Manual de Empacotamento Debian

Ian Jackson <ijackson@gnu.ai.mit.edu>
Revisado por: David A. Morris <bweaver@debian.org>
Mantenedor: Christian Schwarz <schwarz@debian.org>
Mantenedor: Manoj Srivastava <srivasta@debian.org>
Mantenedor: Julian Gilbey <J.D.Gilbey@qmw.ac.uk>
Mantenedor: The Debian Policy group <debian-policy@lists.debian.org>
Traduzido por: Gustavo Noronha Silva (KoV) <dockov@zaz.com.br>

version 3.2.1.0, 2000-08-24

Resumo

Esse manual descreve os aspectos técnicos de criar pacotes de fonte e binários Debian. Ele não lida com os requerimentos da política do Projeto Debian e assume familiaridade com as funções do `dpkg` da perspectiva de administrador de sistemas. Esse pacote em si é mantido por um grupo de mantenedores que não tem poderes editoriais. No memento a lista de mantenedores é:

1 Michael Alan Dorman <mdorman@debian.org>

2 Richard Braakman <dark@xs4all.nl>

3 Philip Hands <phil@hands.com>

4 Julian Gilbey <J.D.Gilbey@qmw.ac.uk>

5 Manoj Srivastava <srivasta@debian.org>

Nota de Copyright

Sumário

Capítulo 1

Introdução e escopo desse manual

`dpkg` é um conjunto de programas para criar pacotes binários e instalar e removê-los em sistemas Unix.[1]

Os pacotes binários são feitos para o manejo de programas executáveis instalados (usualmente binários compilados) e seus dados de associação, apesar de código fonte e documentação serem providos como parte de alguns pacotes.

Esse manual descreve os aspectos técnicos de criar pacotes binários Debian (`arquivos .deb`). Ele documenta o comportamento dos programas de manejo de pacotes `dpkg`, `dselect` e o jeito com que eles interagem com pacotes.

Ele também documenta a interação entre o núcleo do `dselect` e os scripts de métodos de acesso que ele usa para instalar os pacotes selecionados e descreve como criar um novo método de acesso.

Esse manual não entra em detalhes sobre as opções e o uso das ferramentas de construção e instalação de pacotes. Ele deve, portanto, ser lido em conjunto com as páginas de manual de tais programas.

Os programas utilitários que são providos com o `dpkg` para manejar várias configurações de sistema e assuntos similares, como `update-rc.d` e `install-info`, não são descritos em detalhes aqui - por favor veja suas páginas de manual.

Eles *não* descreve os requerimentos da política imposta aos pacotes Debian, como as permissões dos arquivos e diretórios, requerimentos de documentação, procedimento de envio e por aí vai. Você deve ler o manual de política Debian de empacotamento para esses detalhes. (Muitos deles provavelmente irão se tornar de grande ajuda mesmo se você não planeja enviá-lo e fazê-lo disponível como parte da distribuição.)

Ele assume que o leitor é razoavelmente familiar com o `dpkg` Manual do Administrador de Sistemas. Infelizmente esse manual ainda não existe.

A versão Debian do programa hello GNU da FSF é provido como um example para as pessoas que querem criar pacotes Debian. O pacote Debian `debmake` é recomendado como uma

[1] `dpkg` é primeiramente apontado para o Debian GNU/Linux, mas pode funcionar ou ser portado para outros sistemas.

ferramenta de grande ajuda em criar e manter pacotes Debian. No entanto, enquanto essas ferramentas e exemplos são de ajuda, eles não substituem a necessidade de se ler e seguir a Política e o Manual dos Programadores.

Capítulo 2

Pacotes Binários

O pacote binário tem duas seções principais. A primeira parte consiste de vários arquivos e scripts de informações de controle usados pelo `dpkg` quando instalando e removendo. Veja 'Arquivos de informação de controle do pacote' on the following page.

A segunda parte é um arquivo contendo os arquivos e diretórios a serem instalados.

No futuro, pacotes binários poderão também conter outros componentes, como checksums e assinaturas digitais. O formato para o arquivo é descrito por completo na página de manual `deb(5)`.

2.1 Criando arquivos do pacote - `dpkg-deb`

Toda a manipulação dos arquivos do pacote binário é feita pelo `dpkg-deb`; é o único programa que tem conhecimento do format. (`dpkg-deb` pode ser chamado chamando o `dpkg`, pois o `dpkg` irá saber que as opções requesitadas são apropriadas para o `dpkg-deb` e chamá-lo ao invés de a si com os mesmos argumentos.)

Para criar pacotes binários você precisa fazer uma árvore de diretórios que contenha todos os arquivos e diretórios que você quer ter na parte de dados de sistema de arquivo do pacote. No formato Debian de pacotes fonte esse diretório é, normalmente, `debian/tmp`, relativo ao topo da árvore de fonte do pacote.

Eles devem ter a localização (relativo ao diretório raiz que você está construindo), donos e permissões que você quer que eles tenham no sistema quando eles estão instalados.

Com a versão atual do `dpkg` os mapeamentos de uid/nomedousuário e gid/nomedogrupo para os usuários e grupos sendo usados devem ser os mesmos no sistema onde o pacote é construído e onde for instalado.

Você precisa adicionar um diretório especial à raiz do sistema de arquivos em miniatura que você está construindo: `DEBIAN`. Ele deve conter os arquivos de informação de controle, notávelmente o arquivo de controle do pacote binário (veja 'O arquivo principal de informação de controle: `control`' on page 5).

O diretório `DEBIAN` não irá aparecer no arquivo de sistema de arquivos do pacote e portanto, não será instalado pelo `dpkg` quando o pacote é instalado.

Quando você preparou o pacote, você deve chamar:

```
dpkg --build diretório
```

Isso irá construir o pacote em *`diretório`*`.deb`. (`dpkg` sabe que o `--build` é uma opção do `dpkg-deb`, portanto ele chama o `dpkg-deb` com os mesmo argumentos para construir o pacote.)

Veja a página de manual `dpkg-deb(8)` para detalhes em como examinar o conteúdo desse arquivo recém-criado. Você pode achar a saída dos seguintes comandos esclarecedora:

```
dpkg-deb --info nomedoarquivo.deb
dpkg-deb --contents nomedoarquivo.deb
dpkg --contents nomedoarquivo.deb
```

Para ver o arquivo de copyright do pacote você pode usar esse comando:

```
dpkg --fsys-tarfile nomedoarquivo.deb | tar xof usr/doc/\*copyright | less
```

2.2 Arquivos de informação de controle do pacote

A porção de informações de controle de um pacote binário é uma coleção de arquivos com nomes conhecidos do `dpkg`. Ele irá tratar o conteúdo desses arquivos de modo especial - alguns deles contêm informação usada pelo `dpkg` ao instalar ou remover o pacote; outros são scripts que o mantenedor do pacote quer que o `dpkg` execute.

É possível colocar outros arquivos na área de controle do sistema, mas isso não é, geralmente, uma boa idéia (apesar deles serem no máximo, ignorados).

Aqui está uma pequena lista dos arquivos de controle suportados pelo `dpkg` e um sumário de suas funções.

`control` Esse é a descrição chave usada pelo `dpkg`. Ele especifica o nome e versão do pacote, dá a descrição ao usuário, verifica suas relações com outros pacotes e daí para frente. Veja 'O arquivo principal de informação de controle: `control`' on the facing page.

Ele é usualmente gerado automaticamente da informação contida no pacote fonte pelo `dpkg-gencontrol` e com assistência do `dpkg-shlibdeps`. Veja 'Ferramentas para processar pacotes fonte' on page 7.

`postinst`, `preinst`, `postrm`, `prerm` Esses são arquivos executáveis (normalmente scripts) que o `dpkg` roda durante a instalação, atualização e remoção de pacotes. Eles permitem

que o pacote lide com problemas que são particulares para aquele pacote ou requerem processamento mais complexo que o provido pelo `dpkg`. Detalhes sobre quando e como eles são chamados estão em: 'Scripts de mantenedor de pacotes e procedimento de instalação' on page 33.

É muito imporante fazer esses scripts com igual potência. [1] Isso é para que se um erro ocurrer, o usuário interromper o `dpkg` ou alguma outra circunstância anormal acontecer você não irá deixar o usuário com um pacote muito quebrado.

Os scripts dos mantenedores são garantidos de rodar num terminal e podem interagir com o usuário. Se eles precisam pedir por senhas, fazer interação de tela cheia ou algo assim você deve fazê-los para e de `/dev/tty`, já que o `dpkg` irá em algum ponto redirecionar a saída e a entrada padrões para fazer log do processo de instalação. Do mesmo modo, por causa desses scripts poderem ser executados com com a saída padrão redirecionada para um "encanamento" por propósitos de log, os scripts Perl devem ser configurados para saída sem buffer configurando `$|=1` para que a saída seja impressa imediatamente ao invés de ser guardada num buffer.

Cada script deve retornar um estado 0 para sucesso ou não zero para falhas.

`conffiles` Esse arquivo contém uma lista de arquivos de configuração que serão manuseados pelo `dpkg` (veja 'Manuseio de arquivo de configuração' on page 51). Note que não necessariamente todo arquivo de configuração deve estar listado aqui.

`shlibs` Esse arquivo contém uma lista das bibliotecas compartilhadas providas pelo pacote, com detalhes de dependência para cada uma. Ele é usado pelo `dpkg-shlibdeps` quando determina as dependências requeridas num arquivo control de pacote. O formato do arquivo `shlibs` é descrito em 'O Formato de Arquivo do `shlibs`' on page 58.

2.3 O arquivo principal de informação de controle: `control`

O mais importante arquivo de informação de controle usado pelo `dpkg` quando ele instala um pacote é o `control`. Ele contém todas as 'estatísticas vitais' do pacote.

Os arquivos de controle de pacote binário, construídos dos fontes Debian, são construídos por uma ferramenta especial, `dpkg-gencontrol`, que lê o `debian/control` e `debian/changelog` para encontrar as informações necessárias. Veja 'Pacotes de Fontes' on page 7 para mais detalhes.

Os campos nos arquivos de controle do pacote binário são:

- `Package` (origatório)
- `Version` (obrigatório)
- `Architecture` (obrigatório) [2]
- `Depends`, `Provides`

[1] Isso significa que se ele rodar com sucesso ou falhar e for rodado denovo ele não irá explodir, mas apenas ter certeza de que tudo correu do jeito que deveria.

[2] Esse campo deve aparecer em todos os pacote, apesar de o `dpkg` não o requerir ainda para que pacotes mais antigos possam ser instalados.

- `Essential`
- `Maintainer`
- `Section`, `Priority`
- `Source`
- `Description`
- `Installed-Size`

Uma descrição da sintaxe dos arquivos de controle e o propósito desses campos está disponível em 'Arquivos de controle e seus campos' on page 21.

2.4 Estampas de Tempo

Os mantenedores são encorajados a preserver as datas de modificação do código fonte externo num pacote, tanto quanto seja razoávelmente possível. [3]

[3] A razão é que há alguma informação são transportadas pelo conhecimento da idade do arquivo, por examplo, você pode reconhecer que uma documentação é muito velha, olhando a data de modificação, então seria bom se o tempo de modificação da fonte externa fosse preservado.

Capítulo 3

Pacotes de Fontes

Os pacotes binários do Debian na distribuição são gerados de pacotes fontes Debian, que estão num formato especial para ajudar na fácil e automática construção dos binários.

Houva uma versão anterior do formato fonte do Debian, que está agora acabando. As instruções para converter um pacote do velho estilo são dadas no manual de política do Debian.

3.1 Ferramentas para processar pacotes fonte

Várias ferramentas são dispostas para manipulação de pacotes fonte; elas empacotam e desempacotam fontes, ajudam a construir os pacotes binários e ajudam a manejar a distribuição de novas versões.

Eles são introduzidos e usos típicos são descritos aqui; veja `dpkg-source(1)` para documentação completa sobre seus argumentos e operação.

Para exemplos de como construir um pacote fonte do Debian e como usar essas ferramentas que são usadas pelos pacotes fonte do Debian, por favor veja o pacote exemplo `hello`.

3.1.1 `dpkg-source` - empacota e desenpacota pacotes código do Debian

Esse programa é freqüentemente usado manualmente e é também chamado por scripts de construção automática independentes de pacote como o `dpkg-buildpackage`.

Para desempacotar um pacote é tipicamente chamado com

```
dpkg-source -x .../cominho/para/nomedoarquivo.dsc
```

com o *`nomedoarquivo`*`.tar.gz` e *`nomedoarquivo`*`.diff.gz` (se aplicável) no mesmo diretório. Ele desempacota em *`pacote-versão`*, e se aplicável *`pacote-versão`*`.orig`, no diretório corrente.

Para criar um pacote fonte é normalmente chamado:

```
dpkg-source -b pacote-versão
```

Isso irá criar o `.dsc`, `.tar.gz` e `.diff.gz` (se apropriado) no diretório atual. `dpkg-source` não limpa a árvore de fontes primeiro - isso precisa ser feito separadamente se requerido.

Veja também 'Pacotes fonte como arquivos' on page 18.

3.1.2 `dpkg-buildpackage` - script de controle de criação de pacote.

`dpkg-buildpackage` é um script que chama o `dpkg-source`, o`debian/rules` alvos `clean`, `build` e `binary`, `dpkg-genchanges` e `pgp` para construir um fonte assinado e um pacote binário para envio.

Ele é usualmente chamdo manualmente do topo da árvore de diretórios do fonte, construído ou não. Ele pode ser chamado sem argumentos; argumentos úteis incluem:

`-uc`, `-us` Não assina o arquivo `.changes` ou arquivo `.dsc` do pacote fonte, respectivamente com o PGP.

`-p`*`commando-pgp`* Chama *commando-pgp* ao invés de achar o `pgp` no `PATH`. *commando-pgp* deve se comportar como o `pgp`.

`-r`*`commando-root`* Quando privilégio root é requerido, chama o comando *commando-root*. *comando-root* deve chamar seu primeiro argumento como um comando, do `PATH` se necessário, e passar seus outros argumentos para o comando que ele chamar. Se não houver *comando-root* na linha de comando, então o *dpkg-buildpackage* não irá fazer nada para ganhar privilégio root, então para a maior parte dos pacotes, ele deverá ser chamado como root para iniciar.

`-b`, `-B` Two types of binary-only build and upload - see `dpkg-source(1)`.

3.1.3 `dpkg-gencontrol` - geera arquivos de controle de pacote binário

Esse programa é usualmente chamado do `debian/rules` (veja 'A árvore fonte Debianisada' on page 11) no topo da árvore de diretórios do fonte.

Esse é usualmente feito antes dos arquivos e diretórios na arvoré de diretórios temporária onde o pacote está sendo construído tem suas permissões e são configurados seus donos e o pacote é construído usando o `dpkg-deb` `dpkg-deb/` [1].

`dpkg-gencontrol` precisa ser chamado depois de todos os arquivos que irão estar no pacote terem sido colocados no diretório de construção temporário, para que o cálculo do tamanho do pacote instalado seja correto.

É também necessário que o `dpkg-gencontrol` seja rodado depois do `dpkg-shlibdeps` para que as substituições de variáveis criadas pelo `dpkg-shlibdeps` em `debian/substvars` estejam disponíveis.

Para um pacote que gera apenas um pacote binário e que constrói em `debian/tmp` relativo ao topo do pacote fonte, é geralmente suficiente chamar o `dpkg-gencontrol`.

[1] Isso é para que o arquivo de controle que é produzido tenha as permissões corretas.

Fontes que controem vários binários tipicamente precisarão de algo como:

```
dpkg-gencontrol -Pdebian/tmp-pct -ppacote
```

O `-P` diz ao `dpkg-gencontrol` que o pacote está sendo construído em um diretório não padrão, e o `-p` diz qual arquivo de controle do pacote devem ser gerados.

`dpkg-gencontrol` também adiciona informação à lista de arquivos em `debian/files`, para benefício de por exemplo, uma futura chamada ao `dpkg-genchanges`.

3.1.4 `dpkg-shlibdeps` - calcula dependências de bibliotecas compartilhadas

Esse programa é usalmanete chamado de `debian/rules` antes do `dpkg-gencontrol` (veja 'A árvore fonte Debianisada' on page 11), no topo da árvore de diretórios do fonte.

Seus argumentos são executáveis. [2] para o qual as dependências de bibliotecas compartilhadas devem ser incluídas no arquivo de controle do pacote binário.

