

Guia do Novo Mantenedor Debian

Josip Rodin <joy-mg@debian.org>

Traduzido por: Mahdi <mahdi@dcc.ufmg.br>

Revisado por: Priscilla Pimenta <priscilla@minaslivre.org>

version 1.2, 6 April 2002.

Nota de Copyright

Copyright © 1998-2002 Josip Rodin.

Este documento pode ser utilizado sob os termos do GNU General Public License versão 2 ou superior.

Este documento foi criado utilizando os seguintes documentos como exemplos:

Making a Debian Package (AKA the Debmake Manual), copyright © 1997 Jaldhar Vyas.

The New-Maintainer's Debian Packaging Howto, copyright © 1997 Will Lowe.

Sumário

1	Começando do jeito certo:	1
1.1	Programas que você precisa para o desenvolvimento:	1
1.2	Outras informações:	4
2	Primeiros passos	5
2.1	Escolha seu programa	5
2.2	Pegue o programa e teste	6
2.3	Nome do pacote e versão	7
2.4	“Debianização” inicial	8
3	Modificando o código-fonte	9
3.1	Instalação em um subdiretório	9
3.2	Mudando bibliotecas	12
4	Coisas necessárias no debian/	13
4.1	arquivo ‘control’	13
4.2	O arquivo ‘copyright’	17
4.3	O arquivo ‘changelog’	18
4.4	O arquivo ‘rules’	19
5	Outros arquivo no debian/	25
5.1	README.Debian	25
5.2	conffiles.ex	26
5.3	cron.d.ex	26
5.4	dirs	26

5.5	docs	27
5.6	emacsen-*.ex	27
5.7	init.d.ex	28
5.8	manpage.1.ex, manpage.sgml.ex	28
5.9	menu.ex	29
5.10	watch.ex	29
5.11	ex.package.doc-base	30
5.12	postinst.ex, preinst.ex, postrm.ex, prerm.ex	30
6	Construindo o pacote	31
6.1	Reconstrução completa	31
6.2	Reconstrução rápida	32
7	Procurando por erros no pacote	35
8	Enviando o pacote	37
9	Atualizando o pacote	39
9.1	Nova revisão Debian	39
9.2	Nova distribuição do programa	39
9.3	Verificando atualizações de pacotes	40
10	Onde pedir ajuda	41

Capítulo 1

Começando do jeito certo:

O intuito deste documento é descrever a construção de um pacote Debian para o usuário comum do Debian, aspirante desenvolvedor. Ele utiliza uma linguagem bastante informal e é bem ilustrado com exemplos funcionais. Existe um velho ditado Romano, *Longum iter est per preaecepta, breve et efficax per exempla!* (É um longo caminho o das regras, mas curto e eficiente se exemplificado!).

Um dos fatores que fazem do Debian uma distribuição diferenciada é o seu sistema de pacotes. Mesmo já existindo uma vasta quantidade de programas já distribuídos no formato Debian, as vezes é necessário instalar programas que não o são. Você deve estar imaginando como criar seus próprios pacotes, e talvez pense que seja uma tarefa um tanto difícil. Bem, se você é realmente um iniciante no Linux, é complicado, mas se você já tivesse alguma experiência você nem estaria lendo esta documentação agora; :-). Você precisa saber algo sobre programação em Unix, mas certamente não precisa ser um especialista.

Uma coisa é certa, entretanto: para criar e manter pacotes Debian você precisa de bastante tempo. Não cometa erros. Para o nosso sistema funcionar corretamente, os mantenedores precisam ser tanto tecnicamente competentes como aplicados.

Este documento irá explicar passo a passo (mesmo os que a princípio alguns pareçam irrelevantes) e irá ajudá-lo a criar um primeiro pacote e ganhar alguma experiência na criação das próximas distribuições desse mesmo pacote ou até mesmo de outros pacotes.

Novas versões deste documento deverão estar sempre disponíveis em <http://www.debian.org/doc/maint-guide/> e no pacote `'maint-guide-pt'`.

