIR` é o caminho a ser incluído nos arquivos que estão no sistema do cliente. `CLIENT_BASEDIR` existe se `BASEDIR` existir e é idêntico ao `BASEDIR` se não houver `PKG_INSTALL_ROOT`.
`INST_DATADIR`	O diretório no qual está o pacote que está sendo lido agora. Se o pacote estiver sendo lido de uma fita, esta variável será o local de um diretório temporário para o qual o pacote foi transferido em formato de diretório. Em outras palavras, pressupondo que o nome do pacote não apresenta nenhuma extensão (por exemplo, `SUNWstuff.d`), o script `request` do pacote atual se encontraria em `$INST_DATADIR/$PKG/install`.
`PATH`	A lista de pesquisa usada pelo `sh` para encontrar comandos na chamada de script. `PATH` é geralmente definido como `/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin` .
`PKGINST`	O identificador de instância do pacote que está sendo instalado. Se outra instância do pacote ainda não estiver instalada, o valor é a abreviatura do pacote (por exemplo, `SUNWcadap`). Do contrário, o valor é abreviatura do pacote seguida de um sufixo, tal como `SUNWcadap.4`.
`PKGSAV`	O diretório no qual os arquivos podem ser salvos para serem usados pelos scripts de remoção ou no qual os arquivos salvos anteriormente podem ser encontrados. Disponível somente no Solaris 2.5 e versões compatíveis.
`PKG_CLIENT_OS`	O sistema operacional do cliente no qual o pacote está sendo instalado. O valor desta variável é `Solaris`.
`PKG_CLIENT_VERSION`	A versão do Solaris em formato `x.y`.
`PKG_CLIENT_REVISION`	A revisão da compilação do Solaris.

Variável de Ambiente	Descrição
PKG_INSTALL_ROOT	O sistema de arquivos raiz no sistema de destino no qual o pacote está sendo instalado. Esta variável existe somente se os comandos pkgadd e pkgrm forem invocados com a opção -R. Esta existência condicional facilita seu uso em scripts de procedimento na forma ${PKG_INSTALL_ROOT}/*algum_caminho*.
PKG_NO_UNIFIED	Uma variável de ambiente que fica definida se os comandos pkgadd e pkgrm forem invocados com as opções -M e -R. Esta variável de ambiente é passada para qualquer script de instalação de pacote ou comando de pacote que faça parte do ambiente do pacote.
UPDATE	Esta variável de ambiente não existe na maioria dos ambientes de instalação. Se esta variável não existir (com o valor yes), isso pode ter duas explicações. Ou que já existe um pacote com o mesmo nome, versão e arquitetura instalado no sistema. Ou que este pacote está substituindo um pacote instalado com o mesmo nome no diretório do administrador. Nestes casos, é sempre usado o diretório base original.

Obtendo informações do pacote para um script

Dois comandos podem ser usados para solicitar informações dos scripts sobre um pacote:

- O comando pkginfo retorna informações sobre os pacotes de software, tais como o identificador de instância e o nome do pacote.
- O pkgparam retorna valores para as variáveis de ambiente solicitadas.

  Consulte a página do manual pkginfo(1), a página do manual pkgparam(1) e o Capítulo 4, "Verificando e transferindo um pacote" para obter mais informações.

Códigos de saída para scripts

Cada script deve sair com um dos códigos de saída ilustrados na tabela seguinte.

TABELA 3–2 Códigos de saída de script instalação

Código	Significado
0	Conclusão de script bem-sucedida.
1	Erro fatal. O processo de instalação foi interrompido neste ponto.
2	Aviso ou possível condição de erro. A instalação continua. Uma mensagem de aviso é exibida no momento da conclusão.
3	O comando pkgadd é detido perfeitamente. Somente o script checkinstall retorna este código.
10	O sistema deve ser reiniciado quando a instalação de todos os pacotes selecionados for concluída. (Este valor deve ser adicionado a um dos códigos de saída de um dígito.)

TABELA 3–2 Códigos de saída de script instalação *(Continuação)*

Código	Significado
20	O sistema deve ser reiniciado imediatamente na conclusão da instalação do pacote atual. (Este valor deve ser adicionado a um dos códigos de saída de um dígito.)

Consulte Capítulo 5, "Estudos de caso de criação de pacote" para ver exemplos de códigos de saída que são retornados por scripts de instalação.

Observação – Todos os scripts de instalação entregues com o pacote devem ter uma entrada no arquivo `prototype`. O tipo de arquivo deve ser `i` (para script de instalação de pacote).

Escrevendo um script `request`

O script `request` é a única forma de que seu pacote possa interagir diretamente com o administrador que está instalando tal pacote. Este script pode ser usado, por exemplo, para perguntar ao administrador se as partes opcionais de um pacote devem ser instaladas.

A saída de um script `request` deve ser uma lista de variáveis de ambiente e seus valores. Esta lista pode incluir qualquer um dos parâmetros criados no arquivo `pkginfo` e os parâmetros `CLASSES` e `BASEDIR`. A lista também pode introduzir variáveis de ambiente que ainda não foram definidas em outro lugar. No entanto, o arquivo `pkginfo` deve sempre fornecer valores padrão quando for prático. Para obter mais informações sobre as variáveis de ambiente do pacote, consulte "Variáveis de ambiente do pacote" na página 24.

Quando o seu script `request` atribuir valores a uma variável de ambiente, ele deve tornar tais valores disponíveis para o comando `pkgadd` e para outros scripts do pacote.

Comportamentos do script `request`

- O script `request` não pode modificar nenhum arquivo. Este script interage somente com administradores que estão instalando o pacote e cria uma lista de atribuições de variáveis de ambiente baseada em tal interação. O script `request` é executado como o usuário `noaccess`não privilegiado se tal usuário existir. Do contrário, o script é executado como usuário `root`.
- O comando `pkgadd` chama o script `request` com um argumento que nomeia o arquivo de resposta do script. O arquivo de resposta armazena as respostas do administrador.
- O script `request` não é executado durante a remoção do pacote. No entanto, as variáveis de ambiente atribuídas pelo script são salvas e estão disponíveis durante a remoção do pacote.

Regras de criação para scripts `request`

- Pode haver somente um script `request` por pacote. O script deve ser nomeado `request`.

- As atribuições de variável de ambiente devem ser adicionadas ao ambiente de instalação para serem usadas pelo comando `pkgadd` e outros scripts de empacotamento escrevendo-as no arquivo de resposta (conhecido pelo script como `$1`).
- As variáveis de ambiente do sistema e as variáveis de ambiente de instalação padrão, exceto para os parâmetros `CLASSES` e `BASEDIR`, não podem ser modificadas por um script `request`. Todas as outras variáveis de ambiente que você criou podem ser alteradas.

Observação – Um script `request` pode modificar somente o parâmetro `BASEDIR` a partir do Solaris 2.5 e versões compatíveis.

- No arquivo `pkginfo`, a cada variável que o script `request` pode manipular deve ser atribuído um valor padrão.
- O formato da lista de saída deve ser *PARAM=value*. Por exemplo:

```
CLASSES=none class1
```

- O terminal do administrador é definido como entrada padrão do script `request`.
- Não realiza nenhuma análise especial do sistema de destino em um script `request`. É arriscado testar o sistema em busca de determinados binários ou comportamentos, e definir variáveis de ambiente com base em tal análise. Não há garantias de que o script `request` seja executado no tempo de instalação. O administrador que estiver instalando o pacote pode fornecer um arquivo de resposta que inserirá as variáveis de ambiente sem nunca chamar o script `request`. Se o script `request` também estiver avaliando o sistema de arquivos de destino, tal avaliação pode não ocorrer. É melhor que a análise do sistema de destino seja realizada pelo script `checkinstall` para obter tratamento especial.

Observação – Se os administradores que instalarão o seu pacote puderem usar o produto JumpStart, então a instalação do seu pacote não deve ser interativa. Você deve fornecer um script `request` com seu pacote ou deve comunicar aos administradores que eles devem usar o comando `pkgask` antes da instalação. O comando `pkgask` armazena as repostas deles no script `request`. Para obter mais informações sobre o comando `pkgask`, consulte a página do manual `pkgask`(1M).

▼ Como escrever um script `request`

1 Torne o diretório que contém os arquivos de informação no diretório de trabalho atual.

2 Crie um arquivo nomeado `request` com o seu editor de texto preferido.

3 Salve as alterações e saia do editor quando acabar.

4 **Realize uma das seguintes tarefas.**

- Se quiser criar scripts de instalação, vá para a próxima etapa, "Como coletar dados do sistema de arquivos" na página 67.
- Se você não tiver criado o arquivo `prototype`, realize o procedimento "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43. Vá para a Etapa 5.
- Se você já criou o arquivo `prototype`, edite-o e adicione uma entrada para cada script de instalação recém-criado.

5 **Construa o pacote.**

Consulte "Como construir um pacote" na página 46, se necessário.

Exemplo 3–5 Escrevendo um script `request`

Quando um script `request` atribui valor a variáveis de ambiente, ele deve disponibilizar tais valores para o comando `pkgadd`. Este exemplo mostra um segmento do script `request` que realiza esta tarefa em quatro variáveis de ambiente: `CLASSES`, `NCMPBIN`, `EMACS` e `NCMPMAN`. Suponha que estas variáveis tenham sido definidas em uma sessão interativa com o administrador anteriormente no script.

```
# make environment variables available to installation
# service and any other packaging script we might have

cat >$1 <<!
CLASSES=$CLASSES
NCMPBIN=$NCMPBIN
EMACS=$EMACS
NCMPMAN=$NCMPMAN
!
```

Consulte também Depois de construir o pacote, instale-o para confirmar que ele é instalado corretamente e verificar sua integridade. O Capítulo 4, "Verificando e transferindo um pacote" explica estas tarefas e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

Coletando dados do sistema de arquivos com o script `checkinstall`

O script `checkinstall` é executado brevemente depois do script `request` opcional. O script de `checkinstall` é executado como o usuário `noaccess`, se tal usuário existir. O script `checkinstall` não tem autoridade para alterar os dados do sistema de arquivos. No entanto, com base nas informações que o script coleta, ele pode criar ou modificar as variáveis de ambiente a fim de controlar o curso da instalação resultante. O script também pode deter perfeitamente o processo de instalação.

O script `checkinstall` está programado para realizar verificações básicas em um sistema de arquivos que não é normal para o comando `pkgadd`. Por exemplo, este script pode ser usado para verificar arquivos adiante a fim de determinar se tais arquivos do pacote atual substituirão arquivos existentes, ou gerenciar as dependências gerais do software. O arquivo `depend` gerencia somente dependências no nível do pacote.

Diferente do script `request`, o script `checkinstall` é executado se um arquivo de resposta for ou não fornecido. A presença do script não marca pacote como interativo. O script `checkinstall` pode ser usado quando um script `request` for esquecido ou quando a interação do administrador não for útil.

Observação – O script `checkinstall` está disponível a partir do Solaris 2.5 e versões compatíveis.

Comportamentos do script `checkinstall`

- O script `checkinstall` não pode modificar nenhum arquivo. Este script analisa somente o estado do sistema e cria uma lista de atribuições de variáveis de ambiente com base em tal interação. Para fazer cumprir esta limitação, o script `checkinstall` é executado como o usuário `noaccess` não privilegiado se tal usuário existir. Do contrário, este script é executado como usuário `nobody` não privilegiado. O script `checkinstall` não tem autoridade de superusuário.
- O comando `pkgadd` chama o script `checkinstall` com um argumento que nomeia o arquivo de resposta do script. O arquivo de resposta do script é o arquivo que armazena as respostas do administrador.
- O script `checkinstall` não é executado durante a remoção do pacote. No entanto, as variáveis de ambiente atribuídas pelo script são salvas e estão disponíveis durante a remoção do pacote.

Regras de criação para scripts `checkinstall`

- Pode haver somente um script `checkinstall` por pacote. O script deve ser nomeado `checkinstall`.
- As atribuições de variável de ambiente devem ser adicionadas ao ambiente de instalação para serem usadas pelo comando `pkgadd` e outros scripts de empacotamento escrevendo-as no arquivo de resposta (conhecido pelo script como `$1`).
- As variáveis de ambiente do sistema e as variáveis de ambiente de instalação padrão, exceto para os parâmetros `CLASSES` e `BASEDIR`, não podem ser modificadas por um script `checkinstall`. Todas as outras variáveis de ambiente que você criou podem ser alteradas.
- No arquivo `pkginfo`, a cada variável que o script `checkinstall` pode manipular deve ser atribuído um valor padrão.
- O formato da lista de saída deve ser *PARAM=value*. Por exemplo:

```
CLASSES=none class1
```

- A interação do administrador não é permitida durante a execução de um script `checkinstall`. Todas as interações do administrador estão limitadas ao script `request`.

▼ Como coletar dados do sistema de arquivos

1. **Torne o diretório que contém os arquivos de informação no diretório de trabalho atual.**
2. **Crie um arquivo nomeado `checkinstall` com o seu editor de texto preferido.**
3. **Salve as alterações e saia do editor quando acabar.**
4. **Realize uma das seguintes tarefas.**
   - Se quiser criar scripts de instalação adicionais, vá para a próxima etapa, "Como escrever scripts de procedimento" na página 69.
   - Se você não tiver criado o arquivo `prototype`, realize o procedimento "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43. Vá para a Etapa 5.
   - Se você já criou o arquivo `prototype`, edite-o e adicione uma entrada para cada script de instalação recém-criado.
5. **Construa o pacote.**

   Consulte "Como construir um pacote" na página 46, se necessário.

Exemplo 3–6 Escrevendo um script `checkinstall`

Este exemplo do script `checkinstall` realiza uma verificação para ver se o software de banco de dados que o pacote `SUNWcadap` precisa está instalado.

```
# checkinstall script for SUNWcadap
#
# This confirms the existence of the required specU database

# First find which database package has been installed.
pkginfo -q SUNWspcdA    # try the older one

if [ $? -ne 0 ]; then
   pkginfo -q SUNWspcdB    # now the latest

      if [ $? -ne 0 ]; then    # oops
            echo "No database package can be found. Please install the"
            echo "SpecU database package and try this installation again."
            exit 3        # Suspend
      else
            DBBASE="`pkgparam SUNWsbcdB BASEDIR`/db"    # new DB software
      fi
else
      DBBASE="`pkgparam SUNWspcdA BASEDIR`/db"    # old DB software
```

```
fi

# Now look for the database file we will need for this installation
if [ $DBBASE/specUlatte ]; then
      exit 0        # all OK
else
      echo "No database file can be found. Please create the database"
      echo "using your installed specU software and try this"
      echo "installation again."
      exit 3        # Suspend
fi
```

Consulte também Depois de construir o pacote, instale-o para confirmar que ele é instalado corretamente e verificar sua integridade. O Capítulo 4, “Verificando e transferindo um pacote” explica estas tarefas e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

Escrevendo scripts de procedimento

Os scripts de procedimento oferecem um conjunto de instruções para serem realizadas em determinados pontos da instalação ou remoção do pacote. Os quatro scripts de procedimento devem ser nomeados com um dos nomes predefinidos, dependendo de quando as instruções serão executadas. Os scripts são executados sem argumentos.

- O script `preinstall`

  É executado antes do início da instalação da classe. Nenhum arquivo deve ser instalado por este script.
- O script `postinstall`

  É executado depois que todos os volumes tiverem sido instalados.
- O script `preremove`

  É executado antes do início da remoção da classe. Nenhum arquivo deve ser removido por este script.
- O script `postremove`

  É executado depois que todas as classes tiverem sido removidas.

Comportamentos do script de procedimento

Os scripts de procedimento são executados como `uid=root` e `gid=other`.

Regras de criação dos scripts de procedimento

- Cada script deve poder ser executado mais de uma vez porque ele é executado uma vez em cada volume de um pacote. Isso significa que executar um script várias vezes com a mesma entrada produz os mesmos resultados que executar o script somente uma vez.

- Cada script de procedimento que não instalar um objeto de pacote no arquivo `pkgmap` deve usar o comando `installf` para notificar o banco de dados do pacote que ele está adicionando ou modificando um nome de caminho. Depois que todas as adições e modificações forem concluídas, este comando deve ser chamado com a opção `-f`. Somente os scripts `postinstall` e `postremove` podem instalar objetos de pacote desta forma. Consulte a página do manual `installf`(1M) e o Capítulo 5, "Estudos de caso de criação de pacote" para obter mais informações.
- Não é permitida a interação com o administrador durante a execução de um script de procedimento. Todas as interações do administrador estão limitadas ao script `request`.
- Cada script de procedimento que remove arquivos não instalados do arquivo `pkgmap` deve usar o comando `removef` para notificar o banco de dados que ele está removendo um nome de caminho. Depois que a remoção tiver sido concluída, este comando deve ser chamado com a opção `-f`. Consulte a página do manual `removef`(1M) e o Capítulo 5, "Estudos de caso de criação de pacote" para obter detalhes e exemplos.

Observação – Os comandos `installf` e `removef` devem ser usados porque os scripts de procedimentos não estão associados automaticamente a nenhum nome de caminho listado no arquivo `pkgmap`.

▼ Como escrever scripts de procedimento

1 Torne o diretório que contém os arquivos de informação no diretório de trabalho atual.

2 Crie um ou mais scripts de procedimento com o editor de texto de sua preferência.

Um script de procedimento deve ser nomeado com um dos nomes predefinidos: `preinstall`, `postinstall`, `preremove` ou `postremove`.

3 Salve as alterações e saia do editor.

4 Realize uma das seguintes tarefas.

- Se quiser criar scripts de ação de classe, vá para a próxima etapa, "Como escrever scripts de ação de classe" na página 77.
- Se você não tiver criado o arquivo `prototype`, realize o procedimento "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43. Vá para a Etapa 5.
- Se você já criou o arquivo `prototype`, edite-o e adicione uma entrada para cada script de instalação recém-criado.

5 Construa o pacote.

Consulte "Como construir um pacote" na página 46, se necessário.

Consulte também Depois de construir o pacote, instale-o para confirmar que ele é instalado corretamente e verificar sua integridade. O Capítulo 4, "Verificando e transferindo um pacote" explica estas tarefas e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

Escrevendo scripts de ação de classe

Definindo classes de objeto

As classes de objeto possibilitam uma série de ações a serem realizadas em grupo de objetos de pacote na instalação ou remoção. Você atribui objetos a uma classe no arquivo `prototype`. A todos os objetos de pacote deve ser fornecida uma classe, embora a classe `none` seja usada por padrão em objetos que não requerem nenhuma ação especial.

O parâmetro de instalação `CLASSES`, definido no arquivo `pkginfo`, é uma lista de classes a ser instalada (incluindo a classe `none`).

Observação – Os objetos definidos no arquivo `pkgmap` que pertencem a uma classe não listada neste parâmetro do arquivo `pkginfo` *não* serão instalados.

A lista de `CLASSES` determina a ordem de instalação. A classe `none` é sempre instalada primeiro, se estiver presente, e é removida por último. Visto que os diretórios são a estrutura de suporte fundamental de todos os outros objetos do sistema de arquivos, todos eles devem ser atribuídos à classe `none`. Pode haver exceções, mas, como regra geral, a classe `none` é a mais segura. Esta estratégia garante que os diretórios sejam criados antes dos objetos que irão conter. Além disso, nenhuma tentativa de excluir um diretório é feita antes que este esteja vazio.

Como as classes são processadas durante a instalação do pacote

Abaixo estão descritas as ações de sistema que ocorrem quando uma classe é instalada. As ações são repetidas uma vez para cada volume de um pacote, conforme tal volume está sendo instalado.

1. O comando `pkgadd` cria uma lista de nomes de caminho.

   O comando `pkgadd` cria uma lista de nomes de caminho sobre a qual o script de ação opera. Cada linha desta lista contém os nomes de caminho de origem e de destino, separados por um espaço. O nome de caminho de origem indica onde o objeto a ser instalado se localiza no volume de instalação. O nome de caminho de destino indica o local no sistema de destino onde o objeto deve ser instalado. O conteúdo da lista está limitado pelos seguintes critérios:

   - A lista contém somente nomes de caminho que pertencem à classe associada.

- Se a tentativa de criar o objeto de pacote falhar, então os diretórios, os pipes nomeados, os dispositivos de caractere, os dispositivos de bloco e os links simbólicos são incluídos na lista com o nome de caminho de origem definido como `/dev/null`. Normalmente, estes itens são criados automaticamente pelo comando `pkgadd` (se ainda não existirem) e a eles são dados atributos apropriados (modo, proprietário, grupo) conforme definido no arquivo `pkgmap`.
- Os arquivos vinculados em que o tipo de arquivo for `l` não são incluídos na lista sob nenhuma circunstância. Os links de disco rígido de uma determinada classe são criados no item 4.

2. Se nenhum script de ação de classe for fornecido para a instalação de uma determinada classe, os nomes de caminho da lista gerada são copiados do volume no local de destino apropriado.
3. Se houver um script de ação de classe, este é executado.

   O script de ação de classe é chamado com a entrada padrão que contém a lista gerada no item 1. Se este volume for o último volume do pacote, ou se não houver mais objetos nesta classe, o script é executado com o único argumento `ENDOFCLASS`.

   Observação – Mesmo se não houver nenhum arquivo regular desta classe no pacote, o script de ação de classe é chamado pelo menos uma vez com uma lista vazia e o argumento `ENDOFCLASS`.

4. O comando `pkgadd` realiza uma auditoria de conteúdo e atributo, e cria links de disco rígido.

   Após a execução bem-sucedida dos itens 2 ou 3, o comando `pkgadd` realiza a auditoria das informações de conteúdo e de atributo da lista de nomes de caminho. O comando `pkgadd` cria automaticamente os links associados à classe. As incoerências de atributo detectadas são corrigidas em todos os nomes de caminho da lista gerada.

Como as classes são processadas durante a remoção do pacote

Os objetos são removidos classe por classe. As classes que existem em um pacote, mas que não estão listadas no parâmetro `CLASSES`, são removidas primeiro (por exemplo, um objeto instalado com o comando `installf)`. As classes listadas no parâmetro `CLASSES` são removidas na ordem inversa. A classe `none` é removida sempre por último. Abaixo estão descritas as ações de sistema que ocorrem quando uma classe é removida:

1. O comando `pkgrm` cria uma lista de nomes de caminho.

   O comando `pkgrm` cria uma lista de nomes de caminho instalados que pertencem à classe indicada. Os nomes de caminho com referência de outro pacote são excluídos da lista, a menos que o tipo de arquivo deles seja `e`. O tipo de arquivo `e` significa que o arquivo deve ser editado na instalação ou na remoção.

Se o pacote que estiver sendo removido tiver modificado algum arquivo de tipo `e` durante a instalação, ele deve remover apenas as linhas que foram adicionadas. Não exclua um arquivo editável que não estiver vazio. Remova as linhas que o pacote adicionou.

2. Se não houver nenhum script de ação de classe, os nomes de caminho são excluídos.

   Se o seu pacote não tiver nenhum script de ação de classe na classe, todos os nomes de caminho da lista gerada pelo comando `pkgrm` são excluídos.

Observação – Os arquivos com um tipo de arquivo `e` (editável) não são atribuídos a uma classe nem a um script de ação de classe associado. Estes arquivos são removidos neste ponto, mesmo se o nome de caminho for compartilhado com outros pacotes.

---

3. Se houver um script de ação de classe, o script é executado.

   O comando `pkgrm` chama o script de ação de classe com a entrada padrão do script que contém a lista gerada no item 1.

4. O comando `pkgrm` realiza uma auditoria.

   Após a execução bem-sucedida do script de ação de classe, o comando `pkgrm` remove as referências aos nomes de caminho do banco de dados do pacote, a menos que outro pacote faça referência a um nome de caminho.

O script de ação de classe

O script de ação de classe define um conjunto de ações a serem executadas durante a instalação e a remoção de um pacote. As ações são realizadas em um grupo de nomes de caminho com base na definição de classe. Consulte Capítulo 5, “Estudos de caso de criação de pacote” para obter exemplos de scripts de ação de classe.

O nome de um script de ação de classe é baseado na classe na qual ele deve operar e se tais operações devem ocorrer durante a instalação ou remoção do pacote. Os dois formatos de nome são mostrados na tabela seguinte:

Formato de nome	Descrição
`i.`*class*	Opera em nomes de caminho na classe indicada durante a instalação do pacote.
`r.`*class*	Opera em nomes de caminho na classe indicada durante a remoção do pacote.

Por exemplo, o nome do script de instalação de uma classe denominada `manpage` seria `i.manpage`. O script de remoção seria denominado `r.manpage`.

Observação – Este formato de nome de arquivo não é usado em arquivos que pertencem às classes de sistema `sed`, `awk` ou `build`. Para obter mais informações sobre estas classes especiais, consulte "As classes de sistema especiais" na página 73.

Comportamentos do script de ação de classe

- Os scripts de ação de classe são executados como `uid=root` e `gid=other`.
- Um script é executado em todos os arquivos de uma determinada classe no volume atual.
- Os comandos `pkgadd` e `pkgrm` criam uma lista de todos os objetos listados no arquivo `pkgmap` que pertencem à classe. Como conseqüência, um script de ação de classe pode agir somente sobre nomes de caminho definidos no `pkgmap` que pertencem a uma determinada classe.
- Quando um script de ação de classe for executado pela última vez (isto é, não há mais nenhum arquivo que perteça a tal classe), o script de ação de classe é executado uma vez com o argumento de palavra-chave `ENDOFCLASS`.
- Não é permitida a interação com o administrador durante a execução de um script de ação de classe.

Regras de criação para sripts de ação de classe

- Se um pacote abranger mais de um volume, o script de ação de classe é executado uma vez para cada volume que contém pelo menos um arquivo que pertença a uma classe. Conseqüentemente, cada script deve poder ser executado mais de uma vez. Isso significa que executar um script várias vezes com a mesma entrada deve produzir os mesmos resultados que executar o script somente uma vez.
- Quando um arquivo fizer parte de uma classe que tem um script de ação de classe, o script deve instalar o arquivo. O comando `pkgadd` não instala os arquivos em cada classe que tenha um script de ação de classe, embora ele verifique a instalação.
- Um script de ação de classe nunca adiciona, remove ou modifica um nome de caminho ou um atributo de sistema que não apareça na lista gerada pelo comando `pkgadd`. Para obter mais informações sobre esta lista, consulte o item 1 em "Como as classes são processadas durante a instalação do pacote" na página 70.
- Quando o seu script interpretar o argumento `ENDOFCLASS`, coloque as ações de pós-processamento, tal como limpeza, no seu script.
- Todas as interações do administrador estão limitadas ao script `request`. Não tente obter informações do administrador usando um script de ação de classe.

As classes de sistema especiais

O sistema fornece quatro classes especiais:

- A classe `sed`

Oferece um método para uso de instruções `sed` para editar arquivos na instalação e na remoção do pacote.

- A classe `awk`

  Oferece um método para uso de instruções `awk` para editar arquivos na instalação e na remoção do pacote.

- A classe `build`

  Oferece um método para construir ou modificar dinamicamente um arquivo usando comandos de shell.

- A classe `preserve`

  Oferece um método para preservar arquivos que não devem ser substituídos por futuras instalações de pacote.

- A classe `manifest`

  Oferece instalação e desinstalação automática dos serviços SMF (Service Management Facility) associados a um manifest. A classe `manifest` deve ser usada em todos os manifests SMF de um pacote.

Se vários arquivos de um pacote precisarem de processamento especial que possa ser definido completamente através dos comandos `sed`, `awk` ou `sh`, a instalação é mais rápida usando as classes de sistema em vez de usar várias classes e os scripts de ação de classes correspondentes .