Se algumas das bibliotecas compartilhadas encontradas devem apenas garantir um `Recommends` ou `Suggests`, ou se algumas garantem uma `Pre-Depends`, isso pode ser conseguido usando a opção `-d`*campo-de-dependência* antes daqueles executáveis. (Cada `-d` toma efeito até a próxima `-d`.)

`dpkg-shlibdeps` não causa diretamente a saída do arquivo de controle a ser modificado. Ao contrário por padrão, ele adiciona ao arquivo `debian/substvars` configurações de variáveis como `shlibs:Depends`. Essas configurações de variáveis devem ser referênciadas em campos de dependência nas seções por-pacote-binário (per-binary-package) apropriadas do arquivo de controle do fonte.

Por exemplo, o pacote `procps` gera dois tipos de binários, binários C simples como `ps` que requer uma pré-dependência e binários de tela cheia ncurses como o `top` que requer apenas uma recomendação. Ele pode dizer em seu `debian/rules`:

```
dpkg-shlibdeps -dPre-Depends ps -dRecommends top
```

e então no seu arquivo de controle principal `debian/control`:

```
...
Package: procps
Pre-Depends: ${shlibs:Pre-Depends}
Recommends: ${shlibs:Recommends}
...
```

[2] Em uma próxima versão do dpkg, `dpkg-shlibdeps` seria requerido a ser chamado em bibliotecas compartilhadas também. Eles podem ser especificados ou nos locais na árvore de diretórios em que eles serão criados ou em locais na árvore de construção temporária onde eles são instalados antes da criação do pacote.

Fontes que produzem muitos pacotes binários com requerimentos de dependência de bibliotecas compartilhadas podem usar a opção `-p`*`nomedoprefixo`* para sobrescrever o prefixo `shlib:` padrão (uma chamada de `dpkg-shlibdeps` por configuração dessa opção). Eles podem então produzor vários conjuntos de variáveis de dependência, cada um na forma *`nomedoprefixo`*`:`*`campodedependência`*, que podem ser referidos nas partes apropriadas dos arquivos de controle do pacote binário.

3.1.5 `dpkg-distaddfile` - adiciona arquivos ao `debian/files`

Alguns envios de pacotes precisam incluir arquivos diferentes dos arquivos dos pacotes binário e de fonte.

`dpkg-distaddfile` adiciona um arquivo ao `debian/files` para que seja incluído no arquivo `.changes` quando o `dpkg-genchanges` for rodado.

Ele é normalmente chamado do alvo `binário` do `debian/rules`:

```
dpkg-distaddfile nomedoarquivo seção prioridade
```

O *nomedoarquivo* é relativo ao diretório onde `dpkg-genchanges` irá esperar encontrá-lo - ele é normalmente o diretório acima do diretório topo da árvore de código fonte. O alvo do `debian/rules` deve por o arquivo lá antes de chamar o `dpkg-distaddfile`.

O *seção* e o *prioridade* são passados intocados no arquivo `.changes` resultante. Veja '`Section` e `Priority`' on page 24.

3.1.6 `dpkg-genchanges` - gera um arquivo `.changes`de controle de upload

Esse programa é usalmente chamado por scripts de construção automática pacote-independente como o `dpkg-buildpackage`, mas deve ser chamado manualmente.

É chamado normalmente no level topo da árvore de construção de código e quando chamado sem argumentos imprime um arquivo `.changes` baseado na informação que está nos arquivos changelog e control do pacote fonte e os pacotes fonte que deveriam ter sido construídos.

3.1.7 `dpkg-parsechangelog` - produz representação analisada de um changelog

Esse programa é usado internamente pelo `dpkg-source`. Ele pode também ser as vezes útil no `debian/rules` e algum outro lugar. Ele analisa um changelog, `debian/changelog` por padrão e imprime uma representação no formato de arquivo de controle da informação nele para a saída padrão.

3.1.8 `dpkg-architecture` - informação sobre o sistema host e construtor

Esse programa pode ser usado manualmente, mas também é chamado pelo `dpkg-buildpackage` ou `debian/rules` para configurar o ambiente ou fazer variáveis que especificam a arquitetura host e construtora para o processo de construição do pacote.

3.2 A árvore fonte Debianisada

O esquema de arquivo fonte descrito depois é para permitir uma árvore fonte Debianisada com algumas informações de controle associadas para ser reproduzido e transportado facilmente. A árvore fonte Debianisada é uma versão do programa original com certos arquivos adicionados para o benefício do processo de Debianisação e com outras mudanças requeridas feitas ao resto do código fonte do programa e aos scripts de instalação.

Os arquivos extras criador pelo Debian estão no subdiretório `debian` do diretório topo da árvore fonte Debianisada. Eles são descritos abaixo.

3.2.1 `debian/rules` - o script principal de construção

Esse arquivo é um makefile executável e contém as entradas especificas do programa pra compilar o pacote e constuir o(s) pacote(s) binários do código.

Ele deve começar com a linha `#!/usr/bin/make -f`, para que ele possa ser chamado pelo seu nome ao invés de chamar-se o `make` explicitamente.

Já que um script `debian/rules` interativo torna impossível autocompilar aquele pacote e torna também difícil a reprodução do mesmo pacote binário por outras pessoas, todos os **alvos requeridos** devem ser não interativos. No mínimo, os alvos requeridos são os chamado pelo `dpkg-buildpackage`, nomeados, *clean*, *binary*, *binary-arch* e *build*. Também segue que qualquer alvo do qual dependam esses alvos devemser não intereativos.

Os alvos com presença requerida são:

`build` Esse deve executar toda a configuração e compilação não interativa do pacote. Se um pacote tem uma configuração interativa pré-compilação, o pacote fonte Debianisado deve ser construído depois disso ser executado, para que ele possa ser contruído sem que haja reexecução da configuração.

Para alguns pacote, notávelmente aqueles em que a mesma árvore de diretório é compilada de diferentes maneiras para produzir dois pacotes binários, o alvo `build` não faz muito sentido. Para esses pacotes é bom o bastante prover dois (ou mais) alvos (`build-a` e `build-b` ou o que for) para cada uma das maneiras de se compilar o pacote e um alvo `build` que não faz nada. O alvo `binary`terá de construir o pacote em cada uma das maneiras possíveis e fazer o pacote binário de cada um.

O alvo `build` não deve fazer nada que possa requerer privilégios root.

O alvo `build` pode precisar rodar `clena` antes - veja abaixo.

Quando um pacote tem uma rotina de configuração que leva um longo tempo, ou quando os makefiles são desenhados pobremente, ou quando `build` precisa rodar `clean` antes, é uma boa idéia fazer `touch build` quando o processo de construção estiver completo. Isso irá assegurar que se `debian/rules build` for rodado novamente ele não irá reconstruir o programa todo.

`binary`, `binary-arch`, `binary-indep` O alvo `binary` deve ser tudo que é necessário ao usuário para construir o pacote binário. Todos esses alvos devem ser não interativos. Ele é dividido em duas partes: `binary-arch` constrói os arquivos de saída do pacote que são específicos de uma arquitetura particular e `binary-indep` constrói aqueles que não são.

`binary` deve normalmente ser um alvo sem comandos que simplesmente depende do `binary-arch` e do `binary-indep`.

Ambos os alvos `binary-*` devem depender do alvo `build`, acima, para que o pacote seja construído se não foi ainda. Ele deve entào criar o(s) pacote(s) binário(s) relevante(s), usando o `dpkg-gencontrol` para fazer seus arquivos de controle e `dpkg-deb` para construí-los e colocá-los no diretório pai do diretório topo do fonte.

Se nenhum dos alvos `binary-*` tem algo para fazer (isso irá sempre ser o caso, se o fonte gerar apenas um único pacote binário, dependente de arquitetura ou não) ele *deve* ainda existir, mas deve sempre ter sucesso.

'Pacotes Binários' on page 3 descreve como construir pacotes binários.

O alvo `binary` deve ser chamado como root.

`clean` Esse deve desfazer efeitos que os alvos `build` e `binary` podem ter tido, exceto que ele deve deixar qualquer arquivo de saída criado no diretório pai pela execução do `binary`. Esse alvo deve ser não interativo.

Se um arquivo `build` foi passado pelo touch no fim do `build`, como sugerido acima, ele deve ser removido como a primeira coisa que o `clean` faz para que rodar `build` de novo depois de um `clean` interrompido não o faça pensar que tudo já foi feito.

O alvo `clean` deve ser chamado como root se `binary` foi chamado desde o último clean `clean` ter sido chamado como root (já que o `build` pode criar diretórios por exemplo).

`get-orig-source` (opcional) Esse alvo pega a versão mais recente do código fonte original de um site arquivo canônico (via FTP ou WWW, por exemplo), faz qualquer rearranjamento necessário para torná-lo o formato original de arquivo tar de fonte descrito abaixo e o deixa no diretório atual.

Esse alvo deve ser chamado em qualquer diretório e deve tomar cuidade de limpar qualquer arquivo temporário que possa deixar.

Esse alvo é opcional, mas provê-lo se possível é possivelmente uma boa idéia.

Os alvos `build`, `binary` e `clean` devem ser chamados dentro do diretório atual do diretório principal do pacote.

Alvos adicionais podem existir em `debian/rules`, ou publicados ou interfaces não documentadas ou para o uso interno do pacote.

A arquitetura para que nós construímos é determinada pelas variáveis make via dpkg-architecture (veja '`dpkg-architecture` - informação sobre o sistema host e construtor' on page 11). Você pode pegar a arquitetura Debian e a string de especificação de arquitetura estilo GNU para a máquina de construção tanto quanto a máquina host. Aqui está a lista de variáveis make:

- `DEB_*_ARCH` (o arquitetura Debian)
- `DEB_*_GNU_TYPE` (a string de especificação de arquitetura estilo GNU)
- `DEB_*_GNU_CPU` (a parte CPU de DEB_*_GNU_TYPE)
- `DEB_*_GNU_SYSTEM` (a parte de Sistema do DEB_*_GNU_TYPE)

onde * é ou `BUILD` para especificação da máquina de construção ou `HOST` para especificação da máquina para a qual construímos.

Compatibilidade anterior pode ser provida no arquivo rules configurando as variáveis necessárias para valores padrão encaixáveis, por favor se refira à documentação do dpkg-architecture para detalhes.

É importante entender que a string `DEB_*_ARCH` faz apenas determinar para qual arquitetura Debian nós construímos. Ela não deve ser usada para conseguir informações do sistema ou da CPU, as variáveis do estilo GNU deve ser usado para isso.

3.2.2 `debian/control`

Esse arquivo contém detalhes independentes de versão sobre o pacote fonte e sobre o pacote binário que cria.

Ele é uma série de conjuntos de campos de controle, cada um sintaticamente similar a um arquivo de controle de pacote binário. Os conjuntos são separados por uma ou mais linhas em branco. O primeiro conjunto é de informações sobre o pacote fonte em geral; cada conjunto subsequente descreve um pacote binário que a árvore fonte constrói.

A sintática e a semântica dos campos são descritas abaixo em 'Arquivos de controle e seus campos' on page 21.

Os campos (independentes de pacote binário) gerais são:

- `Source` (obrigatório)
- `Maintainer`
- `Section` e `Priority` (classificação, obrigatório)
- `Build-Depends` (interrelações de pacotes fonte)
- `Standards-Version`

Os campos por pacote binário são:

- `Package` (obrigatório)
- `Architecture` (obrigatório)
- `Description`
- `Section` e `Priority` (classificação)

- `Essential`
- `Depends` (interrelações de pacotes binários)

Esses campos são usados pelo `dpkg-gencontrol` para gerar arquivos de controle para pacotes binário (veja abaixo), pelo `dpkg-genchanges` para gerar o arquivo `.changes` para acompanhar o upload, e pelo `dpkg-source` quando ele cria o arquivo de controle fonte `.dsc` como parte de um arquivo fonte.

Os campos aqui podem conter referências a variáveis - seus valores irão ser substituídos pelo `dpkg-gencontrol`, `dpkg-genchanges` ou `dpkg-source` quando eles geram arquivos de controle de saída. Veja ‘`debian/substvars` e substituições de variáveis’ on page 17 para detalhes.

Campos definidos pelo usuário

Campos adicionais adicionados pelo usuário podem ser adicionados ao arquivo de controle do pacote fonte. Esses campos irão ser ignorados e não copiados aos (por exemplo) arquivos de controle do pacote binário ou fonte ou arquivos de controle de upload.

Se você deseja adicionar campos não suportados adicionais a esses arquivos de saída você deve usar o mecanismo descrito aqui.

Campos no arquivo de informações de controle fonte principal com nomes começando com `X` seguido de uma ou mais letras `BCS` e um hífen `-` serão copiados aos arquivos de saída. Só a parte do nome do campo depois do hífen será usado no arquivo de saída. Onde a letra `B` é usada o campo aparecerá nos arquivos de controle do pacote binário, onde a letra `S` é usada nos arquivo de controle do pacote fonte e onde `C` é usado nos arquivos de controle (`.changes`) de upload.

Por exemplo, se o arquivo de controle de informação de fonte principal contém o campo

```
XBS-Comment: Eu fico entre a vela e a estrela.
```

então os arquivos de controle dos pacotes binário e fonte conterão o campo

```
Comment: Eu fico entre a vela e a estrela.
```

3.2.3 `debian/changelog`

Esse arquivo grava as mudanças específicas do Debian do pacote. [3].

Ele tem um formato especial que permite às ferremantas de construção de pacote descobrirem qual versão do pacote está sendo construída e descobrir outras informações específicas de lançamento.

O formato é uma série de entradas como essa::

[3] No entanto não há nada que impeça um autor que é também mantenedor do Debian de usá-lo para todas as suas mudanças, ele terá de ser renomeado se os mantenedores Debian e upstream (externo) são pessoas diferentes.

```
pacote (versão) distribuição(ões); urgency=urgência

 * detalhes de mudanças
 mais detalhes de mudanças
 * ainda mais detalhes de mudanças

-- nome do mantenedor e endereço de e-mail  data
```

pacote e *versão* são a fonte do nome do pacote e número de versão.

distribuição(ões) lista as distribuições nas quais essa versão deve ser instalado quando for enviada - ela é copiada para o campo `Distribution` no arquivo `.changes`. Veja ‘`Distribution`’ on page 26.

urgência é o valor do campo `Urgency` no arquivo `.changes` para o envio. Veja ‘`Urgency`’ on page 27. Não é possível especificar uma urgência contendo vírgulas; vírgulas sào usadas para separar configurações *`palavra-chave=valor`* no formato changelog do `dpkg` (apesar de haer atualmente apenas uma única *palavra-chave* útil, `urgency`).

Os detalhes de mudança podem de fato ser uma série de linhas começando com pelo menos dois espaços, mas convencionalmente cada mudança começa com um asterisco e um espaço separatório e a continuação das linhas são indentados para alinhá-las com o início do texto acima. Linhas em branco podem ser usadas para separar grupos de mudanças, se desejado.

O nome e endereço de e-mail do mantenedor *não* devem necessáriamente ser aqueles do mantenedor do pacote usual. Eles devem ter detalhes da pessoa que fez *essa* versão. A informação aqui será copiada para o arquivo `.changes` e depois usada para enviar uma resposta quando o envio for feito.

A *data* deve ser no formato RFC822 [4]; deve incluir a zona de tempo especificada numericamente, com o nome dela ou abreviação opcionalmente presente como comentário.

A primeira linha ’título’ com o nome do pacote deve começar na margem da esquerda, a linha ‘trailer’ com os detalhes do mantenedor e da data devem ser precedidos por exatamente um espaço. Os detalhes do mantenedor e a data devem ser separadas por, exatamente, dois espaços.

Um modo de Emacs para editar esse formato está disponível ele é chamado `debian-changelog-mode`. Você pode tê-lo selecionado automaticamente quando você edita um changelog Debian adicionando uma cláusula de variáveis locais ao fim do changelog.

Definindo formatos alternativos de changelog

É possível usar um formato diferente do padrão provendo É um analisador para o formato que você quer usar.

[4] Isso é gerado pelo programa `822-date`.

Para que o `dpkg-parsechangelog` rode seu analisador você deve incluir uma linha dentre as 40 últimas do seu arquivo identificando a expressão regular de perl: `\schangelog-format:\s+([0-9a-z]+)\W` A parte em parenteses deve ser o nome do formato. Por exemplo, você pode colcoar:

```
@@@ changelog-format: joebloggs @@@
```

Nomes de formatos de Changelog são strings não vazias de alfan-uméricos.