1.1 Programas que você precisa para o desenvolvimento:

Antes de mais nada, você deve estar certo de que tem instalado alguns pacotes adicionais necessários para o desenvolvimento. Note que a lista não contém nenhum pacote marcado 'essential' ou 'required' - supomos que você já tenha tais pacotes instalados.

Esta revisão deste documento foi atualizada para os pacotes no Debian ('sid')

Os seguintes pacotes vêm na instalação padrão do Debian, e você provavelmente já os tem. Mesmo assim, é prudente conferir com `'dpkg -s <package>'`.

- `dpkg-dev` - este pacote contém ferramentas necessárias para desempacotar, construir e enviar arquivos-fonte do Debian. (veja `dpkg-source(1)`)
- `file` - este prático programa pode determinar o tipo de um arquivo. (veja `file(1)`)
- `gcc` - O Compilador C GNU, necessário se o seu programa, como muitos outros, foi escrito em C. (veja `gcc(1)`) Este pacote irá também instalar vários outros pacotes, como `binutils` que contém programas utilizados para compilar e montar arquivos-objeto.(veja 'info binutils' no pacote `binutils-doc`) e `cpp`, o preprocessador C. (veja `cpp(1)`)
- `g++` - O Compilador C++ GNU, necessário se o seu programa foi escrito em C++. (veja `g++(1)`)
- `libc6-dev` - as bibliotecas para o C e arquivos de header que o gcc precisa criar arquivos-objeto.(veja 'info libc' no pacote `glibc-doc`)
- `make` - normalmente a criação de um programa é feito em várias etapas, e assim sendo é melhor utilizar este programa para automatizar as etapas criando um Makefile ao invés de executar os comandos sempre. (veja 'info make')
- `patch` - esta ferramenta é muito útil para analisar um arquivo contendo uma lista de diferenças (criado pelo programa `diff`) e aplica-la no arquivo original, produzindo uma versão "patcheada". (veja `patch(1)`)
- `perl` - Perl é uma das linguagens de scripting interpretadas mais utilizadas nos sistemas baseados em Unix atuais. (veja `perl(1)`)

Você provavelmente vai querer instalar os seguintes pacotes também:

- `autoconf` e `automake` - muitos dos programas atuais utilizam scripts de configuração e Makefiles preprocessados com ajuda desses programas. (veja 'info autoconf', 'info automake')
- `dh-make` and `debhelper` - `dh-make` é necessário para criar o esqueleto do nosso pacote-exemplo e vai utilizar algumas ferramentas do `debhelper` para criação de pacotes. Elas não são essenciais para a criação de pacotes, mas são **fortemente** recomendadas para novos mantenedores. Elas tornam todo o processo muito mais fácil de começar e controlar, no final das contas. (veja `dh_make(1)`, `debhelper(1)`, `/usr/share/doc/debhelper/README`)
- `devscripts` - este pacote contém alguns scripts muito úteis que podem ser de interesse para os mantenedores, mas não são necessários para a criação de pacotes. (veja `/usr/share/doc/devscripts/README.gz`)

- `fakeroot` - este utilitário permite que você emule ser usuário `root`, o que é necessário em algumas etapas do processo de criação. (veja `fakeroot(1)`)
- `gnupg` - uma ferramenta que permite que você *assine* digitalmente seus pacotes. Isto é especialmente importante se você quer distribuí-los para outras pessoas, e você certamente o fará quando seu trabalho for incluído na distribuição Debian. (veja `gpg(1)`)
- `g77` - o compilador Fortran 77 GNU, necessário se o seu programa foi escrito em Fortran. (see `g77(1)`)
- `gpc` - o compilador Pascal GNU, necessário se o seu programa foi escrito em Pascal. É bom notar também o pacote `fp-compiler`, o compilador Pascal gratuito, que exerce a mesma função igualmente bem. (veja `gpc(1)`, `ppc386(1)`)
- `xutils` - alguns programas, normalmente os criados para o X11, também utilizam esses programas para gerar Makefiles de conjuntos de funções de macro.(see `imake(1)`, `xmkmf(1)`)
- `lintian` - este é o testador de pacotes Debian. Ele permite que você saiba se cometeu alguns dos erros mais comuns, depois de criado o pacote, e explica os erros encontrados. (veja `lintian(1)`, `/usr/share/doc/lintian/lintian.html/index.html`)