O script de classe sed

A classe `sed` oferece um método para modificar um objeto existente em um sistema de destino. O script de ação de classe `sed` é executado automaticamente na instalação se houver um arquivo que pertença à classe `sed`. O nome do script de ação de classe `sed` deve ser igual ao nome do arquivo no qual as instruções são executadas.

Um script de ação de classe `sed` entrega as instruções de `sed` no formato seguinte:

Dois comandos indicam quando as instruções devem ser executadas. As instruções de `sed` que vêm depois do comando `!install` são executadas durante a instalação do pacote. As instruções de `sed` que vêm depois do comando `!remove` são executadas durante a remoção do pacote. Não importa a ordem na qual estes comandos sejam usados no arquivo.

Para obter mais informações sobre as instruções de `sed` consulte a página do manual `sed(1)` Para obter exemplos dos scripts de ação de classe `sed`, consulte o Capítulo 5, “Estudos de caso de criação de pacote”.

O script de classe awk

A classe `awk` oferece um método para modificar um objeto existente em um sistema de destino. As modificações são entregues como instruções `awk` em um script de ação de classe `awk`.

O script de ação de classe awk é executado automaticamente na instalação se houver um arquivo que pertença à classe awk. Tal arquivo contém instruções para o script de classe awk no seguinte formato:

Dois comandos indicam quando as instruções devem ser executadas. As instruções de awk que vêm depois do comando `!install` são executadas durante a instalação do pacote. As instruções que vêm após o comando `!remove` são executadas durante a remoção do pacote. Estes comandos podem ser usados em qualquer ordem.

O nome do script de ação de classe awk deve ser igual ao nome do arquivo no qual as instruções são executadas.

O arquivo que será modificado é usado como entrada no comando awk e a saída do script substitui o objeto original. As variáveis de ambiente talvez não sejam passadas para o comando awk com esta sintaxe.

Para obter mais informações sobre as instruções de awk consulte a página do manual awk(1).

O script de classe build

A classe `build` cria ou modifica um arquivo de objeto de pacote executando as instruções do shell Bourne. Estas instruções são entregues como o objeto de pacote. As instruções são executadas automaticamente na instalação se houver um arquivo que pertença á classe `build`.

O nome do script de ação de classe `build` deve ser igual ao nome do arquivo no qual as instruções são executadas. O nome também deve ser executável pelo comando `sh`. A saída do script se torna a nova versão do arquivo à medida que ele é construído e modificado. Se o script não produzir saída, o arquivo não é criado nem modificado. Portanto, o script pode modificar ou criar o arquivo por si mesmo.

Por exemplo, se um pacote entrega um arquivo padrão, `/etc/randomtable`, e se o arquivo ainda não existir no sistema de destino, a entrada do arquivo `prototype` pode ser a seguinte:

```
e build /etc/randomtable ? ? ?
```

O objeto de pacote, `/etc/randomtable`, pode ser semelhante ao seguinte:

```
!install
# randomtable builder
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
        echo "/etc/randomtable is already in place.";
        else
        echo "# /etc/randomtable" > $PKG_INSTALL_ROOT/etc/randomtable
        echo "1121554    # first random number" >> $PKG_INSTALL_ROOT/etc/randomtable
fi

!remove
# randomtable deconstructor
if [ -f $PKG_INSTALL_ROOT/etc/randomtable ]; then
```

```
        # the file can be removed if it's unchanged
        if [ egrep "first random number" $PKG_INSTALL_ROOT/etc/randomtable ]; then
            rm $PKG_INSTALL_ROOT/etc/randomtable;
        fi
fi
```

Consulte o Capítulo 5, "Estudos de caso de criação de pacote" para obter outros exemplos que usem a classe `build`.

O script de classe preserve

A classe `preserve` preserva um arquivo do objeto de pacote determinando se um arquivo existente deve ou não ser substituído quando o pacote é instalado. Duas situações possíveis que podem ocorrer ao usar um script de classe `preserve` são:

- Se o arquivo que será instalado ainda não existir no diretório de destino, o arquivo será instalado normalmente.
- Se o arquivo que será instalado existir no diretório de destino, é exibida uma mensagem descrevendo que o arquivo existe e este não será instalado.

Ambos os resultados das situações são considerados satisfatórios pelo script `preserve`. Ocorre uma falha somente na segunda situação quando o arquivo não pode ser copiado no diretório de destino.

A partir do Solaris 7, o script `i.preserve` e uma cópia deste script, `i.CONFIG.prsv`, podem ser encontrados no diretório `/usr/sadm/install/scripts` com outros scripts de ação de classe.

Modifique o script para incluir o nome ou os nomes de arquivo que você gostaria de conservar.

O script de classe manifest

A classe `manifest` instala e desinstala automaticamente os serviços SMF (Service Management Facility) associados a um manifest SMF.. Se você não conhece o SMF, consulte o Capítulo 18, "Managing Services (Overview)," no *Oracle Solaris Administration: Basic Administration* para obter informações sobre como usar o SMF para gerenciar serviços.

Todos os manifests de serviço dentro dos pacotes devem ser identificados com a classe `manifest`. Os scripts de ação de classe que instalam e removem os manifests de serviço estão incluídos no subsistema de empacotamento. Quando `pkgadd`(1M) é chamado, o manifest de serviço é importado. Quando o `pkgrm`(1M) é chamado, as instâncias desativadas do manifest de serviço são excluídas. Os serviços no manifest que não têm instâncias restantes também são excluídos. Se a opção `-R` é fornecida a `pkgadd`(1M) ou `pkgrm`(1M), essas ações do manifest de serviço serão realizadas na próxima vez que os sistema for reinicializado com tal caminho de raiz alternativa.

A seguinte parte de código do arquivo de informações de um pacote mostra o uso da classe `manifest`.

```
# packaging files
i pkginfo
i copyright
i depend
i preinstall
i postinstall
#
# source locations relative to the prototype file
#
d none var 0755 root sys
d none var/svc 0755 root sys
d none var/svc/manifest 0755 root sys
d none var/svc/manifest/network 0755 root sys
d none var/svc/manifest/network/rpc 0755 root sys
f manifest var/svc/manifest/network/rpc/smserver.xml 0444 root sys
```

Observação – Não inclua os arquivos `i.manifest` e `r.manifest` nos seus pacotes, pois esses scripts de ação de classe são parte do SO do Oracle Solaris e variam entre as versões. A inclusão desses arquivos tornará seus pacotes menos portáveis entre as diferentes versões do Oracle Solaris.

▼ Como escrever scripts de ação de classe

1 **Torne o diretório que contém os arquivos de informação no diretório de trabalho atual.**

2 **Atribua aos objetos de pacote do arquivo `prototype` os nomes de classe desejados.**

Por exemplo, a atribuição de objetos a uma classe `application` e `manpage` poderia ser semelhante a:

```
f manpage /usr/share/man/manl/myappl.1l
f application /usr/bin/myappl
```

3 **Modifique o parâmetro `CLASSES` no arquivo `pkginfo` para que contenha os nomes de classe que você deseja usar no seu pacote.**

Por exemplo, as entradas da classe `application` e `manpage` poderiam ser semelhantes a:

```
CLASSES=manpage application none
```

Observação – A classe `none` é instalada sempre primeiro e removida por último, independente de onde ela aparece na definição do parâmetro `CLASSES`.

---

4 **Se estiver criando um script de ação de classe para um arquivo que pertença à classe `sed`, `awk` ou `build`, torne o diretório que contém o objeto de pacote seu diretório de trabalho atual.**

5 **Crie os scripts de ação de classe ou objetos de pacote (em arquivos que pertençam à classe `sed`, `awk` ou `build`).**

Por exemplo, o script de instalação de uma classe denominada `application` seria denominado `i.application` e o script de remoção seria denominado `r.application`.

Lembre-se, quando um arquivo fizer parte de uma classe que tem um script de ação de classe, o script deve instalar o arquivo. O comando `pkgadd` não instala os arquivos em cada classe que tenha um script de ação de classe, embora ele verifique a instalação. E, se você definir uma classe, mas não entregar um script de ação de classe, a única ação assumida por tal classe é copiar os componentes do meio de instalação no sistema de destino (o comportamento padrão de `pkgadd`).

6 **Realize *uma* das seguintes tarefas:**

- Se *não* tiver criado o arquivo `prototype`, realize o procedimento "Como criar um arquivo `prototype` usando o comando `pkgproto`" na página 43, e vá para a Etapa 7.
- Se você já criou o arquivo `prototype`, edite-o e adicione uma entrada para cada script de instalação recém-criado.

7 **Construa o pacote.**

Consulte "Como construir um pacote" na página 46, se necessário.

Mais Informações Próximo passo

Depois de construir o pacote, instale-o para confirmar que ele é instalado corretamente e verificar sua integridade. O Capítulo 4, "Verificando e transferindo um pacote" explica como fazê-lo e oferece instruções detalhadas sobre como transferir o pacote verificado a um meio de distribuição.

Criando pacotes assinados

O processo de criação de pacotes assinados abrange várias etapas e requer um pouco de compreensão sobre novos conceitos e terminologias. Esta seção oferece informações sobre pacotes assinados, sua terminologia e sobre o gerenciamento de certificados. Esta seção oferece também procedimentos detalhados sobre como criar um pacote assinado.

Pacotes assinados

Um pacote assinado é um pacote com formato de fluxo normal que possui uma assinatura digital (assinatura digital PKCS7 codificada em PEM definida abaixo) que verifica o seguinte:

- O pacote vem da entidade que o assinou
- A entidade o assinou realmente

- O pacote não foi modificado depois que a entidade o assinou
- A entidade que o assinou é de confiança

Um pacote assinado é idêntico a um pacote não assinado, exceto pela assinatura. Um pacote assinado é compatível em binário com um pacote não assinado. Portanto, um pacote assinado pode ser usado com as versões mais antigas das ferramentas de empacotamento. No entanto, a assinatura é ignorada nesse caso.

A tecnologia de empacotamento assinado introduz algumas novas terminologias e abreviações, que estão descritas na tabela seguinte.

Termo	Definição
ASN.1	Notação de sintaxe abstrata 1 - Um forma de representar objetos abstratos. A ASN.1 define, por exemplo, um certificado de chave pública, todos os objetos que fazem parte do certificado e a ordem na qual os objetos são coletados. No entanto, a ASN.1 não específica como os objetos são serializados para armazenamento e transmissão.
X.509	Recomendação ITU-T X.509 - Especifica a sintaxe amplamente adotada de certificado de chave pública X.509.
DER	Regras de codificação diferenciadas - Uma representação binária de um objeto ASN.1 e define como um objeto ASN.1 é serializado para armazenamento e transmissão em ambientes de computação.
PEM	Mensagem de privacidade melhorada - Uma forma de codificar um arquivo (em DER ou em outro formato binário) usando a codificação de base 64 e alguns cabeçalhos opcionais. A PEM era usada inicialmente para codificação de mensagens de e-mail de tipo MIME. A PEM também é usada amplamente para codificação de certificados e chaves privadas em um arquivo que existe em um sistema de arquivos ou em uma mensagem de e-mail.
PKCS7	Criptografia de chave pública de padrão #7 - Este padrão descreve uma sintaxe geral de dados que pode apresentar criptografia aplicada a eles, tal como assinaturas e envelopes digitais. Um pacote assinado contém uma assinatura PKCS7 integrada. Esta assinatura contém como mínimo a síntese criptografada do pacote, junto com o certificado de chave pública X.509 do assinante. O pacote assinado também pode conter cadeia de certificados. A cadeia de certificados pode ser usada ao formar uma cadeia de confiança a partir do certificado do assinante até um certificado de confiança armazenado localmente.
PKCS12	Criptografia de chave pública de padrão #12 - Este padrão descreve uma sintaxe para armazenamento de objetos criptografados em disco. A chave de armazenamento de pacote é mantida neste formato.

Termo	Definição
Chave de armazenamento de pacote	Um repositório de certificados e chaves que pode ser consultado pelas ferramentas do pacote.

Gerenciamento de certificado

Antes de criar um pacote assinado, você deve ter uma chave de armazenamento de pacote. Esta chave de armazenamento de pacote contém certificados na forma de objetos. Existem dois tipos de objetos em uma chave de armazenamento de pacote:

Certificados de confiança
: Um certificado de confiança que contém um certificado único de chave pública que pertence a outra entidade. O certificado de confiança tem essa denominação porque o proprietário da chave de armazenamento confia realmente que a chave pública do certificado pertence à identidade indicada pelo "sujeito" (proprietário) do certificado. O emissor do certificado se responsabiliza por esta confiança assinando o certificado.

  Os certificados de confiança são usados na verificação de assinaturas e ao iniciar uma conexão a um servidor seguro (SSL).

Chave de usuário
: Uma chave de usuário conserva informações criptográficas importantes que diferenciam maiúsculas e minúsculas. Esta informação é armazenada em um formato protegido para evitar acesso não autorizado. Uma chave de usuário está formada pela chave privada do usuário e pelo certificado de chave pública que corresponde à chave privada.

  As chaves de usuário são usadas ao criar um pacote assinado.

Por padrão, a chave de armazenamento de pacote é armazenada no diretório `/var/sadm/security`. Os usuários individuais também podem ter suas próprias chaves de armazenamento armazenadas por padrão no diretório `$HOME/.pkg/security`.

No disco, uma chave de armazenamento de pacote pode se apresentar em dois formatos: formato de vários arquivos e formato de arquivo único. O formato de vários arquivos armazena seus objetos em vários arquivos. Cada tipo de objeto é armazenado em um arquivo diferente. Todos os arquivos devem ser criptografados usando a mesma frase-senha. Uma chave de armazenamento de arquivo único armazena todos os seus objetos em um único arquivo no sistema de arquivos.

O principal utilitário usado para gerenciar os certificados e a chave de armazenamento de pacote é o comando `pkgadm`. As subseções seguintes descrevem as tarefas mais comuns usadas para gerenciamento de chaves de armazenamento de pacote.

Adicionando certificados de confiança à chave de armazenamento de pacote

Um certificado de confiança pode ser adicionado à chave de armazenamento de pacote usando o comando `pkgadm`. O certificado pode estar no formato PEM ou DER. Por exemplo:

```
$ pkgadm addcert -t /tmp/mytrustedcert.pem
```

Neste exemplo, o certificado em formato PEM denominado `mytrustedcert.pem` é adicionado à chave de armazenamento de pacote.

Adicionando um certificado de usuário e chave privada à chave de armazenamento de pacote

O comando `pkgadm` não gera certificados de usuário nem chaves privadas. Os certificados de usuário e chaves privadas são obtidos normalmente de uma Autoridade de certificado, tal como Verisign. Ou, são gerados localmente tal como um certificado auto-assinado. Uma vez a chave e o certificado tenham sido obtidos, eles podem ser importados para a chave de armazenamento de pacote usando o comando `pkgadm`. Por exemplo:

```
pkgadm addcert -n myname -e /tmp/myprivkey.pem /tmp/mypubcert.pem
```

Neste exemplo, são usadas as seguintes opções:

`-n` *myname*	Identifica a entidade (*myname*) da chave de armazenamento de pacote na qual você deseja operar. A entidade *myname* se torna o alias no qual os objetos são armazenados.
`-e /tmp/`*myprivkey.pem*	Especifica o arquivo que contém a chave privada. Neste caso, o arquivo é *myprivkey.pem*, que está localizado no diretório `/tmp`.
`/tmp/`*mypubcert.pem*	Especifica o arquivo de certificado no formato PEM denominado *mypubcert.pem* .

Verificando o conteúdo na chave de armazenamento de pacote

O comando `pkgadm` também é usado para visualizar o conteúdo da chave de armazenamento de pacote. Por exemplo:

```
$ pkgadm listcert
```

Este comando exibe os certificados de confiança e as chaves privadas da chave de armazenamento de pacote.

Excluindo certificados de usuário e chaves privadas da chave de armazenamento de pacote

O comando `pkgadm` pode ser usado para excluir certificados de confiança e chaves privadas da chave de armazenamento de pacote.

Ao excluir os certificados de usuários, o alias do par certificado/chave deve ser especificado. Por exemplo:

```
$ pkgadm removecert -n myname
```

O alias do certificado é o nome comum do certificado, que pode ser identificado usando o comando `pkgadm listcert`. Por exemplo, este comando exclui um certificado de confiança intitulado Trusted CA Cert 1:

```
$ pkgadm removecert -n "Trusted CA Cert 1"
```

Observação – Se você possuir tanto um certificado de confiança quanto um certificado de usuário armazenado usando o mesmo alias, ambos são excluídos quando a opção `-n` for especificada.

Criação de pacotes assinados

O processo de criação de pacotes assinados apresenta três etapas básicas:

1. A criação de um pacote não assinado no formato de diretório.
2. A importação do certificado assinado, certificados CA, e da chave privada para a chave de armazenamento de pacote.
3. A assinatura do pacote da Etapa 1 com os certificados da Etapa 2.

Observação – As ferramentas de empacotamento não criam certificados. Estes certificados devem ser obtidos de uma Autoridade de certificado, tal como Verisign ou Thawte.

Cada etapa da criação de pacotes assinados está descrita nos procedimentos seguintes.

▼ Como criar um pacote não assinado no formato de diretório

O procedimento para a criação de um pacote não assinado no formato de diretório é igual ao procedimento para a criação de um pacote normal, conforme descrito previamente neste manual. O procedimento seguinte descreve o processo de criação do pacote não assinado no formato de diretório. Se você precisar de mais informações, consulte as seções anteriores sobre a construção de pacotes.

1 Crie o arquivo `pkginfo`.

O arquivo `pkginfo` deve ter o seguinte conteúdo básico:

```
PKG=SUNWfoo
BASEDIR=/
NAME=My Test Package
ARCH=sparc
VERSION=1.0.0
CATEGORY=application
```

2 Crie um arquivo `prototype`.

O arquivo `prototype` deve ter o seguinte conteúdo básico:

```
$cat prototype
i pkginfo
d none usr 0755 root sys
d none usr/bin 0755 root bin
f none usr/bin/myapp=/tmp/myroot/usr/bin/myapp 0644 root bin
```

3 Liste o conteúdo do diretório de origem do objeto.

Por exemplo:

```
$ ls -lR /tmp/myroot
```

A saída seria semelhante à seguinte:

```
/tmp/myroot:
total 16
drwxr-xr-x   3 abc      other        177 Jun  2 16:19 usr

/tmp/myroot/usr:
total 16
drwxr-xr-x   2 abc      other        179 Jun  2 16:19 bin

/tmp/myroot/usr/bin:
total 16
-rw-------   1 abc      other       1024 Jun  2 16:19 myapp
```

4 Crie o pacote não assinado.

```
pkgmk -d ‘pwd‘
```

A saída seria semelhante à seguinte:

```
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter <PSTAMP> set to "syrinx20030605115507"
WARNING: parameter <CLASSES> set to "none"
## Attempting to volumize 3 entries in pkgmap.
part  1 -- 84 blocks, 7 entries
## Packaging one part.
/tmp/SUNWfoo/pkgmap
/tmp/SUNWfoo/pkginfo
/tmp/SUNWfoo/reloc/usr/bin/myapp
## Validating control scripts.
## Packaging complete.
```

O pacote existe agora no diretório atual.

▼ Como importar os certificados para a chave de armazenamento de pacote

O certificado e a chave privada que serão importados devem existir como chave privada e certificado X.509 codificado em DER ou PEM. Além disso, o intermediário ou a "cadeia" de certificados que unem seu certificado assinado à Autoridade de certificado devem ser importados para a chave de armazenamento de pacote antes que um pacote seja assinado.

Observação – Cada Autoridade de certificado pode emitir certificados em vários formatos. Para extrair os certificados e a chave privada do arquivo PKCS12 e colocar no arquivo X.509 codificado em PEM (apropriado para a importação à chave de armazenamento de pacote), use um utilitário de conversão de software gratuito como o OpenSSL.

Se a sua chave privada estiver criptografada (que deveria ser geralmente o caso), você é solicitado a introduzir uma frase-senha. Você também será solicitado a introduzir uma senha para proteger a chave de armazenamento de pacote resultante. Você tem a opção de não fornecer a senha, mas como conseqüência a chave de armazenamento de pacote não será criptografada.

O procedimento seguinte descreve como importar os certificados usando o comando `pkgadm` uma vez o certificado esteja no formato apropriado.

1 **Importe todos os certificados de Autoridade de certificado encontrados no seu arquivo de certificado X.509 codificado em DER ou PEM.**

Por exemplo, para importar todos os certificados de Autoridade de certificado encontrados no arquivo `ca.pem`, você deve digitar o seguinte:

```
$ pkgadm addcert -k ~/mykeystore -ty ca.pem
```

A saída seria semelhante à seguinte:

```
Trusting certificate <VeriSign Class 1 CA Individual \
Subscriber-Persona Not Validated>
Trusting certificate </C=US/O=VeriSign, Inc./OU=Class 1 Public \
Primary Certification Authority
Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
For Verification: Type a Keystore protection Password.
Press ENTER for no protection password (not recommended):
Certificate(s) from <ca.pem> are now trusted
```

A fim de importar sua chave assinada para a chave de armazenamento de pacote, você deve fornecer um alias que será usado mais tarde ao assinar o pacote. Este alias também pode ser usado se quiser excluir a chave da chave de armazenamento de pacote.

Por exemplo, para importar sua chave assinada do arquivo `sign.pem`, você deve digitar o seguinte:

```
$ pkgadm addcert -k ~/mykeystore -n mycert sign.pem
```

A saída seria semelhante à seguinte:

```
Enter PEM passphrase:
Enter Keystore Password:
Successfully added Certificate <sign.pem> with alias <mycert>
```

2 **Verifique se os certificados estão na chave de armazenamento de pacote.**

Por exemplo, para visualizar os certificados na chave de armazenamento criada na etapa anterior, você deve digitar o seguinte:

```
$ pkgadm listcert -k ~/mykeystore
```

▼ Como assinar o pacote

Uma vez os certificados tenham sido importados para a chave de armazenamento de pacote, você pode assinar o pacote. A assinatura real do pacote é realizada com o comando `pkgtrans`.

- **Assine o pacote usando o comando `pkgtrans`. Forneça o local do pacote não assinado e o alias da chave para assinar o pacote.**

Por exemplo, usando os exemplos dos procedimentos anteriores, você deve digitar o seguinte para criar um pacote assinado denominado `SUNWfoo.signed`:

```
$ pkgtrans -g -k ~/mykeystore -n mycert . ./SUNWfoo.signed SUNWfoo
```

A saída deste comando seria semelhante à seguinte:

```
Retrieving signing certificates from keystore </home/user/mykeystore>
Enter keystore password:
Generating digital signature for signer <Test User>
Transferring <SUNWfoot> package instance
```

O pacote assinado é criado no arquivo `SUNWfoo.signed` e está no formato de fluxo de pacote. Este pacote assinado é apropriado para copiar em um site e ser instalado usando o comando `pkgadd` e um URL.

◆ ◆ ◆ CAPÍTULO 4

Verificando e transferindo um pacote

Este capítulo descreve como verificar a integridade do seu pacote e como transferi-lo a um meio de distribuição, tal como um disquete ou um CD-ROM.

A lista abaixo traz as informações gerais encontradas neste capítulo.

Verificando e transferindo um pacote (Mapa de tarefas)

A tabela abaixo descreve as etapas que você deve seguir para verificar a integridade de seu pacote e transferi-lo para uma mídia de distribuição.

TABELA 4–1 Mapa de tarefas da verificação e transferência de um pacote

Tarefa	Descrição	Instruções
1. Construir o pacote	Construa o pacote no disco.	Capítulo 2, "Construindo um Pacote"
2. Instalar o pacote	Teste o pacote instalando-o e comprove que a instalação foi realizada sem erros.	"Como instalar um pacote em um sistema ou servidor independente" na página 89
3. Verificar a integridade do pacote	Use o comando `pkgchk` para verificar a integridade do pacote.	"Como verificar a integridade de um pacote" na página 91
4. Obter outras informações do pacote	*Opcional*. Use os comandos `pkginfo` e `pkgparam` para realizar verificações específicas no pacote.	"Exibindo informações adicionais sobre pacotes instalados" na página 92

TABELA 4–1 Mapa de tarefas da verificação e transferência de um pacote *(Continuação)*

Tarefa	Descrição	Instruções
5. Remover o pacote instalado	Use o comando `pkgrm` para remover o pacote instalado do sistema.	"Como remover um pacote" na página 97
6. Transferir o pacote para um meio de distribuição	Use o comando `pkgtrans` para transferir o pacote (no formato de pacote) para um meio de distribuição.	"Como transferir um pacote para um meio de distribuição" na página 98

Instalando os pacotes de software

Os pacotes de software são instalados usando o comando `pkgadd`. Este comando transfere o conteúdo de um pacote de software do meio de distribuição ou diretório e o instala em um sistema.

Esta seção oferece instruções básicas de instalação do pacote a fim de verificar se a instalação é realizada corretamente.

O banco de dados do software de instalação

As informações de todos os pacotes instalados no sistema são mantidas no banco de dados do software de instalação. Em um pacote, há uma entrada para cada objeto, com informações como o nome do componente, onde ele está estabelecido e o seu tipo. Uma entrada contém um registro do pacote ao qual o componente pertence, outros pacotes que devem fazer referência ao componente e informações como o nome do caminho, onde o componente está estabelecido e o tipo do componente. As entradas são adicionadas e removidas automaticamente pelos comandos `pkgadd` e `pkgrm`. É possível ver as informações no banco de dados com os comandos `pkgchk` e `pkginfo`.

Dois tipos de informações estão associados com cada componente do pacote. As informações do atributo descrevem o componente em si. Por exemplo, as permissões de acesso do componente, o ID de proprietário e o ID de grupo são informações do atributo. As informações descrevem o conteúdo do componente, tal como o tamanho do arquivo e a data da última modificação.

O banco de dados do software de instalação mantém um controle do status do pacote. Um pacote pode ser instalado completamente (o processo de instalação é completado com sucesso) ou instalado parcialmente (o processo de instalação não é completado com sucesso).

Quando um pacote é instalado parcialmente, partes do pacote podem ter sido instaladas antes que a instalação fosse concluída, conseqüentemente, uma parte do pacote é instalada e registrada no banco de dados e outra parte não. Quando o pacote é reinstalado, você é solicitado a iniciar do ponto onde a instalação foi interrompida porque o comando `pkgadd` pode acessar o banco de dados e detectar que partes já foram instaladas. Também é possível remover as partes que foram instaladas com base nas informações do banco de dados do software de instalação usando o comando `pkgrm`.

Interagindo com o comando pkgadd

Se o comando pkgadd encontrar um problema, ele verifica o arquivo de administração da instalação em busca de instruções. (Consulte admin(4) para obter mais informações.) Se não houver nenhuma instrução ou se o parâmetro relevante no arquivo de administração estiver definido como ask, o pkgadd exibe uma mensagem descrevendo o problema e solicita uma resposta. A solicitação é geralmente Deseja continuar esta instalação?. Você deve responder yes, no ou quit.

Se você tiver especificado mais de um pacote, o no interrompe a instalação do pacote que está sendo instalado, mas pkgadd continua a instalação dos outros pacotes. quit indica que pkgadd deve interromper a instalação de todos os pacotes.

Instalando pacotes em sistemas ou servidores independentes em um ambiente homogêneo

Esta seção descreve como instalar os pacotes em um sistema ou servidor independente em um ambiente homogêneo.

▼ Como instalar um pacote em um sistema ou servidor independente

1. **Construa o pacote.**

   Consulte "Construindo um Pacote" na página 45, se necessário.