Se tal linha existir então o `dpkg-parsechangelog` procurará pelo analisador como `/usr/lib/dpkg/parsechangelog/`*`nome-do-formato`* ou `/usr/local/lib/dpkg/parsechangelog/`*`nome-do-formato`*; é um erro não encontrá-lo ali ou não ser um executável. O formato de changelog padrão é o `dpkg` e um analisador é provido com o pacote `dpkg`.

O analisador será chamado com o changelog aberto na entrada padrão no início do arquivo. Ele deve ler o arquivo (deve procurar se deseja) para determinar as informações requeridas e retornar a informação analisada para a saída padrão no formato de uma série de campos de controle no formato padrão. Por padrão ele deve retornar informação apenas sobre a versão mais recente no changelog; ele deve aceitar uma opção `-v`*`versão`* para retornar mudanças de todas as versões presentes *estritamente depois* de *versão* e deve então ser um erro a *versão* não estar presente no changelog.

Os campos são:

- `Source`
- `Version` (obrigatório)
- `Distribution` (obrigatório)
- `Urgency` (obrigatório)
- `Maintainer` (obrigatório)
- `Date`
- `Changes` (obrigatório)

Se várias versões estão sendo retornadas (por causa de uso do `-v`, o valor de urgency deve ser o do maior código de urgência listada no início de qualquer sas versões pedidas seguidas dos comentários (separados por espaços) concatenados de todas as versões pedidas; o mantenedor, versão, distribuição e data devem sempre ser da mais recente versão.

Para o formato do campo `Changes` veja ‘`Changes`’ on page 27.

Se o formato do changelog que está sendo analisado sempre ou quase sempre deixa uma linha em branco entre as notas de mudanças individuais, essas linhas devem ser retiradas, para fazer o resultado de saída compacto.

Se o formato de changelog não contém data ou nome do pacote essa informação deve ser omitida da saída. O analisador não deve tentar sintetizá-la ou encontrá-la de outras fontes.

Se o changelog não tem o formato esperado o analisador deve sair com um estado de saída de não-zero, ao invés de tentar “dar um jeito” e possivelmente mostrar saída errada.

Um analisador de changelog não deve interagir com o usuário.

3.2.4 `debian/substvars` e substituições de variáveis

Quando `dpkg-gencontrol`, `dpkg-genchanges` e `dpkg-source` gera arquivos de controle eles fazem substituições de variáveis em sua saída logo antes de escrevê-la. Substituição de variável tem sua forma `${nome-da-variável}`. O arquivo opocional `debian/substvars` contém substituições de variáveis a serem usadas; variáveis podem também ser configuradas diretamente do `debian/rules` usando a opção `-V` para os comandos de empacotamento de fonte e certas veriáveis predefinidas estão a disposição.

Ele é normalmente gerado e modificado dinamicamente pelos alvos do `debian/rules`; nesse caso ele deve ser removido pelo alvo `clean`.

Veja `dpkg-source(1)` para detalhes mais completos sobre substituição de variáveis do código, incluindo o formato do `debian/substvars`.

3.2.5 `debian/files`

Esse arquivo não é uma parte permanente da árvore fonte; ele é usado enquanto construindo pacotes para gravar quais arquivos estão sendo gerados. `dpkg-genchanges` o usa quando gera um arquivo `.changes`.

Ele não deve existir em um pacote fonte lançado e então ele (e qualquer outro arquio de backup ou arquivo temporário como `files.new` [5]) deve ser removido pelo alvo `clean` Ele deve também ser sábio para garantir um início fresco esvaziando ou removendo-o no início do alvo `binary`.

`dpkg-gencontrol` adiciona uma entrada para esse arquivo para o arquivo `.deb` que será criado pelo `dpkg-deb` do arquivo de controle que ele gera para que para a maioria dos pacote tudo o que precise ser feito com esse arquivo é deletado no `clean`.

Se o upload de um pacote inclui arquivos que não o pacote fonte e qualquer pacote binário os quais os arquivos de controle foram feitos com `dpkg-gencontrol` então eles devem ser colocados no diretório pai do diretório principal do pacote e `dpkg-distaddfile` deve ser chamado para adicionar o arquivo à lista em `debian/files`.

3.2.6 `debian/tmp`

Esse é a localização temporária canônica para a construção dos pacotes binários pelo alvo `binary`. O diretório `tmp` serve como o raiz da árvore do sistema de arquivos porque ele está sendo construído (por exemplo, usando os alvos install dos makefiles do pacote externo e redirecionando a saída lá) e também contém o subdiretório `DEBIAN`. Veja 'Criando arquivos do pacote - `dpkg-deb`' on page 3.

Se vários pacotes binários são gerados da mesma árvore de fonte é usual usar vários diretórios `debian/tmp`*algo* por exemplo `tmp-a` ou `tmp-doc`.

[5] `files.new` é usado como um arquivo temporário pelo `dpkg-gencontrol` e `dpkg-distaddfile` - eles ecrevem uma nova versão de `files` aqui antes de renomeá-lo, para evitar deixar uma cópia corrompida se um erro ocorrer

Quaisquer diretórios `tmp` criados e usados pelo `binary` devem ser, claro, removidos pelo alvo `clean`.

3.3 Pacotes fonte como arquivos

Como existe no site FTP, um pacote fonte Debian consiste de três arquivlos relacionados. Você deve ter as versões certas de cada um para estar apto a usá-los.

Arquivo de controle de fonte Debian - `.dsc` Esse arquivo contém uma série de campos, identificados e separados como os campos no arquivo control do pacote binário. Os campos são listados abaixo; sua sintaxe está descrita axima em 'Arquivos de controle e seus campos' on page 21.

- `Source`
- `Version`
- `Maintainer`
- `Binary`
- `Architecture`
- `Build-Depends` (interrelações de pacotes fonte)
- `Standards-Version`
- `Files`

O arquivo de controle do pacote fonte é gerado pelo `dpkg-source` quando ele constrói o arquivo fonte, de outros arquivos no pacote fonte, descrito acima. Quando desempacotando ele é checado com os arquivos e diretórios nas outras partes do pacote fonte, como descrito abaixo.

Arquivo fonte original - *`pacote_versão-externa`*`.orig.tar.gz` Esse é um arquivo comprimido (com `gzip -9`) `tar` contendo o código fonte do programa do autor externo. O arquivo tar desempacota num diretório *`pacote-versão-externa`*`.orig`, e não contém arquivos em lugar nenhum que nele ou seus subdiretórios.

diff de debianisação - *`pacote_revisão-de-versão-externa`*`.diff.gz` Esse é um diff unificado de contexto (`diff -u`) dando as mudanças que são requeridas para tornar o código original em código Debian. Essas mudanças podem incluir apenas editar e criar arquivos texto. As permissões dos arquivos, os alvos de links simbólicos e as características de arquivos especiais ou pipes não podem ser mudadas e nenhum arquivo pode ser renomeado ou removido.

Todos os diretórios no diff devem existir, exceto o diretório `debian` do diretório principal da árvore do fonte, que será criado pelo `dpkg-source` se necessário quando desempacotando.

O programa `dpkg-source` vai tornar o arquivo `debian/rules` executável automaticamente (veja abaixo).

Se não há código fonte original - por exemplo, se o pacote for especialmente preparado para o Debian ou o mantenedor Debian é o mesmo mantenedor externo - o formato é um pouco

diferente: então não há diff e o arquivo tar é chamdo *pacote_versão*`.tar.gz` e contém um diretório *pacote-versão*.

3.4 Desempacotando um pacote fonte Debian sem o `dpkg-source`

`dpkg-source` é o método recomendado de desempacotar um pacote fonte Debian. No entanto, se não está disponível é possível desempacotar assim:

1 Descompacte o arquivo tar, que será criará um diretório `.orig`
2 Renomeie o diretório `.orig` para *pacote-versão*.
3 Crie o subdiretório `debian` no diretório principal.
4 Aplique o diff usando `patch -p0`.
5 Descompacte o arquivo tar de novo se você quer uma cópia do código fonte original

Não é possível gerar um arquivo fonte válido do Debian sem usar o `dpkg-source`. Em particular, tentar usar o `diff` diretamente para gerar o arquivo `diff.gz` não funcionará.

3.4.1 Restrições de objetos em pacotesfonte

O pacote fonte não pode conter nenhum link duro (hard lynk) [6] [7], aparelhos, arquivos especiais, sockets ou arquivos setgid ou setuid. [8]

As ferramentas de empacotamento de fonte controlam as mudanças entre o pacote original e o Debianisado usando o `diff` e o `patch`. Mudar a árvore fonte original incluída no `.orig.tar.gz` para o fonte Debianisado não deve involver nenhuma mudança que não possa ser administradapor essas ferramentas. Mudanças problemáticas que causem a parada do `dpkg-source` com um erro quando construindo o pacote fonte são:

- Adicionar ou remover links simbólicos, sockets ou pipes.
- Mudar o alvo dos links simbólicos.
- Criar diretórios outros que não o `debian`.
- Mudanças nos conteúdos dos arquivos binários.

Mudanças que causam a apresentação de um aviso mas permite ao `dpkg-source` a continuar de qualquer modo são:

- Remover arquivos, diretórios ou symlinks. [9]
- Arquivos texto modificados nos quais falta a linha final usual (na árvore fonte ou na modificada.).

Mudanças que não são representadas, mas que não são detectadas pelo `dpkg-source`, são:

- Mudar permissões de arquivos (que não `debian/rules`) e diretórios.

O diretório `debian` e o `debian/rules` são manuseados especialmente pelo `dpkg-source` - antes de aplicar as mudanças ele cria o diretório `debian` e depois irá fazer o `debian/rules` executável por qualquer usuário.

[6] Isso não é detectado enquanto se constrói o pacote mas quando se extrai.
[7] Links duros podem ser permitidos em algum ponto no futuro, mas requeriria um grande trabalho.
[8] Diretórios setgid são permitidos.
[9] Renomear um arquivo que não é tratado especialmente - é visto como a remoção do antigo arquivo (que gera um aviso, mas é de outro modo ignorado) e a criação do novo.

Capítulo 4

Arquivos de controle e seus campos

Muitas das ferramentas na suíte `dpkg` munipulam dados num formato comun, conhecido como arquivos de controle. Pacotes binário e de fonte tem dados de controle como os arquivos `.changes` que controlam a instalação dos arquivos enviados e o banco de dados interno do `dpkg` estão em formato similar.

4.1 A sintaxe dos arquivos de controle

Um arquivo consiste de dois ou mais parágrafos de campos. Os parágrafos são separados por linhas em branco. Alguns arquivos de controle apenas permitem um parágrafo; outros permitem vários, casos nos quais cada parágrafo geralmente refere a um pacote diferente.

Cada parágrafo é uma série de campos e valores; cada campo consiste de um nome, seguido por dois pontos (:) e um valor. Ele termina no fim da linha. Espaços horizontais (espaços e tabulações) podem estar antes ou depois do valor e são ignorados; é convenção colocar um único espaço após os dois pontos.

Alguns valores de campos podem pegar várias linhas; nesse caso cada linha de continuação *deve* começar com um espaço ou tabulação. Qualquer espaço ou tabulação no fim de linhas individuais de um valor de campo são ignorados.

Exceto onde onde de outra forma apenas uma linha de dados é permitida e espaços em branco não são significantes num corpo de campo. Espaços não podem nunca aparecer dentro de nomes (de pacotes, arquiteturas, arquivos ou qualquer outra coisa), números de versões ou entre os caracteres de relações de versão de múltiplos caracteres.

Os nomes dos campos não são sensíveis a maíusculas minúsculas, mas é usual capitalizar (deixar maíusculas) os nomes dos campos usando maíusculas e minúsculas mixadas como mostrado abaixo.

Linhas em branco ou linhas consistindo de apenas espaços e tabulações não são permitidas dentro de valores de campos ou entre campos - que seria tomado como um novo parágrafo.

É importante notar que há vários campos que são opcionais já que o `dpkg` e as ferramentas relativas estão interessadas, mas que devem aparecer em todo pacote Debian ou as quais a omissão pode causar problemas. Enquando se escreve arquivos de controle para pacotes Debian você *deve* ler o manual de políticas do Debian em conjunto com os detalhes abaixo e a lista de campos para o arquivo particular.

4.2 Lista de campos

4.2.1 `Package`

O nome do pacote binário. Nomes de pacote consistem de alfa-numéricos e `+ - .` (mais, menos e ponto final). [1]

Eles devem ter pelo menos dois caracteres e devem começar com um alfanumérico. Nas versões atuais do dpkg eles são sensíveis a maíusculas e minúsculas[2]; use nomes de pacotes em minúsculas a menos que o pacote que você esteja construindo (ou referindo em outros campos) já está em letras maiúsculas.

4.2.2 `Version`

Esse lista o número de versão do pacote binário ou fonte - veja 'Numeração de versão' on page 29.

4.2.3 `Architecture`

Essa é a string de arquitetura; é uma palavra apenas para a arquitetura Debian.

`dpkg` irá checar a arquitetura declarada do pacote binário com seu próprio valor que foi compilado com ele antes de instalá-lo.

O valor especial `all` indica que o pacote é independente de arquitetura.

No arquivo `debian/control` principal no pacote fonte ou no arquivo de controle do pacote fonte `.dsc`, uma lista das arquiteturas (separadas por espaços) é também permitido, como o valor especial `any`. Uma lista indica que o fonte irá construir um pacote dependente de arquitetura e irá apenas trabalhar corretamente nas arquiteturas listadas. `any` indica que apesar do pacote fonte não depender de nenhuma arquitetura particular e deve compilar bem em qualquer uma, os pacotes binários produzidos não são independentes de arquitetura mas irão ao invés disso ser especificados para qualquer que seja a arquitetura de construção atual.

Num arquivo `.changes` o campo `Architeture` lista a(s) arquitetura(s) do(s) pacote(s) atualmente sendo enviadas. Irá ser uma lista; se o fonte para o pacote está sendo enviada também a entrada especial `source` está presente.

[1] Os caracteres `@ : = % _` (arroba, dois pontos, igual, por cento e traço em baixo) costumam ser permitidos e ainda são aceitos quando encontrados num arquivo de pacote, mas não pode ser usado em novos pacotes

[2] Isso é um bug,

Veja 'debian/rules - o script principal de construção' on page 11 para informações sobre como conseguir a arquitetura para o processo de construção.

4.2.4 Maintainer

O nome do mantenedor e o endereço de email. O nome deve vir antes, então o endereço de email dentro de sinais de menor e maior <> (no formato RFC822).

Se o nome do mantenedor contém um ponto final então o campo todo não funcionará diretamente como um endereço de email por causa de um erro na sintaxe especificada no RFC822; um programa usando esse campo como um endereço deve checar isso e corrigir o problema se necessário (por exemplo colocando o nome em volta de parenteses e movendo-o para o fim e trazendo o endereço de email para a frente).

Em uns arquivos .changes ou dados changelog analizados ele contém o nome e o endereço de email da pessoa responsável pela versão particular em questão - pode não ser o mantenedor usual do pacote.

Esse campo é usualmente opcional pelo dpkg estar interessado, mas sua ausência durante a construção de um pacote usualmente gera um aviso.

4.2.5 Source

Esse campo identifica o nome do pacote fonte.

Numa informação de fonte principal ou num .changes ou .dsc ou num changelog analizado, ele pode conter apenas o nome do pacote fonte.

No arquivo de controle de um pacote binário ou em um arquivo Packages) ele deve ser seguido pelo número de versão em parenteses. [3] This version number may be omitted (and is, by dpkg-gencontrol) if it has the same value as the Version field of the binary package in question. The field itself may be omitted from a binary package control file when the source package has the same name and version as the binary package.

4.2.6 Campos de interrelações de pacotes: Depends, Pre-Depends, Recommends Suggests, Conflicts, Provides, Replaces

Esses campos descrevem as relações do pacote com outros pacotes. Sua sintaxe e semântica estão descritas em 'Declarando relacionamentos entre os pacotes' on page 43.

4.2.7 Description

Em um pacote binário o arquivo Packages ou o arquivo de controle principal do fonte esse campo contém uma descrição do pacote binário, num formato especial. Veja 'Descrição de pacotes - o campo Description' on page 39 para detalhes.