O seguinte pacote é uma documentação *muito* importante que você deve ler junto com este documento:

- `debian-policy` - o Policy contém explicações sobre a estrutura e conteúdo do repositório Debian, muitas questões sobre arquitetura de sistemas operacionais, a hierarquia padrão do sistema de arquivos (que diz onde cada arquivo e diretório deve ficar) etc. Para você, o importante é que ele descreve os pre-requisitos que cada pacote deve satisfazer para ser incluído na distribuição. (veja `/usr/share/doc/debian-policy/policy.html/index.html`)
- `developers-reference` - para todos os casos não especificamente sobre detalhes técnicos de empacotamento, como a estrutura do repositório, como renomear, tornar órfão adotar pacotes, como fazer NMUs, como tratar de bugs, onde e quando enviar os pacotes, etc. (veja `/usr/share/doc/developers-reference/developers-reference.html/index.html`)

As curtas descrições que são dadas acima servem apenas para apresentar-lhe o que cada pacote faz. Antes de continuar, por favor leia cuidadosamente a documentação de cada programa, pelo menos na parte do uso padrão. Pode parecer muita coisa para estudar, mas isto lhe será de *enorme* valia no final.

Nota: `debmake` é um pacote que contém alguns programas similares ao `dh-make`, mas seu uso específico **não** é tratado neste documento, porque ele é *obsoleto*. Para mais informações, veja the Debmake manual (<http://www.debian.org/~jaldhar/>)

1.2 Outras informações:

Existem dois tipos de pacotes que você pode criar: fonte e binários. Um pacote fonte contém um código-fonte que você possa compilar. Um pacote binário contém somente o programa pronto. Não confunda os termos, como o código-fonte do programa e o pacote-fonte do programa. Por favor leia outros manuais se você precisa de maiores informações sobre essa terminologia.

No Debian, o termo 'mantenedor' é usado para uma pessoa que cria pacotes, 'autor' para a pessoa que criou o programa e 'mantenedor superior' para a pessoa que atualmente mantém o programa fora do Debian. Normalmente o autor e o mantenedor superior são a mesma pessoa - e as vezes até mesmo o mantenedor é a mesma pessoa. Se você criou um programa, e que colocá-lo no Debian, sinta-se a vontade para enviar-nos sua aplicação para se tornar um mantenedor.

Depois que você criou o seu pacote (ou enquanto o faz), você terá de se tornar um mantenedor oficial da Debian para que seu programa seja colocado na próxima distribuição do Debian (se o seu programa é realmente útil, por que não?). Esse processo é tratado na Referência do Desenvolvedor (Developer's Reference). Por favor leia-o.

Capítulo 2

Primeiros passos

2.1 Escolha seu programa

Você provavelmente escolheu o pacote que quer criar. A primeira coisa que você precisa fazer é conferir se o pacote já não está na distribuição. Se você usa a distribuição 'stable', talvez seja melhor vc ir na página de busca de pacotes Debian (<http://www.debian.org/distrib/packages>). Se você usa a distribuição 'unstable' **atual** confira com os seguintes comandos:

```
dpkg -s program
dpkg -l '*program*'
```

Se o pacote já existe, bem, é só instalar! :-) Se ele estiver órfão – se seu mantenedor estiver marcado no “Grupo QA do Debian”, você pode adotá-lo. Consulte a lista de pacotes órfãos (<http://www.debian.org/devel/wnpp/orphaned>) e a lista de pacotes para adoção (http://www.debian.org/devel/wnpp/rfa_bypackage) para verificar se o pacote está realmente disponível.