2. **Efetue log-in no sistema como superusuário.**

3. **Adicione um pacote de software ao sistema.**

   ```
   # pkgadd -d device-name [pkg-abbrev...]
   ```

-d *device-name*	Especifique o local do pacote. Observe que *device-name* pode ser o nome completo do caminho de um diretório ou os identificadores de uma fita, um disquete ou um disco removível.
pkg-abbrev	É o nome de um ou mais pacotes (separado por espaços) que serão adicionados. Se for omitido, o pkgadd instala todos os pacotes disponíveis.

Exemplo 4–1 Instalando os pacotes em servidores e sistemas independentes

Para instalar um pacote de software chamado `pkgA` de um fita chamada `/dev/rmt/0`, você deve inserir o comando seguinte:

```
# pkgadd -d /dev/rmt/0 pkgA
```

Você também pode instalar vários pacotes ao mesmo tempo, contanto que os nomes dos pacotes sejam separados por espaços, da seguinte forma:

```
# pkgadd -d /dev/rmt/0 pkgA pkgB pkgC
```

Se você não nomear o dispositivo no qual o pacote reside, o comando verifica o diretório spool padrão (`/var/spool/pkg`). Se o pacote não estiver lá, a instalação falha.

Consulte também Se você já estiver preparado para ir para a próxima tarefa, consulte "Como verificar a integridade de um pacote" na página 91.

Verificando a integridade de um pacote

O comando `pkgchk` permite verificar a integridade de pacotes, se eles estão instalados em um sistema ou em um formato de pacote (pronto para ser instalado com o comando `pkgadd`). Ele confirma a estrutura do pacote ou os arquivos e diretórios instalados, ou exibe informações sobre os objetos do pacote. O comando `pkgchk` pode listar ou verificar o seguinte:

- Os scripts de instalação do pacote.
- O conteúdo ou os atributos, ou ambos, dos objetos atualmente instalados no sistema.
- O conteúdo de um pacote em spool desinstalado.
- O conteúdo ou atributos, ou ambos, dos objetos descritos no arquivo `pkgmap`.

Para obter mais informações sobre este comando, consulte `pkgchk(1M)`.

O comando `pkgchk` realiza dois tipos de verificação. Verifica os atributos do arquivo (as permissões e a propriedade de um arquivo e os números maior/menor de dispositivos especiais de caracteres ou bloco) e o conteúdo do arquivo (o tamanho, a soma de verificação e a data de modificação). Por padrão, o comando verifica os atributos e o conteúdo do arquivo.

O comando `pkgchk` também compara os atributos e o conteúdo do arquivo do pacote instalado com o banco de dados do software de instalação. As entradas referentes a um pacote podem ter sido alteradas desde o momento em que tal pacote foi instalado. Por exemplo, outro pacote pode ter alterado o componente de tal pacote. O banco de dados reflete tal alteração.

▼ Como verificar a integridade de um pacote

1 Instale o pacote.

Consulte "Como instalar um pacote em um sistema ou servidor independente" na página 89, se necessário.

2 Verifique a integridade do pacote.

```
# pkgchk [-v] [-R root-path] [pkg-abbrev...]
```

`-v`	Lista os arquivos à medida que são processados.
`-R` *root-path*	Especifica o local do sistema de arquivos raiz do sistema do cliente.
pkg-abbrev	É o nome de um ou mais pacotes (separado por espaços) que serão verificados. Se for omitido, o `pkgchk` verifica todos os pacotes disponíveis.

Exemplo 4–2 Verificando a integridade de um pacote

Este exemplo mostra o comando que você deve usar para verificar a integridade de um pacote instalado.

```
$ pkgchk pkg-abbrev
$
```

Se houver erros, o comando `pkgchk` os imprime. Do contrário, ele não imprime nada e retorna um código de saída de 0. Se você não fornecer a abreviatura de um pacote, ele verificará todos os pacotes do sistema.

Outra alternativa é usar a opção `-v`, que imprimirá uma lista de arquivos do pacote se não houver erros. Por exemplo:

```
$ pkgchk -v SUNWcadap
/opt/SUNWcadap
/opt/SUNWcadap/demo
/opt/SUNWcadap/demo/file1
/opt/SUNWcadap/lib
/opt/SUNWcadap/lib/file2
/opt/SUNWcadap/man
/opt/SUNWcadap/man/man1
/opt/SUNWcadap/man/man1/file3.1
/opt/SUNWcadap/man/man1/file4.1
/opt/SUNWcadap/man/windex
/opt/SUNWcadap/srcfiles
/opt/SUNWcadap/srcfiles/file5
/opt/SUNWcadap/srcfiles/file6
$
```

Se precisar verificar um pacote instalado no sistema de arquivos raiz de um sistema do cliente, use este comando:

```
$ pkgchk -v -R root-path pkg-abbrev
```

Consulte também Se estiver preparado para ir para a próxima tarefa, consulte "Como obter informações com o comando pkginfo" na página 96.

Exibindo informações adicionais sobre pacotes instalados

Você pode usar outros dois comandos para exibir as informações sobre os pacotes instalados:

- O comando `pkgparam` exibe os valores dos parâmetros.
- O comando `pkginfo` exibe informações do banco de dados do software de instalação.

O comando `pkgparam`

O comando `pkgparam` permite exibir os valores associados aos parâmetros que você especificou na linha de comando. Os valores são recuperados do arquivo `pkginfo` de um pacote específico ou do arquivo cujo nome foi dado. É exibido um valor de parâmetro por linha. É possível exibir somente os valores ou os parâmetros e seus valores.

▼ Como obter informações com o comando `pkgparam`

1 Instale o pacote.

Consulte "Como instalar um pacote em um sistema ou servidor independente" na página 89, se necessário.

2 Exiba as informações adicionais sobre o pacote.

```
# pkgparam [-v] pkg-abbrev [param...]
```

`-v`	Exibe o nome do parâmetro e seu valor.
pkg-abbrev	É o nome de um pacote específico.
param	Especifica um ou mais parâmetros cujo valor é exibido.

Exemplo 4–3 Obtendo informações com o comando `pkgparam`

Por exemplo, para exibir somente os valores, use este comando.

```
$ pkgparam SUNWcadap
none
/opt
```

```
US/Mountain
/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin
/usr/sadm/sysadm
SUNWcadap
Chip designers need CAD application software to design abc
chips.  Runs only on xyz hardware and is installed in the usr
partition.
system
release 1.0
SPARC
venus990706083849
SUNWcadap
/var/sadm/pkg/SUNWcadap/save
Jul 7 1999 09:58
$
```

Para exibir os parâmetros e seus valores, use o comando a seguir.

```
$ pkgparam -v SUNWcadap
pkgparam -v SUNWcadap
CLASSES='none'
BASEDIR='/opt'
TZ='US/Mountain'
PATH='/sbin:/usr/sbin:/usr/bin:/usr/sadm/install/bin'
OAMBASE='/usr/sadm/sysadm'
PKG='SUNWcadap'
NAME='Chip designers need CAD application software to design abc chips.
Runs only on xyz hardware and is installed in the usr partition.'
CATEGORY='system'
VERSION='release 1.0'
ARCH='SPARC'
PSTAMP='venus990706083849'
PKGINST='SUNWcadap'
PKGSAV='/var/sadm/pkg/SUNWcadap/save'
INSTDATE='Jul 7 1999 09:58'
$
```

Ou, se desejar exibir o valor de um parâmetro específico, use este formato:

```
$ pkgparam SUNWcadap BASEDIR
/opt
$
```

Para obter mais informações, consulte `pkgparam(1)`.

Consulte também Se você já estiver preparado para ir para a próxima tarefa, consulte "Como remover um pacote" na página 97.

O comando `pkginfo`

Você pode exibir informações sobre os pacotes instalados com o comando `pkginfo`. Este comando possui várias opções que permitem personalizar o formato e o conteúdo da exibição.

Você pode solicitar informações sobre qualquer quantidade de instâncias de pacotes.

A exibição padrão do `pkginfo`

Quando o comando `pkginfo` é executado sem opções, ele exibe a categoria, a instância e o nome de todos os pacotes que estiverem completamente instalados no sistema. A exibição é ordenada por categorias conforme mostra o exemplo a seguir.

```
$ pkginfo
.
.
.
system      SUNWinst       Install Software
system      SUNWipc        Interprocess Communications
system      SUNWisolc      XSH4 conversion for ISO Latin character sets
application SUNWkcspf      KCMS Optional Profiles
application SUNWkcspg      KCMS Programmers Environment
application SUNWkcsrt      KCMS Runtime Environment
.
.
.
$
```

Personalizando o formato da exibição de `pkginfo`

Você pode ter a exibição de `pkginfo` em três formatos: breve, extraído e longo.

O formato breve é o padrão. Mostra somente a categoria, a abreviatura do pacote e o nome completo do pacote, conforme ilustrado em "A exibição padrão do `pkginfo`" na página 94.

O formato extraído mostra a abreviatura, o nome, a arquitetura (se disponível) e a versão (se disponível) do pacote. Use a opção `-x` para solicitar o formato extraído conforme ilustrado no próximo exemplo.

```
$ pkginfo -x
.
.
.
SUNWipc         Interprocess Communications
                (sparc) 11.8.0,REV=1999.08.20.12.37
SUNWisolc       XSH4 conversion for ISO Latin character sets
                (sparc) 1.0,REV=1999.07.10.10.10
SUNWkcspf       KCMS Optional Profiles
                (sparc) 1.1.2,REV=1.5
SUNWkcspg       KCMS Programmers Environment
                (sparc) 1.1.2,REV=1.5
.
.
.
$
```

O uso da opção `-l` proporciona a exibição no formato longo, mostrando todas as informações disponíveis sobre um pacote, conforme ilustra o exemplo a seguir.

```
$ pkginfo -l SUNWcadap
   PKGINST:  SUNWcadap
      NAME:  Chip designers need CAD application software to
```

```
design abc chips.  Runs only on xyz hardware and is installed
in the usr partition.
  CATEGORY:  system
      ARCH:  SPARC
   VERSION:  release 1.0
   BASEDIR:  /opt
    PSTAMP:  system980706083849
  INSTDATE:  Jul 7 1999 09:58
    STATUS:  completely installed
     FILES:     13 installed pathnames
                 6 directories
                 3 executables
              3121 blocks used (approx)
$
```

Descrições de parâmetros no formato longo de `pkginfo`

A tabela abaixo descreve os parâmetros que podem ser exibidos de cada pacote. O parâmetro e o seu valor são exibidos somente quando o parâmetro tem um valor atribuído a ele.

TABELA 4–2 Parâmetros do pacote

Parâmetro	Descrição
ARCH	A arquitetura suportada por este pacote.
BASEDIR	O diretório base no qual se encontra o pacote de software (mostrado se o pacote for relocável).
CATEGORY	A categoria, ou categorias, do software de qual este pacote é membro (por exemplo, `system` ou `application`).
CLASSES	Lista de classes definida de um pacote. A ordem da lista determina a ordem na qual as classes serão instaladas. A classes listadas primeiro serão instaladas primeiro (mídia por mídia). Este parâmetro pode ser modificado pelo script de solicitação.
DESC	Texto que descreve o pacote.
EMAIL	O endereço eletrônico para as perguntas do usuário.
HOTLINE	Informações sobre como receber ajuda direta sobre este pacote.
INTONLY	Indica que pacote deve ser instalado somente de forma interativa quando definido com um valor não nulo.
ISTATES	Lista de estados de execução admissíveis para a instalação do pacote (por exemplo, `S s 1`).
MAXINST	O número máximo de instâncias do pacote que deve ser permitido em uma máquina ao mesmo tempo. Por padrão, é permitida somente uma instância de um pacote.
NAME	O nome do pacote, geralmente um texto descrevendo a abreviatura do pacote.

TABELA 4–2 Parâmetros do pacote *(Continuação)*

Parâmetro	Descrição
ORDER	Lista de classes que define a ordem na qual elas devem ser colocadas no meio. Usado pelo comando `pkgmk` na criação do pacote. As classes não definidas neste parâmetro são colocadas em um meio usando os procedimentos de ordenação padrão.
PKGINST	Abreviatura do pacote que está sendo instalado.
PSTAMP	O carimbo de produção deste pacote.
RSTATES	Lista de estados de execução admissíveis para a remoção do pacote (por exemplo, `S s 1`).
ULIMIT	Se definido, este parâmetro é passado como um argumento para o comando `ulimit`, que estabelece o tamanho máximo de um arquivo durante a instalação. Aplica-se somente a arquivos criados por scripts de procedimento.
VENDOR	O nome do fornecedor que forneceu o pacote de software.
VERSION	A versão deste pacote.
VSTOCK	Número de lote fornecido pelo fornecedor.

Para obter informações detalhadas sobre o comando `pkginfo`, consulte a página do manual `pkginfo(1)`.

▼ Como obter informações com o comando `pkginfo`

1 Instale o pacote.

Consulte "Como instalar um pacote em um sistema ou servidor independente" na página 89, se necessário.

2 Exiba as informações adicionais sobre o pacote.

```
# pkginfo [-x | -l] [pkg-abbrev]
```

`-x`	Exibe as informações do pacote em formato extraído.
`-l`	Exibe as informações do pacote em formato longo.
pkg-abbrev	É o nome de um pacote específico. Se for omitido, o comando `pkginfo` exibe as informações sobre todos os pacotes instalados no formato padrão.

Mais Informações

Próximo passo

Se você já estiver preparado para ir para a próxima tarefa, consulte "Como remover um pacote" na página 97.

Removendo um pacote

Por atualizar as informações do banco de dados dos produtos de software, o uso do comando `pkgrm` é importante ao remover um pacote, embora você possa se sentir tentado a usar o comando `rm`. Você pode, por exemplo, usar o comando `rm` para remover um arquivo executável binário, mas não é o mesmo usar o comando `pkgrm` para remover o pacote de software que inclui tal executável binário. O uso do comando `rm` para remover os arquivos do pacote corromperá o banco de dados dos produtos do software. (Se quiser realmente remover apenas um arquivo, você pode usar o comando `removef`, que atualizará o banco de dados do produto de software corretamente.

▼ Como remover um pacote

1 **Efetue log-in no sistema como superusuário.**

2 **Remova um pacote instalado.**

```
# pkgrm pkg-abbrev ...
```

pkg-abbrev	É o nome de um ou mais pacotes (separado por espaços). Se for omitido, o `pkgrm` remove todos os pacotes disponíveis.

3 **Para verificar se o pacote foi removido com sucesso, use o comando `pkginfo`.**

```
$ pkginfo | egrep pkg-abbrev
```

Se *pkg-abbrev*estiver instalado, o comando `pkginfo` retorna uma linha de informação sobre ele. Do contrário, o `pkginfo` retorna uma solicitação do sistema.

Transferindo um pacote para um meio de distribuição

O comando `pkgtrans` move os pacotes e realiza traduções de formato do pacote. Você pode usar o comando `pkgtrans` para realizar as seguintes traduções de um pacote instalável:

- Do formato de sistema de arquivos para o formato de fluxo de dados
- Do formato de fluxo de dados para o formato de sistema de arquivos
- De um formato do sistema de arquivos para outro formato do sistema de arquivos

▼ Como transferir um pacote para um meio de distribuição

1 **Construa o pacote, criando um pacote de formato de diretório, se ainda não tiver criado.**

Para obter mais informações, consulte "Como construir um pacote" na página 46.

2 **Instale o pacote para verificar se ele é instalado corretamente.**

Consulte "Como instalar um pacote em um sistema ou servidor independente" na página 89, se necessário.

3 **Verifique a integridade do pacote.**

Consulte "Como verificar a integridade de um pacote" na página 91, "Como obter informações com o comando `pkginfo`" na página 96 e "Como obter informações com o comando `pkgparam`" na página 92, se necessário.

4 **Remova o pacote instalado do sistema.**

Consulte "Como remover um pacote" na página 97, se necessário.

5 **Transfira o pacote (em formato de pacote) para um meio de distribuição.**

Para realizar uma transferência básica, execute o comando seguinte:

```
$ pkgtrans device1 device2 [pkg-abbrev...]
```

device1	É o nome do dispositivo no qual o pacote está atualmente.
device2	É o nome do dispositivo no qual o pacote transferido será gravado.
[*pkg-abbrev*]	É uma ou mais abreviaturas de pacotes.

Se nenhum nome for fornecido, todos os pacotes que estão no *device1* serão transferidos e gravados no *device2* .

Observação – Se mais de uma instância de um pacote estiver no *device1*, você deve usar um identificador de instância do pacote. Para obter uma descrição de um identificador de instâncias, consulte "Definindo uma instância de pacote" na página 27. Quando uma instância do pacote que está sendo transferido já existir no *device2*, o comando `pkgtrans` não realiza a transferência. Você pode usar a opção `-o` para fazer com que o comando `pkgtrans` substitua as instâncias existentes no dispositivo de destino e a opção `-n` para que ele crie uma nova instância se já existir uma. Observe que esta verificação não se aplica quando o *device2* oferece suporte ao formato de fluxo de dados.

Mais Informações

Próximo passo

Neste ponto você completou as etapas necessárias para criar, construir, verificar e transferir seu pacote. Se estiver interessado em analisar alguns estudos, consulte o Capítulo 5, “Estudos de caso de criação de pacote”. Se estiver interessado em idéias avançadas de criação de pacotes, consulte o Capítulo 6, “Técnicas avançadas para a criação de pacotes”.

◆ ◆ ◆ CAPÍTULO 5

Estudos de caso de criação de pacote

Este capítulo oferece estudos de caso para mostrar exemplos de empacotamento tais como instalar objetos condicionalmente, determinar em tempo de execução quantos arquivos criar e modificar um arquivo de dados existente durante a instalação e a remoção do pacote.

Cada estudo de caso começa com uma descrição, seguida de uma lista das técnicas de empacotamento usadas, uma descrição narrativa da abordagem adotada ao usar tais técnicas e scripts e arquivos de amostra associados ao estudo de caso.

A lista abaixo traz os estudos de caso encontrados neste capítulo.

Solicitando entrada do administrador

O pacote deste estudo de caso possui três tipos de objetos. O administrador pode escolher qual dos três tipos instalar e onde colocar os objetos na máquina de instalação.

Técnicas

Este estudo de caso demonstra as seguintes técnicas:

- Uso de nomes de caminho paramétricos (variáveis em nomes de caminho do objeto) usados para estabelecer vários diretórios base

Para obter informações sobre nomes de caminho paramétricos, consulte "Nomes de caminho paramétrico" na página 34.

- Uso de um script `request` para solicitar entrada do administrador

  Para obter informações sobre scripts `request`, consulte "Escrevendo um script `request`" na página 63.
- Definição de valores condicionais para um parâmetro de instalação.

Abordagem

Para configurar a instalação seletiva neste estudo de caso, você deve concluir as seguintes tarefas:

- Definir uma classe para cada tipo de objeto que pode ser instalado.

  Neste estudo de caso, os três tipos de objetos são as executáveis do pacote, as páginas do manual e as executáveis `emacs`. Cada tipo possui sua própria classe: `bin`, `man` e `emacs`, respectivamente. Observe que no arquivo `prototype` todos os arquivos do objeto pertencem a uma dessas três classes.
- Inicializar o parâmetro `CLASSES` no arquivo `pkginfo` como nulo.

  Normalmente, quando uma classe é definida, ela deve ser listada no parâmetro `CLASSES` no arquivo `pkginfo`. Do contrário, nenhum objeto é instalado em tal classe. Para este estudo de caso, o parâmetro é inicialmente definido como nulo, o que significa que nenhum objeto será instalado. O parâmetro `CLASSES` será alterado pelo script `request`, com base nas escolhas do administrador. Desta forma, o parâmetro `CLASSES` é definido somente para os tipos de objetos que o administrador deseja instalar.

  Observação – Em geral, é uma boa idéia definir os parâmetros com um valor padrão. Se este pacote tiver componentes comuns aos três tipos de objetos, você pode atribuí-los a classe `none` e, então, definir o parâmetro `CLASSES` igual a `none`.

- Inserir nomes de caminho paramétricos no arquivo `prototype`.

  O script `request` define estas variáveis de ambiente com o valor fornecido pelo administrador. Nesse caso, o comando `pkgadd` resolve estas variáveis de ambiente no tempo de instalação e sabe onde instalar o pacote.

  As três variáveis de ambiente usadas deste exemplo estão definidas com seus valores padrão no arquivo `pkginfo` e servem para os seguintes propósitos:

  - `$NCMPBIN` define o local das executáveis do objeto
  - `$NCMPMAN` define o local das páginas do manual
  - `$EMACS` define o local das executáveis `emacs`

O arquivo `prototype` do exemplo mostra como definir os nomes de caminho do objeto com as variáveis.

- Criar um script `request` para perguntar ao administrador que partes do pacote devem ser instaladas e onde devem ser colocadas.

  O script `request` deste pacote faz duas perguntas ao administrador:

  - Esta parte do pacote deve ser instalada?

    Quando a resposta for sim, o nome de classe apropriado é adicionado ao parâmetro `CLASSES`. Por exemplo, quando o administrador opta por instalar as páginas do manual associadas a este pacote, a classe `man` é adicionada ao parâmetro `CLASSES`.

  - Se afirmativo, onde esta parte deve ser colocada?

    A variável de ambiente apropriada é definida de acordo com a resposta a esta pergunta. No exemplo da página do manual, a variável `$NCMPMAN` é definida como o valor da resposta.

  Estas duas perguntas são repetidas em cada um dos três tipos de objetos.

  No final do script `request`, os parâmetros são disponibilizados para o ambiente de instalação do comando `pkgadd` e qualquer outro script de empacotamento. O script `request` faz isso gravando estas definições no arquivo fornecido pelo utilitário de chamada. Para este estudo de caso, não é fornecido nenhum outro script.

  Ao observar o script `request` deste estudo de caso, note que as perguntas são geradas pelas ferramentas de validação de dados `ckyorn` e `ckpath`. Para obter mais informações sobre estas ferramentas, consulte `ckyorn(1)` e `ckpath(1)`.

Arquivos de estudo de caso

O arquivo `pkginfo`

```
PKG=ncmp
NAME=NCMP Utilities
CATEGORY=application, tools
BASEDIR=/
ARCH=SPARC
VERSION=RELEASE 1.0, Issue 1.0
CLASSES=""
NCMPBIN=/bin
NCMPMAN=/usr/man
EMACS=/usr/emacs
```

O arquivo `prototype`

```
i pkginfo
i request
x bin $NCMPBIN 0755 root other
```

```
f bin $NCMPBIN/dired=/usr/ncmp/bin/dired 0755 root other
f bin $NCMPBIN/less=/usr/ncmp/bin/less 0755 root other
f bin $NCMPBIN/ttype=/usr/ncmp/bin/ttype 0755 root other
f emacs $NCMPBIN/emacs=/usr/ncmp/bin/emacs 0755 root other
x emacs $EMACS 0755 root other
f emacs $EMACS/ansii=/usr/ncmp/lib/emacs/macros/ansii 0644 root other
f emacs $EMACS/box=/usr/ncmp/lib/emacs/macros/box 0644 root other
f emacs $EMACS/crypt=/usr/ncmp/lib/emacs/macros/crypt 0644 root other
f emacs $EMACS/draw=/usr/ncmp/lib/emacs/macros/draw 0644 root other
f emacs $EMACS/mail=/usr/ncmp/lib/emacs/macros/mail 0644 root other
f emacs $NCMPMAN/man1/emacs.1=/usr/ncmp/man/man1/emacs.1 0644 root other
d man $NCMPMAN 0755 root other
d man $NCMPMAN/man1 0755 root other
f man $NCMPMAN/man1/dired.1=/usr/ncmp/man/man1/dired.1 0644 root other
f man $NCMPMAN/man1/ttype.1=/usr/ncmp/man/man1/ttype.1 0644 root other
f man $NCMPMAN/man1/less.1=/usr/ncmp/man/man1/less.1 0644 inixmr other
```

O script `request`

```
trap 'exit 3' 15
# determine if and where general executables should be placed
ans=‘ckyorn -d y \
-p "Should executables included in this package be installed"
‘ || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES bin"
   NCMPBIN=‘ckpath -d /usr/ncmp/bin -aoy \
   -p "Where should executables be installed"
   ‘ || exit $?
fi
# determine if emacs editor should be installed, and if it should
# where should the associated macros be placed
ans=‘ckyorn -d y \
-p "Should emacs editor included in this package be installed"
‘ || exit $?
if [ "$ans" = y ]
then
   CLASSES="$CLASSES emacs"
   EMACS=‘ckpath -d /usr/ncmp/lib/emacs -aoy \
   -p "Where should emacs macros be installed"
   ‘ || exit $?
fi
```

Observe que um script `request` pode sair sem deixar nenhum arquivo no sistema de arquivos. Em instalações em versões do Oracle Solaris anteriores à 2.5 e versões compatíveis (onde nenhum script `checkinstall` pode ser usado), o script `request` é o lugar correto para testar o sistema de arquivos de todas as formas necessárias para garantir que a instalação será bem-sucedida. Quando o script `request` existir com o código 1, a instalação sairá perfeitamente.

Estes arquivos de exemplo mostram o uso de caminhos paramétricos para estabelecer vários diretórios base. No entanto, o método preferido envolve o uso do parâmetro `BASEDIR` que é gerenciado e validado pelo comando `pkgadd`. Sempre que vários diretórios base forem usados, tenha especial cuidado em prover a instalação de várias versões e arquiteturas na mesma plataforma.

Criando um arquivo na instalação e salvando-o durante a remoção

Este estudo de caso cria um arquivo de banco de dados no tempo de instalação e salva um cópia do banco de dados quando o pacote é removido.

Técnicas

Este estudo de caso demonstra as seguintes técnicas:

- Uso de classes e script de ação de classe para realizar ações especiais em diferentes conjuntos de objetos

  Para obter informações, consulte "Escrevendo scripts de ação de classe" na página 70.
- Uso do arquivo `space` para informar o comando `pkgadd` que é necessário espaço extra para instalar este pacote adequadamente

  Para mais informações sobre o arquivo `space`, consulte "Reservando espaço adicional em um sistema de destino" na página 57.
- Uso do comando `installf` para instalar um arquivo não definido nos arquivos `prototype` e `pkgmap`.

Abordagem

Para criar um arquivo de banco de dados na instalação e salvar uma cópia na remonação neste estudo de caso, você deve realizar as tarefas seguintes:

- Definir três classes.

  O pacote deste estudo de caso requer que as três classes seguintes sejam definidas no parâmetro `CLASSES`:

  - A classe `none` padrão, que contém um conjunto de processos pertencente ao subdiretório `bin`.
  - A classe `admin`, que contém um arquivo executável `config` e um diretório contendo arquivos de dados.
  - A classe `cfgdata`, que contém um diretório.
- Tornar o pacote relocável coletivamente.

  Observe que no arquivo `prototype` nenhum dos nomes de caminho começa com um barra ou uma variável de ambiente. Isso indica que são relocáveis coletivamente.
- Calcular a quantidade de espaço que o arquivo de banco de dados necessita e criar um arquivo `space` para ser distribuído com o pacote. Este arquivo notifica o comando `pkgadd` que o pacote requer espaço extra e especifica quanto.

- Criar um script de ação de classe para a classe `admin` (`i.admin`).