[3] É usual deixar um espaço depois do nome do pacote se um número de versão for especificado.

Num arquivo `.changes` ele contém um sumário das descrições para os pacotes sendo enviados. A parte do campo antes da primeira linha nova fica em branco; depois disso cada linha tem o nome de um pacote binário e a linha de descrição do pacote binário. Cada linha é indentada com um espaço.

4.2.8 `Essential`

Esse é um campo booleano que pode ocorrer apenas em um arquivo de controle de um pacote binário (ou no arquivo `Packages`) ou em um parágrafo de campos por-pacote de um arquivo de controle principal.

Se ajustado para `yes` então o `dpkg` e o `dselect` irá se recusar a remover o pacote (no entanto pode ser atualizado ou reposto). O outro valor possível é `no`, que é o mesmo que não ter esse campo, de qualquer modo.

4.2.9 `Section` e `Priority`

Esses dois campos classificam o pacote. O `Priority` representa quão importante é que o usuário o tenha instalado; o `Section` representa uma área de aplicação em qual o pacote foi classificado.

Quando eles aparecem no arquivo `debian/control` esses campos dão valores para os subcampos section e priority do campo `Fields` do arquivo `.changes` e dá padrões para o section e priority dos pacotes binários.

section e priority são representadas, apesar de não como campos separados, na informaçào para cada arquivo no campo `-File` de um arquivo `.changes`. O valor de section num `.changes` é usado para decidir onde instalar um pacote no arquivo FTP.

Esses campos não são usados pelo `dpkg` mas pelo `dselect` quando ele organiza pacotes e seleciona os padrões. Veja o manual de política Debian para as prioridades em uso e o critério de seleção de prioridade para um pacote e olhe no arquivo FTP do Debian para uma lista das prioridades atualmente em uso.

Esses campos podem aparecer nos arquivos de controle do pacote binário em cujo caso eles irão prover um valor padrão no caso dos arquivos `Packages` estarem faltando informações. `dpkg` e `dselect` irão apenas usar o valor de um arquivo `.deb` se eles não tiverem outras informações; um valor listado em um arquivo `Packages` irá sempre ter preferência. Por padrão `dpkg-gencontrol` não inclui a seção e a prioridade no arquivo control de um pacote binário - use as opções `-isp`, `-is` ou `-ip` para conseguir tal efeito.

4.2.10 `Binary`

Esse campo é uma lista dos pacotes binários.

Quando ele aparece no arquivo `.dsc` ele é a lista de pacotes binários que um pacote fonte pode produzir. Ele não necessariamente produz todos esses pacotes binários para cada arquitetura.

O arquivo de informação de controle não contém detalhes de quais arquiteturas são apropriadas para quais dos pacotes binários.

Quando ele aparece num arquivo `.changes` ele lista os nomes dos pacotes binários atualmente sendo enviados.

A sintaxe é uma lista de pacotes binários separados por vírgulas. [4] Atualmente os pacotes devem ser separados usando apenas espaços no arquivo `.changesM`

4.2.11 `Installed-Size`

Esse campo aparece nos arquivos de controle dos pacotes binários e nos arquivos `Packages`. Ele dá o total de espaço em disco requerido para instalar o pacote nomeado.

O espaço em disco é representado em kbytes como um número decimal simples.

4.2.12 `Files`

Esse campo contém uma lista de arquivos com informação sobre cada um. A informação e sintaxe exatas sobre cada um. A informação e sintaxe exata variam com o contexto. Em todos os casos a parte do conteúdo do campo na mesma linha que o nome do campo fica vazia. O resto do campo é uma linha por arquivo, cada linha sendo indentada por um espaço e contendo um número de sub-campos separados por espaços.

No arquivo `.dsc` (Debian source control ou controle de fonte Debian) cada linha contém o checksum MD5, tamanho e nome do arquivo do arquivo tar e (se aplicável) arquivo diff que faz o resto do pacote fonte. [5] As formas exatas dos nomes de arquivos são descritas em 'Pacotes fonte como arquivos' on page 18.

No arquivo `.changes` ele contém uma linha por aquivo sendo enviado. Cada linha contém o checksum MD5, tamanho, seção e prioridade e o nome do arquivo. Seção e prioridade são os valores dos campos correspondentes no arquivo de controle principal - veja '`Section` e `Priority`' on the preceding page. Se não foram especificados esses valores então `-` deve ser usado, no entanto, section e priority devem ter valores especificados em novos pacotes para serem instalados corretamente.

O valor especial `byhand` para o section num arquivo `.changes` indica que o arquivo em questão não é um pacote ordinário e deve ser instalado manualmente pelos mantenedores da distribuição. Se o section é `byhand` o priority deve ser `-`.

Se uma revisão Debian do pacote está sendo lançada e nenhum arquivo fonte original está sendo distribuído o `.dsc` deve ainda conter o campo `Files` para o arquivo fonte original *`pacote-versão-externa`*`.orig.tar.gz`, mas o `.changes` deve deixá-la de fora. Nesse caso o fonte original no site de distribuição deve ser exatamente igual, byte a byte ao arquivo original de fonte que foi usado para gerar o arquivo `.dsc` que está sendo enviado.

[4] Um espaço depois de cada vírgula é convencional.
[5] Isto é, as partes que não são o `.dsc`.

4.2.13 `Standards-Version`

A versão mais recente dos padrões (os manuais de política, programadores do `dpkg` e associados) com que o pacote acerta. Isso é atualizado manualmente quando editando o pacote fonte para conformar com os novos padrões; ele pode algumas vezes ser usado para dizer quando um pacote precisa de atenção.

Seu formato é o mesmo daqueles de número de versão exceto que revisões e data não são permitidos - veja 'Numeração de versão' on page 29.

4.2.14 `Distribution`

Num arquivo `.changes` ou saída de changelog analizado ele contém o(s) nome(s) (separados por espaços) da(s) distribuição(ões) onde essa versão do pacote deve ser ou foi instalado. Nomes de distribuições seguem as regras para nomes de pacotes. (Veja '`Package`' on page 22).

Valores atuais de distribuições são:

stable Essa é a versão 'lançada' atual do Debian GNU/Linux. Uma nova versão é lançada aproximadamente a cada 3 meses depois do código em *desenvolvimento* ter sido *congelado* por um mês para testes. Depois de a distribuição estar *estável* apenas consertos de bugs significativos são permitidos. Quando mudanças são feitas a essa distribuição, o número de lançamento é aumentado (por exemplo: 1.2r1 se torna 1.2r2 e então 1.2r3, etc).

unstable Esse valor de distribuição se refere à parte em *desenvolvimento* da distribuição Debian. Novos pacotes, novas versões externas de pacotes e consertos de erros vão na árvore de diretórios *unstable*. Baixe dessa distribuição a seu próprio risco.

contrib Os pacotes dentro desta distribuição não tem o critério para inclusão na distribuição principal do Debian como definido pelo Manual de Política, mas tem o critério para a distribuição *contrib* Não há distinção atualmente entre os pacotes estáveis e instáveis do *contrib* nem do *non-free*. Use seu próprio julgamento para baixar dessa distribuição.

non-free Como os pacote na seção *contrib*, os pacote em *non-free* não tem critério para inclusão na distribuição principal definido pelo Manual de Política. Novamente, use seu julgamento ao baixar dessa distribuição.

experimental Os pacotes com esse valor de distribuição estão marcadas por seus mantenedores como sendo de alto risco. Muitas vezes eles representam betas novos ou pacotes em desenvolvimento de várias fontes que os mantenedores querem que as pessoas experimentem mas não estão prontos para serem parte das outras partes da árvore de distribuição Debian. Pegue esses a seu próprio risco.

frozen De tempos em tempos, (atualmente a cada 3 meses) a distribuição *unstable* entra num estado de 'congelamento de código' antecipando um lançamento como versão *stable*. Durante esse período de teste (normalmente 4 semanas) apenas consertos para erros existentes ou descobertos serão permitidos.

Você deve listar *todas* as distribuições nas quais o pacote deve ser instalado. Exceto em circustâncias não usuais, instalações na *stable* devem também ir na *frozen* (se existir) e *unstable*. Do mesmo modo, instalações na *frozen* devem ir para a *unstable*.

4.2.15 `Urgency`

Esse é a descrição de quão importante é atualizar para essa versão da anterior. Consiste de uma única palavra chave normalmente tendo um dos valores `LOW` `MEDIUM` ou `HIGH` seguidos por um comentário opcional (separado por um espaço) que fica normalmente entre parenteses. Por exemplo:

```
Urgency: LOW (HIGH para usuários de deflexões (diversions))
```

Esse campo aparece no arquivo `.changes` e nos changelogs analizados; seu valor aparece como o valor do atributo `urgency` num changelog do estilo `dpkg` (veja '`debian/changelog`' on page 14).

Palavras chave de urgência não são sensíveis à caixa alta ou baixa.

4.2.16 `Date`

No arquivo `.changes` e em changelogs analizados, esse dá a data em que o pacote foi construído ou editado pela última vez.

4.2.17 `Format`

Esse campo ocorre nos arquivos `.changes` e especifica uma revisão de formato para o arquivo. O formato descrito aqui é da versão `1.5`. A sintaxe do valor do formato é o mesmo que o de versão de pacote exceto que não são permitidos datas ou revisões - veja 'Numeração de versão' on page 29.

4.2.18 `Changes`

Num arquivo `.changes` ou num changelog analizado esse campo contém os dados das mudanças legíveis para humanos, descrevendo as diferenças entre a última versão e a atual.

Não deve haver nada nesse campo antes da primeira linha nova; todas as linhas subsequentes devem ser indentadas por pelo menos um espaço; linhas em branco devem ser representadas por uma linha consistindo apenas de um espaço e um ponto final.

Cada informação de mudança de versão deve ser precedida por uma linha 'título' dando pelo menos a versão, a(s) distribuição(ões) e a urgência, de um jeito legível para o humano.

Se dados de várias versões estão sendo retornados a entrada para a versão mais recente deve ser retornada primeiro e as entradas devem ser separadas por uma representação de linha em branco (a linha 'título' deve também ser seguida por uma linha em branco de representação).

4.2.19 `Filename` e `MSDOS-Filename`

Esses campos no arquivo `Packages` dão o(s) nome(s) de arquivo(s) de (partes de) um pacote no diretório da distribuição, relativo à raiz da hierarquia Debian. Se o pacote foi dividido em várias partes estão todos listados em ordem, separados por espaços.

4.2.20 `Size` e `MD5sum`

Esses campos nos arquivos `Packages` dão o tamanho (em bytes, expressados em decimal) e os checksums MD5 do(s) arquivo(s) que faz(em) um pacote binário na distribuição. Se o pacote é dividido em várias partes os valores para as partes são listados em ordem, separados por espaços.

4.2.21 `Status`

Esse campo no arquivo de estados do `dpkg` grava se o usuário quer um pacote instalado, removido ou deixado em paz, se ele está quebrado (requerendo reinstalação) ou não e qual o estado atual no sistema. Cada uma dessas peças de informação é uma palvra única.

4.2.22 `Config-Version`

Se um pacote não é instalado ou não configurado, esse campo no arquivo de estados do `dpkg` grava a última versão do pacote que foi configurada com sucesso.

4.2.23 `Conffiles`

Esse campo no arquivo de estados do `dpkg` contém informações sobre os arquivos de configuração automaticamente manejados de um pacote. Esse campo *não* deve aparecer em lugar algum em um pacote!

4.2.24 Campos obsoletos

Esses ainda são reconhecidos pelo `dpkg` mas não devem aparecer mais em lugar nenhum.

`Revision`
`Package-Revision`
`Package_Revision` A parte revisão da versão do pacote estava antes num campo separado do arquivo control. Esse campo teve vários outros nomes.
`Recommended` Nome antigo para `Recommends`
`Optional` Nome antigo para `Suggests`.
`Class` Nome antigo para `Priority`.

Capítulo 5

Numeração de versão

Todo pacote tem um número de versão, em seu campo de arquivo de controle chamado `Version`.

o `dpkg` impõe uma ordenação nos números de versão, para que ele possa dizer se os pacotes estão sendo atualizados ou desatualizados e para que o `dselect` possa dizer se um pacote que ele encontra disponível é mais novo que aquele instalado no sistema. O formato de número de versão tem as partes mais significantes (quando uma comparação é realizada) no começo.

O formato de número de versão é: [*época*`/:`]*versão-externa*[`-/revisão-debian`].

Os três componentes aqui são:

época Esse é um inteiro único e positivo, que deve normalmente ser pequeno. Pode ser omitido, caso no qual zero é assumido. Se ele for omitido então a *versão-externa* pode conter alguns ':'s (dois pontos).

É provido permitir que erros nos números de versão de versões antigas de um pacote e também de esquemas de numeração de versões anteriores a serem deixados para trás.

o `dpkg` não irá normalmente mostrar a época a menos que seja essencial (não-zero, ou se a *versão-externa* contém um ':'); o `dselect` não irá mostrar datas na parte principal da mostragem de seleção de pacotes.

versão-externa Essa é a parte principal da versão. É normalmente o número de versão original do pacote ('externo') de qual o pacote `.deb` foi feito, se isso for aplicável. Normalmente ele irá ser do mesmo formato como especificado pelo(s) autor(es) externo(s); no entanto, ele pode precisar ser reformatado para servir no formato e esquema de comparação do `dpkg`.

O comportamento de comparação do `dpkg` com respeito à parte de *versão-externa* do número de versão é obrigatório.

A *versão-externa* pode conter apenas alfanuméricos e os caracteres `. + - :` (ponto final, mais, hífen e dois pontos) e deve começar com um dígito. Se não há *revisão-debian* então os hifens não são permitidos; se não há *época* então dois pontos não são permitidos.

revisão-debian Essa parte da versão representa a versão das modificações que foram feitas ao pacote para fazê-lo um pacote binário do Debian. Está no mesmo formato que a *versão-externa* e `dpkg` compara-o do mesmo jeito.

É opcional; se não está presente então a *versão-externa* não pode conter um hífen. Esse formato representa o caso onde um programa foi escrito especialmente para se tornar um pacote binário Debian e entào há apenas uma 'debianização' dele e assim não é necessária revisão.

É convencional reiniciar a *revisão-debian* em `1` cada vez que a *versão-externa* for aumentado.

`dpkg` irá quebrar a *versão-externa* e a *revisão-debian* usando o último hífen na string. A ausência de uma *revisão-debian* é comparada antes da presença de um (mas note que a *revisão-debian* é a parte menos significante do número de versão).

A *revisão-debian* pode conter apenas alfanuméricos e os caracteres `+` e `.` (mais e ponto final).

As partes *versão-externa* e a *revisão-debian* são comparadas pelo `dpkg` usando o mesmo algoritmo:

As strings são comparadas da esquerda para a direita.

Primeiro a parte inicial de cada string, consistindo inteiramente de caracteres não-dígitos, é determinada. Essas duas partes (uma das quais pode estar vazia) são comparadas lexicograficamente. Se uma diferença é encontrada ela é retornada. A comparação lexicográfica é uma comparação de valores ASCII modificados para que todas as letras se organizem antes das não-letras.

Então a parte inicial do resto de cada string que consiste inteiramente de caracteres dígitos é determinada. Os valores numéricos dessas duas partes são comparados e qualquer diferença encontrada é retornada como o resultado da comparação. Para esses propósitos uma string vazia (que pode apenas ocorrer no fim de um ou ambas strings de versões sendo comparadas) conta como zero.

Esses dois passos são repetidos (cortando a string inicial não-numeral e a de dígitos iniciais para fora do início) até que uma diferença seja encontrada ou ambas as strings tenham se acabado.

Note que o propósito da época é permitir-nos a deixar para trás erros na numeração de versão e lidar com situações nas quais o número de versões muda. *Não* há meios de lidar com números de versões contendo strings de letras as quais o `dpkg` não pode interpretar (como `ALPHA` ou `pre-`, ou com ordenações idiotas (o autor desse manual soube de um pacote cujas versões foram `1.1`, `1.2`, `1.3`, `1`, `2.1`, `2.2`, `2` e daí para frente).

Se um pacote externo tem números de versões problemáticas eles devem ser convertidas para uma forma sã para usar no campo `Version`.

Se você precisa comparar números de versão em um script, você pode usar `dpkg --compare-versions ...`. Digite `dpkg --help` para detalhes em argumentos.