Se você pode adotar o pacote, pegue o seu código-fonte (com algo como, `apt-get source packagename`) e analise-o. Este documento infelizmente não inclui informações sobre adoção de pacotes. Felizmente você não deve ter problemas descobrindo como o pacote funciona uma vez que alguém já fez todo o trabalho inicial para você. Mesmo assim continue lendo, muito do que vem a seguir ainda será aplicável para você.

Se o pacote ainda não existe, e você gostaria de vê-lo no Debian, prossiga como seguinte:

- Confira se ninguém mais está trabalhando no pacote na lista de pacotes em desenvolvimento (http://www.de.debian.org/devel/wnpp/being_packaged). Se alguém já está criando-o, entre em contato com essa pessoa se você achar desejável. Senão - procure outro programa interessante que ninguém mantém ainda.
- Programas **têm** de ter uma licença, se possível gratuita como descrito no Guia Debian de Software Livre (http://www.debian.org/social_contract#guidelines). Se o

programa não concorda com alguma dessas regras, mas ainda assim pode ser distribuído, ele ainda pode ser incluído nas seções 'contrib' ou 'non-free'. Se você ainda não tem certeza sobre onde ele deve ser incluído, envie o texto da licença para <debian-legal@lists.debian.org> e peça ajuda.

- Programas certamente **não** devem ser executados com privilégios de root (setuid root), ou ainda melhor - eles não devem precisar depender de nenhum privilégio específico (nem setuid nem setgid)
- Programas não devem ser daemons, nem nada que tenha de ir para diretórios */sbin, nem abrir portas como root.
- Programas devem estar na forma executável, bibliotecas são mais difíceis de lidar.
- Tudo deve ser bem documentado, e/ou o código tem de ser bem legível e inteligível.
- Você deve entrar em contato com o(s) autor(es) do programa para confirmar se eles concordam com o seu empacotamento. É importante ser possível o contato com o(s) autor(es) no caso de quaisquer problemas específicos do programa, então não faça pacotes de software sem suporte.
- E por último, mas não menos importante, você deve ter certeza que o tudo funciona corretamente, utilizando algum tempo.

Obviamente tudo isso são medidas de segurança, e têm como intuito evitar usuários zangados com você, se você fez algo errado em algum daemon setuid root... Quando você tiver mais experiência na criação de pacotes, você poderá criar tais pacotes, mas mesmo os desenvolvedores mais experientes consultam a lista debian-mentors quando em dúvida. E as pessoas lá ficarão felizes em poder ajudar.

Para mais ajuda sobre isso, leia a Referência do Desenvolvedor (Developer's Reference).

2.2 Pegue o programa e teste

A primeira coisa a ser feita é encontrar e baixar o pacote original. Eu presudo que você já tem o código-fonte pegado na página do autor. Códigos-fonte para software livre em Unix normalmente vêm no formato tar/gzip, com a extensão .tar.gz, e normalmente contém um subdiretório com o nome nome_do_programa-versão e todos os códigos-fonte dele dentro. Se o código-fonte do seu programa vem em outro tipo de arquivamento (como por exemplo um arquivo de extensão ".Z" ou ".zip"), desempacote-o com as ferramentas adequadas ou peça ajuda na lista debian-mentors se você não tem certeza de como desempacota-lo corretamente (dica: execute 'file arquivo.extensão').

Como exemplo, eu usarei um programa chamado 'gentoo', um manipulador de arquivos para o X em GTK+. Note que o programa já está empacotado, e mudou bastante desde a última versão em que este texto foi escrito.

Crie um subdiretório no seu home chamado 'debian' ou 'deb' ou algo que você ache mais apropriado (ex: ~/gentoo/ seria suficiente neste caso). Coloque o arquivo baixado dentro dele, e descompacte-o (com 'tar xzf gentoo-0.9.12.tar.tz'). Certifique-se que não ocorreram erros, mesmo os mais "irrelevantes", pois provavelmente poderão ocorrer erros nos sistemas de outras pessoas, que têm ferramentas de descompactamento que podem ou não ignorar tais anomalias.