  O script de amostra inicializa um banco de dados usando os arquivos de dados pertencentes à classe `admin`. Para realizar esta tarefa, ele adota o seguinte procedimento:

  - Copia o arquivo de dados de origem ao destino apropriado
  - Cria um arquivo vazio chamado `config.data` e o atribui a uma classe de `cfgdata`
  - Executa o comando `bin/config` (distribuido com o pacote e já instalado) para preencher o arquivo de banco de dados `config.data` usando os arquivos de dados pertencentes à classe `admin`
  - Executa o comando `installf -f` para finalizar a instalação de `config.data`

  Não é necessário realizar nenhuma ação especial na classe `admin` no tempo de remoção já que nenhum script de ação de classe de remoção é criado. Isso significa que todos os arquivos e diretórios da classe `admin` são removidos do sistema.

- Criar um script de ação de remoção para a classe `cfgdata` (`r.cfgdata`).

  O script de remoção faz uma cópia do arquivo de banco de dados antes de excluí-lo. Não é necessário realizar nenhuma ação especial nesta classe no tempo de instalação, já que nenhum script de classe de instalação é necessário.

  Lembre-se de que a entrada de um script de remoção é uma lista de nomes de caminho a serem removidos. Os nomes de caminho sempre aparecem na ordem alfabética inversa. Este script de remoção copia os arquivos em um diretório chamado `$PKGSAV`. Quando todos os nomes de caminho tiverem sido processados, o script volta e remove todos os diretórios e arquivos associados à classe `cfgdata`.

  O resultado deste script de remoção é copiar `config.data` em `$PKGSAV` e, em seguida, remover o arquivo `config.data` e o diretório de dados.

Arquivos de estudo de caso

O arquivo `pkginfo`

```
PKG=krazy
NAME=KrAzY Applications
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1
CLASSES=none cfgdata admin
```

O arquivo `prototype`

```
i pkginfo
i request
i i.admin
i r.cfgdata
```

```
d none bin 555 root sys
f none bin/process1 555 root other
f none bin/process2 555 root other
f none bin/process3 555 root other
f admin bin/config 500 root sys
d admin cfg 555 root sys
f admin cfg/datafile1 444 root sys
f admin cfg/datafile2 444 root sys
f admin cfg/datafile3 444 root sys
f admin cfg/datafile4 444 root sys
d cfgdata data 555 root sys
```

O arquivo space

```
# extra space required by config data which is
# dynamically loaded onto the system
data 500 1
```

O script de ação de classe i.admin

```
# PKGINST parameter provided by installation service
# BASEDIR parameter provided by installation service
while read src dest
do
   cp $src $dest || exit 2
done
# if this is the last time this script will be executed
# during the installation, do additional processing here.
if [ "$1" = ENDOFCLASS ]
then
# our config process will create a data file based on any changes
# made by installing files in this class; make sure the data file
# is in class ‘cfgdata’ so special rules can apply to it during
# package removal.
   installf -c cfgdata $PKGINST $BASEDIR/data/config.data f 444 root
   sys || exit 2
   $BASEDIR/bin/config > $BASEDIR/data/config.data || exit 2
   installf -f -c cfgdata $PKGINST || exit 2
fi
exit 0
```

Isso ilustra uma instância rara na qual `installf` é apropriada em um script de ação de classe. Pelo fato do arquivo `space` ser usado para reservar espaço em um sistema de arquivos específico, este novo arquivo pode ser adicionado seguramente, mesmo se não estiver incluído no arquivo `pkgmap`.

O script de remoção r.cfgdata

```
# the product manager for this package has suggested that
# the configuration data is so valuable that it should be
# backed up to $PKGSAV before it is removed!
while read path
do
# path names appear in reverse lexical order.
   mv $path $PKGSAV || exit 2
```

```
   rm -f $path || exit 2
done
exit 0
```

Definindo compatibilidades e dependências de pacotes

O pacote deste estudo de caso usa arquivos de informação opcionais para definir as compatibilidades e dependências do pacote, e para apresentar uma mensagem de copyright durante a instalação.

Técnicas

Este estudo de caso demonstra as seguintes técnicas:

- Uso do arquivo `copyright`
- Uso do arquivo `compver`
- Uso do arquivo `depend`

Para obter mais informações sobre estes arquivos, consulte "Criando arquivos de informação" na página 52.

Abordagem

Para satisfazer os requisitos da descrição, você deve:

- Criar um arquivo `copyright`.

  Um arquivo `copyright` contém o texto ASCII de uma mensagem de copyright. A mensagem que aparece no arquivo de amostra é exibida na tela durante a instalação do pacote.

- Criar um arquivo `compver`.

  O arquivo `pkginfo` mostrado na figura a seguir define a versão do pacote como 3.0. O arquivo `compver` define a versão 3.0 como compatível com as versões 2.3, 2.2, 2.1, 2.1.1, 2.1.3 e 1.7.

- Criar um arquivo `depend`.

  Os arquivos listados em um arquivo `depend` já devem estar instalados no sistema quando um pacote for instalado. O arquivo de exemplo possui 11 pacotes que já devem estar no sistema no momento da instalação.

Arquivos de estudo de caso

O arquivo `pkginfo`

```
PKG=case3
NAME=Case Study #3
CATEGORY=application
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 3.0
CLASSES=none
```

O arquivo `copyright`

```
Copyright (c) 1999 company_name
All Rights Reserved.
THIS PACKAGE CONTAINS UNPUBLISHED PROPRIETARY SOURCE CODE OF
company_name.
The copyright notice above does not evidence any
actual or intended publication of such source code
```

O arquivo `compver`

```
Version 3.0
Version 2.3
Version 2.2
Version 2.1
Version 2.1.1
Version 2.1.3
Version 1.7
```

O arquivo `depend`

```
P acu Advanced C Utilities
Issue 4 Version 1
P cc C Programming Language
Issue 4 Version 1
P dfm Directory and File Management Utilities
P ed Editing Utilities
P esg Extended Software Generation Utilities
Issue 4 Version 1
P graph Graphics Utilities
P rfs Remote File Sharing Utilities
Issue 1 Version 1
P rx Remote Execution Utilities
P sgs Software Generation Utilities
Issue 4 Version 1
P shell Shell Programming Utilities
P sys System Header Files
Release 3.1
```

Modificando um arquivo usando classes padrão e scripts de ação de classe

Este estudo de caso modifica um arquivo existente durante a instalação do pacote usando classes padrão e scripts de ação de classe. Usa um dos três métodos de modificação. Os outros dois métodos estão descritos em "Modificando um arquivo usando a classe `sed` e um script `postinstall`" na página 112 e "Modificando um arquivo usando a classe `build`" na página 114. O arquivo modificado é `/etc/inittab` .

Técnicas

Este estudo de caso demonstra como usar os scripts de ação de classe de instalação e de remoção. Para obter informações, consulte "Escrevendo scripts de ação de classe" na página 70.

Abordagem

Para modificar `/etc/inittab` durante a instalação usando classes e scripts de ação de classe, você deve realizar as tarefas seguintes:

- Criar uma classe.

  Crie uma classe chamada `inittab` . Você deve fornecer um script de ação de classe de instalação e um de remoção para esta classe. Defina a classe `inittab` no parâmetro `CLASSES` no arquivo `pkginfo`.

- Criar um arquivo `inittab`.

  Este arquivo contém a informação da entrada que você adicionará ao `/etc/inittab` . Observe que no arquivo `prototype` aparece que `inittab` é um membro da classe `inittab` e possui um tipo de arquivo de `e` para editável.

- Criar um script de ação de classe de instalação (`i.inittab`).

  Lembre-se que os scripts de ação de classe devem produzir os mesmos resultados sempre que forem executados. O script de ação de classe realiza os seguintes procedimentos:

  - Verifica se esta entrada tinha sido adicionada anteriormente
  - Se tiver sido, remove as versões anteriores da entrada
  - Edita o arquivo `inittab` e adiciona as linhas de comentário para que você saiba de onde é a entrada
  - Move o arquivo temporário de volta para `/etc/inittab`
  - Executa o comando `init q` quando recebe o indicador `ENDOFCLASS`

  Observe que o comando `init q` pode ser realizado por este script de instalação. Um script `postinstall` de uma linha não é necessário nesta abordagem.

- Criar um script de ação de classe de remoção (`r.inittab`).

  O script de remoção é muito parecido ao script de instalação. A informação adicionada pelo script de instalação é removida e o comando `init q` é executado.

Este estudo de caso é mais complicado do que o estudo de caso a seguir, consulte "Modificando um arquivo usando a classe `sed` e um script `postinstall`" na página 112. Em vez de fornecer dois arquivos, são necessários três arquivos e o arquivo `/etc/inittab` integrado é, em realidade, apenas um espaço reservado contendo um fragmento da entrada que será inserida. Poderia ter sido colocado no arquivo `i.inittab` exceto que o comando `pkgadd` deve ter um arquivo para passar ao arquivo `i.inittab`. O procedimento de remoção também deve ser colocado em outro arquivo (`r.inittab`). Embora este método funcione bem, é melhor reservá-lo para os casos que envolvam instalações complicadas de vários arquivos. Consulte "Modificando arquivos `crontab` durante a instalação" na página 116.

O programa `sed` usado em "Modificando um arquivo usando a classe `sed` e um script `postinstall`" na página 112 oferece suporte a várias instâncias de pacote desde que o comentário no final da entrada `inittab` esteja baseado em uma instância de pacote. O estudo de caso em "Modificando um arquivo usando a classe `build`" na página 114 mostra uma abordagem mais dinâmica para editar `/etc/inittab` durante a instalação.

Arquivos de estudo de caso

O arquivo `pkginfo`

```
PKG=case5
NAME=Case Study #5
CATEGORY=applications
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1d05
CLASSES=inittab
```

O arquivo `prototype`

```
i pkginfo
i i.inittab
i r.inittab
e inittab /etc/inittab ? ? ?
```

O script de ação de classe de instalação i.inittab

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
```

```
exit 2
sed -e "s/$/#$PKGINST" $src >> /tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
if [ "$1" = ENDOFCLASS ]
then
/sbin/init q ||
exit 2
fi
exit 0
```

O script de ação de classe de remoção r.inittab

```
# PKGINST parameter provided by installation service
while read src dest
do
# remove all entries from the table that
# are associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" $dest >
/tmp/$$itab ||
exit 2
mv /tmp/$$itab $dest ||
exit 2
done
/sbin/init q ||
exit 2
exit 0
```

O arquivo inittab

```
rb:023456:wait:/usr/robot/bin/setup
```

Modificando um arquivo usando a classe sed e um script postinstall

Este estudo de caso modifica um arquivo que existe na máquina de instalação durante a instalação do pacote. Usa um dos três métodos de modificação. Os outros dois métodos estão descritos em "Modificando um arquivo usando classes padrão e scripts de ação de classe" na página 110 e "Modificando um arquivo usando a classe build" na página 114. O arquivo modificado é /etc/inittab.

Técnicas

Este estudo de caso demonstra as seguintes técnicas:

- Uso da classe sed

  Para obter mais informações sobre a classe sed, consulte "O script de classe sed" na página 74.

- Uso de um script postinstall

  Para obter mais informações sobre este script, consulte "Escrevendo scripts de procedimento" na página 68.

Abordagem

Para modificar /etc/inittab no momento da instalação usando a classe sed, você deve realizar as tarefas seguintes:

- Adicionar o script de classe sed ao arquivo prototype.

  O nome de um script deve ser o nome do arquivo que será editado. Neste caso, o arquivo a ser editado é /etc/inittab e o script sed também é nomeado /etc/inittab. Não há requisitos de modo, proprietário e grupo de um script sed (representado no prototype de amostra por pontos de interrogação). O tipo de arquivo do script sed deve ser e (indicando que é editável).
- Definir o parâmetro CLASSES para incluir a classe sed.

  Conforme mostrado no arquivo de exemplo, sed é a única classe instalada. No entanto, poderia ser uma entre várias classes.
- Criar um script de ação de classe sed.

  Seu pacote não pode distribuir uma cópia de /etc/inittab que tenha a aparência que você precisa, já que /etc/inittab é um arquivo dinâmico e você não tem como saber como será sua aparência no momento da instalação do pacote. No entanto, o uso do script sed permitirá modificar o arquivo /etc/inittab durante a instalação do pacote.
- Criar um script postinstall.

  Você precisa executar o comando init q para informar o sistema que /etc/inittab foi modificado. O único lugar no qual você pode realizar esta ação neste exemplo é em um script postinstall. Ao observar o script postinstall de exemplo, você verá que o propósito dele é executar o comando init q.

Esta abordagem de edição de /etc/inittab durante a instalação tem uma desvantagem. Você tem que distribuir um script completo (o script postinstall) simplesmente para realizar o comando init q.

Arquivos de estudo de caso

O arquivo pkginfo

```
PKG=case4
NAME=Case Study #4
CATEGORY=applications
BASEDIR=/opt
```

```
ARCH=SPARC
VERSION=Version 1d05
CLASSES=sed
```

O arquivo `prototype`

```
i pkginfo
i postinstall
e sed /etc/inittab ? ? ?
```

O script de ação de classe sed (/etc/inittab)

```
!remove
# remove all entries from the table that are associated
# with this package, though not necessarily just
# with this package instance
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
!install
# remove any previous entry added to the table
# for this particular change
/^[^:]*:[^:]*:[^:]*:[^#]*#ROBOT$/d
# add the needed entry at the end of the table;
# sed(1) does not properly interpret the '$a'
# construct if you previously deleted the last
# line, so the command
# $a\
# rb:023456:wait:/usr/robot/bin/setup #ROBOT
# will not work here if the file already contained
# the modification. Instead, you will settle for
# inserting the entry before the last line!
$i\
rb:023456:wait:/usr/robot/bin/setup #ROBOT
```

O script `postinstall`

```
# make init re-read inittab
/sbin/init q ||
exit 2
exit 0
```

Modificando um arquivo usando a classe build

Este estudo de caso modifica um arquivo que existe na máquina de instalação durante a instalação do pacote. Usa um dos três métodos de modificação. Os outros dois métodos estão descritos em "Modificando um arquivo usando classes padrão e scripts de ação de classe" na página 110 e "Modificando um arquivo usando a classe `sed` e um script `postinstall`" na página 112. O arquivo modificado é `/etc/inittab`.

Técnicas

Este estudo de caso demonstra como usar a classe `build`. Para obter mais informações sobre a classe `build`, consulte "O script de classe `build`" na página 75.

Abordagem

Esta abordagem de modificação de `/etc/inittab` usa a classe `build`. Um script de classe `build` é executado com um script shell e sua saída se torna a nova versão do arquivo que está sendo executado. Em outras palavras, o arquivo de dados `/etc/inittab` distribuído com este pacote será executado e a saída de tal execução será `/etc/inittab`.

O script de classe `build` é executado durante a instalação e a remoção do pacote. O argumento `install` é passado para o arquivo se ele estiver sendo executado no tempo de instalação. Observe no script de classe `build` de amostra que as ações de instalação são definidas ao testar este argumento.

Para editar `/etc/inittab` usando a classe `build`, você deve realizar as tarefas seguintes:

- Definir o arquivo build no arquivo `prototype`.

  A entrada do arquivo build no arquivo `prototype` deve colocá-lo na classe `build` e definir seu tipo de arquivo como `e`. Certifique-se de que o parâmetro `CLASSES` do arquivo `pkginfo` esteja definido como `build`.
- Criar o script de classe `build`.

  O script de classe `build` de amostra realiza os seguintes procedimentos:

  - Edita o arquivo `/etc/inittab` para remover as alterações existentes deste pacote. Observe que o nome do arquivo `/etc/inittab` está definido de modo determinado e inalterável no comando `sed`.
  - Se o pacote estiver sendo instalado, adiciona a nova linha ao final de `/etc/inittab`. Uma marca de comentário é incluída na nova entrada para descrever de onde tal entrada provém.
  - Executa o comando `init q`.

Esta solução trata das desvantagens descritas nos estudos de caso em "Modificando um arquivo usando classes padrão e scripts de ação de classe" na página 110 e em "Modificando um arquivo usando a classe `sed` e um script `postinstall`" na página 112. É necessário somente um arquivo breve (além dos arquivos `pkginfo` e `prototype`). O arquivo funciona com várias instâncias de um pacote desde que o parâmetro `PKGINST` seja usado, e o script `postinstall` não é necessário desde que o comando `init q` possa ser executado da classe `build`.

Arquivos de estudo de caso

O arquivo `pkginfo`

```
PKG=case6
NAME=Case Study #6
CATEGORY=applications
BASEDIR=/opt
```

```
ARCH=SPARC
VERSION=Version 1d05
CLASSES=build
```

O arquivo prototype

```
i pkginfo
e build /etc/inittab ? ? ?
```

O arquivo build

```
# PKGINST parameter provided by installation service
# remove all entries from the existing table that
# are associated with this PKGINST
sed -e "/^[^:]*:[^:]*:[^:]*:[^#]*#$PKGINST$/d" /etc/inittab ||
exit 2
if [ "$1" = install ]
then
# add the following entry to the table
echo "rb:023456:wait:/usr/robot/bin/setup #$PKGINST" ||
exit 2
fi
/sbin/init q ||
exit 2
exit 0
```

Modificando arquivos crontab durante a instalação

Este estudo de caso modifica arquivos crontab durante a instalação do pacote.

Técnicas

Este estudo de caso demonstra as seguintes técnicas:

- Uso de classes e scripts de ação de classe

  Para obter informações, consulte "Escrevendo scripts de ação de classe" na página 70.
- Uso do comando crontab dentro de um script de ação de classe.

Abordagem

A forma mais eficiente de editar mais de um arquivo durante a instalação é definir uma classe e fornecer um script de ação de classe. Se você usou a abordagem da classe build, será necessário distribuir um script de classe build para cada arquivo crontab editado. A definição de uma classe cron proporciona uma abordagem mais geral. Para editar arquivos crontab com esta abordagem, você deve:

- Definir os arquivos crontab que serão editados no arquivo prototype.

Criar uma entrada no arquivo prototype para o arquivo crontab que será editado. Definir a classe como cron e o tipo de arquivo como e de cada arquivo. Usar o nome real do arquivo a ser editado.

- Criar os arquivos crontab no pacote.

  Estes arquivos contêm as informações que você deseja adicionar aos arquivos crontab existentes de mesmo nome.

- Criar um script de ação de classe de instalação para a classe cron.

  O script i.cron de amostra realiza os seguintes procedimentos:

  - Determina o ID de usuário (UID).

    O script i.cron define a variável *user* como o nome base do script de classe cron que está sendo processado. Tal nome é o UID. Por exemplo, o nome base de /var/spool/cron/crontabs/root é root, que também é o UID.

  - Executa o crontab usando o UID e a opção - l.

    A opção -l diz ao crontab para enviar o conteúdo do arquivo crontab do usuário definido para a saída padrão.

  - Conduz a saída do comando crontab ao script sed que remove as entradas adicionadas anteriormente com esta técnica de instalação.
  - Coloca a saída editada em um arquivo temporário.
  - Adiciona o arquivo de dados do UID root (distribuído com o pacote) ao arquivo temporário e adiciona uma marca para que você saiba de onde provêm estas entradas.
  - Executa o crontab com o mesmo UID e o fornece ao arquivo temporário como entrada.

- Criar um script de ação de classe de remoção cron.

  O script r.cron é semelhante ao script de instalação, exceto que não há nenhum procedimento para adicionar informações ao arquivo crontab.

  Estes procedimentos são realizados por todos os arquivos na classe cron.

Arquivos de estudo de caso

Os scripts i.cron e r.cron descritos abaixo são executados pelo superusuário. A edição de outro arquivo crontab do usuário como superusuário pode ter conseqüências imprevisíveis. Se necessário, altere a seguinte entrada em cada script de:

```
crontab $user < /tmp/$$crontab ||
```

para

```
su $user -c "crontab /tmp/$$crontab" ||
```

O comando pkginfo

```
PKG=case7
NAME=Case Study #7
CATEGORY=application
BASEDIR=/opt
ARCH=SPARC
VERSION=Version 1.0
CLASSES=cron
```

O arquivo prototype

```
i pkginfo
i i.cron
i r.cron
e cron /var/spool/cron/crontabs/root ? ? ?
e cron /var/spool/cron/crontabs/sys ? ? ?
```

O script de ação de classe de instalação i.cron

```
# PKGINST parameter provided by installation service
while read src dest
do
user=‘basename $dest‘ ||
exit 2
(crontab -l $user |
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
sed -e "s/$/#$PKGINST/" $src >> /tmp/$$crontab ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
rm -f /tmp/$$crontab
done
exit 0
```

O script de ação de classe de remoção r.cron

```
# PKGINST parameter provided by installation service
while read path
do
user=‘basename $path‘ ||
exit 2
(crontab -l $user |
sed -e "/#$PKGINST$/d" > /tmp/$$crontab) ||
exit 2
crontab $user < /tmp/$$crontab ||
exit 2
rm -f /tmp/$$crontab
done
exit
```

Arquivo crontab #1

```
41,1,21 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
45 23 * * * ulimit 5000; /usr/bin/su uucp -c
"/usr/lib/uucp/uudemon.cleanup" >
```

```
/dev/null 2>&1
11,31,51 * * * * /usr/lib/uucp/uudemon.poll > /dev/null
```

Arquivo `crontab` #2

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

Observação – Se a edição de um grupo de arquivos for aumentar o tamanho total do arquivo em mais de 10K, forneça um arquivo `space` para que o comando `pkgadd` possa permitir esse aumento. Para mais informações sobre o arquivo `space`, consulte "Reservando espaço adicional em um sistema de destino" na página 57.

Instalando e removendo um driver com scripts de procedimento

Este pacote instala um driver.

Técnicas

Este estudo de caso demonstra as seguintes técnicas:

- Instalação e carregamento de um driver com um script `postinstall`
- Carregamento de um driver com um script `preremove`

Para obter mais informações sobre estes scripts, consulte "Escrevendo scripts de procedimento" na página 68.

Abordagem

- Criar um script `request`.

  O script `request` determina onde o administrador quer que os objetos de driver sejam instalados, perguntando ao administrador e atribuindo as respostas ao parâmetro `$KERNDIR`.

  O script termina com uma rotina para tornar os dois parâmetros `CLASSES` e `KERNDIR` disponíveis para o ambiente de instalação e o script `postinstall`.

- Criar um script `postinstall`.

  O script `postinstall` realiza, na verdade, a instalação do driver. Ele é executado depois que os arquivos `buffer` e `buffer.conf` tiverem sido instalados. O arquivo `postinstall` mostrado neste exemplo realiza as seguintes ações:

- Usa o comando `add_drv` para carregar o driver no sistema.
- Cria um link para o dispositivo usando o comando `installf`.
- Finaliza a instalação usando o comando `installf -f`.
- Cria um script `preremove`.

  O script `preremove` usa o comando `rem_drv` para descarregar o driver do sistema e, em seguida, remove o link `/dev/buffer0` .

Arquivos de estudo de caso

O arquivo `pkginfo`

```
PKG=bufdev
NAME=Buffer Device
CATEGORY=system
BASEDIR=/
ARCH=INTEL
VERSION=Software Issue #19
CLASSES=none
```

O arquivo `prototype`

Para instalar um driver no momento da instalação, você deve incluir os arquivos de objeto e de configuração do driver no arquivo `prototype`.

Neste exemplo, o módulo executável do driver é nomeado `buffer`. O comando `add_drv` opera neste arquivo. O kernel usa o arquivo de configuração, `buffer.conf`, para ajudar a configurar o driver.

```
i pkginfo
i request
i postinstall
i preremove
f none $KERNDIR/buffer 444 root root
f none $KERNDIR/buffer.conf 444 root root
```

Ao observar o arquivo `prototype` deste exemplo, note o seguinte:

- Visto que os objetos de pacote não precisam de nenhum tratamento especial, você pode colocá-los na classe `none` padrão. O parâmetro `CLASSES` está definido como `none` no arquivo `pkginfo`.
- Os nomes de caminho de `buffer` e `buffer.conf` começam com a variável `$KERNDIR`. Esta variável está definida no script `request` e permite que o administrador decida onde os arquivos de driver devem ser instalados. O diretório padrão é `/kernel/drv`.
- Há uma entrada para o script `postinstall` (o script que realizará a instalação do driver).

O script `request`

```
trap 'exit 3' 15
# determine where driver object should be placed; location
# must be an absolute path name that is an existing directory
KERNDIR=`ckpath -aoy -d /kernel/drv -p \
"Where do you want the driver object installed"` || exit $?