5.1 Números de versão baseados em datas

Em geral, pacotes Debian devem usar a mesma versão que os fontes externas.

No entanto, em alguns casos onde o número de versão é baseado em uma data (ex.: um lançamento de pré-versão em desenvolvimento) o dpkg não pode lidar com esses números de versão atualmente, sem épocas. Por exemplo, dpkg irá considerar '96May01' maior que '96Dec24'.

Para prevenir ter de usar épocas para cada nova versão externa, o número de versão deve ser mudado para o formato a seguir em tais casos: '19960501', '19961224'. A decisão de colocar para o mantenedor externo que se troque o formato do número de versão externo fica a critério do mantenedor.

Note que outros formatos de versões baseados em datas que são analizados corretamente pelo dpkg *não* devem ser mudados.

Pacotes nativos do Debian (ex: pacotes que tenham sido escritos especialmente para o Debian) os quais os números de versão incluem datas devem sempre usar o formato 'YYYYMMDD'.

Capítulo 6

Scripts de mantenedor de pacotes e procedimento de instalação

6.1 Introdução a scripts de mantenedor de pacote

É possível colocar scripts como parte de um pacote os quais o `dpkg` irá rodar para você quando seu pacote for instalado, atualizado ou removido.

Esses scripts devem ser os arquivos `preinst`, `postinst`, `prerm` e `postrm` na área controle do pacote. Eles devem ser apropriadamente executáveis; se eles são scripts (o que é recomendado) eles devem começar com o usual `#!`. Eles devem ser legíveis e executáveis para qualquer um, e não passíveis de escrita por qualquer um.

`dpkg` olha o estado de saída desses scripts. É importante que eles saiam com um estado não-zero se houver um erro, para que o `dpkg` pode parar seu processamento. Para scripts shell isso significa que você *quase sempre* precisa usar `set -e` (isso é normalmente verdade quando se escreve scripts shell, de fato). É também importante, claro, que eles não saiam com um estado não-zero se tudo for bem.

É necessário para os procedimentos de recuperação de erro que os scripts sejam idem-potentes: ex: chamar o mesmo script várias vezes na mesma situação não deve criar problemas. Se a primeira chamada falhou ou foi abortada no meio do caminho por alguma razão, a segunda chamada deve meramente fazer as coisas que foram deixadas não-feitas da primeira vez, se houver, e sair com um estado de erro.

Quando um pacote é atualizado uma combinação dos scripts do velho pacote e do novo é chamada entre outros passos do procedimento de atualização. Se seus scripts vão ser complicados você deve estar avisado disso e pode precisar checar os argumentos para seus scripts.

Falando mais amplamente o `preinst` é chamado antes de (uma versão particular de) um pacote ser instalado e o `postinst` depois; o `prerm` antes (uma versão de) um pacote ser removido e o `postrm` depois.

Programas chamados de dentro dos scripts de mantenedor não devem normalmente ter um caminho antes deles. Antes da instalação ser iniciada o `dpkg` checa para ver se os programas

`ldconfig`, `start-stop-daemon`, `install-info` e `update-rc.d` podem ser encontrados via variável `PATH`. Esses programas e qualquer outro programa que se espera estar no `PATH` devem, portante, serem chamados sem um caminho absoluto. Scripts de mantenedor não devem resetar o `PATH`, mas podem escolher modificá-lo colocando diretórios específicos de pacotes antes ou depois. Essas considerações se aplicam, realmente a todo script shell.

6.2 Resumo das maneiras pelas quais os scripts de mantenedor são chamados

- *novo-preinst* `install`
- *novo-preinst* `install` *versão-antiga*
- *novo-preinst* `upgrade` *versão-antiga*
- *velho-preinst* `abort-upgrade` *nova-versão*
- *postinst* `configure` *versão-mais-recentemente-configurada*
- *velho-postinst* `abort-upgrade` *nova versão*
- *postinst-do-conflitante* `abort-remove in-favour` *pacote nova-versão*
- *postinst-do-desconfigurado* `abort-deconfigure in-favour` *pacote-de-instalação-falha versão* `removing` *pacote-conflitante versão*
- *prerm* `remove`
- *velho-prerm* `upgrade` *nova-versão*
- *nova-prerm* `failed-upgrade` *velho-versão*
- *prerm-do-conflitante* `remove in-favour` *pacote nova-versão*
- *prerm-do-desconfigurado* `deconfigure in-favour` *pacote-sendo-instalado versão* `removing` *pacote-conflitando versão*
- *postrm* `remove`
- *postrm* `purge`
- *velho-postrm* `upgrade` *nova-versão*
- *nova-postrm* `failed-upgrade` *versão-antiga*
- *nova-postrm* `abort-install`
- *novo-postrm* `abort-install` *versão-antiga*
- *novo-postrm* `abort-upgrade` *versão-antiga*
- *postrm-do-que-vai-desaparecer* `disappear` *que-vai-sobrescrever versão-do-que-vai-sobrescrever*

6.3 Detalhes da fase de desempacotamento de instalação ou atualização

O procedimento de instalação/atualização/sobrescrição/ desaparecimento (ex: quando executando `dpkg --unpack`, ou o estágio de desempacotamente do `dpkg --install`) é como segue. Em cada caso se um erro ocorrer a ação é, em geral, é voltada - isso significa que os scripts de mantenedor são rodados com argumentos diferentes em ordem reversa. São as chamadas 'defazedoras de erro' listadas abaixo.

1 1 Se uma versão do pacote já está instalada, chama

```
antigo-prerm upgrade nova-versão
```

2 Se isso retornar erro (ex: um estado de saída não-zero), o dpkg irá tentar ao invés:

```
novo-prerm failed-upgrade antiga-versão
```

Reversão de erros, para ambos os casos acima:

```
antigo-postinst abort-upgrade nova-versão
```

2 Se um pacote 'conflitante' está sendo removido ao mesmo tempo:

1 Se algum pacote dependia daquele pacote conflitante e `--auto-deconfigure` está especificado, chama, para cada pacote desses:

```
prerm-do-desconfigurado deconfigure \
 in-favour pacote-sendo-instalado versão \
  removing pacote-conflitante versão
```

Reversão de erro:

```
postinst-do-desconfigurado abort-deconfigure \
  in-favour pacote-sendo-instalado-mas-falhou versão \
    removing pacote-conflitante versão
```

Os pacotes desconfigurados são marcados como requerendo configuração, para que se o `--install` for usado eles irão se configurados novamente se possível.

2 Para preparar para a remoção de um pacote conflitante, chama:

```
prerm-do-conflitante remove in-favour pacote nova-versão
```

Reversão de erro:

```
postinst-do-conflitante abort-remove \
  in-favour pacote nova-versão
```

3 1 Se o pacote estiver sendo atualizado, chama:

```
novo-preinst upgrade antiga-versão
```

2 De outro modo, se o pacote tem algum arquivo de configuração de uma versão anterior instalada (ex: está no estado 'configuration files only' (arquivos de configuração apenas)):

```
novo-preinst install antiga-versão
```

3 De outro modo (ex: o pacote foi completamente eliminadi):

```
novo-preinst install
```

Desfazendo erros nas versão, respectivamente:

novo-postrm abort-upgrade *antiga-versão*
novo-postrm abort-install *antiga-versão*
novo-postrm abort-install

4 Os arquivos do novo pacote são desempacotados, sobrescrevendo qualquer um que já esteja no sistema, por exemplo, qualquer um da versão passado da versão velha do mesmo pacote ou de outro pacote (backups dos arquivos antigos são deixados lá e se alguma coisa der errado o dpkg irá tentar colocá-los de volta como parte de uma desfeitura de erros.).

É um erro um pacote conter arquivos que estão no sistema em algum outro pacote, a menos que o `Replaces` seja usado (veja 'Replaces - sobrescrevendo arquivos e substituindo pacotes' on page 48). Atualmente a opção `--force-overwrite` está habilitada, fazendo disso apenas um aviso, mas pode não ser o caso.

É um erro mais sério um pacote conter um arquivo plano ou outro tipo de arquivo onde outro pacote tem um diretório (de novo, a menos que o `Replaces` seja usado). Esse erro pode ser passado se desejada usando `--force-overwrite-dir`, mas isso não é aconselhável.

Pacotes que sobrescrevem arquivos de outros produzem comportamentos que apesar determinável é difícil de ser compreendido pelo administrador do sistema. Ele pode levar a causar 'falta' de programas se, por exemplo, um pacote é instalado e sobrescreve um arquivo de outro pacote e é então removido de novo. [1]

Um diretório nunca será substituído por um link simbólico para um diretório ou vice-versa; ao invés disso, o estado existente (link ou não) será deixado como está e o `dpkg` irá seguir o link se existir um.

5 1 Se o pacote está sendo atualizado, chama

antigo-postrm upgrade *nova-versão*

2 Se isso falhar, `dpkg` irá tentar:

novo-postrm failed-upgrade *antiga-versão*

Reversão de erros, para ambos os casos:

antigo-preinst abort-upgrade *nova-versão*

Esse é um ponto sem volta - se o `dpkg` chegar aqui, ele não vai voltar desse ponto se um erro ocorrer. Isso irá deixar o pacote em um estado bem mal, que irá requerir uma reinstalação com sucesso para limpar, mas é quando o `dpkg` começa a fazer coisas que são irreverssíveis.

6 Qualquer arquivo que existia na velha versão do pacote mas não na nova versão são removidos.

7 A nova lista de arquivo substitui a antiga.

[1] Parte do problema é por causa de um discutível erro no `dpkg`.

8 Os novos scripts de mantenedor substituem os velhos.

9 Quaisquer pacotes cujos arquivos foram sobrescritos durante a instalação e que não são atualmente requeridos para dependências, são considerados para serem removidos. Para cada desses pacotes,

   1 `dpkg` chama:

```
postrm-daquele-a-ser-removido disappear \
  o-que-sobrescreverá versão-do-que-sobrescreverá
```

   2 Os scripts de mantenedor são removidos.

   3 É notado no banco de dados de estado como estando em um estado são, chamado não instalado (qualquer arquivos de configuração que ele possa ter são ignorados, ao invés de serem removidos pelo `dpkg`). Note que pacotes que são removidos não tem seu prerm chamados, por causa do `dpkg` não saber antes que o pacote será extinto.

10 Quaisquer arquivos no pacote que estamos desempacotando que estão também listados nas listas de arquivo de outro pacotes são removidos dessas listas. (Isso irá *FIXME: lobotomise?* a lista de arquivo do pacote 'conflitante' se houver um.)

11 Os arquivos de backup feitos durante a instalação, acima, são deletados.

12 O estado do novo pacote é agora são e gravado como 'unpacked' (desempacotado). Aqui está outro ponto sem volta - se a remoção do pacote conflitante falhar nós não revertemos o resto da instalação; o pacote conflitante é deixado em um limbo meio-removido.

13 Se haia um pacote conflitante nós vamos e fazemos as ações de remoção (descritas abaixo), começando com a remoção dos arquivos do pacote conflitante (quaisquer que estejam também em um pacote sendo instalado já foram removidas da lista de arquivos do pacote conflitante e então não são removidas agora).

6.4 Detalhes de configuração

Quando configuramos um pacote (isso acontece com `dpkg --install`, ou com `--configure`), nós primeiro atualizamos o conffiles e então chamamos:

```
postinst configure versão-mais-recentemente-configurada
```

Não é feita tentativa de reverter erros depois deles acontecerem durante a configuração.

Se não há versão mais recentemente configurada o `dpkg` irá passar um argumento vazio; versão velhas do dpkg podem passar `<unknown>` (incluindo os sinais) nesse caso. Mesmo mais antigas não passam um segundo argumento, em nenhuma circusntância.

6.5 Detalhes de remoção e/ou eliminação de configuração

1 *prerm* remove

2 Os arquivos do pacote são removidos (exceto arquivos de configuração).

3 *postrm* remove

4 Todos os scripts de mantenedor exceto o postrm são removidos.

Se nós não estamos eliminando o pacote, paramos aqui. Note que pacotes que não tem postrm e nem arquivos de configuração são automaticamente eliminados quando removidos, como não há diferença exceto para o estado do `dpkg`.

5 Os arquivos de configuração e quaisquer arquivos de backup (arquivos com `~`, arquivos `#*#`, arquivos `%`, `.dpkg-{old,new,tmp}`, etc) são removidos.

6 *postrm* purge

7 A lista de arquivos do pacote é removida.

Não é feita tentativa para reverter erros durante a remoção.

Capítulo 7

Descrição de pacotes - o campo `Description`

O campo `Description` do arquivo control é usado pelo `dselect` quando o usuários está secionando quais pacotes instalar e pelo `dpkg` quando ele mostra informações sobre o estado do pacote. Ele é incluído no site FTP nos arquivos `Packages` e pode também ser usado pelas páginas WWW do Debian.

A descrição é para descrever o programa para um usuário que nunca o viu antes para que ele saiba se irá querer instalá-lo ou não. Ele deve também dar informações sobre as dependências significantes e conflitos entre esse pacote e outros para que o usuário saiba por que essas dependências e conflitos foram declarados.

O formato do campo é:

```
Description: resumo de uma linha
descrição extendida com várias linhas
```

O resumo é frequentemente impresso nas listas de pacotes e ourtos e deve ser tanto informativo quanto possível. Cada pacote deve também ter uma descrição extendida.

7.1 Tipos de formatação de linhas na descrição extendida

- As que começam com um único espaço são parte de um parágrafo. Linhas sucessivas dessa forma serão mostradas em linhas contínuas quebrando a linha ao chegar a um certo limite quando mostradas. O espaço inicial é normalmente tirado.
- As que começam com dois ou mais espaços. Essas irão ser mostradas como apresentadas. Se o mostrador não pode mostrá-la toda em horizontal o programa que mostrar irá quebrar alinha “duramente” (ou seja, sem levar em conta os espaços entre palavras). Se possível elas serão permitidas a passar para fora da tela para a direita. Nenhum, um

ou dois espaços inicias podem ser deletados, mas o número de espaços deletados de cada linha terá o mesmo tamanho (para que você a "indentação" corretamente, por exemplo).

- Aquelas contendo um único espaço seguidos por um único ponto final. Essas serão mostradas como linhas em branco. Essa é a *única* maneira de ter uma linha em branco - veja abaixo.
- As que contém um espaço, um ponto final e alguns outros caracteres. Essas são para expansão futura. Não as use.

7.2 Notas sobre escrever descrições

Sempre comece as linhas de descrições extendidas com pelo menos um espaço em branco. Os campos no arquivo control e no Packages são separados pelos nomes dos campos começando na primeira coluna, como um cabeçalho de mensagem no RFC822. Esquecer do espaço em branco irá fazer o `dpkg-deb` [1] produzir um erro de sintaxe quando tentar construir o pacote. Se você forçá-lo a construir de qualquer modo o `dpkg` irá se recusar a instalar o resultado disso.

Não inclua nenhuma linha *em branco* completamente. Essas separam registros diferentes em um arquivo Packages e pacotes diferentes no arquivo `debian/control` e são proibidas em arquivos de controle de pacotes. Veja o parágrafo anterior para saber o que acontece se você fizer isso errado.

O resumo de uma linha deve ser pequeno - certamente menor que 80 caracteres. `dselect` mostra entre 25 e 49 caracteres sem sair da tela se você está usando um terminal de 80 colunas, dependendo de quais opções de mostragem estão efetivas.

Não inclua o nome do pacote na linha de resumo. O programa que mostra sabe como mostrar isso e você não precisa colocá-lo. Lembre que em muitas situações o usuário pode apenas ver a linha de resumo - faça-a tão informativa quanto lhe for possível.

A descrição extendida deve descrever o que o pacote faz e como ele se relaciona com o resto do sistema (em termos de, por exemplo, que subsistema do qual ele é parte).

A introdução que vem com um programa em seu anúncio e/ou arquivo `README` é raramente cabível para uso em uma descrição. Ele é normalmente destinado a pessoas que já estão na comunidade em que o pacote é usado. O campo de descrição precisa ter sentido para qualquer um, mesmo as pessoas que não têm idéia de nada que o pacote faz.

Coloque as informações importantes antes em ambos, resumo e descrição extensa. Algumas vezes apenas a primeira parte do resumo ou da descrição serão mostrados. Você pode assumir que haverá normalmente um jeito de ver a descrição completa.

Você pode incluir informação sobre dependências e daí pra frente na descrição extensa se quiser.

Não use tabulações. Seu efeito não é previsível.

[1] Versão 0.93.23 ou maior

Não tente mudar de linha no resumo (a parte que fica na mesma linha que o nome do campo `Description`) dentro da descrição extensa. Isso não irá funcionar corretamente quando a descrição completa for mostrada e não faz sentido onde apenas o resumo está disponível.

7.3 Descrições de exemplo no arquivo de controle para o Smail

```
Package: smail
Version: 3.1.29.1-13
Maintainer: Ian Jackson <iwj10@cus.cam.ac.uk>
Recommends: pine | mailx | elm | emacs | mail-user-agent
Suggests: metamail
Depends: cron, libc5
Conflicts: sendmail
Provides: mail-transport-agent
Description: Sistema de transporte de correio eletrônico.
Smail é o agente de transporte de correspondência (MTA) para o Debian.
.
Um MTA é o alojamento do sistema de correios - ele pega mensagens
dos programas amigáveis de e-mail e faz com que elas sejam entregued
localmente ou passadas para outros sistemas como requerido.
.
Para fazer uso dele você deve ter um ou mais programas leitores de
e-mail como elm, pine, mailx ou Emacs (que tem Rmail e VM como
leitores de e-mail) instalado. Se você deseja enviar mensagens para
outros locais que não seu próprio sistema você tem de ter suporte a
rede apropriado, nas formas IP ou UUCP.
```

*Nota do Tradutor:*Essa descrição está em português apenas para facilitar sua leitura. Para pacotes que serão distribuídos (para a distribuição do Debian, por exemplo), a descrição deve estar em inglês. Pelo menos até que se resolva o problema da internacionalização do sistema de empacotamento.

Capítulo 8

Declarando relacionamentos entre os pacotes

Pacotes podem declarar em seu arquivo de controle que eles têm certos relacionamentos com outros pacote - por exemplo, que eles não podem ser instalados na mesma hora em que um certo outro pacote e/ou que eles dependem da presença de outros, ou que eles devem sobrescrever arquivos em certos outros pacotes se presente.

Isso é feito usando os campos de arquivo control a seguir: `Depends`, `Recommends`, `Suggests`, `Conflicts`, `Provides` e `Replaces`.

Pacotes fonte podem declarar relacionamentos com pacotes binários, dizendo que eles requerem que certos pacotes binários estejam instalados ou ausentes quando da construção do pacote.

Isso é feito usando os campos de arquivo de controle a seguir: `Build-Depends`, `Build-Depends-Indep`, `Build-Conflicts`, and `Build-Conflicts-Indep`.

8.1 Sintaxe de campos de relacionamento

Esses campos todos tem uma sintaxe uniforme. Eles são uma lista de nomes de pacotes separados por vírgulas.

Em `Depends`, `Recommends`, `Suggests`, `Pre-Depends`, `Build-Depends` e `Build-Depends-Indep`(os campos que declaram dependências do pacote em que eles ocorrem em outros pacote) esses nomes de pacotes podem também ser listas de nomes de pacotes alternativos, separados pela barra vertical `|` (símbolo de encanamento).

Todos os campos exceto o `Provides` podem restingir sua aplicabilidade para versões particulares de cada pacote nomeado. Isso é feito em parenteses depois de cada nome de pacote individual; o parenteses deve conter uma relação da lista abaixo seguida pelo número de versão no formato descrito em 'Numeração de versão' on page 29.

As relações permitidas são <<, <=, =, >= e >> para anterior a, anterior ou igual a, exatamente igual a, posterior ou igual a e posterior a, respectivamente. As formas < e > foram usadas para significar anterior/posterior ou igual, ao invés de anterior/posterior, então eles não devem aparacer em novos pacotes (apesar do `dpkg` ainda suportá-los).

Espaços em branco podem aparecer em qualquer ponto na especificação da versão e devem aparecer quando for necessário tirar ambigüidade; não é significante fora isso. Para consistência e em caso de mudanças futuras no `dpkg` é recomendado que um espaço único seja usado depois de uma relação de versão e antes de um número de versão; é usual também colocar um espaço único depois de cada vírgula, em ambos os lados de cada barra vertical e antes de cada parentese de abertura.

Por exemplo:

```
Package: metamail
Version: 2.7-3
Depends: libc5 (>= 5.2.18-4), mime-support, csh | tcsh
```

Todos os campos que especificam relações de tempo de construção (`Build-Depends`, `Build-Depends-Indep`, `Build-Conflicts` e `Build-Conflicts-Indep`) podem ser restritos a certos conjuntos de arquiteturas. Isso é feito dentro de colchetes depois de cada nome de pacote e a especificação opcional da versão. Os colchetes englobam uma lista de nomes de arquiteturas Debian separadas por espaços. Um ponto de exclamação pode ser colocado antes de cada nome. Se a arquitetura host atual não está nessa lista e não há pontos de exclamação nessa lista, ou se ela está nessa lista com um ponto de exclamação na lista, o nome do pacote e a especificação de versão associada são ignoradas completamente para os propósitos de definição de relacionamentos.

Por exemplo:

```
Source: glibc
Build-Depends-Indep: texinfo
Build-Depends: kernel-headers-2.2.10 [!hurd-i386],
               hurd-dev [hurd-i386], gnumach-dev [hurd-i386]
```

8.2 Dependências Binary - `Depends`, `Recommends`, `Suggests`, `Pre-Depends`

Esses quatro campos são usados para declarar uma dependência de um pacote por outro. Eles aparecem no arquivo de controle do pacote que depende.

Todos exceto o `Pre-Depends` (discutido abaixo) tem efeito *apenas* quando um pacote está para ser configurado. Eles não previnem um pacote de estar no sistema de forma não configurada enquanto suas dependências estão insatisfeitas, e é possível substituir um pacote cujas dependências são satisfeitas e que está propriamente instalado com uma versão diferente cujas

dependências não estão e não podem ser satisfeitas; quando isso é feito o pacote dependente irá ser deixado não-configurado (já que tentativa de configurar irá dar erros) e não irá funcionar propriamente.

Por essa razão pacotes em uma instalação são normalmente desempacotados, todos, e depois configurados; isso dá a versão maiores de pacotes com dependências em versão posteriores de outros pacotes a oportunidade de ter suas dependências satisfeitas.

Assim o `Depends` permite aos mantenedores do pacote imporem uma ordem na qual os pacotes devem ser configurados.

`Depends` Isso declara uma dependência absoluta.

O `dpkg` não irá configurar pacotes cujas dependências não estão satisfeitas. Se for pedido para fazer uma instalação que cause a quebra de dependências de um pacote instalado ele irá reclamar [1], a menos que `--auto-deconfigure` esteja especificado, nesse caso esses pacotes irão ser desconfigurados antes que a instalação proceda.

`dselect` torna difícil para o usuário a seleção de pacotes para instalação, remoção ou atualização de um jeito que significaria que os campos `Depends` seriam tornados insatisfeitos. O usuário pode passar por cima se quizer, por exemplo se ele sabe que o `dselect` tem uma visão desatualizada de relacionamentos de pacotes.

O campo `Depends` deve ser usado se o pacote de que se depende é necessário para o pacote dependente para prover uma quantidade significante de funcionalidades.

`Recommends` Esse declara uma dependência forte mas não absoluta.

`Recommends` é ignorado pelo `dpkg`, para que usuários usando a linha de comando (que se presume saibam o que estão fazendo) não irão ser impedidos.

Ele é tratado pelo `dselect` exatamente como o `Depends`; isso torna difícil para o usuário selecionar coisas que deixa o campo `Recommends` insatisfeito, mas ele pode fazê-lo sendo persistente.

O campo `Recommends` deve listar pacotes que seriam achados juntos com esse em todas as instalações exceto nas não usuais.

`Suggests` Esse é usado para declarar que um pacote pode ser mais útil com um ou mais outros pacotes. Usar esse campo diz ao sistema de empacotamento e ao usuário que os pacotes listados são relacionados com esse e pode talvez melhorar sua utilidade mas que instalá-lo sem eles é perfeitamente razoavel.

O `dselect` vai oferecer os pacotes sugeridos para o administrados de sistemas quando ele selecionar o pacote sugerido, mas o padrão é não instalar o pacote sugerido.

`Pre-Depends` Esse campo é como o `Depends`, exceto que ele também força o `dpkg` a completar a instalação dos pacotes nomeados antes mesmo de começar a instalação do pacote que declara a pré-dependência.

[1] Versão atuais (1.2.4) do `dpkg` tem um erro nessa área que irá fazer com que alguns desses problemas sejam ignorados.

O `dselect` checa para dependências quando está fazendo uma instalação e tentará achar os pacotes cuja instalação é necessária em primeiro momento e então na ordem certa.

No entanto, esse processo é lento (porque requer chamadas repetidas ao `dpkg`) e problemática (porque requer uma adivinhação de onde achar os arquivos apropriados).

Por essas razões e por causa desse campo impor restrições na ordem em que os pacotes podem ser desempacotados (o que pode ser difícil em instalações de mídia multi-parte, por exemplo), `Pre-Depends` deve ser usado muito pouco, preferívelmente apenas por pacotes em que atualização prematura ou instalação iria romper a habilidade do sistema de continuar com qualquer atualização que pode estar em progresso.

Quando o pacote declarando que ele está sendo configurado, uma `Pre-Dependencia` será considerada satisfeita apenas se o pacote de que se depende tiver sido configurado como se um `Depends` ordinário tivesse sido usado.

No entanto quando um pacote declarando uma predependência está sendo desempacotado a precedência pode ser satisfeita mesmo se o(s) pacote(s) de(os) que(ais) se depende estiver apenas desempacotado ou meio-configurado, provendo que ele(s) tenha(m) sido configurado(s) corretamente em algum ponto do passado (e não removido(s) ou parcialmente removido(s) desde então). Nesse caso ambos o previamente-configurado e a atualmente desempacotada ou meio-configurada devem satisfazer qualquer cláusula de versão do campo `Pre-Depends`.

Quando selecionando qual level de dependência usar você deve considerar quão importante o pacote de que se depende é para a funcionalidade do declarante da dependência. Alguns pacotes são compostos de componentes de vários graus de importancia. Tais pacotes devem listar usando o `Depends` o(s) pacote(s) que é(são) necessário(s) pelos componentes mais importantes. Os outros requerimentos dos outros componentes devem ser mencionados como sugestões (Suggests) ou recomendações (Recommends), como é apropriado para a importância relativa dos componentes.

8.2.1 Dependências de bibliotecas compartilhadas

Os campos de dependências listados acima são usados por pacotes que precisam de bibliotecas compartilhadas para declarar dependências para os pacotes apropriados.

Essas dependências são normalmente determinadas automaticamente usando o `dpkg-shlibdeps` e inserido no arquivo de controle do pacote usando o mecanismo de substituição de variáveis do arquivo de controle; veja ‘`debian/substvars` e substituições de variáveis’ on page 17 e ‘Ferramentas para processar pacotes fonte’ on page 7.

8.2.2 Desconfiguração por causa de remoção durante instalações massivas

Se o `dpkg` quisesse remover um pacote por causa de um conflito, como descrito acima, mas isso fosse violar a dependência de algum outro pacote no sistema, o `dpkg` normalmente não removerá o pacote conflitante e sairá com um erro.

No entanto se o `--auto-deconfigure` (`-B`) for usado o `dpkg` irá automaticamente 'desconfigurar' o pacote com a dependência problemática para que o pacote conflitante possa ser removido e o pacote que estamos tentando instalar seja instalado. Se o `dpkg` está sendo usado para instalar pacotes (ao invés de apenas desempacotá-los) ele irá tentar reconfigurar o pacote quando ele desempacotou todos os seus argumentos, na esperança de um desses pacotes sendo instalados irão satisfazer a dependência problemática.

O `dselect` dispõe esse argumento para o `dpkg` quando o chmama, para que instalações em massa possam prosesseguir sem erros.

8.3 Pacotes binários alternativos - `Conflicts` e `Replaces`

Quando um pacote binário declara um conflito com outro o `dpkg` irá se recusar a permitir a instalação deles no sistema ao mesmo tempo.

Se um pacote está para ser instalado, o outro precisa se removido antes - se o pacote sendo instalado está marcado como substitituindo ('`Replaces` - sobrescrevendo arquivos e substituindo pacotes' on the following page) o que está no sistema ou o do sistema está marcada como desselecionado ou ambos os pacotes estão marcados `Essential` (essencial), então o `dpkg` irá remover o pacote que está causando o conflito automaticamente, senão ele irá parar a instalação do novo pacote com um erro. Esse mecanismo especificamente não funciona quando o pacote instalado é `Essential`, mas o novo pacote não.

O `dselect` torna difícil a seleção de pacote conflitantes, mas o usuário pode passar por cima disso se ele quiser. Se ele não passar o `dselect` irá selecionar um dos pacotes para remoção e o usuário deve ter certeza de que é o certo. No futuro o `dselect` procurará pela presença de um campo `Replaces` para ajudar a decidir qual pacote deve ser instalado e qual deve ser removido.

Um pacote não irá causar um conflito meramente por causa de seus arquivos de configuração ainda estarem instalados; ele deve estar pelo menos meio-instalado.

Uma excessão especial é para os pacotes que declaram um conflito com seu próprio nome de pacote ou com um pacote virtual que ele provê (veja abaixo): isso não previne sua instalação e permite ao pacote conflitar com outros provendo um substituto para ele. Você usa essa função quando quer o pacote em questão seja o único provendo alguma coisa.

Uma entrada de `Conflicts` deve quase nunca ter uma cláusula de versão 'anterior a'. Isso iria impedir o `dpkg` de atualizar ou instalar o pacote que declara tal conflito até a atualização ou remoção do pacote com que se conflita ter sido completada. Esse aspecto da ordem da instalação não é manejada pelo `dselect`, por isso o uso do `Conflicts` desse jeito é passível de causar problemas para atualizações e instalações massivas.

8.4 Pacotes virtuais - `Provides`

Bem como os nomes do pacotes ('concretos'), os campos de relacionamento de pacotes `Depends`, `Build-Depends`, `Build-Depends-Indep`, `Recommends`, `Suggests`,

`Conflicts`, `Build-Conflicts` e `Build-Conflicts-Indep` poden mencionar pacotes virtuais.

Um pacote virtual é um que aparece no campo `Provides` do arquivo control de outro pacote. O efeito é como se o pacote que provê um pacote virtual particular tivesse sido listado em todos os lugares que o nome do pacote virtual aparece.

Se há ambos um pacote real e um virtual de mesmo nome então a dependência pode ser satisfeita (ou o conflito causado) ou pelo pacote real ou por qualquer um dos pacotes virtuais que ele provê. Isso é tanto que, por exemplo, supondo que tenhamos

```
Package: vm
Depends: emacs
```

e em alguma outra pessoa lança um pacote xemacs eles podem dizer

```
Package: xemacs
Provides: emacs
```

e tudo irá funcionar (até um nome de pacote virtual ser decidido e o `emacs` e o `vm` serem mudados para usá-lo).

Se uma dependência ou um conflito tem um número de versão atachado então apenas pacotes reais serão considerados para ver se o relacionamento é satisfeito (ou a proibição violada por um conflito) - é assumido que um pacote real que provê pacote virtual não é a versão 'certa'. Então, um campo `Provides` não pode conter números de versão, e o número de versão do pacote concreto que provê um pacote virtual particular não será olhado quando considerando uma dependência ou um conflito com o nome do pacote virtual.

É provavel que a habilidade será adicionado em uma versão futura do `dpkg` para especificar um número de versão para cada pacote virtual que provê. Essa função não está presente ainda, no entanto, é esperado que seja usada não muito frequentemente.

Se você quer especificar que um conjunto de pacotes reais devem ser o padrão para satisfazer uma dependência particular em um pacote virtual, você deve listar o pacote real como uma alternativa antes do virtual.

8.5 `Replaces` - sobrescrevendo arquivos e substituindo pacotes

O campo `Replaces` do arquivo de controle tem dois propósitos, que vêm ao caso em diferentes situações.

Pacotes virtuais ('Pacotes virtuais - `Provides`' on the page before) não são considerados quando olhando o campo `Replaces` - os pacotes declarados como sendo substitutos devem ser mencionados pelos seus nomes reais.