# make parameters available to installation service, and
# so to any other packaging scripts
cat >$1 <<!

CLASSES='$CLASSES'
KERNDIR='$KERNDIR'
!
exit 0
```

O script `postinstall`

```
# KERNDIR parameter provided by 'request' script
err_code=1                    # an error is considered fatal
# Load the module into the system
cd $KERNDIR
add_drv -m '* 0666 root sys' buffer || exit $err_code
# Create a /dev entry for the character node
installf $PKGINST /dev/buffer0=/devices/eisa/buffer*:0 s
installf -f $PKGINST
```

O script `preremove`

```
err_code=1                    # an error is considered fatal
# Unload the driver
rem_drv buffer || exit $err_code
# remove /dev file
removef $PKGINST /dev/buffer0 ; rm /dev/buffer0
removef -f $PKGINST
```

Instalando um driver usando a classe sed e scripts de procedimento

Este estudo de caso descreve como instalar um driver usando a classe `sed` e scripts de procedimento. Também é diferente do estudo de caso anterior (consulte "Instalando e removendo um driver com scripts de procedimento" na página 119) porque este pacote está composto de objetos absolutos e relocáveis.

Técnicas

Este estudo de caso demonstra as seguintes técnicas:

- Construção de um arquivo `prototype` com objetos absolutos e relocáveis.

Para obter informações sobre construção de um arquivo `prototype`, consulte "Criando um arquivo `prototype`" na página 31.

- Uso de um script `postinstall`

  Para obter mais informações sobre este script, consulte "Escrevendo scripts de procedimento" na página 68.

- Uso de um script `preremove`

  Para obter mais informações sobre este script, consulte "Escrevendo scripts de procedimento" na página 68.

- Uso de um arquivo `copyright`

  Para obter mais informações sobre este arquivo, consulte "Escrevendo uma mensagem de copyright" na página 55.

Abordagem

- Criar um arquivo `prototype` contendo objetos de pacote absolutos e relocáveis.

  Este tema é tratado detalhadamente em "O arquivo `prototype`" na página 123.

- Adicionar o script de classe `sed` ao arquivo `prototype`.

  O nome de um script deve ser o nome do arquivo que será editado. Neste caso, o arquivo a ser editado é `/etc/devlink.tab` e o script `sed` também é nomeado `/etc/devlink.tab`. Não há requisitos de modo, proprietário e grupo de um script `sed` (representado no `prototype` de amostra por pontos de interrogação). O tipo de arquivo do script `sed` deve ser `e` (indicando que é editável).

- Definir o parâmetro `CLASSES` para incluir a classe `sed`.
- Criar um script de ação de classe `sed` (`/etc/devlink.tab`).
- Criar um script `postinstall`.

  O script `postinstall` precisa executar o comando `add_drv` para adicionar o driver do dispositivo ao sistema.

- Criar um script `preremove`.

  O script `preremove` precisa executar o comando `rem_drv` para remover o driver do dispositivo do sistema, antes que o pacote seja removido.

- Criar um arquivo `copyright`.

  Um arquivo `copyright` contém o texto ASCII de uma mensagem de copyright. A mensagem que aparece no arquivo de amostra é exibida na tela durante a instalação do pacote.

Arquivos de estudo de caso

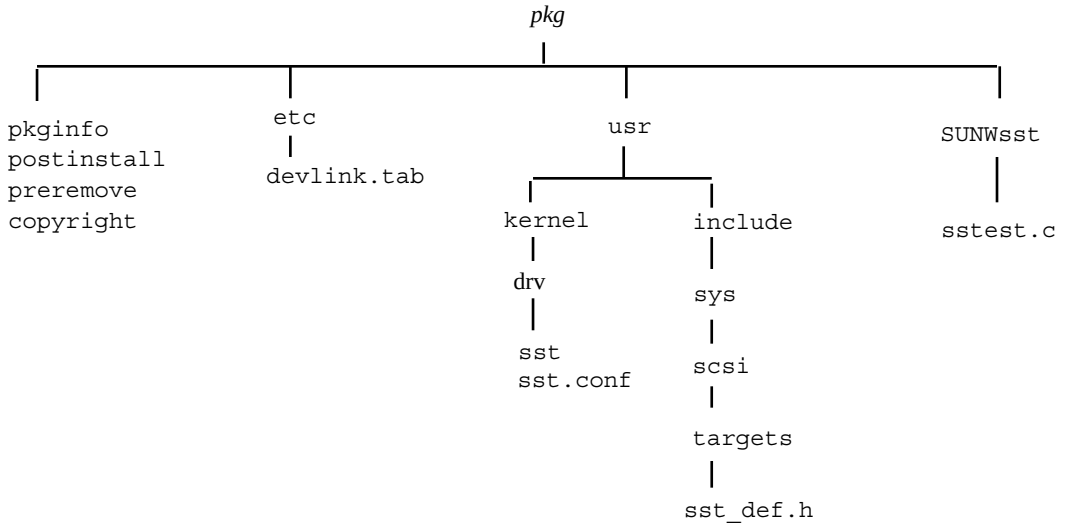
O arquivo `pkginfo`

```
PKG=SUNWsst
NAME=Simple SCSI Target Driver
VERSION=1
CATEGORY=system
ARCH=sparc
VENDOR=Sun Microsystems
BASEDIR=/opt
CLASSES=sed
```

O arquivo `prototype`

Este estudo de caso, por exemplo, usa o layout de hierarquia dos objetos de pacote mostrados na figura abaixo.

FIGURA 5–1 Estrutura hierárquica do diretório do pacote



Os objetos de pacote são instalados nos mesmos locais que no diretório `pkg` acima. Os módulos do driver (`sst` e `sst.conf`) são instalados no `/usr/kernel/drv` e o arquivo incluído é instalado em `/usr/include/sys/scsi/targets`. Os arquivos `sst`, `sst.conf` e `sst_def.h` são objetos absolutos. O programa de teste, `sstest.c`, e seu diretório `SUNWsst` são relocáveis. O seu local de instalação é definido pelo parâmetro `BASEDIR`.

Os componentes restantes do pacote (todos os arquivos de controle) se encontram na parte superior do diretório do pacote na máquina de desenvolvimento, exceto o script de classe `sed`. Denomina-se `devlink.tab` depois do arquivo que ele modifica, e entra em `etc`, o diretório que contém o arquivo `devlink.tab` real.

Do diretório `pkg`, execute o comando `pkgproto` da seguinte forma:

```
find usr SUNWsst -print | pkgproto > prototype
```

A saída do comando acima se assemelha à saída seguinte:

```
d none usr 0775 pms mts
d none usr/include 0775 pms mts
d none usr/include/sys 0775 pms mts
d none usr/include/sys/scsi 0775 pms mts
d none usr/include/sys/scsi/targets 0775 pms mts
f none usr/include/sys/scsi/targets/sst_def.h 0444 pms mts
d none usr/kernel 0775 pms mts
d none usr/kernel/drv 0775 pms mts
f none usr/kernel/drv/sst 0664 pms mts
f none usr/kernel/drv/sst.conf 0444 pms mts
d none SUNWsst 0775 pms mts
f none SUNWsst/sstest.c 0664 pms mts
```

Este arquivo `prototype` ainda não está completo. Para completar este arquivo, você precisa fazer as seguintes modificações:

- Insira as entradas dos arquivos de controle (tipo de arquivo `i`), porque apresentam um formato diferente do formato dos outros objetos de pacote.
- Remova as entradas dos diretórios que já existem no sistema de destino.
- Altere a permissão de acesso e a propriedade de cada entrada.
- Anteponha uma barra nos objetos de pacote absolutos.

Este é o arquivo `prototype` final:

```
i pkginfo
i postinstall
i preremove
i copyright
e sed /etc/devlink.tab ? ? ?
f none /usr/include/sys/scsi/targets/sst_def.h 0644 bin bin
f none /usr/kernel/drv/sst 0755 root sys
f none /usr/kernel/drv/sst.conf 0644 root sys
d none SUNWsst 0775 root sys
f none SUNWsst/sstest.c 0664 root sys
```

Os pontos de interrogação na entrada de cada script `sed` indicam que as permissões de acesso e a propriedade do arquivo existentes na máquina de instalação não devem ser alterados.

O script de ação de classe sed (/etc/devlink.tab)

No exemplo de driver, um script de classe `sed` é usado para adicionar uma entrada ao arquivo `/etc/devlink.tab` no driver. Este arquivo é usado pelo comando `devlinks` para criar links simbólicos de `/dev` em `/devices`. Este é o script `sed`:

```
# sed class script to modify /etc/devlink.tab
!install
/name=sst;/d
$i\
type=ddi_pseudo;name=sst;minor=character    rsst\\A1

!remove
/name=sst;/d
```

O comando `pkgrm` não executa a remoção de parte do script. Você pode precisar adicionar uma linha ao script `preremove` para executar `sed` diretamente para remover a entrada do arquivo `/etc/devlink.tab`.

O script de instalação `postinstall`

Neste exemplo, o script precisa apenas executar o comando `add_drv`.

```
# Postinstallation script for SUNWsst
# This does not apply to a client.
if [$PKG_INSTALL_ROOT = "/" -o -z $PKG_INSTALL_ROOT]; then
   SAVEBASE=$BASEDIR
   BASEDIR=””; export BASEDIR
   /usr/sbin/add_drv sst
   STATUS=$?
   BASEDIR=$SAVEBASE; export BASEDIR
   if [ $STATUS -eq 0 ]
   then
         exit 20
   else
         exit 2
   fi
else
   echo "This cannot be installed onto a client."
   exit 2
fi
```

O comando `add_drv` usa o parâmetro `BASEDIR`, de modo que o script tem que cancelar a definição de `BASEDIR` antes de executar o comando e restaurá-lo depois.

Uma das ações do comando `add_drv` é executar `devlinks`, que usa a entrada colocada em `/etc/devlink.tab` pelo script de classe `sed` para criar as entradas `/dev` no driver.

O código de saíca do script `postinstall` é importante. O código de saída `20` manda o comando `pkgadd` dizer ao usuário para reiniciar o sistema (necessário depois da instalação de um driver), e o código de saída `2` manda o comando `pkgadd` dizer ao usuário que a instalação falhou parcialmente.

O script de remoção `preremove`

No estudo de caso deste exemplo de driver, ele remove os links em `/dev` e executa o comando `rem_drv` no driver.

```
# Pre removal script for the sst driver
echo “Removing /dev entries”
/usr/bin/rm -f /dev/rsst*

echo “Deinstalling driver from the kernel”
SAVEBASE=$BASEDIR
BASEDIR=””; export BASEDIR
/usr/sbin/rem_drv sst
BASEDIR=$SAVEBASE; export BASEDIR

exit
```

O script remove as entradas `/dev`. As entradas `/devices` são removidas pelo comando `rem_drv`.

O arquivo `copyright`

É um arquivo ASCII simples que contém o texto de um aviso de copyright. O aviso é exibido no início da instalação do pacote exatamente como aparece no arquivo.

```
    Copyright (c) 1999 Drivers-R-Us, Inc.
    10 Device Drive, Thebus, IO 80586

All rights reserved. This product and related documentation is
protected by copyright and distributed under licenses
restricting its use, copying, distribution and decompilation.
No part of this product or related documentation may be
reproduced in any form by any means without prior written
authorization of Drivers-R-Us and its licensors, if any.
```

◆ ◆ ◆ CAPÍTULO 6

Técnicas avançadas para a criação de pacotes

Todos os recursos do pacote System V, como implementado no SO Oracle Solaris, oferecem uma ferramenta eficiente para a instalação de produtos de software. Como um criador de pacotes, você pode tirar vantagem dessas capacidades. Os pacotes que não são parte do SO Oracle Solaris (pacotes sem conjunto) podem usar o mecanismo de classe para personalizar instalações do servidor/cliente. Os pacotes relocáveis podem ser criados para satisfazer os desejos do administrador. Um produto complexo pode ser entregue como um conjunto de pacotes compostos que resolve automaticamente as dependências de pacote. As atualizações e patches podem ser personalizados pelo criador do pacote. Os pacotes com patch podem ser entregues da mesma forma que os pacotes sem patch, e os arquivos de backout também podem ser incluídos no produto.

A lista abaixo traz as informações gerais encontradas neste capítulo.

Especificando o diretório base

Você pode usar vários métodos para especificar onde um pacote será instalado e é importante poder alterar a base de instalação dinamicamente no tempo de instalação. Se isto for atingido satisfatoriamente, um administrador pode instalar várias versões e várias arquiteturas se complicações.

Esta seção trata primeiro dos métodos comuns, depois das abordagens que melhoram as instalações em sistemas heterogêneos.

O arquivo de padrões administrativos

Os administradores responsáveis pela instalação de pacotes podem usar arquivos de administração para controlar a instalação do pacote. No entanto, como criador de pacotes, você precisa conhecer os arquivos de administração e saber como um administrador pode alterar a instalação pretendida do pacote.

Um arquivo de administração diz ao comando `pkgadd` se ele deve realizar as verificações ou solicitações que normalmente realiza. Conseqüentemente, os administradores devem entender por completo o processo de instalação do pacote e os scripts envolvidos antes de usar os arquivos de administração.

Um arquivo de padrões administrativos é entregue com o sistema operacional SunOS em `/var/sadm/install/admin/default`. Trata-se do arquivo que estabelece o nível mais básico de regras administrativas com relação à instalação de produtos de software. O arquivo apresenta a seguinte aparência quando entregue:

```
#ident "@(#)default
1.4 92/12/23 SMI"    /* SVr4.0 1.5.2.1    */
mail=
instance=unique
partial=ask
runlevel=ask
idepend=ask
rdepend=ask
space=ask
setuid=ask
conflict=ask
action=ask
basedir=default
```

O administrador pode editar este arquivo para estabelecer novos comportamentos padrões, ou criar um arquivo de administração e especificar sua existência usando a opção `-a` no comando `pkgadd`.

Podem ser definidos onze parâmetros no arquivo de administração, mas nem todos precisam ser definidos. Para obter mais informações, consulte `admin(4)`.

O parâmetro `basedir` especifica como o diretório base será obtido quando um pacote for instalado. A maioria dos administradores o mantém como `padrão`, mas `basedir` pode ser definido como:

- `ask`, que significa sempre pedir que o administrador indique um diretório base
- Um nome de caminho absoluto
- Um nome de caminho absoluto contendo a construção `$PKGINST`, que significa instalar sempre em um diretório base obtido da instância do pacote

Observação – Se o comando `pkgadd` for chamado com o argumento `-a none`, ele sempre pede que o administrador indique um diretório base. Infelizmente, isso também define *todos* os parâmetros do arquivo com o valor padrão `quit`, que pode causar problemas adicionais.

Conformando-se com a incerteza

Um administrador tem controle sobre todos os pacotes que estão sendo instalados em um sistema usando o arquivo de administração. Infelizmente, um arquivo de padrões administrativos alternativo é, com freqüência, fornecido pelo *criador de pacotes*, ignorando os desejos do administrador.

Os criadores de pacotes incluem, às vezes, um arquivo de administração alternativo para que eles, e não o administrador, controlem a instalação de um pacote. Por substituir todos os diretórios base, a entrada `basedir` do arquivo de padrões administrativos proporciona um método simples para selecionar o diretório base apropriado no tempo de instalação. Em todas as versões do SO Oracle Solaris anteriores à versão Solaris 2.5, esse era considerado um método mais simples de controlar o diretório de base.

Entretanto, é necessário aceitar os desejos do administrador com relação à instalação do produto. Proporcionar um arquivo de padrões administrativos temporário para fins de controle de instalação leva a uma desconfiança por parte dos administradores. Você deve usar um script `request` e um script `checkinstall` para controlar estas instalações sob a supervisão do administrador. Se o script `request` envolver fielmente o administrador no processo, o empacotamento System V servirá tanto aos administradores como aos criadores de pacotes.

Usando o parâmetro `BASEDIR`

O arquivo `pkginfo` de qualquer pacote relocável deve incluir um diretório base padrão em forma de entrada como esta:

`BASEDIR=`*absolute_path*

Este é o único diretório base padrão. Ele pode ser alterado pelo administrador durante a instalação.

Enquanto alguns pacotes requerem mais de um diretório base, a vantagem de usar este parâmetro para posicionar o pacote está no fato de que garante que o diretório base esteja na sua posição e seja gravável como um diretório válido no momento em que a instalação começar. O caminho correto para o diretório base do servidor e do cliente está disponível para todos os scripts de procedimento na forma de variáveis de ambiente reservadas, e o comando `pkginfo -r SUNWstuf` exibe a base de instalação atual do pacote.

No script `checkinstall`, `BASEDIR` é o parâmetro exatamente conforme definido no arquivo `pkginfo` (ele ainda foi condicionado). A fim de inspecionar o diretório de destino base, a construção `${PKG_INSTALL_ROOT}$BASEDIR` é necessária. Isso significa que o script `request` ou

checkinstall podem alterar o valor de BASEDIR no ambiente de instalação com resultados previsíveis. Neste momento o script preinstall é chamado, o parâmetro BASEDIR é o indicador completamente condicionado do diretório base real no sistema de destino, mesmo que o sistema for um cliente.

Observação – O script request utiliza o parâmetro BASEDIR de forma diferente em diferentes versões do sistema operacional SunOS. A fim de testar um parâmetro BASEDIR em um script request, o seguinte código deve ser usado para determinar o diretório base real em uso.

```
# request script
constructs base directory
if [ ${CLIENT_BASEDIR} ]; then
      LOCAL_BASE=$BASEDIR
else
      LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
fi
```

Usando diretórios base paramétricos

Se um pacote precisar de vários diretórios base, você pode estabelecê-los com nomes de caminho paramétricos. Este método tem se tornado muito popular, embora apresente as desvantagens seguintes.

- Um pacote com nomes de caminho paramétricos se comporta geralmente como um pacote absoluto, mas é tratado pelo comando pkgadd como um pacote relocável. O parâmetro BASEDIR deve ser definido mesmo que não seja usado.
- O administrador não pode verificar a base de instalação do pacote usando os utilitários System V (o comando pkginfo -r não funcionará).
- O administrador não pode usar o método estabelecido para relocar o pacote (é denominado relocável, mas age como absoluto).
- As instalações de várias arquiteturas e várias versões requerem plano de contingência para cada um dos diretórios de destino base, o que significa freqüentemente vários scripts de ação de classe complexos.

Embora os parâmetros que determinam os diretórios base sejam definidos no arquivo pkginfo, eles podem ser modificados pelo script request. Esta é uma das principais razões da popularidade desta abordagem. As desvantagens, entretanto, são crônicas e esta configuração deve ser levada em consideração em último caso.

Exemplos — Usando diretórios base paramétricos

O arquivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
EZDIR=/usr/stuf/EZstuf
HRDDIR=/opt/SUNWstuf/HRDstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert980707141632
```

O arquivo pkgmap

```
: 1 1758
1 d none $EZDIR 0775 root bin
1 f none $EZDIR/dirdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/usrdel 0555 bin bin 40 773 751310229
1 f none $EZDIR/filedel 0555 bin bin 40 773 751310229
1 d none $HRDDIR 0775 root bin
1 f none $HRDDIR/mksmart 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mktall 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkcute 0555 bin bin 40 773 751310229
1 f none $HRDDIR/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f none /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Gerenciando o diretório base

Os pacotes disponíveis em várias versões ou de várias arquiteturas devem ser criados para *deslocar* o diretório base, se necessário. Deslocar um diretório base significa que, se já existir uma versão anterior ou uma arquitetura diferente do pacote que estiver sendo instalado no diretório base, o pacote resolve este problema, criando talvez um novo diretório base com um nome levemente diferente. Os scripts `request` e `checkinstall` do Solaris 2.5 e versões compatíveis têm a capacidade de modificar a variável de ambiente `BASEDIR`. Isso não é válido para as versões anteriores do sistema operacional Oracle Solaris.

Mesmo em versões mais antigas do sistema operacional do Oracle Solaris, o script `request` tem a autoridade para redefinir os diretórios que estão na base de instalação. O script `request` pode fazê-lo de uma forma que ainda oferece suporte à maioria das preferências administrativas.

Acomodando a relocação

Embora você possa selecionar os diretórios base de vários pacotes que são exclusivos de uma arquitetura ou versão, esse procedimento cria níveis desnecessários de hierarquia de diretórios. Por exemplo, em um produto criado para processadores baseados em SPARC e em x86, você poderia organizar os diretórios base por processador e versão conforme ilustrado abaixo.

Diretório base	Versão e processador
`/opt/SUNWstuf/sparc/1.0`	Versão 1.0, SPARC
`/opt/SUNWstuf/sparc/1.2`	Versão 1.2, SPARC
`/opt/SUNWstuf/x86/1.0`	Versão 1.0, x86

Esse procedimento está bem e funciona, mas você está tratando os nomes e os números como se eles tivessem algum significado para o administrador. Uma abordagem mais apropriada é fazê-lo automaticamente *depois* de explicar ao administrador e pedir permissão.

Isso significa que você pode realizar todo o trabalho no pacote sem pedir ao administrador que o faça manualmente. Você pode atribuir o diretório base arbitrariamente e, a seguir, estabelecer com clareza os links de cliente apropriados em um script `postinstall`. Você também pode usar o comando `pkgadd` para instalar todo o pacote ou parte dele nos clientes do script `postinstall`. Você pode, até mesmo, perguntar ao administrador quais os usuários ou clientes que precisam ter informações sobre este pacote e atualizar automaticamente as variáveis de ambiente `PATH` e os arquivos `/etc`. Isso é totalmente aceitável contanto que qualquer ação realizada pelo pacote na instalação seja desfeita durante a remoção.

Deslocando diretórios base

Você pode tirar vantagem dos dois métodos para controlar o diretório base no momento da instalação. O primeiro é melhor para novos pacotes que serão instalados somente no Solaris 2.5 e versões compatíveis. Ele oferece dados muito úteis para o administrador e oferece suporte a várias versões e arquiteturas instaladas, além de exigir pouco trabalho especial. O segundo método pode ser usado por qualquer pacote e faz uso do controle inerente do script `request` sobre os parâmetros de construção para garantir que as instalações sejam bem-sucedidas.

Usando o parâmetro BASEDIR

O script checkinstall pode selecionar o diretório base apropriado no tempo de instalação, o que significa que o diretório base pode ser posicionado bem abaixo na árvore de diretórios. Este exemplo aumenta o diretório base seqüencialmente, levando a diretórios na forma /opt/SUNWstuf, /opt/SUNWstuf.1 e /opt/SUNWstuf.2. O administrador pode usar o comando pkginfo para determinar que arquitetura e versão são instalados em cada diretório base.

Se o pacote SUNWstuf (que contém um conjunto de utilitários que realizam ações) usasse este método, os arquivos pkginfo e pkgmap teriam a seguinte aparência.

O arquivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt/SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

O arquivo pkgmap

```
: 1 1758
1 d none EZstuf 0775 root bin
1 f none EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none HRDstuf 0775 root bin
1 f none HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

Exemplo — Scripts de análise que deslocam um BASEDIR

Suponha que a versão x86 de SUNWstuf já esteja instalada no servidor em /opt/SUNWstuf. Quando o administrador usa o comando pkgadd para instalar a versão SPARC, o script request precisa detectar a existência da versão x86 e interagir com o administrador em relação à instalação.

Observação – O diretório base pode ser deslocado sem a interação do administrador em um script checkinstall, mas se operações arbitrárias como essa ocorrerem constantemente, os administradores perdem a confiança no processo.

Os scripts request e checkinstall de um pacote que manipula esta situação podem ter a seguinte aparência.

O script request

```
# request script
for SUNWstuf to walk the BASEDIR parameter.

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

GENMSG="The base directory $LOCAL_BASE already contains a \
different architecture or version of $PKG."

OLDMSG="If the option \"-a none\" was used, press the  \
key and enter an unused base directory when it is requested."

OLDPROMPT="Do you want to overwrite this version? "

OLDHELP="\"y\" will replace the installed package, \"n\" will \
stop the installation."

SUSPEND="Suspending installation at user request using error \
code 1."

MSG="This package could be installed at the unused base directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed base directory? "

HELP="A response of \"y\" will install to the proposed directory and continue,
\"n\" will request a different directory. If the option \"-a none\" was used,
press the  key and enter an unused base directory when it is requested."

DIRPROMPT="Select a preferred base directory ($WRKNG_BASE) "

DIRHELP="The package $PKG will be installed at the location entered."

NUBD_MSG="The base directory has changed. Be sure to update \
any applicable search paths with the actual location of the \
binaries which are at $WRKNG_BASE/EZstuf and $WRKNG_BASE/HRDstuf."

OldSolaris=""
```

```
Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0
      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older version it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR
            OldSolaris="true"
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR
fi

WRKNG_BASE=$LOCAL_BASE

    # See if the base directory is already in place and walk it if
    # possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
               # So this is a different architecture or
               # version than what is already there.
               # Walk the base directory
               Suffix=`expr $Suffix + 1`
               WRKNG_BASE=$LOCAL_BASE.$Suffix
               Changed="true"
         fi
done

    # So now we can propose a base directory that isn't claimed by
    # any of our other versions.
if [ $Changed ]; then
         puttext "$GENMSG"
         if [ $OldSolaris ]; then
               puttext "$OLDMSG"
               result=`ckyorn -Q -d "a" -h "$OLDHELP" -p "$OLDPROMPT"`
               if [ $result="n" ]; then
                     puttext "$SUSPEND"
                     exit 1    # suspend installation
               else
                     exit 0
               fi
         else    # The latest functionality is available
               puttext "$MSG"
```

```
                result=‘ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"‘
                if [ $? -eq 3]; then
                      echo quitinstall >> $1
                      exit 0
                fi

                if [ $result="n" ]; then
                      WRKNG_BASE=‘ckpath -ayw -d "$WRKNG_BASE" \
                      -h "$DIRHELP" -p "$DIRPROMPT"‘
                else if [ $result="a" ]
                      exit 0
                fi
            fi
            echo "BASEDIR=$WRKNG_BASE" >> $1
            puttext "$NUBD_MSG"
      fi
fi
exit 0
```

O script checkinstall

```
# checkinstall
script for SUNWstuf to politely suspend

grep quitinstall $1
if [ $? -eq 0 ]; then
    exit 3        # politely suspend installation
fi

exit 0
```

Esta abordagem não funcionaria muito bem se o diretório base fosse simplesmente `/opt`. Este pacote tem que chamar mais precisamente o `BASEDIR` visto que `/opt` seria difícil de deslocar. De fato, dependendo do esquema de montagem, isso pode não ser possível. O exemplo desloca o diretório base criando um novo diretório em `/opt`, que não apresenta nenhum problema.

Este exemplo usa um script `request` e um script `checkinstall` embora as versões do Oracle Solaris anteriores à versão 2.5 não possam executar o script `checkinstall`. O script `checkinstall` deste exemplo é usado a fim de interromper educadamente a instalação em resposta a uma mensagem privada na forma da seqüência `quitinstall`. Se este script for executado no Solaris 2.3, o script `checkinstall` é ignorado e o script `request` interrompe a instalação com uma mensagem de erro.

Lembre-se que nas versões anteriores ao Solaris 2.5 e versões compatíveis, o parâmetro `BASEDIR` é um parâmetro somente de leitura e não pode ser alterado pelo script `request`. Por essa razão, se uma versão antiga do sistema operacional SunOS for detectada (através do teste de uma variável de ambiente `CLIENT_BASEDIR` condicionada), o script `request` tem apenas duas opções — continuar ou sair.

Usando caminhos paramétricos relativos

Se o seu produto de software puder ser instalado em versões antigas do sistema operacional SunOS, o script `request` precisa realizar todas as ações necessárias. Esta abordagem também pode ser usada para manipular vários diretórios. Se forem necessários diretórios adicionais, estes ainda precisam ser incluídos em um único diretório base a fim de oferecer um produto facilmente administrável. Embora o parâmetro `BASEDIR` não ofereça o nível de granularidade disponível na última versão do Oracle Solaris, seu pacote ainda pode deslocar o diretório base usando o script `request` para manipular caminhos paramétricos. Os arquivos `pkginfo` e `pkgmap` podem ter a aparência seguinte.

O arquivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
SUBBASE=SUNWstuf
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

O arquivo pkgmap

```
: 1 1758
1 d none $SUBBASE/EZstuf 0775 root bin
1 f none $SUBBASE/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none $SUBBASE/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none $SUBBASE/HRDstuf 0775 root bin
1 f none $SUBBASE/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none $SUBBASE/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 f daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i i.daemon 509 39560 752978103
1 i r.daemon 320 24573 742152591
```

Este exemplo não é perfeito. Um comando `pkginfo -r` retorna `/opt` para a base de instalação, que é um pouco ambíguo. Muitos pacotes estão em `/opt`, mas pelo menos é um diretório

significativo. Igual ao exemplo anterior, o próximo exemplo oferece suporte completo a várias arquiteturas e versões. O script `request` pode ser adaptado às necessidades de determinado pacote e resolver quaisquer dependências aplicáveis.

Exemplo — Um script `request` que desloca um caminho paramétrico relativo

```
# request script
for SUNWstuf to walk a parametric path

PATH=/usr/sadm/bin:${PATH}    # use admin utilities

MSG="The target directory $LOCAL_BASE already contains \
different architecture or version of $PKG. This package \
could be installed at the unused target directory $WRKNG_BASE."

PROMPT="Do you want to use to the proposed directory? "

HELP="A response of \"y\" will install to the proposed directory \
and continue, \"n\" will request a different directory. If \
the option \"-a none\" was used, press the <RETURN> key and \
enter an unused base directory when it is requested."

DIRPROMPT="Select a relative target directory under $BASEDIR/"

DIRHELP="The package $PKG will be installed at the location entered."

SUSPEND="Suspending installation at user request using error \
code 1."

NUBD_MSG="The location of this package is not the default. Be \
sure to update any applicable search paths with the actual \
location of the binaries which are at $WRKNG_BASE/EZstuf \
and $WRKNG_BASE/HRDstuf."

Changed=""
Suffix="0"

#
# Determine if this product is actually installed in the working
# base directory.
#
Product_is_present () {
      if [ -d $WRKNG_BASE/EZstuf -o -d $WRKNG_BASE/HRDstuf ]; then
            return 1
      else
            return 0

      fi
}

if [ ${BASEDIR} ]; then
      # This may be an old version of Solaris. In the latest Solaris
      # CLIENT_BASEDIR won't be defined yet. In older versions it is.
      if [ ${CLIENT_BASEDIR} ]; then
            LOCAL_BASE=$BASEDIR/$SUBBASE
```

```
      else    # The base directory hasn't been processed yet
            LOCAL_BASE=${PKG_INSTALL_ROOT}$BASEDIR/$SUBBASE
      fi

WRKNG_BASE=$LOCAL_BASE

# See if the base directory is already in place and walk it if
# possible
while [ -d ${WRKNG_BASE} -a Product_is_present ]; do
         # There is a conflict
         # Is this an update of the same arch & version?
         if [ ${UPDATE} ]; then
               exit 0    # It's out of our hands.
         else
               # So this is a different architecture or
               # version than what is already there.
               # Walk the base directory
               Suffix=`expr $Suffix + 1`
               WRKNG_BASE=$LOCAL_BASE.$Suffix
               Changed="true"
         fi
done

# So now we can propose a base directory that isn't claimed by
# any of our other versions.
if [ $Changed ]; then
         puttext "$MSG"
         result=`ckyorn -Q -d "a" -h "$HELP" -p "$PROMPT"`
         if [ $? -eq 3 ]; then
               puttext "$SUSPEND"
               exit 1
         fi

         if [ $result="n" ]; then
               WRKNG_BASE=`ckpath -lyw -d "$WRKNG_BASE" -h "$DIRHELP" \
               -p "$DIRPROMPT"`

            elif [ $result="a" ]; then
                  exit 0
            else
                  exit 1
            fi
            echo SUBBASE=$SUBBASE.$Suffix >> $1
            puttext "$NUBD_MSG"
      fi
fi
exit 0
```

Oferecendo suporte a relocação em um ambiente heterogêneo

O conceito original do empacotamento System V pressupõe uma arquitetura por sistema. O conceito de um servidor não tinha um papel importante na criação. Agora, é claro, um único servidor pode oferecer suporte a várias arquiteturas, o que significa que pode haver várias

cópias de um mesmo software em um servidor, cada uma de uma arquitetura diferente. Embora os pacotes do Oracle Solaris sejam sequestrados dentro dos limites recomendados do sistema de arquivos (por exemplo, `/` e `/usr`), com bancos de dados do produto no servidor e em cada cliente, nem todas as instalações suportam, necessariamente, essa divisão. Determinadas implementações oferecem suporte a uma estrutura completamente diferente e abrangem um banco de dados comum do produto. Embora seja simples indicar aos clientes diferentes versões, instalar pacotes System V em diferentes diretórios base pode realmente causar complicações ao administrador.

Ao criar seu pacote, você deve também levar em consideração os métodos comuns usados pelos administradores para introduzir novas versões de software. Os administradores procuram freqüentemente instalar e testar a última versão lado-a-lado com a versão atualmente instalada. O procedimento envolve a instalação da nova versão em um diretório base diferente do diretório da versão atual e o envio de uma nova versão a um grupo de clientes não-críticos como teste. Assim como as construções de confiança, o administrador reenvia a mais e mais clientes a nova versão. Eventualmente, o administrador retém a versão antiga somente para emergências e, em seguida, finalmente a exclui.

Isso significa que os pacotes destinados a sistemas heterogêneos modernos devem oferecer suporte a relocações verdadeiras, o quer dizer que o administrador pode colocá-los em qualquer lugar razoável do sistema de arquivos e ainda obter total funcionalidade. O Solaris 2.5 e versões compatíveis oferecem várias ferramentas úteis que permitem a instalação perfeita de várias arquiteturas e versões no mesmo sistema. O Solaris 2.4 e versões compatíveis também oferecem suporte à relocação verdadeira, mas a conclusão das tarefas não é tão evidente.

Abordagem tradicional

Pacotes relocáveis

A ABI do System V dá a entender que a intenção original do pacote relocável era tornar a instalação do pacote mais fácil para o administrador. Agora, a necessidade em ter pacotes relocáveis vai mais além. A facilidade não é o único fim, é mais apropriado dizer que é bem possível que, durante a instalação, um produto de software ativo já esteja instalado no diretório padrão. Um pacote que não está desenhado para lidar com este tipo de situação, ou substitui o produto existente ou falha ao instalar. Entretanto, um pacote desenhado para manipular várias arquiteturas e versões pode ser instalado tranqüilamente e oferece ao administrador uma grande variedade de opções totalmente compatíveis com as tradições administrativas existentes.

De alguma forma, o problema das arquiteturas e versões múltiplas é o mesmo. Deve ser possível instalar uma variante do pacote existente lado-a-lado com outras variantes, e conduzir os clientes e os consumidores independentes de sistemas de arquivos exportados a qualquer uma destas variantes sem perda de funcionalidade. Embora a Sun tenha estabelecido métodos para

lidar com várias arquiteturas em um servidor, o administrador pode não se ater a tais recomendações. Todos os pacotes precisam ser capazes de satisfazer os desejos sensatos dos administradores em relação à instalação.

Exemplo - Um pacote relocável tradicional

Este exemplo mostra a aparência de um pacote relocável tradicional. O pacote será colocado em `/opt/SUNWstuf`, e os arquivos `pkginfo` e `pkgmap` podem ter a seguinte aparência.

O arquivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

O arquivo pkgmap

```
: 1 1758
1 d none SUNWstuf 0775 root bin
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

A isso se refere como método tradicional porque cada objeto de pacote é instalado no diretório base definido pelo parâmetro `BASEDIR` do arquivo `pkginfo`. Por exemplo, o primeiro objeto do arquivo `pkgmap` é instalado como o diretório `/opt/SUNWstuf`.

Pacotes absolutos

Um pacote absoluto é aquele que é instalado em um determinado sistema de arquivos raiz (/). É difícil lidar com estes pacotes do ponto de vista de arquiteturas e versões múltiplas. Como regra geral, todos os pacotes devem ser relocáveis. Há, no entanto, razões muito boas para incluir elementos absolutos em um pacote relocável.

Exemplo - Um pacote absoluto tradicional

Se o pacote `SUNWstuf` fosse um pacote absoluto, o parâmetro `BASEDIR` não deveria estar no arquivo `pkginfo` e o arquivo `pkgmap` teria a seguinte aparência.

O arquivo pkgmap

```
: 1 1758
1 d none /opt ? ? ?
1 d none /opt/SUNWstuf 0775 root bin
1 d none /opt/SUNWstuf/EZstuf 0775 root bin
1 f none /opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none /opt/SUNWstuf/HRDstuf 0775 root bin
1 f none /opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none /opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Neste exemplo, se o administrador tivesse especificado um diretório base alternativo durante a instalação, ele seria ignorado pelo comando `pkgadd`. Este pacote é instalado sempre no `/opt/SUNWstuf` do sistema de destino.

O argumento `-R` do comando `pkgadd` funciona conforme o previsto. Por exemplo,

```
pkgadd -d . -R /export/opt/client3 SUNWstuf
```

instala os objetos em `/export/opt/client3/opt/SUNWstuf`. Porém, isso é o mais perto que este pacote chega de ser um pacote relocável.

Observe o uso do ponto de interrogação (`?`) no diretório `/opt` do arquivo `pkgmap`. Isso indica que os atributos existentes não devem ser alterados. Não significa "criar o diretório com atributos padrão", embora sob certas circunstâncias isso possa ocorrer. Qualquer diretório específico de um novo pacote deve especificar todos os atributos explicitamente.

Pacotes compostos

Qualquer pacote que contenha objetos relocáveis é denominado de pacote relocável. Isso pode induzir a erros porque um pacote relocável pode conter caminhos absolutos em seu arquivo `pkgmap`. O uso de uma entrada raiz (`/`) em um arquivo `pkgmap` pode melhorar os aspectos relocáveis do pacote. Os pacotes que possuem entradas raiz e relocáveis são denominados pacotes *compostos*.

Exemplo - Uma solução tradicional

Suponha que um objeto do pacote `SUNWstuf` seja um script de início executado no nível de execução 2. O arquivo `/etc/rc2.d/S70dostuf` precisa ser instalado como parte do pacote, mas não pode ser colocado no diretório base. Pressupondo que um pacote relocável seja a única solução, os arquivos `pkginfo` e `pkgmap` podem ter a seguinte aparência.

O arquivo pkginfo

```
# pkginfo file
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none
PSTAMP=hubert990707141632
```

O arquivo pkgmap

```
: 1 1758
1 d none opt/SUNWstuf/EZstuf 0775 root bin
1 f none opt/SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none opt/SUNWstuf/HRDstuf 0775 root bin
1 f none opt/SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none opt/SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none etc    ? ? ?
1 d none etc/rc2.d ? ? ?
1 f none etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
```

Não há muita diferença entre esta abordagem e a abordagem do pacote absoluto. De fato, estaria melhor como um pacote absoluto — se o administrador fornecesse um diretório base alternativo a este pacote, ele não funcionaria!

Na verdade, somente um arquivo deste pacote precisa estar relacionado à raiz, o restante pode ser movido para qualquer lugar. Como resolver este problema através do uso de um pacote composto é um tema tratado no restante desta seção.

Não tradicional

A abordagem descrita nesta seção não se aplica a todos os pacotes, mas tem como resultado melhor desempenho durante a instalação em um ambiente heterogêneo. Uma pequena parte se aplica a pacotes entregues como parte do sistema operacional Oracle Solaris (pacotes incorporados). Entretanto, os pacotes avulsos podem praticar empacotamento não tradicional.

A razão pela qual se encoraja o uso de pacotes relocáveis é oferecer suporte a este requisito:

Quando um pacote é adicionado ou removido, os comportamentos desejáveis existentes dos produtos de software instalados não serão alterados.

Os pacotes avulsos devem estar em `/opt` de modo a garantir que o novo pacote não interfira nos produtos existentes.

Outra consideração sobre os pacotes compostos

Há duas regras a seguir ao construir um pacote composto funcional:

- Estabelecer o diretório base baseado em aonde vai a grande maioria dos objetos de pacote.
- Se um objeto de pacote for para um diretório comum que não é o diretório base (por exemplo, `/etc`), especifique-o como nome de caminho absoluto no arquivo `prototype`.

Em outras palavras, visto que "relocável" significa que o objeto pode ser instalado em qualquer parte e ainda funcionar, nenhum script de início executado por `init` no tempo de inicialização pode ser considerado relocável! Embora não exista nada de errado em especificar `/etc/passwd` como um caminho relativo no pacote entregue, há somente um lugar para o qual ele pode ir.

Fazendo nomes de caminho absolutos parecer relocáveis

Se for construir um pacote composto, os caminhos absolutos devem operar de modo que não interfiram no software instalado. Um pacote que pode estar totalmente contido em `/opt` contorna este problema, já que não há arquivos existentes no meio do caminho. Quando um arquivo em `/etc` estiver incluído no pacote, você deve garantir que nomes de caminho absolutos se comportem da mesma forma que os nomes de caminho relativos. Leve em consideração os dois exemplos seguintes.

Exemplo — Modificando um arquivo

Descrição

Uma entrada está sendo adicionada a uma tabela, ou o objeto é uma nova tabela que será provavelmente modificada por outros programas ou pacotes.

Implementação

Defina o objeto como tipo de arquivo e e pertencente à classe `build`, `awk` ou `sed`. O script que realiza esta tarefa deve remover a si mesmo de maneira tão eficaz como quando se adiciona.

Exemplo

Uma entrada precisa ser adicionada a `/etc/vfstab` em apoio ao novo disco rígido de estado sólido.

A entrada no arquivo `pkgmap` pode ser

```
1 e sed /etc/vfstab ? ? ?
```

O script `request` pergunta ao operador se `/etc/vfstab` deve ser modificado pelo pacote. Se o operador responder "não", então o script request imprimirá as instruções sobre como realizar o trabalho manualmente e executará

```
echo "CLASSES=none" >> $1
```

Se o operador responder "sim", então ele executa

```
echo "CLASSES=none sed" >> $1
```

que ativa o script de ação de classe que fará as modificações necessárias. A classe `sed` significa que o arquivo de pacote `/etc/vfstab` é um programa `sed` que contém as operações de instalação e remoção do arquivo com o mesmo nome no sistema de destino.

Exemplo — Criando um novo arquivo

Descrição

O objeto é um arquivo totalmente novo que provavelmente não será editado mais adiante ou está substituindo um arquivo de outro pacote.

Implementação

Defina o objeto de pacote como tipo de arquivo `f` e o instale usando um script de ação de classe capaz de desfazer alterações.

Exemplo

Um novo arquivo de marca é requerido em `/etc` para proporcionar as informações necessárias para suportar o disco rígido de estado sólido, denominado `/etc/shdisk.conf` . A entrada no arquivo `pkgmap` pode ter esta aparência:

```
.
.
.
1 f newetc /etc/shdisk.conf
.
.
.
```

O script de ação de classe `i.newetc` é responsável pela instalação e por qualquer outro arquivo que precise ir para `/etc`. Ele faz a verificação para se certificar de que não exista outro arquivo lá. Se não existir, ele simplesmente copia o novo arquivo no local. Se já existir um arquivo no local, ele fará o backup do arquivo existente antes de instalar o novo arquivo. O script `r.newetc` remove estes arquivos e restaura os originais, se requerido Abaixo se encontra o fragmento principal do script de instalação.

```
# i.newetc
while read src dst; do
      if [ -f $dst ]; then
        dstfile=‘basename $dst‘
        cp $dst $PKGSAV/$dstfile
      fi
      cp $src $dst
done

if [ "${1}" = "ENDOFCLASS" ]; then
      cd $PKGSAV
      tar cf SAVE.newetc .
      $INST_DATADIR/$PKG/install/squish SAVE.newetc
fi
```

Observe que este usa a variável de ambiente `PKGSAV` para armazenar o backup do arquivo que será substituído. Quando o argumento `ENDOFCLASS` é passado para o script, isso quer dizer que o comando `pkgadd` informa o script que estas são as últimas entradas desta classe, momento em que o script arquiva e compacta os arquivos salvos usando um programa privado de compactação armazenado no diretório de instalação do pacote.

Embora o uso da variável de ambiente `PKGSAV` não seja confiável durante a atualização de um pacote, se o pacote não for atualizado (através de um patch, por exemplo), o arquivo de backup é seguro. O seguinte script de remoção inclui o código para lidar com outros problemas — o fato de que as versões antigas do comando `pkgrm` não passem para os scripts o caminho correto da variável de ambiente `PKGSAV`.

O script de remoção pode ter esta aparência.

```
# r.newetc

# make sure we have the correct PKGSAV
if [ -d $PKG_INSTALL_ROOT$PKGSAV ]; then
      PKGSAV="$PKG_INSTALL_ROOT$PKGSAV"
fi

# find the unsquish program
UNSQUISH_CMD=‘dirname $0‘/unsquish

while read file; do
      rm $file
done

if [ "${1}" = ENDOFCLASS ]; then
      if [ -f $PKGSAV/SAVE.newetc.sq ]; then
         $UNSQUISH_CMD $PKGSAV/SAVE.newetc
      fi

      if [ -f $PKGSAV/SAVE.newetc ]; then
         targetdir=dirname $file    # get the right directory
         cd $targetdir
            tar xf $PKGSAV/SAVE.newetc
            rm $PKGSAV/SAVE.newetc
      fi
fi
```

Este script usa um algoritmo privado desinstalado (`unsquish`) que está no diretório de instalação do banco de dados do pacote. Isso é feito automaticamente pelo comando `pkgadd` no tempo de instalação. Todos os scripts não reconhecidos especificamente como somente de instalação pelo comando `pkgadd` são deixados neste diretório para serem usados pelo comando `pkgrm`. Você não pode saber onde tal diretório está, mas pode ter certeza de que está fixo e contém todos os arquivos de informação e scripts de instalação necessários do pacote. O script encontra o diretório devido ao fato de que o script de ação de classe está com certeza sendo executado a partir do diretório que contém o programa `unsquish`.

Observe, também, que este script não apenas supõe que o diretório de destino é `/etc`. Ele pode realmente estar em `/export/root/client2/etc` . O diretório correto pode ser construído de duas formas.

- Use a construção `${PKG_INSTALL_ROOT}/etc`, ou.
- Tome o nome do diretório de um arquivo passado pelo comando `pkgadd` (que é o que esse script faz).

Usando esta abordagem para cada objeto absoluto do pacote, você pode ter certeza de que o comportamento atual desejável é inalterado ou, pelo menos, recuperável.

Exemplo — Um pacote composto

Abaixo se encontra um exemplo dos arquivos `pkginfo` e `pkgmap` de um pacote composto.

O arquivo pkginfo

```
PKG=SUNWstuf
NAME=software stuff
ARCH=sparc
VERSION=1.0.0,REV=1.0.5
CATEGORY=application
DESC=a set of utilities that do stuff
BASEDIR=/opt
VENDOR=Sun Microsystems, Inc.
HOTLINE=Please contact your local service provider
EMAIL=
MAXINST=1000
CLASSES=none daemon
PSTAMP=hubert990707141632
```

O arquivo pkgmap

```
: 1 1758
1 d none SUNWstuf/EZstuf 0775 root bin
1 f none SUNWstuf/EZstuf/dirdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/usrdel 0555 bin bin 40 773 751310229
1 f none SUNWstuf/EZstuf/filedel 0555 bin bin 40 773 751310229
1 d none SUNWstuf/HRDstuf 0775 root bin
1 f none SUNWstuf/HRDstuf/mksmart 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mktall 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkcute 0555 bin bin 40 773 751310229
1 f none SUNWstuf/HRDstuf/mkeasy 0555 bin bin 40 773 751310229
1 d none /etc    ? ? ?
1 d none /etc/rc2.d ? ? ?
1 e daemon /etc/rc2.d/S70dostuf 0744 root sys 450 223443
1 i i.daemon 509 39560 752978103
1 i pkginfo 348 28411 760740163
1 i postinstall 323 26475 751309908
1 i postremove 402 33179 751309945
1 i preinstall 321 26254 751310019
1 i preremove 320 26114 751309865
1 i r.daemon 320 24573 742152591
```

Enquanto `S70dostuf` pertence à classe `daemon`, os diretórios que lidam com ele (que já estão no local no tempo de instalação) pertencem à classe `none`. Mesmo se os diretórios forem exclusivos deste pacote, você deve deixá-los na classe `none`. A razão disso é que os diretórios precisam ser criados primeiro e excluídos por último, e tal ação é sempre válida para a classe `none`. O comando `pkgadd` cria diretórios. Eles não são copiados do pacote e não são passados a um script de ação de classe a ser criado. Em vez disso, eles são criados pelo comando `pkgadd` antes que ele chame o script de ação de classe de instalação, e o comando `pkgrm` exclui os diretórios após a conclusão do script de ação de classe de remoção.

Isso significa que, se um diretório de uma classe especial contiver objetos na classe `none`, quando o comando `pkgrm` tenta remover o diretório, ele falha porque o diretório não estará vazio a tempo. Se um objeto de classe `none` estiver para ser inserido em um diretório de alguma classe especial, tal diretório não existirá a tempo para aceitar o objeto. O comando `pkgadd` não criará o diretório instantaneamente durante a instalação do objeto e pode não ser capaz de sincronizar os atributos de tal diretório quando exibir finalmente a definição de `pkgmap`.

Observação – Ao atribuir um diretório a uma classe, lembre-se sempre da ordem de criação e exclusão.

Tornando os pacotes instaláveis remotamente

Todos os pacotes *devem* ser remotamente instaláveis. Instalável remotamente significa que você não pressupõe que o administrador que está instalando seu pacote esteja instalando no sistema de arquivos raiz (`/`) do sistema que está executando o comando `pkgadd`. Se, em um dos seus scripts de procedimento, você precisar obter o arquivo `/etc/vfstab` do sistema de destino, será necessário usar a variável de ambiente `PKG_INSTALL_ROOT`. Em outras palavras, o nome de caminho `/etc/vfstab` o levará ao arquivo `/etc/vfstab` do sistema que estiver executando o comando `pkgadd`, mas o administrador pode estar instalando em um cliente em `/export/root/client3`. O caminho `${PKG_INSTALL_ROOT}/etc/vfstab` com certeza o leva ao sistema de arquivos de destino.

Exemplo – Instalando em um sistema cliente

Neste exemplo, o pacote `SUNWstuf` está instalado no `client3`, que está configurado com `/opt` no sistema de arquivos raiz (`/`). Outra versão deste pacote já está instalada no `client3`, e o diretório base está definido como `basedir=/opt/$PKGINST` de um arquivo de administração, `thisadmin`. (Para obter mais informações sobre arquivos de administração, consulte "O arquivo de padrões administrativos" na página 128.) O comando `pkgadd` executado no servidor é:

```
# pkgadd -a thisadmin -R /export/root/client3 SUNWstuf
```

A tabela abaixo lista as variáveis de ambiente e os valores passados aos scripts de procedimento.

TABELA 6–1 Valores passados aos scripts de procedimento

Variável de Ambiente	Valor
`PKGINST`	`SUNWstuf.2`
`PKG_INSTALL_ROOT`	`/export/root/client3`
`CLIENT_BASEDIR`	`/opt/SUNWstuf.2`
`BASEDIR`	`/export/root/client3/opt/SUNWstuf.2`

Exemplo – Instalando em um sistema servidor ou independente

Para instalar em um sistema servidor ou em um sistema independente sob as mesmas circunstâncias do exemplo anterior, o comando é:

```
# pkgadd -a thisadmin SUNWstuf
```

A tabela abaixo lista as variáveis de ambiente e os valores passados aos scripts de procedimento.

TABELA 6–2 Valores passados aos scripts de procedimento

Variável de Ambiente	Valor
PKGINST	SUNWstuf.2
PKG_INSTALL_ROOT	Não definido.
CLIENT_BASEDIR	/opt/SUNWstuf.2
BASEDIR	/opt/SUNWstuf.2

Exemplo – Montando sistemas de arquivos compartilhados

Suponha que o pacote SUNWstuf crie e compartilhe um sistema de arquivos no servidor em /export/SUNWstuf/share. Quando o pacote for instalado no sistema cliente, seus arquivos /etc/vfstab precisam ser atualizados para montar este sistema de arquivos compartilhado. É neste caso que a variável CLIENT_BASEDIR pode ser usada.

A entrada do cliente precisa apresentar o ponto de montagem com relação ao sistema de arquivos do cliente. Esta linha deve ser construída corretamente se a instalação for do servidor ou do cliente. Suponha que o nome do sistema do servidor seja $SERVER. Você pode ir a $PKG_INSTALL_ROOT/etc/vfstab e, usando os comandos sed ou awk, construir a seguinte linha no arquivo /etc/vfstab do cliente.

```
$SERVER:/export/SUNWstuf/share - $CLIENT_BASEDIR/usr nfs - yes ro
```

Por exemplo, para o servidor universe e o sistema cliente client9, a linha no arquivo /etc/vfstab do sistema cliente se assemelharia a:

```
universe:/export/SUNWstuf/share - /opt/SUNWstuf.2/usr nfs - yes ro
```

Ao usar corretamente este parâmetro, a entrada sempre monta o sistema de arquivos do cliente, tanto se estiver sendo construído localmente quanto a partir do servidor.

Atualizando com patches os pacotes

Um patch de um pacote é apenas um pacote pequeno criado para substituir certos arquivos no original. Não há uma verdadeira razão para a distribuição de pacotes pequenos exceto para economizar espaço no meio de distribuição. Você também pode distribuir o pacote original completo com poucos arquivos alterados, ou fornecer acesso ao pacote modificado em uma rede. Contanto que somente estes novos arquivos sejam realmente diferentes (os outros arquivos não foram recompilados), o comando `pkgadd` instala as diferenças. Observe as seguintes diretrizes em relação a atualização com patches dos pacotes.

- Se o sistema for bastante complexo, é sensato estabelecer um sistema de identificação de patches que garanta que dois patches não substituam o mesmo arquivo na tentativa de corrigir comportamentos anômalos. Por exemplo, os números de base do patch da Sun são atribuídos mutuamente a conjuntos exclusivos de arquivos pelos quais eles são responsáveis.
- É necessário poder devolver um patch.

É vital que o número de versão do pacote de patch seja o mesmo do pacote original. Você deve rastrear o status do patch do pacote usando uma entrada diferente do arquivo `pkginfo` com a forma.

`PATCH=`*patch_number*

Se a versão do pacote for alterada em um patch, você cria outra instância do pacote e fica extremamente difícil gerenciar o produto com patch. Este método de patch de instância progressiva oferecia certas vantagens nas primeiras versões do sistema operacional do Oracle Solaris, mas torna chato o gerenciamento de sistemas mais complexos.

Todos os parâmetros de zona do patch devem corresponder aos parâmetros de zona do pacote.

No que diz respeito aos pacotes que formam o sistema operacional do Oracle Solaris, deveria haver somente uma cópia do pacote no banco de dados do pacote, embora possa haver várias instâncias com patches. Para remover um objeto de um pacote instalado (usando o comando `removef`), você precisa saber que instâncias possuem tal arquivo.

Entretanto, se seu pacote (que não faz parte do sistema operacional do Oracle Solaris) precisar determinar o nível do patch de um determinado pacote que *faz* parte do sistema operacional do Oracle Solaris, isso se torna um problema que será resolvido aqui. Os scripts de instalação podem ser um pouco grandes sem causar impacto importante desde que não sejam armazenados no sistema de destino. Com o uso de scripts de ação de classe e vários outros scripts de procedimento, você pode salvar os arquivos alterados usando a variável de ambiente `PKGSAV` (ou em outro diretório mais permanente) a fim de possibilitar a devolução dos patches instalados. Você pode monitorar o histórico do patch definindo as variáveis de ambiente adequadas através de scripts `request`. Os scripts das seções seguintes supõem que pode haver vários patches cujo esquema de numeração tem um significado quando aplicado a um único

pacote. Neste caso, os números de patches individuais representam um subconjunto de funcionalidades relacionado aos arquivos do pacote. Dois números diferentes de patch não podem alterar o mesmo arquivo.

Para criar um pacote pequeno regular em um pacote de patch, os scripts descritos nas seções seguintes podem ser simplesmente colocados no pacote. Todos eles podem ser reconhecidos como componentes de pacote padrão com exceção dos dois últimos denominados `patch_checkinstall` e `patch_postinstall`. Estes dois scripts podem ser incorporados no pacote de devolução se você quiser incluir a capacidade de devolver o patch. Os scripts são bastante simples e as suas tarefas são fáceis.

Observação – Este método de atualizar com patches pode ser usado para colocar patches nos sistemas clientes, mas os diretórios-raiz do cliente no servidor deve ter as permissões corretas para que o usuário `install` ou `nobody` possa ler.

O script `checkinstall`

O script `checkinstall` verifica se tal patch é apropriado para este pacote em particular. Uma vez confirmado, ele constrói a *lista de patches* e a lista de *informações do patch* e, em seguida, insere-as no arquivo de resposta para que sejam incorporadas no banco de dados do pacote.