8.5.1 Sobrescrevendo arquivos em outros pacotes

Primeiramente, como mencionado antes, é normalmente um erro para um pacote conter arquivos que estão no sistema em outro pacote, apesar de atualmente a opção `--force-overwrite` estar ligada por padrão, diminuindo o erro para um aviso.

Se o pacote que está sobrescrevendo declara que ele substitui o que contém os arquivos sendo sobrescritos então o `dpkg` irá proceder e substituir o arquivo do pacote antigo com o do novo. O arquivo não será listado como 'possuído' pelo pacote velho.

Se um pacote for completamente substituído dessa forma, de forma que o `dpkg` não saiba de nenhum arquivo que ele ainda contém, ele é considerado como removido. Será marcado como não-quisto no sistema (selecionado para remoção) e não instalado. Quaisquer detalhes de conffiles notados no pacote serão ignorados, já que eles serão tomados pelo(s) pacote(s) substituto(s). O script `postrm` será rodado para permitir ao pacote a execução de qualquer limpeza requerida. Veja 'Resumo das maneiras pelas quais os scripts de mantenedor são chamados' on page 34.

No futuro o `dpkg` irá discartar arquivos que sobrescrevem aqueles de outro pacote que declara que ele substitui o que está sendo instalado (para que você instalar uma versão mais velha de um pacote sem problemas).

Esse uso de `Replaces` só toma efeito quando ambos os pacotes esão pelo menos parcialmente no sistema de uma vez, então isso só pode acontecer se eles não conflitarem ou se o conflito for sobrepassado.

8.5.2 Substituindo pacotes inteiros, forçando a remoção

Segundo, `Replaces` permite ao `dpkg` e ao `dselect` resolver quais pacotes devem ser removidos quando há um conflito - veja 'Pacotes binários alternativos - `Conflicts` e `Replaces`' on page 47. Esse uso toma efeito apenas quando os dois pacote *conflitam*, para que cada um dos dois processos não interfiram o outro.

8.6 Padrões para satisfazer dependências - ordenando

Ordenar é significante em campos de dependência.

Normalmente o dselect irá sugerir ao usuário que ele selecione o pacote com a classe mais 'fundamental' (ex: irá preferir pacotes Base a Optional), ou o que eles 'mais queriam' selecionar em algum senso.

Na ausência de outra informação o `dselect` irá oferecer uma seleção padrão do primeiro pacote nomeado na lista de alternativas.

No entanto, não há um jeito de especificar a 'ordem' de vários pacotes que provêm a mesma coisa, quando essa coisa é listada como dependência.

Assim uma denpendência em um pacote virtual deve conter um nome de pacote concreto como a primeira alternativa, para que ele seja o padrão.

Por exemplo, considere o conjunto de pacotes:

```
Package: glibcdoc
Recommends: info-browser

Package: info
Provides: info-browser

Package: emacs
Provides: info-browser
```

Se `emacs` e `info` têm a mesma prioridade então a escolha do `dselect` é será totalmente aleatória. Melhor seria usar

```
Package: glibcdoc
Recommends: info | info-browser
```

para que o `dselect` pegue como padrão selecionar o navegador info menor, leve e que roda sozinho.

8.7 Relações entre pacotes fonte e binários - `Build-Depends`, `Build-Depends-Indep`, `Build-Conflicts`, `Build-Conflicts-Indep`

Um pacote fonte pode declarar uma dependência ou um conflito com um pacote binário. Isso é feito com os campos de controle `Build-Depends`, `Build-Depends-Indep`, `Build-Conflicts`, e `Build-Conflicts-Indep`. Sua semântica é que as dependências e conflitos que eles definem devem ser satisfeitas (como definido antes para pacotes binários), quando um dos alvos no `debian/rules` que o campo particular aplica é chamado.

`Build-Depends`, `Build-Conflicts` Os campos `Build-Depends` e `Build-Conflicts` aplicam aos alvos `build`, `binary`, `binary-arch` e `binary-indep`.

`Build-Depends-Indep`, `Build-Conflicts-Indep` Os campos `Build-Depends-Indep` e `Build-Conflicts-Indep` aplicam aos alvos `binary` e `binary-indep`.

Capítulo 9

Manuseio de arquivo de configuração

`dpkg` pode fazer um certo tanto de manuseio de arquivos de configuração de pacote.

Se esse mecanismo é apropriado depende de um número de fatores, mas basicamente há duas visões para qualquer arquivo de configuração particular.

O método mais fácil é lançar com uma configuração bem feita no pacote e usar o mecanismo de arquivos de configuração do `dpkg` para manusear atualizações. Se o usuário não quer editar o arquivo, mas você precisa que eles sejam capazes disso sem perder suas mudanças e um novo pacote com uma versão mudada do arquivo é lançada não muito frequentemente, essa é uma boa visão.

O meio difícil é construir o arquivo de configuração do início no script `postinst`, e levar a responsabilidade de consertar erros feitos em versões anteriores do pacote automáticamente. Isso será o apropriado se o arquivo deve ser necessário em cada sistema.

9.1 Manuseio automático dos arquivos de configuração pelo `dpkg`

Um pacote pode conter um arquivo de controle de área chamado `conffiles`. Esse arquivo deve ser uma lista de nomes de arquivos de configuração que precisam manuseio automático, separados por novas linhas. Os nomes de arquivos devem ser caminhos absolutos e os arquivos referidos devem atualmente existir no pacote.

Quando um pacote é atualizado o `dpkg` irá processar os arquivos de configuração durante o estágio de configuração, pouco antes de rodar o `postinst` do pacote,

Para cada arquivo ele checa para ver se a versão do arquivo incluído no pacote é a mesma daquela que foi incluída na última versão do pacote (a que está sendo atualizada); também compara a versão atualmente no sistema com a enviada com a última versão.

Se nem o usuário nem o mantenedor do pacote mudaram o arquivo, ele é deixado para lá. Se um ou outro mudou sua versão, então a versão modificada é preferida - ex: se o usuário edita seu arquivo, mas o mantenedor do pacote não manda uma versão diferente, as mudanças do usuário ficarão, silenciosamente, mas se o mantenedor envia uma nova versão e o usuário

não editou a sua a nova versão será instalada (com uma mensagem informativa). Se ambos mudaram suas versões o usuário é questionado sobre o problema e deve resolver as diferenças ele mesmo.

As comparações são feitas calculando as mensagens de MD5 dos arquivos e guardando o MD5 do arquivo como ele foi incluído na mais recente versão do pacote.

Quando um pacote é instalado pela primeira vez o `dpkg` irá instalar o arquivo que vem com ele, a menos que isso signifique sobrescrever um arquivo já no sistema de arquivos.

No entanto, note que o `dpkg` *não* irá substituir um conffile que foi removido pelo usuário (ou por um script). Isso é necessário porque com alguns programas um arquivo faltando produz um efeito difícil ou impossível de se conseguir de outro jeito, então o arquivo faltante precisa ser mantido da mesma forma que o usuário fez.

Note que um pacote *não* deve modificar um conffile manuseado pelo `dpkg` em seus scripts de mantenedor. Fazer isso irá fazer com que o `dpkg` dê ao usuário opções confusas e possivelmente perigosas para atualização de um conffile quando o pacote for atualizado.

9.2 Scripts de mantenedor lidando totalmente com arquivos de configuração

Para arquivos que contém informação específica-de-máquina como o hostname e detalhes de rede e por aí, é melhor criar o arquivo no script `postinst` do pacote.

Isso irá tipicamente involver examinar do estado do resto do sistema para determinar valores e outras informações e pode involver perguntas ao usuário para alguma informação que não pode ser obtida de nenhum outra forma.

Quando usar esse método há alguns assuntos importantes que devem ser considerados:

Se você descobrir um erro no programa que gera o arquivo de configuraçào ou se o formato do arquivo muda de uma versão para a outra, você terá de arranjar para que o postinst faça algum sensível - normalmente isso significa editar o arquivo de configuração instalado para remover o problema ou mudar a sintaxe. Você terá de fazê-lo muito cuidadosamente já que o usuário pode ter mudado o arquivo, talvez para consertar o problema que seu script está tentando consertar - você terá que detectar essas situações e lidar com elas corretamente.

Se você for por esse caminho é provavelmente uma boa idéia fazer o programa que gera os arquivos de configuração um programa separado no `/usr/sbin`, por convenção chamado *`pacote`*`config` e então rodá-lo se apropriado pelo script postinst. O programa *`pacote`*`>config` não deve inquestionávelmente sobrescrever uma configuração existente - se seu modo de operação é a de configurar um pacote pela primeira vez (ao invés de uma reconfiguração arbitrária depois) você deve fazê-lo checar se a configuração já existe e requerir uma opção `--force` para sobrescrevê-lo.

Capítulo 10

Versões alternativas de uma interface - `update-alternatives`

Quando vários pacotes todos provêm versões diferentes do mesmo programa ou arquivo é útil que o sistema selecione um padrão, mas permitir ao administrador do sistema mudar e ter sua decisão respeitada.

Por exemplo, há várias versões do editor `vi` e não há razão para prevenir que todos eles estejam instalados de uma vez, cada um sob seu próprio nome (`nvi`, `vim` ou o que for). Mesmo assim é desejável que se tenha o nome `vi` referindo a algo, pelo menos por padrão.

Se todos os pacotes involvidos cooperarem, isso pode ser feito com o `update-alternatives`.

Cada pacote provê sua própria versão sob seu próprio nome e chama o `update-alternatives` em seu postinst para registrar sua versão (e novamente em seu prerm para desregistrá-lo).

Veja a página de manual `update-alternatives(8)` para detalhes.

Se o `update-alternatives` não parece apropriado você pode desejar considerar usar "diversions" (diversões) [1] ao invés disso.

[1] Não as diversões de ir a Disney, por exemplo, e sim algo como divergir. (N.T.)

Capítulo 11

Diversões - sobrepassando uma versão de um arquivo de um pacote

É possível fazer com que o `dpkg` não sobrescreva um arquivo quando reinstala o pacote ao qual ele pertence e que ele coloque o arquivo do pacote em algum outro lugar, ao invés disso.

Isso pode ser usado localmente para sobrepassar uma versão de um arquivo de um pacote ou por um pacote sobrepassar uma versão de outro (ou prover um "mudador" para ele).

Antes de decidir usar uma diversão, leia o 'Versões alternativas de uma interface - `update-alternatives`' on page 53 para ver se você realmente quer uma diversão ao invés de várias versões alternativas de um programa.

Há uma lista de diversões, que é lida pelo `dpkg`, e atualizada por um programa especial `dpkg-divert`. Por favor veja `dpkg-divert(8)` para detalhes completas dessa operação.

Quando um pacote quer divergir um arquivo de outro, ele deve chamar `dpkg-divert` em seu preinst para adicionar a diversão e renomear o arquivo existente. Por exemplo, supondo que o pacote `smailwrapper` quer instalar um "mudador" para o `/usr/bin/smail`:

```
if [ install = "$1" -o upgrade = "$1" ]; then
   dpkg-divert --package smailwrapper --add --rename \
      --divert /usr/sbin/smail.real /usr/sbin/smail
fi
```

Testar `$1` é necessário para que o script não tente adicionar a diversão de novo quando o `smailwrapper` for atualizado. O `--package smailwrapper` assegura que a cópia de `/usr/bin/smail` do `smailwrapper` pode passar a diversão e ser instalada como a versão verdadeira.

O postrm tem de fazer o reverso:

```
if [ remove = "$1" ]; then
```

```
        dpkg-divert --package smailwrapper --remove --rename \
           --divert /usr/sbin/smail.real /usr/sbin/smail
    fi
```

Não tente divergir um arquivo que é vitalmente importante para a operação do sistema - quando se usa o `dpkg-divert` há um tempo, depois de ele ter sido divergido mas antes de o `dpkg` ter instalado a nova versão, em que o arquivo não existe.

Capítulo 12

Bibliotecas compartilhadas

Pacotes contendo bibliotecas compartilhadas devem ser construídas com um pouco de cuidado para ter certeza de que a biblioteca compartilhada esteja sempre disponível. Isso é especialmente importante para pacotes cujas bibliotecas compartilhadas são vitalmente importantes, como a libc.

Primeiramente, seu pacote deve instalar as bibliotecas compartilhadas sob seus nomes normais. Por exemplo, o pacote `libgdbm1` deve instalar a `libgdm.so.1.7.3` como `/usr/lib/libgdbm.so.1.7.3`. Os arquivos não devem ser renomeados ou relinkados por qualquer scripts prerm ou postrm; o `dpkg` tomará cuidade de renomear as coisas com segurança sem afetar os programas que estão rodando e tentativas de interferir com isso estão sujeitas a erros.

Segundo, seu pacote deve incluir um link simbólico que o `ldconfig` iria criar para as bibliotecas compartilhadas. Por exemplo, o pacote `libgdbm1` deve incluir um link simbólico de `/usr/lib/libgdbm.so.1` para `libgdbm.so.1.7.3`. Isso é necessário para que o `ld.so` possa achar a biblioteca no meio tempo em que o `dpkg` a instala e o `ldconfig` é executado no script `postinst`. Mais, versões antigas do sistema de manejamento de pacotes requeria que a biblioteca fosse colocada antes de o link apontando para ele o fosse no arquivo `.deb`. Isso é para que quando o `dpkg` viesse a instalar o link (sobrescrevendo o link anterior apontando para uma versão mais velha da biblioteca) a nova biblioteca compartilhada já estivesse no lugar. Infelizmente, isso não era sempre possível, já que ele depende altamente do comportamento do sistema de arquivos. Alguns sistemas de arquivos (como o reiserfs) irão reordenar os arquivos de maneira que não importa a ordem em que você os cria. Começando pelo lançamento `1.7.0` o `dpkg` irá reordenar os arquivos ele mesmo quando construir um pacote.

Terceiro, o pacote em desenvolvimento deve conter um link simbólico para a biblioteca compartilhada sem um número de versão. Por exemplo, o pacote `libgdm1-dev` deve incluir um link simbólico do `/usr/lib/libgdm.so` para `libgdm.so.1.7.3`. Esse link é necessário quando o `ld` está compilando pacotes pois ele irá procurar apenas por `libgdm.so` e pelo `libgdm.a` quando compilando dinâmicamente ou estáticamente respectivamente.

Qualquer pacote instalando bibliotecas compartilhadas em um diretório que está listado no `/etc/ld.so.conf` ou em um dos diretórios padrões de bibliotecas do `ld.so` (atualmente

esses são `/usr/lib` e `/lib`) deve chamar `ldconfig` em seu script`postinst` se e apenas se o primeiro agumento for 'configure'. No entanto, é importante não chamar o `ldconfig` nos scripts postrm ou preinst no caso em que o pacote está sendo atualizado (veja 'Detalhes da fase de desempacotamento de instalação ou atualização' on page 34), já que o `ldconfig` verá os nomes temporários que o `dpkg` usa para os arquivos enquanto está os instalando e irá fazer com que a biblioteca compartilhada aponte para eles, antes mesmo de o `dpkg` continuar a instalação e remover os links!

12.1 O Formato de Arquivo do `shlibs`

Esse arquivo é usado pelo `dpkg-shlibdeps` e é requerido quando seu pacote provê bibliotecas compartilhadas.

Cada linha é da forma:

```
nome-da-biblioteca versão-ou-nome.so dependências ...
```

nome-da-biblioteca é o nome da biblioteca compartilhada, por exemplo `libc5`.

versão-ou-nome.so é o nome com .so da biblioteca - ex: a coisa que deve exatamente encaixar para a biblioteca ser reconhecida pelo `ld.so`. Normalmente é o número de versão mais importante da biblioteca.

dependências tem a mesma sintaxe que um campo de dependência em um arquivo de controle de um pacote binário. Ele deve dar detalhes de quais pacotes são requeridos para satisfazer um binário construído com a versão da biblioteca contida no pacote. Veja 'Sintaxe de campos de relacionamento' on page 43.

Por exemplo, se o pacote `foo` contém `libfoo.so.1.2.3`, onde o nome.so da biblioteca é `libfoo.so.1`, e a primeira versão do pacote que continha um número menos importante de pelo menos `2.3` foi *1.2.3-1*, então o *shlibs* do pacote poderia dizer:

```
libfoo 1 foo (>= 1.2.3-1)
```

A dependência específica de versão é para evitar avisos do `ld.so` sobre usar velhas bibliotecas compartilhadas com novos binários.

12.2 Maiores informações técnicas sobre o `shlibs`

12.2.1 *O que* são os arquivos `shlibs`?