Uma lista de patches é a lista de patches que afetam o pacote atual. Esta lista de patches está registrada no arquivo `pkginfo` do pacote instalado com uma linha que se assemelha a:

`PATCHLIST=`*`patch_id patch_id ...`*

Uma lista de informações do patch é a lista de patches da qual o patch atual é dependente. Esta lista de patches também está registrada no arquivo `pkginfo` com uma linha que se assemelha a:

```
PATCH_INFO_103203-01=Installed... Obsoletes:103201-01 Requires: \ Incompatibles: 120134-01
```

Observação – Estas linhas (e seus formatos) são declaradas como interface pública. Qualquer empresa que distribua patches para pacotes Oracle Solaris deve atualizar esta lista adequadamente. Quando um patch é entregue, cada pacote dentro do patch contém um script `checkinstall` que realiza esta tarefa. Este mesmo script `checkinstall` também atualiza alguns parâmetros específicos do patch. Esta é a nova arquitetura do patch, que é denominada Direct Instance Patching

---

Neste exemplo, os pacotes originais e seus patches existem no mesmo diretório. Os dois pacotes originais são denominados `SUNWstuf.v1` e `SUNWstuf.v2`, e seus patches são denominados `SUNWstuf.p1` e `SUNWstuf.p2`. Isso significa que poderia ser muito difícil para um script de procedimento saber de que diretório vêm esses arquivos, visto que tudo no nome do pacote

após o ponto (“.”) é extraído para o parâmetro PKG, e a variável de ambiente PKGINST faz referência à instância instalada, não à instância de origem. Para que os scripts de procedimento possam encontrar o diretório de origem, o script checkinstall (que é executado sempre a partir do diretório de origem) faz a pesquisa e repassa o local como a variável SCRIPTS_DIR. Se houvesse somente um pacote no diretório de origem denominado SUNWstuf, então os scripts de procedimento poderiam tê-lo encontrado usando $INSTDIR/$PKG.

```
# checkinstall script to control a patch installation.
# directory format options.
#
#       @(#)checkinstall 1.6 96/09/27 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

INFO_DIR=‘dirname $0‘
INFO_DIR=‘dirname $INFO_DIR‘    # one level up

NOVERS_MSG="PaTcH_MsG 8 Version $VERSION of $PKG is not installed on this system."
ALRDY_MSG="PaTcH_MsG 2 Patch number $Patch_label is already applied."
TEMP_MSG="PaTcH_MsG 23 Patch number $Patch_label cannot be applied until all \
restricted patches are backed out."

# Read the provided environment from what may have been a request script
. $1

# Old systems can’t deal with checkinstall scripts anyway
if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        exit 0
fi

#
# Confirm that the intended version is installed on the system.
#
if [ "${UPDATE}" != "yes" ]; then
        echo "$NOVERS_MSG"
        exit 3
fi

#
# Confirm that this patch hasn’t already been applied and
# that no other mix-ups have occurred involving patch versions and
# the like.
#
Skip=0
active_base=‘echo $Patch_label | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $Patch_label | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+Patchvers_pfx_lnth) } ’‘

# Is this a restricted patch?
if echo $active_base | egrep -s "Patchstrict_str"; then
        is_restricted="true"
```

```
        # All restricted patches are backoutable
        echo "PATCH_NO_UNDO=" >> $1
else
        is_restricted="false"
fi

for patchappl in ${PATCHLIST}; do
        # Is this an ordinary patch applying over a restricted patch?
        if [ $is_restricted = "false" ]; then
                if echo $patchappl | egrep -s "Patchstrict_str"; then
                        echo "$TEMP_MSG"
                        exit 3;
                fi
        fi

        # Is there a newer version of this patch?
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
                        { print substr($0, match($0, "Patchvers_pfx")\
+Patchvers_pfx_lnth) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        echo "PaTcH_MsG 1 Patch number $Patch_label is \
superceded by the already applied $patchappl."
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        # Not newer, it’s the same
                        if [ "$PATCH_UNCONDITIONAL" = "true" ]; then
                                if [ -d $PKGSAV/$Patch_label ]; then
                                        echo "PATCH_NO_UNDO=true" >> $1
                                fi
                        else
                                echo "$ALRDY_MSG"
                                exit 3;
                        fi
                fi
        fi
done

# Construct a list of applied patches in order
echo "PATCHLIST=${PATCHLIST} $Patch_label" >> $1

#
# Construct the complete list of patches this one obsoletes
#
ACTIVE_OBSOLETES=$Obsoletes_label

if [ -n "$Obsoletes_label" ]; then
        # Merge the two lists
        echo $Obsoletes_label | sed ’y/\ /\n/’ | \
        nawk -v PatchObsList="$PATCH_OBSOLETES" ’
        BEGIN {
                printf("PATCH_OBSOLETES=");
                PatchCount=split(PatchObsList, PatchObsComp, " ");

                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
```

```
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                Inserted=0;
                for(PatchIndex in PatchObs) {
                        if (PatchObs[PatchIndex] == $0) {
                                if (Inserted == 0) {
                                        PatchObsCnt[PatchIndex]=PatchObsCnt\
[PatchIndex]+1;
                                        Inserted=1;
                                } else {
                                        PatchObsCnt[PatchIndex]=0;
                                }
                        }
                }
                if (Inserted == 0) {
                        printf ("%s@1 ", $0);
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] != 0) {
                                printf("%s@%d ", PatchObs[PatchIndex], \
PatchObsCnt[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1
        # Clear the parameter since it has already been used.
        echo "Obsoletes_label=" >> $1

        # Pass it's value on to the preinstall under another name
        echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $1
fi

#
# Construct PATCH_INFO line for this package.
#

tmpRequire=`nawk -F= ' $1 ~ /REQUIR/ { print $2 } ' $INFO_DIR/pkginfo `
tmpIncompat=`nawk -F= ' $1 ~ /INCOMPAT/ { print $2 } ' $INFO_DIR/pkginfo `

if [ -n "$tmpRequire" ] && [ -n "$tmpIncompat" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
          Incompatibles: $tmpIncompat" >> $1
elif [ -n "$tmpRequire" ]
then
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: $tmpRequire \
Incompatibles: " >> $1
elif [ -n "$tmpIncompat" ]
then
```

```
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: \
$tmpIncompat" >> $1
else
        echo "PATCH_INFO_$Patch_label=Installed: `date` From: `uname -n` \
          Obsoletes: $ACTIVE_OBSOLETES Requires: Incompatibles: " >> $1
fi

#
# Since this script is called from the delivery medium and we may be using
# dot extensions to distinguish the different patch packages, this is the
# only place we can, with certainty, trace that source for our backout
# scripts. (Usually $INST_DATADIR would get us there).
#
echo "SCRIPTS_DIR=`dirname $0`" >> $1

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

O script `preinstall`

O script `preinstall` inicializa o arquivo `prototype`, os arquivos de informação e os scripts de instalação para que o pacote de devolução seja construído. Este script é muito simples e os scripts restantes deste exemplo permitem somente que um pacote de devolução descreva arquivos regulares.

Se quiser restaurar os links simbólicos, os links de disco rígido, os dispositivos e os pipes nomeados em um pacote de devolução, você pode modificar o script `preinstall` para que use o comando `pkgproto` para comparar o arquivo `pkgmap` entregue aos arquivos instalados e, então, criar uma entrada do arquivo `prototype` para cada não-arquivo a ser alterado no pacote de devolução. O método que você deve usar é semelhante ao método do script de ação de classe.

Os scripts `patch_checkinstall` e `patch_postinstall` são inseridos na árvore de origem do pacote do script `preinstall`. Estes dois scripts desfazem as ações do patch.

```
# This script initializes the backout data for a patch package
# directory format options.
#
#       @(#)preinstall 1.5 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH
recovery="no"

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
```

```
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
    if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] || [ -d \
"$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
        recovery="yes"
    fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$BUILD_DIR/files/reloc
ROOT_DIR=$BUILD_DIR/files/root
PROTO_FILE=$BUILD_DIR/prototype
PKGINFO_FILE=$BUILD_DIR/pkginfo
THIS_DIR=`dirname $0`

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#

        exit 0
fi

#
# Unless specifically denied, initialize the backout patch data by
# creating the build directory and copying over the original pkginfo
# which pkgadd saved in case it had to be restored.
#
if [ "$PATCH_NO_UNDO" != "true" ] && [ "$recovery" = "no" ]; then
        if [ -d $BUILD_DIR ]; then
                rm -r $BUILD_DIR
        fi

        # If this is a retry of the same patch then recovery is set to
        # yes. Which means there is a build directory already in
        # place with the correct backout data.

        if [ "$recovery" = "no" ]; then
                mkdir $BUILD_DIR
                mkdir -p $RELOC_DIR
                mkdir $ROOT_DIR
        fi

        #
        # Here we initialize the backout pkginfo file by first
        # copying over the old pkginfo file and themn adding the
        # ACTIVE_PATCH parameter so the backout will know what patch
        # it's backing out.
        #
```

```
# NOTE : Within the installation, pkgparam returns the
# original data.
#
pkgparam -v $PKGINST | nawk ’
        $1 ~ /PATCHLIST/        { next; }
        $1 ~ /PATCH_OBSOLETES/  { next; }
        $1 ~ /ACTIVE_OBSOLETES/ { next; }
        $1 ~ /Obsoletes_label/  { next; }
        $1 ~ /ACTIVE_PATCH/     { next; }
        $1 ~ /Patch_label/      { next; }
        $1 ~ /UPDATE/   { next; }
        $1 ~ /SCRIPTS_DIR/      { next; }
        $1 ~ /PATCH_NO_UNDO/    { next; }
        $1 ~ /INSTDATE/ { next; }
        $1 ~ /PKGINST/  { next; }
        $1 ~ /OAMBASE/  { next; }
        $1 ~ /PATH/     { next; }
        { print; } ’ > $PKGINFO_FILE
echo "ACTIVE_PATCH=$Patch_label" >> $PKGINFO_FILE
echo "ACTIVE_OBSOLETES=$ACTIVE_OBSOLETES" >> $PKGINFO_FILE

# And now initialize the backout prototype file with the
# pkginfo file just formulated.
echo "i pkginfo" > $PROTO_FILE

# Copy over the backout scripts including the undo class
# action scripts
for script in $SCRIPTS_DIR/*; do
        srcscript=‘basename $script‘
        targscript=‘echo $srcscript | nawk ’
                { script=$0; }
                /u\./ {
                        sub("u.", "i.", script);
                        print script;
                        next;
                }
                /patch_/ {
                        sub("patch_", "", script);
                        print script;
                        next;
                }
                { print "dont_use" } ’‘
        if [ "$targscript" = "dont_use" ]; then
                continue
        fi

        echo "i $targscript=$FILE_DIR/$targscript" >> $PROTO_FILE
        cp $SCRIPTS_DIR/$srcscript $FILE_DIR/$targscript
done
 #
# Now add entries to the prototype file that won’t be passed to
# class action scripts. If the entry is brand new, add it to the
# deletes file for the backout package.
#
Our_Pkgmap=‘dirname $SCRIPTS_DIR‘/pkgmap
BO_Deletes=$FILE_DIR/deletes

nawk -v basedir=${BASEDIR:-/} ’
        BEGIN { count=0; }
```

```
                {
                        token = $2;
                        ftype = $1;
                }
                $1 ~ /[#\!:]/ { next; }
                $1 ~ /[0123456789]/ {
                        if ( NF >= 3) {
                                token = $3;
                                ftype = $2;
                        } else {
                                next;
                        }
                }
                { if (ftype == "i" || ftype == "e" || ftype == "f" || ftype == \
"v" || ftype == "d") { next; } }
                {
                        equals=match($4, "=")-1;
                        if ( equals == -1 ) { print $3, $4; }
                        else { print $3, substr($4, 0, equals); }
                }
                ' < $Our_Pkgmap | while read class path; do
                        #
                        # NOTE: If pkgproto is passed a file that is
                        # actually a hard link to another file, it
                        # will return ftype "f" because the first link
                        # in the list (consisting of only one file) is
                        # viewed by pkgproto as the source and always
                        # gets ftype "f".
                        #
                        # If this isn't replacing something, then it
                        # just goes to the deletes list.
                        #
                        if valpath -l $path; then
                                Chk_Path="$BASEDIR/$path"
                                Build_Path="$RELOC_DIR/$path"
                                Proto_From="$BASEDIR"
                        else    # It's an absolute path
                                Chk_Path="$PKG_INSTALL_ROOT$path"
                                Build_Path="$ROOT_DIR$path"
                                Proto_From="$PKG_INSTALL_ROOT"
                        fi
                         #
                        # Hard links have to be restored as regular files.
                        # Unlike the others in this group, an actual
                        # object will be required for the pkgmk.
                        #
                        if [ -f "$Chk_Path" ]; then
                                mkdir -p `dirname $Build_Path`
                                cp $Chk_Path $Build_Path
                                cd $Proto_From
                                pkgproto -c $class "$Build_Path=$path" 1>> \
$PROTO_FILE 2> /dev/null
                                cd $THIS_DIR
                        elif [ -h "$Chk_Path" -o \
                             -c "$Chk_Path" -o \
                             -b "$Chk_Path" -o \
                             -p "$Chk_Path" ]; then
                                pkgproto -c $class "$Chk_Path=$path" 1>> \
$PROTO_FILE 2> /dev/null
```

```
                        else
                                echo $path >> $BO_Deletes
                        fi
                done
fi

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

O script de ação de classe

O script de ação de classe cria uma cópia de cada arquivo que substitui um arquivo existente e adiciona uma linha correspondente ao arquivo `prototype` do pacote de devolução. Tudo isso é feito com scripts `nawk` muito simples. O script de ação de classe recebe uma lista de pares origem/destino composta de arquivos comuns que não combinam com os arquivos instalados correspondentes. Os links simbólicos e outros não-arquivos devem ser tratados no script `preinstall`.

```
# This class action script copies the files being replaced
# into a package being constructed in $BUILD_DIR. This class
# action script is only appropriate for regular files that
# are installed by simply copying them into place.
#
# For special package objects such as editable files, the patch
# producer must supply appropriate class action scripts.
#
# directory format options.
#
#       @(#)i.script 1.6 96/05/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

ECHO="/usr/bin/echo"
SED="/usr/bin/sed"
PKGPROTO="/usr/bin/pkgproto"
EXPR="/usr/bin/expr"    # used by dirname
MKDIR="/usr/bin/mkdir"
CP="/usr/bin/cp"
RM="/usr/bin/rm"
MV="/usr/bin/mv"

recovery="no"
Pn=$$
procIdCtr=0

CMDS_USED="$ECHO $SED $PKGPROTO $EXPR $MKDIR $CP $RM $MV"
```

```
LIBS_USED=""

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

# Check to see if this is a patch installation retry.
if [ "$INTERRUPTION" = "yes" ]; then
        if [ -d "$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST" ] ||
\
[ -d "$PATCH_BUILD_DIR/$Patch_label.$PKGINST" ]; then
                recovery="yes"
        fi
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
PROGNAME=‘basename $0‘

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        PATCH_NO_UNDO="true"
fi

# Since this is generic, figure out the class.
Class=‘echo $PROGNAME | nawk ’ { print substr($0, 3)  }’‘

# Since this is an update, $BASEDIR is guaranteed to be correct
BD=${BASEDIR:-/}

cd $BD

#
# First, figure out the dynamic libraries that can trip us up.
#
if [ -z "$PKG_INSTALL_ROOT" ]; then
        if [ -x /usr/bin/ldd ]; then
                LIB_LIST=‘/usr/bin/ldd $CMDS_USED | sort -u | nawk ’
                        $1 ~ /\// { continue; }
                        { printf "%s ", $3 } ’‘
        else
                LIB_LIST="/usr/lib/libc.so.1 /usr/lib/libdl.so.1
\
/usr/lib/libw.so.1 /usr/lib/libintl.so.1 /usr/lib/libadm.so.1 \
/usr/lib/libelf.so.1"
        fi
fi

#
# Now read the list of files in this class to be replaced. If the file
# is already in place, then this is a change and we need to copy it
# over to the build directory if undo is allowed. If it’s a new entry
```

```
# (No $dst), then it goes in the deletes file for the backout package.
#
procIdCtr=0
while read src dst; do
        if [ -z "$PKG_INSTALL_ROOT" ]; then
                Chk_Path=$dst
                for library in $LIB_LIST; do
                        if [ $Chk_Path = $library ]; then
                                $CP $dst $dst.$Pn
                                LIBS_USED="$LIBS_USED $dst.$Pn"
                                LD_PRELOAD="$LIBS_USED"
                                export LD_PRELOAD
                        fi
                done
        fi

        if [ "$PATCH_PROGRESSIVE" = "true" ]; then
                # If this is being used in an old-style patch, insert
                # the old-style script commands here.

                #XXXOld_CommandsXXX#
                echo >/dev/null # dummy
        fi

        if [ "${PATCH_NO_UNDO}" != "true" ]; then
                #
                # Here we construct the path to the appropriate source
                # tree for the build. First we try to strip BASEDIR. If
                # there's no BASEDIR in the path, we presume that it is
                # absolute and construct the target as an absolute path
                # by stripping PKG_INSTALL_ROOT. FS_Path is the path to
                # the file on the file system (for deletion purposes).
                # Build_Path is the path to the object in the build
                # environment.
                #
                if [ "$BD" = "/" ]; then
                        FS_Path=`$ECHO $dst | $SED s@"$BD"@@`
                else
                        FS_Path=`$ECHO $dst | $SED s@"$BD/"@@`
                fi

                # If it's an absolute path the attempt to strip the
                # BASEDIR will have failed.
                if [ $dst = $FS_Path ]; then
                        if [ -z "$PKG_INSTALL_ROOT" ]; then
                                FS_Path=$dst
                                Build_Path="$ROOT_DIR$dst"
                        else
                                Build_Path="$ROOT_DIR`echo $dst | \
                                    sed s@"$PKG_INSTALL_ROOT"@@`"
                                FS_Path=`echo $dst | \
                                    sed s@"$PKG_INSTALL_ROOT"@@`
                        fi
                else
                        Build_Path="$RELOC_DIR/$FS_Path"
                fi

                if [ -f $dst ]; then    # If this is replacing something
                        cd $FILE_DIR
```

```
                        #
                        # Construct the prototype file entry. We replace
                        # the pointer to the filesystem object with the
                        # build directory object.
                        #
                        $PKGPROTO -c $Class $dst=$FS_Path | \
                            $SED -e s@=$dst@=$Build_Path@ >> \
                            $BUILD_DIR/prototype

                        # Now copy over the file
                        if [ "$recovery" = "no" ]; then
                                DirName=‘dirname $Build_Path‘
                                $MKDIR -p $DirName
                                $CP -p $dst $Build_Path
                        else
                                # If this file is already in the build area skip it
                                if [ -f "$Build_Path" ]; then
                                        cd $BD
                                        continue
                                else
                                        DirName=‘dirname $Build_Path‘
                                        if [ ! -d "$DirName" ]; then
                                                $MKDIR -p $DirName
                                        fi
                                        $CP -p $dst $Build_Path
                                fi
                        fi

                        cd $BD
                else    # It’s brand new
                        $ECHO $FS_Path >> $BO_Deletes
                fi
        fi

        # If special processing is required for each src/dst pair,
        # add that here.
        #
        #XXXSpecial_CommandsXXX#
        #

        $CP $src $dst.$$$procIdCtr
        if [ $? -ne 0 ]; then
                $RM $dst.$$$procIdCtr 1>/dev/null 2>&1
        else
                $MV -f $dst.$$$procIdCtr $dst
                for library in $LIB_LIST; do
                        if [ "$library" = "$dst" ]; then
                                LD_PRELOAD="$dst"
                                export LD_PRELOAD
                        fi
                done
        fi
        procIdCtr=‘expr $procIdCtr + 1‘
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#
```

```
#
# Release the dynamic libraries
#
for library in $LIBS_USED; do
        $RM -f $library
done

exit 0
```

O script `postinstall`

O script `postinstall` cria o pacote de devolução usando as informações fornecidas pelos outros scripts. Visto que os comandos `pkgmk` e `pkgtrans` não requerem o banco de dados do pacote, eles podem ser executados na instalação de um pacote.

No exemplo, é permitido desfazer o patch construindo um pacote em formato de fluxo no diretório salvo (usando a variável de ambiente `PKGSAV)`. Não é óbvio, mas este pacote não deve estar em formato de fluxo, porque o diretório salvo se desloca durante uma operação com `pkgadd`. Se o comando `pkgadd` for aplicado em um pacote no seu próprio diretório salvo, é muito arriscado supor onde a origem do pacote está em um dado momento. Um pacote em formato de fluxo é desempacotado em um diretório temporário e é instalado a partir de tal diretório. (Um pacote em formato de fluxo poderia começar a instalação a partir do diretório salvo e descobrir inesperadamente que foi relocado durante uma operação com `pkgadd` à prova de falha.)

Para determinar quais patches se aplicam a um pacote, use este comando:

```
$ pkgparam SUNWstuf PATCHLIST
```

Com exceção de `PATCHLIST`, que é uma interface pública da Sun, não há nada significativo nos nomes de parâmetro deste exemplo. Em vez de `PATCH`, você pode usar o `SUNW_PATCHID` tradicional, e as outras listas, tais como `PATCH_EXCL` e `PATCH_REQD`, podem ser renomeadas do devido modo.

Se determinados pacotes de patches dependerem de outros pacotes de patches que estão disponíveis no mesmo meio, o script `checkinstall` pode determinar tal dependência e criar um script que será executado pelo script `postinstall` da mesma forma que no exemplo de atualização (consulte "Atualizando pacotes" na página 171).

```
# This script creates the backout package for a patch package
#
# directory format options.
#
# @(#) postinstall 1.6 96/01/29 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#
```

```
# Description:
#       Set the TYPE parameter for the remote file
#
# Parameters:
#       none
#
# Globals set:
#       TYPE

set_TYPE_parameter () {
        if [ ${PATCH_UNDO_ARCHIVE:?????} = "/dev" ]; then
                # handle device specific stuff
                TYPE="removable"
        else
                TYPE="filesystem"
        fi
}

#
# Description:
#       Build the remote file that points to the backout data
#
# Parameters:
#       $1:     the un/compressed undo archive
#
# Globals set:
#       UNDO, STATE

build_remote_file () {
        remote_path=$PKGSAV/$Patch_label/remote
        set_TYPE_parameter
        STATE="active"

        if [ $1 = "undo" ]; then
                UNDO="undo"
        else
                UNDO="undo.Z"
        fi

        cat > $remote_path << EOF
# Backout data stored remotely
TYPE=$TYPE
FIND_AT=$ARCHIVE_DIR/$UNDO
STATE=$STATE
EOF
}

PATH=/usr/sadm/bin:$PATH

if [ "$PKG_INSTALL_ROOT" = "/" ]; then
        PKG_INSTALL_ROOT=""
fi

if [ -n "$PATCH_BUILD_DIR" -a -d "$PATCH_BUILD_DIR" ]; then
        BUILD_DIR="$PATCH_BUILD_DIR/$Patch_label.$PKGINST"
else
        BUILD_DIR="$PKG_INSTALL_ROOT/var/tmp/$Patch_label.$PKGINST"
fi
```

```
if [ ! -n "$PATCH_UNDO_ARCHIVE" ]; then
        PATCH_UNDO_ARCHIVE="none"
fi

FILE_DIR=$BUILD_DIR/files
RELOC_DIR=$FILE_DIR/reloc
ROOT_DIR=$FILE_DIR/root
BO_Deletes=$FILE_DIR/deletes
THIS_DIR=‘dirname $0‘
PROTO_FILE=$BUILD_DIR/prototype
TEMP_REMOTE=$PKGSAV/$Patch_label/temp

if [ "$PATCH_PROGRESSIVE" = "true" ]; then
        # remove the scripts that are left behind
        install_scripts=‘dirname $0‘
        rm $install_scripts/checkinstall \
$install_scripts/patch_checkinstall $install_scripts/patch_postinstall

        # If this is being used in an old-style patch, insert
        # the old-style script commands here.

        #XXXOld_CommandsXXX#

        exit 0
fi
#
# At this point we either have a deletes file or we don’t. If we do,
# we create a prototype entry.
#
if [ -f $BO_Deletes ]; then
        echo "i deletes=$BO_Deletes" >> $BUILD_DIR/prototype
fi

#
# Now delete everything in the deletes list after transferring
# the file to the backout package and the entry to the prototype
# file. Remember that the pkgmap will get the CLIENT_BASEDIR path
# but we have to actually get at it using the BASEDIR path. Also
# remember that removef will import our PKG_INSTALL_ROOT
#
Our_Deletes=$THIS_DIR/deletes
if [ -f $Our_Deletes ]; then
        cd $BASEDIR

        cat $Our_Deletes | while read path; do
                Reg_File=0

                if valpath -l $path; then
                        Client_Path="$CLIENT_BASEDIR/$path"
                        Build_Path="$RELOC_DIR/$path"
                        Proto_Path=$BASEDIR/$path
                else    # It’s an absolute path
                        Client_Path=$path
                        Build_Path="$ROOT_DIR$path"
                        Proto_Path=$PKG_INSTALL_ROOT$path
                fi

                # Note: If the file isn’t really there, pkgproto
```

```
                # doesn't write anything.
                LINE=`pkgproto $Proto_Path=$path`
                ftype=`echo $LINE | nawk '{ print $1 }'`
                if [ $ftype = "f" ]; then
                        Reg_File=1
                fi

                if [ $Reg_File = 1 ]; then
                        # Add source file to the prototype entry
                        if [ "$Proto_Path" = "$path" ]; then
                                LINE=`echo $LINE | sed -e s@$Proto_Path@$Build_Path@2`
                        else
                                LINE=`echo $LINE | sed -e s@$Proto_Path@$Build_Path@`
                        fi

                        DirName=`dirname $Build_Path`
                        # make room in the build tree
                        mkdir -p $DirName
                        cp -p $Proto_Path $Build_Path
                fi

                # Insert it into the prototype file
                echo $LINE 1>>$PROTO_FILE 2>/dev/null

                # Remove the file only if it's OK'd by removef
                rm `removef $PKGINST $Client_Path` 1>/dev/null 2>&1
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Unless specifically denied, make the backout package.
#
if [ "$PATCH_NO_UNDO" != "true" ]; then
        cd $BUILD_DIR   # We have to build from here.

        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                STAGE_DIR="$PATCH_UNDO_ARCHIVE"
                ARCHIVE_DIR="$PATCH_UNDO_ARCHIVE/$Patch_label/$PKGINST"
                mkdir -p $ARCHIVE_DIR
                mkdir -p $PKGSAV/$Patch_label
        else
                if [ -d $PKGSAV/$Patch_label ]; then
                        rm -r $PKGSAV/$Patch_label
                fi
                STAGE_DIR=$PKGSAV
                ARCHIVE_DIR=$PKGSAV/$Patch_label
                mkdir $ARCHIVE_DIR
        fi

        pkgmk -o -d $STAGE_DIR 1>/dev/null 2>&1
        pkgtrans -s $STAGE_DIR $ARCHIVE_DIR/undo $PKG 1>/dev/null 2>&1
        compress $ARCHIVE_DIR/undo
        retcode=$?
        if [ "$PATCH_UNDO_ARCHIVE" != "none" ]; then
                if [ $retcode != 0 ]; then
                        build_remote_file "undo"
```

```
                else
                        build_remote_file "undo.Z"
                fi
        fi
        rm -r $STAGE_DIR/$PKG

        cd ..
        rm -r $BUILD_DIR
        # remove the scripts that are left behind
        install_scripts=`dirname $0`
        rm $install_scripts/checkinstall $install_scripts/patch_\
checkinstall $install_scripts/patch_postinstall
fi

#
# Since this apparently worked, we'll mark as obsoleted the prior
# versions of this patch - installpatch deals with explicit obsoletions.
#
cd ${PKG_INSTALL_ROOT:-/}
cd var/sadm/pkg

active_base=`echo $Patch_label | nawk '
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } '`

List=`ls -d $PKGINST/save/${active_base}*`
if [ $? -ne 0 ]; then
        List=""
fi

for savedir in $List; do
        patch=`basename $savedir`
        if [ $patch = $Patch_label ]; then
                break
        fi

        # If we get here then the previous patch gets deleted
        if [ -f $savedir/undo ]; then
                mv $savedir/undo $savedir/obsolete
                echo $Patch_label >> $savedir/obsoleted_by
        elif [ -f $savedir/undo.Z ]; then
                mv $savedir/undo.Z $savedir/obsolete.Z
                echo $Patch_label >> $savedir/obsoleted_by
        elif  [ -f $savedir/remote ]; then
                `grep . $PKGSAV/$patch/remote | sed 's/STATE=.*/STATE=obsolete/
' > $TEMP_REMOTE`
                rm -f $PKGSAV/$patch/remote
                mv $TEMP_REMOTE $PKGSAV/$patch/remote
                rm -f $TEMP_REMOTE
                echo $Patch_label >> $savedir/obsoleted_by
        elif  [ -f $savedir/obsolete -o -f $savedir/obsolete.Z ]; then
                echo $Patch_label >> $savedir/obsoleted_by
        fi
done

# If additional operations are required for this package, place
# those package-specific commands here.

#XXXSpecial_CommandsXXX#

exit 0
```

O script patch_checkinstall

```
# checkinstall script to validate backing out a patch.
# directory format option.
#
#       @(#)patch_checkinstall 1.2 95/10/10 SMI
#
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#

PATH=/usr/sadm/bin:$PATH

LATER_MSG="PaTcH_MsG 6 ERROR: A later version of this patch is applied."
NOPATCH_MSG="PaTcH_MsG 2 ERROR: Patch number $ACTIVE_PATCH is not installed"
NEW_LIST=""

# Get OLDLIST
. $1

#
# Confirm that the patch that got us here is the latest one installed on
# the system and remove it from PATCHLIST.
#
Is_Inst=0
Skip=0
active_base=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
active_inst=‘echo $ACTIVE_PATCH | nawk ’
        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
for patchappl in ${OLDLIST}; do
        appl_base=‘echo $patchappl | nawk ’
                { print substr($0, 1, match($0, "Patchvers_pfx")-1) } ’‘
        if [ $appl_base = $active_base ]; then
                appl_inst=‘echo $patchappl | nawk ’
                        { print substr($0, match($0, "Patchvers_pfx")+1) } ’‘
                result=‘expr $appl_inst \> $active_inst‘
                if [ $result -eq 1 ]; then
                        puttext "$LATER_MSG"
                        exit 3
                elif [ $appl_inst = $active_inst ]; then
                        Is_Inst=1
                        Skip=1
                fi
        fi
        if [ $Skip = 1 ]; then
                Skip=0
        else
                NEW_LIST="${NEW_LIST} $patchappl"
        fi
done

if [ $Is_Inst = 0 ]; then
        puttext "$NOPATCH_MSG"
        exit 3
fi

#
```

```
# OK, all's well. Now condition the key variables.
#
echo "PATCHLIST=${NEW_LIST}" >> $1
echo "Patch_label=" >> $1
echo "PATCH_INFO_$ACTIVE_PATCH=backed out" >> $1

# Get the current PATCH_OBSOLETES and condition it
Old_Obsoletes=$PATCH_OBSOLETES

echo $ACTIVE_OBSOLETES | sed 'y/\ /\n/' | \
nawk -v PatchObsList="$Old_Obsoletes" '
        BEGIN {
                printf("PATCH_OBSOLETES=");
                PatchCount=split(PatchObsList, PatchObsComp, " ");

                for(PatchIndex in PatchObsComp) {
                        Atisat=match(PatchObsComp[PatchIndex], "@");
                        PatchObs[PatchIndex]=substr(PatchObsComp[PatchIndex], \
0, Atisat-1);
                        PatchObsCnt[PatchIndex]=substr(PatchObsComp\
[PatchIndex], Atisat+1);
                }
        }
        {
                for(PatchIndex in PatchObs) {
                        if (PatchObs[PatchIndex] == $0) {
                                PatchObsCnt[PatchIndex]=PatchObsCnt[PatchIndex]-1;
                        }
                }
                next;
        }
        END {
                for(PatchIndex in PatchObs) {
                        if ( PatchObsCnt[PatchIndex] > 0 ) {
                                printf("%s@%d ", PatchObs[PatchIndex], PatchObsCnt\
[PatchIndex]);
                        }
                }
                printf("\n");
        } ' >> $1

        # remove the used parameters
        echo "ACTIVE_OBSOLETES=" >> $1
        echo "Obsoletes_label=" >> $1

exit 0
```

O script patch_postinstall

```
# This script deletes the used backout data for a patch package
# and removes the deletes file entries.
#
# directory format options.
#
#       @(#)patch_postinstall 1.2 96/01/29 SMI
#
```

```
# Copyright (c) 1995 by Sun Microsystems, Inc.
# All rights reserved
#
PATH=/usr/sadm/bin:$PATH
THIS_DIR=‘dirname $0‘

Our_Deletes=$THIS_DIR/deletes

#
# Delete the used backout data
#
if [ -f $Our_Deletes ]; then
        cat $Our_Deletes | while read path; do
                if valpath -l $path; then
                        Client_Path=‘echo "$CLIENT_BASEDIR/$path" | sed s@//@/@‘
                else    # It’s an absolute path
                        Client_Path=$path
                fi
                rm ‘removef $PKGINST $Client_Path‘
        done
        removef -f $PKGINST

        rm $Our_Deletes
fi

#
# Remove the deletes file, checkinstall and the postinstall
#
rm -r $PKGSAV/$ACTIVE_PATCH
rm -f $THIS_DIR/checkinstall $THIS_DIR/postinstall

exit 0
```

Atualizando pacotes

O processo de atualização de um pacote é muito diferente do processo de substituição de um pacote. Ao mesmo tempo em que há ferramentas especiais que oferecem suporte à atualização de pacotes padrão entregues como parte do sistema operacional do Oracle Solaris, um pacote não incorporado pode estar destinado a oferecer suporte a sua própria atualização — vários exemplos anteriores descreveram os pacotes pensados para o futuro e que controlam o método exato de instalação sob a direção do administrador. Você também pode criar um script `request` para oferecer suporte à atualização direta de um pacote. Se o administrador optar por instalar um pacote de modo a substituir completamente o outro, sem deixar rastro de arquivos obsoletos, os scripts de pacote podem fazê-lo.

O script `request` e o script `postinstall`deste exemplo oferecem um pacote atualizável simples. O script `request` se comunica com o administrador e configura um arquivo simples no diretório `/tmp` para remover a instância de pacote antiga. (Embora o script `request` crie um arquivo (que é proibido), não há problema porque todos têm acesso ao `/tmp`).

O script `postinstall` executa, então, o script shell em `/tmp`, que executa o comando `pkgrm` necessário contra o pacote antigo e, em seguida, se auto-exclui.

Este exemplo ilustra uma atualização básica. Tem menos de cinqüenta linhas de código incluindo algumas mensagens bastante longas. Pode ser estendido para que devolva a atualização ou realize outras transformações principais no pacote conforme requerido pelo criador.

O design de uma opção de atualização da interface de usuário deve estar absolutamente segura de que o administrador está totalmente ciente do processo e solicitou ativamente a atualização em vez da instalação paralela. Não há nada de errado em realizar uma operação complexa como a atualização contanto que a interface de usuário permita que a operação seja realizada com clareza.

O script `request`

```
# request script
control an upgrade installation

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

UPGRADE_MSG="Do you want to upgrade the installed version ?"

UPGRADE_HLP="If upgrade is desired, the existing version of the \
    package will be replaced by this version. If it is not \
    desired, this new version will be installed into a different \
    base directory and both versions will be usable."

UPGRADE_NOTICE="Conflict approval questions may be displayed. The \
    listed files are the ones that will be upgraded. Please \
    answer \"y\" to these questions if they are presented."

pkginfo -v 1.0 -q SUNWstuf.\*

if [ $? -eq 0 ]; then
      # See if upgrade is desired here
      response=‘ckyorn -p "$UPGRADE_MSG" -h "$UPGRADE_HLP"‘
      if [ $response = "y" ]; then
            OldPkg=‘pkginfo -v 1.0 -x SUNWstuf.\* | nawk ’ \
            /SUNW/{print $1} ’‘
            # Initiate upgrade
            echo "PATH=/usr/sadm/bin:$PATH" > $UPGR_SCRIPT
            echo "sleep 3" >> $UPGR_SCRIPT
            echo "echo Now removing old instance of $PKG" >> \
            $UPGR_SCRIPT
            if [ ${PKG_INSTALL_ROOT} ]; then
                  echo "pkgrm -n -R $PKG_INSTALL_ROOT $OldPkg" >> \
                  $UPGR_SCRIPT
            else
                  echo "pkgrm -n $OldPkg" >> $UPGR_SCRIPT
            fi
            echo "rm $UPGR_SCRIPT" >> $UPGR_SCRIPT
            echo "exit $?" >> $UPGR_SCRIPT

            # Get the original package’s base directory
```

```
            OldBD=‘pkgparam $OldPkg BASEDIR‘
            echo "BASEDIR=$OldBD" > $1
            puttext -l 5 "$UPGRADE_NOTICE"
       else
             if [ -f $UPGR_SCRIPT ]; then
                   rm -r $UPGR_SCRIPT
             fi
       fi
fi

exit 0
```

O script `postinstall`

```
# postinstall
to execute a simple upgrade

PATH=/usr/sadm/bin:$PATH
UPGR_SCRIPT=/tmp/upgr.$PKGINST

if [ -f $UPGR_SCRIPT ]; then
      sh $UPGR_SCRIPT &
fi

exit 0
```

Criando pacotes de arquivo de classe

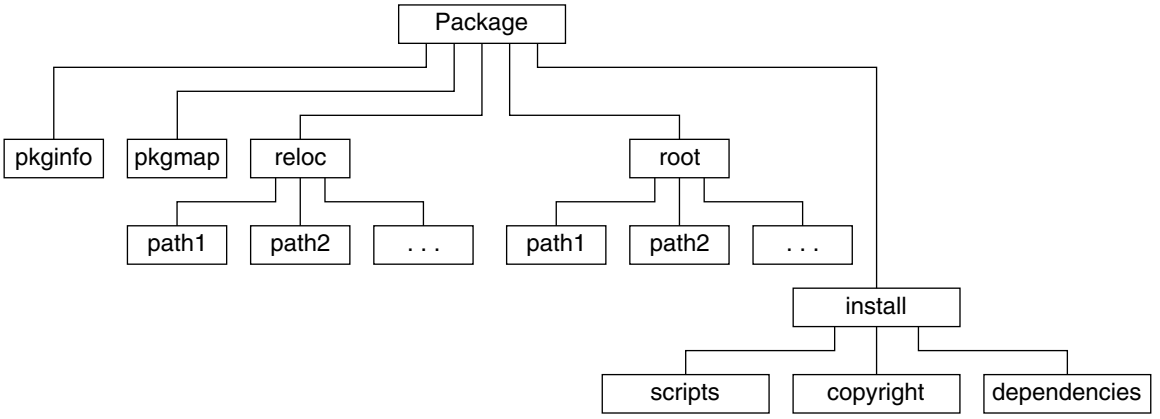
Um pacote de arquivo de classe, que é um aperfeiçoamento da ABI (Application Binary Interface), é aquele no qual certos conjuntos de arquivos são combinados em arquivos simples, ou arquivos de dados, e são opcionalmente compactados ou criptografados. Os formatos de arquivo de classe aumentam a velocidade da instalação inicial em até 30% e melhoram a confiabilidade durante a instalação de pacotes e patches em sistemas de arquivos potencialmente ativos.

As seções seguintes oferecem informações sobre a estrutura do diretório, as palavras-chave e o utilitário `faspac` do pacote de arquivo.

Estrutura do diretório do pacote de arquivo

A entrada do pacote mostrada na ilustração abaixo representa o diretório que contém os arquivos de pacote. Este diretório deve ser o mesmo do pacote.

FIGURA 6–1 Estrutura do diretório do pacote



O quadro abaixo lista as funções dos arquivos e diretórios contidos no diretório do pacote.

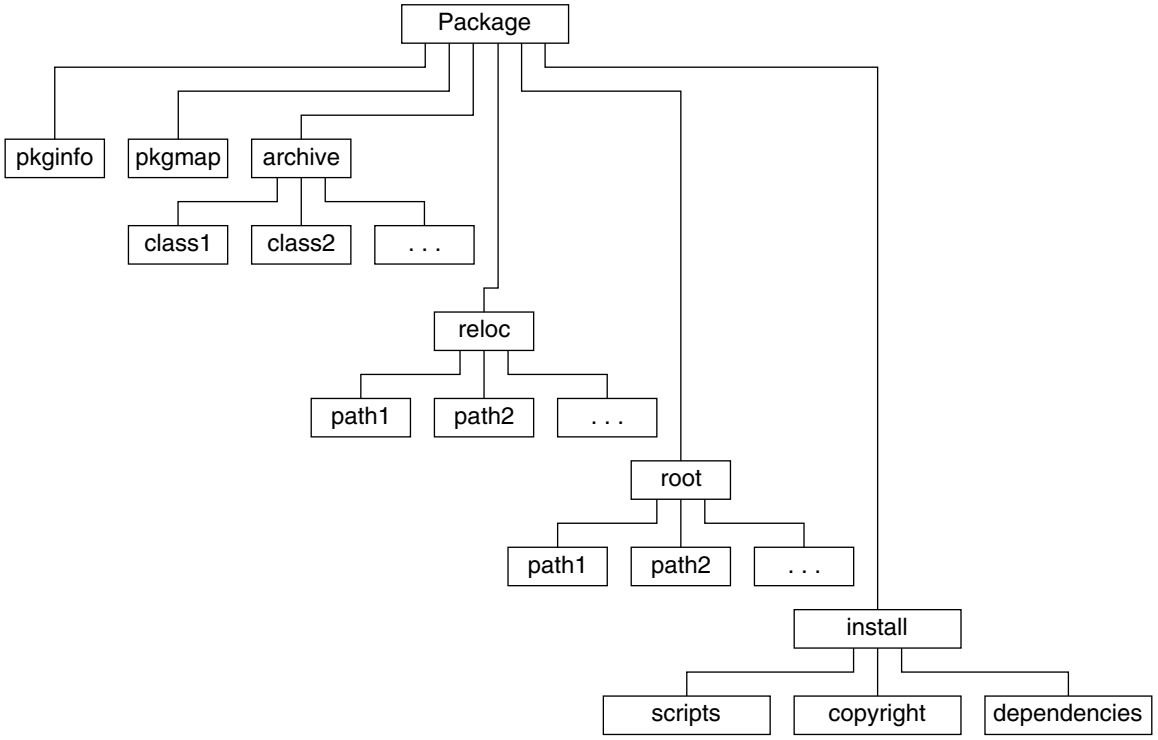
Item	Descrição
`pkginfo`	Arquivo que descreve o pacote como um todo incluindo as variáveis de ambiente especiais e as diretivas de instalação
`pkgmap`	Arquivo que descreve cada objeto que será instalado , como um arquivo, diretório ou pipe
`reloc`	Diretório opcional que contém os arquivos a serem instalados em relação ao diretório base (os objetos relocáveis)
`root`	Diretório opcional que contém os arquivos a serem instalados em relação ao diretório `root` (os objetos-raiz)
`install`	Diretório opcional que contém os scripts e outros arquivos auxiliares (exceto para `pkginfo` e `pkgmap`, todos os arquivos `ftype i` até aqui)

O formato de arquivo de classe permite que o construtor do pacote combine arquivos dos diretórios `reloc` e `root` em arquivos de dados que podem ser compactados, criptografados ou, senão, processados de outra maneira a fim de aumentar a velocidade da instalação, diminuir o tamanho do pacote ou aumentar a segurança deste.

A ABI permite que os arquivos dentro de um pacote sejam atribuídos a uma classe. Todos os arquivos de uma classe específica podem ser instalados no disco usando um método personalizado definido por um script de ação de classe. Este método personalizado pode usar programas disponíveis no sistema de destino ou programas entregues com o pacote. A formato resultante se parece muito ao formato padrão da ABI. Como mostrado na ilustração seguinte, outro diretório é adicionado. Qualquer tipo de arquivos destinado ao arquivo de dados é

simplesmente combinado em um arquivo único e colocado no diretório `archive`. Todos os arquivos arquivados são removidos dos diretórios `reloc` e `root` e um script de ação de classe de instalação é colocado no diretório `install`.

FIGURA 6–2 Estrutura do diretório do pacote de arquivo



Palavras-chave para oferecer suporte aos pacotes de arquivo de classe

Para oferecer suporte a este novo formato de arquivo de classe, três novas interfaces em forma de palavras-chave têm especial significado no arquivo `pkginfo`. Estas palavras-chave são usadas para designar as classes que requerem tratamento especial. A instrução do formato de cada palavra-chave é: `keyword=class1[class2 class3 ...]`. Cada valor da palavra-chave está definido na tabela seguinte.

Palavra-chave	Descrição
PKG_SRC_NOVERIFY	Esta palavra-chave diz a pkgadd para não verificar a existência e as propriedades dos arquivos nos diretórios reloc ou root do pacote entregue se eles pertencerem à classe nomeada. É requerido por todas as classes arquivadas, porque tais arquivos não estão mais em um diretório reloc ou root. São um arquivo de formato privado do diretório archive.
PKG_DST_QKVERIFY	Os arquivos nessas classes são verificados após a instalação usando um algoritmo rápido com pouca ou nenhuma saída de texto. A verificação rápida primeiro define os atributos de cada arquivo corretamente e, então, realiza a verificação para comprovar se a operação foi bem-sucedida. Em seguida, há um teste do tamanho do arquivo e do tempo de modificação em comparação ao pkgmap. Não é realizada nenhuma verificação com checksum e a recuperação de erros é bem pior comparada com a oferecida pelo mecanismo padrão. No caso de blecaute ou de falha do disco durante a instalação, o arquivo do conteúdo pode ficar incompatível com os arquivos instalados. Esta incompatibilidade pode ser resolvida sempre com um comando pkgrm.
PKG_CAS_PASSRELATIVE	Normalmente o script de ação de classe recebe do stdin uma lista de pares de origem e destino constando quais arquivos instalar. As classes atribuídas a PKG_CAS_PASSRELATIVE não obtêm os pares de origem e destino. Em vez disso, elas recebem uma única lista, na qual a primeira entrada é a localização do pacote de origem e o restante são os caminhos de destino. Isso serve especificamente para simplificar a extração de um arquivo de dados. Da localização do pacote de origem, você pode encontrar o arquivo de dados no diretório archive. Os caminhos de destino são, então, passados para a função responsável pela extração do conteúdo do arquivo de dados. Cada caminho de destino fornecido é absoluto ou relativo do diretório base dependendo se o caminho estiver localizado originalmente em root ou reloc. Se esta opção for escolhida, pode ser difícil combinar caminhos absoluto e relativo em uma única classe.

Para cada classe arquivada é necessário um script de ação de classe. É um arquivo que contém comandos do shell Bourne que é executado por pkgadd para realmente instalar os arquivos a partir do arquivo de dados. Se o script de ação de classe for encontrado no diretório install do pacote, o pkgadd passa toda a responsabilidade da instalação para este script. O script de ação de classe é executado com permissões de raiz e pode colocar seus arquivos em quase todos os locais do sistema de destino.

Observação – A única palavra-chave que é absolutamente necessária para implementar um pacote de arquivo de classe é `PKG_SRC_NOVERIFY`. As outras podem ser usadas para melhorar a velocidade da instalação ou conservar o código.

O utilitário `faspac`

O utilitário `faspac` converte um pacote de ABI padrão em um formato de arquivo de classe usado em pacotes incorporados. Este utilitário arquiva usando cpio e compacta usando compress. O pacote resultante tem um diretório adicional na parte superior do diretório denominada `archive`. Neste diretório todos os arquivos de dados terão o nome de classe. O diretório `install` conterá os scripts de ação de classe necessários para desempacotar cada arquivo de dados. Os caminhos absolutos não são arquivados.

O utilitário `faspac` tem o seguinte formato:

```
faspac [-m Archive Method] -a -s -q [-d Base Directory] /
[-x Exclude List] [List of Packages]
```

Cada opção do comando `faspac` está descrita na tabela seguinte.

Opção	Descrição
`-m` *Método de Arquivamento*	Indica um método para arquivar ou compactar. `bzip2` é o utilitário de compactação padrão usado. Para passar para o método zip ou unzip use `-m zip` ou para cpio ou compress use `- m cpio`.
`-a`	Fixa atributos (deve ser raiz para fazê-lo).
`-s`	Indica a tradução do pacote do tipo ABI padrão. Esta opção toma o cpio ou o pacote compactado e o transforma em um formato de pacote compatível com a ABI padrão.
`-q`	Indica o modo silencioso.
`-d` *Diretório Base*	Indica o diretório no qual todos os pacotes presentes agirão de acordo com o requerido pela linha de comando. É mutuamente exclusivo em relação à entrada *List of Packages*.
`-x` *Excluir Lista*	Indica uma lista de pacotes separados por vírgulas, espaços ou entre aspas que serão excluídos do processo.
Lista de Pacotes	Indica a lista de pacotes que serão processados.

Glossário

ABI Consulte interface binária de aplicativos (ABI).

abreviatura de pacote Um nome abreviado de um pacote definido através do parâmetro `PKG` no arquivo `pkginfo`.

armazenamento de chave do pacote Um repositório de certificados e chaves que pode ser consultado pelas ferramentas do pacote.

arquivo `compver` Um método de especificar a compatibilidade com versões anteriores do pacote.

arquivo de controle Arquivo que controla como, onde e se um pacote será instalado. Consulte arquivo de informação e script de instalação.

arquivo de informação Um arquivo que pode definir as dependências do pacote, fornecer uma mensagem de copyright ou reservar espaço em um sistema de destino.

arquivo `depend` Um método de resolver as dependências de pacote básicas. Consulte também o arquivo `compver`.

ASN.1 Consulte notação de sintaxe abstrata 1 (ASN.1)

assinatura digital Uma mensagem codificada usada para verificar a integridade e a segurança de um pacote.

autoridade de certificado Um órgão, tal como a Verisign, que emite certificados usados na autenticação de pacotes.

certificado de confiança Um certificado que contém um certificado único de chave pública que pertence a outra entidade. Os certificados de confiança são usados ao verificar assinaturas digitais e ao iniciar uma conexão a um servidor seguro (SSL).

chave de usuário Uma chave que contém informações criptográficas importantes que diferenciam maiúsculas e minúsculas. Esta informação é armazenada em um formato protegido para evitar uso não autorizado. As chaves de usuário usadas quando um pacote assinado é criado.

chave privada Uma chave de criptografia/descriptografia conhecida somente pela parte ou partes que intercambiam mensagens secretas. Esta chave privada é usada em conjunto com chaves públicas para criar pacotes assinados.

chave pública Um valor gerado como uma chave de criptografia que, combinada como a chave privada derivada da chave pública, pode ser usada para criptografar com eficácia mensagens e assinaturas digitais.

classe Um nome usado para agrupar objetos do pacote. Consulte também script de ação de classe.

copyright O direito de possuir e vender direitos autorais, como software, código-fonte ou documentação. O direito de propriedade deve especificar no CD-ROM e no texto inserido se o copyright pertence à SunSoft ou a terceiros. O proprietário do copyright também deve estar especificado na documentação da SunSoft.

dependência inversa Uma condição na qual um pacote depende da existência de outro pacote. Consulte também o arquivo `depend`.

DER Consulte as regras de codificação distintas.

diretório base O local no qual os objetos relocáveis serão instalados. É definido no arquivo `pkginfo`, usando o parâmetro `BASEDIR`.

identificador de pacote Um sufixo numérico acrescentado à abreviatura de um pacote pelo comando `pkgadd`.

instância do pacote Uma variação de um pacote determinada pela combinação das definições dos parâmetros `PKG`, `ARCH` e `VERSION` no arquivo `pkginfo` do pacote.

interface binária de aplicativos Definição da interface binária do sistema entre aplicativos compilados e o sistema operacional no qual estes são executados.

lista de patches Uma lista de patches que afetam o pacote atual. Esta lista de patches está registrada no pacote instalado no arquivo `pkginfo`.

mensagem de privacidade aprimorada Uma forma de codificar um arquivo usando a codificação base 64 e alguns cabeçalhos opcionais. Usado amplamente para codificação de certificados e chaves privadas em um arquivo que existe em um sistema de arquivos ou em uma mensagem de e-mail.

nome comum Um nome de alias listado no armazenamento de chave de pacote para pacotes assinados.

nome de caminho paramétrico Um nome de caminho que inclui uma especificação de variável.

notação de sintaxe abstrata 1 Uma forma de expressar objetos abstratos. A ASN.1 define, por exemplo, um certificado de chave pública, todos os objetos que fazem parte do certificado e a ordem na qual os objetos são coletados. No entanto, a ASN.1 não específica como os objetos são serializados para armazenamento e transmissão.

objeto de pacote Outro nome de um arquivo de aplicativo que está contido em um pacote a ser instalado em um sistema de destino.

objeto relocável Um objeto de pacote que não precisa de um local de caminho absoluto em um sistema de destino. Em vez disso, seu local é determinado durante o processo de instalação. Consulte também objeto relocável coletivamente e objeto relocável individualmente.

objeto relocável coletivamente Um objeto de pacote que é localizado em relação a uma base de instalação comum. Consulte também diretório base.

objeto relocável individualmente Um objeto de pacote que não está limitado ao mesmo local de diretório que um objeto relocável coletivamente. É definido usando uma variável de instalação no campo *path* no arquivo `prototype`, e o local de instalação é determinado através de um script `request` ou de um script `checkinstall`.

pacote Um conjunto de arquivos e diretórios necessários para um aplicativo de software.

pacote composto Um pacote que contém nomes de caminhos absolutos e relocáveis.

pacote de pré-requisitos Um pacote que depende da existência de outro pacote. Consulte também o arquivo `depend`.

pacote incompatível Um pacote incompatível com um pacote nomeado. Consulte também o arquivo `depend`.

pacote não assinado Um pacote ABI normal sem assinaturas digitais ou criptografia.

pacotes assinados Um pacote com formato de fluxo normal com uma assinatura digital que verifica o seguinte: se o pacote vem de uma entidade que o assinou, se a entidade o assinou, se o pacote não foi modificado depois que a entidade o assinou e se a entidade que o assinou é confiável.

padrão de criptografia de chave pública #12 Um padrão que descreve uma sintaxe para armazenamento de objetos de criptografia em disco. A chave de armazenamento de pacote é mantida neste formato.

padrão de criptografia de chave pública #7 Um padrão que descreve uma sintaxe geral dos dados que podem apresentar criptografia aplicada a eles, tal como assinaturas e envelopes digitais. Um pacote assinado contém uma assinatura PKCS7 integrada.

PEM Consulte mensagem de privacidade aprimorada.

PKCS12 Consulte padrão de criptografia de chave pública #12.

PKCS7 Consulte padrão de criptografia de chave pública #7.

Recomendação X.509 de ITU-T Um protocolo que especifica a sintaxe de certificado de chave pública X.509 amplamente adotada.

regras de codificação distintas Uma representação binária de um objeto ASN.1 e define como um objeto ASN.1 é serializado para armazenamento e transmissão em ambientes de computação. Usado com pacotes assinados.

relocável Um objeto do pacote definido em um arquivo `prototype` com um nome de caminho relativo.

script de ação de classe Um arquivo que define um conjunto de ações que serão realizadas em um grupo de objetos do pacote.

script de instalação Um script que permite que você ofereça procedimentos personalizados de instalação em um pacote.

script de procedimento Um script que define as ações que ocorrem em um determinado ponto durante a instalação e a remoção do pacote.

segmentado Um pacote que não cabe em um volume único, como em um disquete.

`tar` Recuperação de arquivo de gravação. Comando do Oracle Solaris para adicionar ou extrair arquivos de uma mídia.

tempo de construção O período durante o qual um pacote está sendo construído com o comando `pkgmk`.

tempo de instalação O período durante o qual um pacote está sendo instalado com o comando `pkgadd`.

variável de construção Um variável que começa com uma letra minúscula e que é avaliada no tempo de construção.

variável de instalação Um variável que começa com uma letra maiúscula e que é avaliada no tempo de instalação.

X.509 Consulte a recomendação X.509 de ITU-T.

Índice

B

C

R

S

T

U

V