O arquivo `debian/shlibs` provê um meio de checar por dependências de bibliotecas compartilhadas em binários empacotados. Eles têm a intenção de ser usados pelos mantenedores do pacote para tornar a vida mais fácil.

Outros arquivos `shlibs` que existem em um sistema Debian são

- `/etc/dpkg/shlibs.default`
- `/etc/dpkg/shlibs.override`
- `/var/lib/dpkg/info/*.shlibs`
- `debian/shlibs.local`

Esses arquivos são usados pelo `dpkg-shlibdeps` quando criando um pacote binário.

12.2.2 *Como* o `dpkg-shlibdeps` funciona?

`dpkg-shlibdeps` determina as bibliotecas compartilhadas diretamente [1] usado por binários compilados (e bibliotecas, em uma versão do `dpkg-shlibdeps` vindo breve) passado através de sua linha de comando.

para cada biblioteca compartilhada, o `dpkg-shlibdeps` precisa saber
- o pacote contendo a biblioteca e
- o número de versão da biblioteca,

ele analiza os arquivos a seguir nessa ordem.

```
1 debian/shlibs.local
2 /etc/dpkg/shlibs.override
3 /var/lib/dpkg/info/*.shlibs
4 /etc/dpkg/shlibs.default
```

12.2.3 *Quem* mantém os vários arquivos `shlibs`?

- `/etc/dpkg/shlibs.default` - o mantenedor do dpkg
- `/var/lib/dpkg/info/`*`pacote`*`.shlibs` - o mantenedor de cada pacote
- `/etc/dpkg/shlibs.override` - o administrador de sistema local

[1] Atualmente, ele chama o `ldd`, mas em uma versão vindoura ele deve chamar o `objdump` para fazer isso. Isso, mudanças precisarão de um tanto de mudanças no jeito em que os pacotes são construídos. Suponha um binário `foo` diretamente usando uma biblioteca `libbar` se ele está linkado com essa biblioteca. Outras bibliotecas que são necessárias pelo `libbar` são linkados indiretamente a `foo` e o linkador dinâmico irá carregá-las automaticamente quando carregar a `libbar`. Usar o `ldd` lista todos as bibliotecas usadas diretamente e indiretamente; mas o `objdump` só lista as diretamente ligadas. Um pacote só precisa depender nas bibliotecas as quais ele diretamente depende, desde que as dependências para essas bibliotecas devem automaticamente ser colocadas em outras bibliotecas. Essa mudança não significa uma mudança no jeito em que os pacote são construídos: atualmente o dpkg-shlibdeps é rodado apenas em binários. Mas já que nós iremos agora depender das bibliotecas depender das bibliotecas que elas precisam dos pacotes contendo essas bibliotecas irão precisar rodar o dpkg-shlibdeps nas bibliotecas. Um bom exemplo onde isso iria nos ajudar é a atual bagunça com múltiplas versões da biblioteca mesa. Com o sistema baseado no ldd cada pacote que usa o mesa precisa adicionar uma dependência com svgalib | svgalib-dummy para conseguir lidar com a variante mesa glide. Com um sistema baseado em objdump isso não é mais necessário e teria evitado um grande trabalho de muitas pessoas. Outro exemplo: nós poderíamos atualizar libimlib com uma nova versão que suporta um novo formato de gráfico chamdo dgf. Se nós usamos o método ldd seria necessária a compilação de todo os pacotes que usa libimlib para também depender de libdgf ou ele não rodaria por causa de símbolos faltando. No entanto com o novo sistema pacotes que usam libimlib podem depender da libimlib e ter dependências em libgdh e não seria necessária a atualização.

- `debian/shlibs.local` - o mantenedor do pacote

O arquivo `shlibs.default` é manejado pelo `dpkg`. As entradas em `shlibs.default` que são providas pelo `dpkg` estão aqui apenas para consertar coisas até que as bibliotecas compartilhadas todas tenham arquivos `shlibs`.

12.2.4 *Como* usar o `dpkg-shlibdeps` e o arquivo `shlibs`?

Se seu pacote não provê uma biblioteca compartilhada

Ponha uma chamada para `dpkg-shlibsdeps` no seu `debian/rules`. Se seu pacote contém apenas binários (ex: não tem scripts) use:

```
dpkg-shlibdeps debian/tmp/usr/bin/* debian/tmp/usr/sbin/*
```

Se o `dpkg-shlibdeps` não reclamar, você o fez. Se ele reclamar você deve precisar criar seu próprio arquivo `debian/shlibs.local`.

Se seu pacote provê uma biblioteca compartilhada

Crie um arquivo `debian/shlibs` e deixe o `debian/rules` instalá-lo na área de controle:

```
install -m644 debian/shlibs debian/tmp/DEBIAN
```

Se seu pacote contém binários adicionais veja acima.

12.2.5 *Como* escrever um `debian/shlibs.local`

Esse arquivo tem a intenção de apenas ser um conserto *temporário* se seus binários dependem de uma biblioteca que não provê seu próprio arquivo `/var/lib/dpkg/info/*.shlibs` ainda.

Vamos assumir que você está empacotando um binário `foo`. Sua saída quando construir o pacote pode ser assim.

```
$ ldd foo
libbar.so.1 => /usr/X11R6/lib/libbar.so.1.0
libc.so.5 => /lib/libc.so.5.2.18
libX11.so.6 => /usr/X11R6/lib/libX11.so.6.0
```

E quando você rodou o `dpkg-shlibdeps`

```
$ dpkg-shlibdeps -o foo
dpkg-shlibdeps: warning: unable to find dependency information
for shared library libbar
(soname 1, path /usr/X11R6/lib/libbar.so.1.0, dependency field Depends)
shlibs:Depends=elf-x11r6lib, libc5 (>= 5.2.18)
```

O binário `foo` depende da biblioteca compartilhada `libbar`, mas nenhum pacote parece prover um arquivo `*.shlibs` em `/var/lib/dpkg/info`. Vamos determinar o pacote responsável por isso:

```
$ dpkg -S /usr/X11R6/lib/libbar.so.1.0
bar1: /usr/X11R6/lib/libbar.so.1.0
$ dpkg -s bar1 | grep Version
Version: 1.0-1
```

Isso nos mostra que o pacote `bar1`, versão 1.0-1 é o que estamos usando. Agora nós podemos criar nosso próprio `debian/shlibs.local` para arrumar o problema acima temporáriamente. Inclua a linha a seguir no seu arquivo `debian/shlibs.local`.

```
libbar 1 bar1 (>= 1.0-1)
```

Agora a construção do seu pacote deve funcionar. Assim que o mantenedor de `libbar1` prover um arquivo `shlibs`, você pode remover o seu `debian/shlibs.local`.

Capítulo 13

interfaces do `dselect` para seus métodos de instalação

`dselect` chama scripts de seus métodos de instalação quando ele precisa de acessar dados da distribuição. O programa principal `dselect` só chama esses scripts e provê o pacote e as interfaces de seleção de método. Os métodos de instalação são responsáveis por chamar o `dpkg` como apropriado.

Cada método de instalação tem três scripts:

- Configura parâmetros de instalação.
- Atualiza a lista de pacotes disponíveis.
- Instala.

O `dselect` procura por métodos em `/usr/lib/dpkg/methods` e `/usr/local/lib/dpkg/methods`.

13.1 Funções dos scripts de métodos

O script de configuração é rodado após o usuário ter escolhido o método de instalação. Ele deve pedir ao usuário os parâmetros como o site para NFS-mount ou FTP, o diretório para usar ou o diretório ou sistema de arquivos onde os arquivos `.deb` podem ser achados, ou a fita ou disquete de onde instalar. Ele deve guardar as respostas em `/var/lib/dpkg/methods` - veja abaixo. Se não houver listas de pacotes disponíveis ele deve talvez se oferecer para procurar pelos pacotes disponíveis.

O script de atualização deve obter uma lista de pacotes disponíveis se pssível e rodar `dpkg --update-avail`, `dpkg --merge-avail` e/ou `dpkg --forget-old-unavail` para carregá-lo no banco de dados do `dpkg` e do `dselect` de pacotes disponíveis. Se nenhuma lista de pacotes disponíveis e o usuário recebeu a oferta de procurar pelos pacotes disponíveis e aceitou, essa procura deve ser feita aqui, usando o `dpkg --record-avail`.

O script de instalação supri todos os arquivos .deb disponíveis para o `dpkg --iGOEB` (isso é equivalente a `dpkg --install --refuse-downgrade --selected-only`

`--skip-same-version --auto-deconfigure`. A opção `-R` (`--recursive`) para ir atravessando diretórios pode também ser útil aqui).

If any of these scripts needs to display a message for the user, it should wait for the user to hit 'return' before exiting so that dselect doesn't immediately rewrite the screen.

Se um script método tem sucesso (retorna um status de saída zero) o `dselect` retornará imediatamente o menu principal, com a 'próxima' opção iluminada pronta para o usuário selecionar. Se falhar o `dselect` mostrará uma mensagem e esperará até o usuário apertar enter.

13.2 Localização e argumentos dos scripts de método

Um conjunto de scripts (aqui conhecidos como grupo) podem prover vários métodos no 'menu principal' com comportamentos diferentes. Por exemplo, pode haver um grupo genérico pegue-pacotes-pelo-FTP que pode prover métodos no menu principal para instalação direta de um dos sites mirror do Debian assim como para instalação vinda de um site específico do usuário.

Cada grupo de métodos implementados pelo mesmo conjunto de scripts deve ter um subdiretório `/usr/lib/dpkg/methods/`*`goupo`* ou `/usr/local/lib/dpkg/methods/`*`grupo`*, contendo:
`names` uma lista de métodos visíveis ao usuário providos por esses scripts.
`setup`
`update`
`install` programas executáveis, os scripts, eles mesmos.
`desc.`*`opção`* arquivo de descrição.

`names` será formatado como uma lista de linhas, cada uma contendo:

```
sequência método sumário
```

sequência é um número de dois dígitos que serão usados muito parecidamente com os prefixod do `rc.d` para controlar a ordem do menu principal. Se estiver em dúvida, use 50.

método é um nome que é mostrado pelo `dselect` como o nome de um método e que será passado para `setup`, `update` e `unpack` como seu primeiro argumento.

sumário é uma string de descrição resumida para o menu do `dselect`.

Cada um desses três scripts tem os mesmos três argumentos: *vardir*, *grupo* e *método*. *vardir* é o diretório base para guardar o estado do `dpkg` e do `dselect`, normalmente `/var/lib/dpkg`; isso é passado para que assim a opção `--admindir` passada para o `dselect` seja honrada).

Cada opção pode ter uma descrição extendida em `desc.`*`opção`*. Esse deve ser formatado como a parte de descrição extensa do campo `Description` *mudado um caractere para a esquerda*.

`vardir``/methods` existirá e um grupo de métodos pode usar um diretório *`vardir`*`/methods/`*`grupo`* para guardar seu estado.

O nome do grupo e o nome do método devem seguir as regras dos identificadores de C.

Capítulo 14

Procedimento de conversão de velhos pacotes fontes

Esse é um sumário resumido do procedimento para converter um pacote fonte pre-2.0.0.0 para o novo formato.

É extremamente aconselhável que você baixe o pacote `hello` e leia a seção no manual dos programadores do `dpkg` descrevendo as ferramentas de pacotes fontes. Mais detalhes sobre a funcionalidade exata dessas ferramentas estão disponíveis em `dpkg-source(1)`.

- Pegue o código fonte original de qualquer lugar em que ele possa ser encontrado e faça qualquer rearranjamento necessário para fazê-lo parecer a árvore original do fonte Debian. Coloque em *`pacote-versão-externa`*`.orig/` ou *`pacote_versão-externa`*`.orig.tar.gz`.
- Renomeie todos os arquivos `debian.*` para `debian/*`. Podem haver algumas excessões para isso, mas é um bom começo.
- Edite o `debian/changelog` - crie ou renomeie-o se necessário. Adicione uma nova revisão ao topo com os detalhes apropriados e uma entrada de variáveis locais ao final para configurar o Emacs para o modo certo:

  ```
  Local variables:
  mode: debian-changelog
  End:
  ```

- Edite/crie o `debian/control`:
  - Remova o campo `Version`. Se ele for gerado não raramente (não igual à versão fonte) você deve usar a opção -v no dpkg-gencontrol (veja abaixo). `Section`, `Priority`, `Maintainer` vão acima da primeira linha em branco, a maior parte do resto, abaixo.
  - Reordene os campos e adicione uma linha em branco num ponto apropriado, separando os campos do pacote fonte dos campos do pacote binário.

  - Adicione o campo `Source`.
  - Adicione o campo `Standards-Version` (Por favor confira o Manual de Política Debian para detalhes sobre esse campo).
  - Mude o campo `Architecture` para cada pacote para `any`, `all` ou o que for. Se não há um campo `Architecture` inclua um.
  - Se nenhum outro uso de sed ou coisas normais de acontecerem para fazer os arquivos de controle binários usa a função de substituição de variáveis do `dpkg-gencontrol`para conseguir o mesmo efeito. Use `debian/substvars` se você precisa colocar informação normalmente-gerada (tirando os detalhes dos arquivos `.deb`) no arquivo `.changes` também.

- Edite o `debian/rules`:
  - Remova os alvos `source` e `diff` e qualquer `changes` e `dist`. Essas coisas agora acontecem em um jeito idependente de pacote e não são feitos pelo `debian/rules`.
  - Divida o alvo `binary` em `binary-arch` e `binary-indep`; em muitos casos tudo de `binary` deve ir no `binary-arch`. Crie o alvo `binary` e o não usado dos dois alvos `binary-*` se há um - você pode copiar os do pacote `hello`.
  - Mude o alvo `binary` para usar o `dpkg-gencontrol` para fazer o(s) arquivo(s) de controle do pacote. Mova-o depois de todos os arquivos terem sido instalados mas antes dos últimos `chown` e `chmod` no alvo.
  - Mude as ocorrências de `debian-tmp` para `debian/tmp`.
  - Mude as ocorrências de `debian.{post,pre}{inst,rm}` para `debian/*`.
  - Remova a configuração de número de versão do topo se há algum.
  - Assegure-se que os changelogs específico do Debian e da versão externa sejam instalados.

- Mude o pacote para usar `dpkg-shlibdeps` para determinar suas dependências de bibliotecas compartilhadas e substituí-las lá dentro. Dependências em bibliotecas compartilhadas não devem ser colocadas dentro do pacote fonte.

- Confira que o `debian/README` é realmente o arquivo de copyright e se sim, renomeie-no para `debian/copyright` e edite `debian/rules` para lidar com isso e para mudar a instalação do arquivo de copyright de `/usr/doc/`*`pacote`*`/copyright` para `/usr/doc/copyright/`*`pacote`*. Se não é então ache o `debian/copyright` e decida o que fazer com o `README`.

- Confira os vários outros anacronismos e problemas:
  - Remova quaisquer campos `Package_Revision`, `Package-Revision` ou `Revision`.
  - Renomeie `Optional` para `Suggests`, `Recommended` para `Recommends`.
  - Mude `/usr/doc/examples/`*`pacote`* para `/usr/doc/`*`pacote`*`/examples`.
  - Se assegure de que as páginas de manual são instaladas comprimidas.
  - Confira que a descrição tem uma descrição extendida, é bem formatada e significante e de ajuda para as pessoas que querem saber se instalam o pacote.

- Olhando tudo no geral.

- Faça uma construção teste usando `dpkg-buildpackage -us -uc -sa -r`*`qualquercoisa`*. Cheque as permissões e localização dos arquivos no pacote resultante examinando a saída de `dpkg-deb --contents` e cheque se a construção do fonte foi OK. Teste instalar o paocte construído e teste extrair o pacote fonte.

- Assine o lançamento: ou reconstrua tudo com `dpkg-buildpackage -sa`, ou assine com PGP o `.dsc`, reconstrua o `.changes` usando `dpkg-genchanges -sa`, e então assine o `.changes`.

O uso de `-sa` em `dpkg-buildpackage` e `dpkg-genchanges` é importante quando fazendo a primeira construção/envio de um pacote fonte de novo formato. A menos que aconteça de ser uma revisão Debian `0` ou `1` por padrão o fonte original não será incluído nos arquivos listados no arquivo `.changes` e então não será instalado no site FTP. `-sa` pede que o fonte original seja incluído sem considerações.