Agora você tem outro subdiretório, chamado 'gentoo-0.9.12'. Mude para esse diretório e leia **cuidadosamente** a documentação fornecida. Normalmente existem arquivos chamados RE-ADME*, INSTALL*, *.lsm ou *.html. Você deve encontrar instruções de como compilar e instalar o programa corretamente (provavelmente eles irão assumir que você quer instalar o programa no diretório /usr/local/bin, mas você não fará isso. Mais sobre isso será dito em 'Instalação em um subdiretório' on page 9).

O processo varia de programa para programa, mas vários programas atuais vem com um script 'configure' que configura o código-fonte de acordo com o seu sistema e garante que seu sistema tem condições de compilar o programa. Após configurar o código-fonte com './configure', os programas são normalmente compilados com 'make'. Alguns deles suportam 'make check', para rodar incluindo auto-testes. A instalação nos devidos diretórios de destino são normalmente feitos com 'make install'.

Agora tente compilar e executar seu programa, para garantir que ele funciona corretamente e não interfere em nada enquanto ele está sendo instalado ou executado.

Normalmente você também pode executar 'make clean' (ou melhor ainda 'make distclean') para limpar o diretório de compilação. Algumas vezes você pode inclusive executar um 'make uninstall' para remover todos os arquivos instalados.

2.3 Nome do pacote e versão

Você deve começar empacotando com um diretório de código-fonte completamente limpo, ou simplesmente com um código-fonte recém descompactado.

Para que o pacote seja criado corretamente, você deve tornar o nome do programa caixa-baixa (se já não for), e deve mover o diretório fonte para <nome_do_pacote>-<versão>.

Se o nome do programa utiliza mais de uma palavra, reduza-o para uma palavra ou abrevie-o. Por exemplo, o programa "Pequeno Editor do John para X" pode ser nomeado "xpedjohn", ou "xeditorjohn" ou qualquer outra coisa que você decidir, desde que o nome tenha um nome razoável, como 20 caracteres.

Verifique também a versão exata do programa (para ser incluída na versão do pacote). Se este software não utiliza versões numeradas tipo X.Y.Z, mas algo como uma data, sinta-se a vontade para utilizar essa data como a versão, sufixada com "0.0." (para o caso de alguém superior na manutenção do programa resolver um dia distribuir uma versão tipo 1.0). Dessa forma, se a distribuição ou imagem do programa é do dia 19/12/1998, você pode utilizar a versão como sendo 0.0.19981219.

Alguns programas nem têm uma versão definida, caso o qual você deve entrar em contato com o mantenedor superior para certificar-se de algum outro eventual sistema de revisão por ele utilizado.

2.4 “Debianização” inicial

Certifique-se que você está no diretório do código-fonte do programa e execute:

```
dh_make -e seu.endereço@de.mantenedor -f ../gentoo-0.9.12.tar.gz
```

Obviamente você deve substituir “seu.endereço@de.mantenedor” com o seu endereço de e-mail a ser incluído no changelog, outros arquivos e no nome do arquivo com o nome do seu código-fonte. Veja `dh_make(1)` para maiores informações.

Algumas informações serão mostradas. Vai ser perguntado que tipo de pacote você quer criar. O gentoo é um pacote de um único binário - ele só cria um arquivo binário, e logo um arquivo `.db -`, logo, vamos selecionar a primeira opção utilizando a tecla ‘s’. Confira a informação impressa na tela e confirme pressionando `<enter>`.

Novamente, como novo mantenedor, não é recomendável que você crie pacotes de múltiplos binários ou bibliotecas. Não é muito complicado, mas exige um pouco mais de conhecimento, e assim sendo não será descrito aqui.

Note que você deve executar o `dh_make` **somente uma vez**, e que ele não se comportará corretamente se você executá-lo novamente num diretório já “debianizado”. Isso também significa que você irá utilizar um método diferente para criar uma nova revisão ou versão do seu pacote no futuro. Leia mais sobre isso em ‘Atualizando o pacote’ on page [39](#)

Capítulo 3

Modificando o código-fonte

Normalmente, os programas são instalados em subdiretórios em `/usr/local`, mas os pacotes do Debian não devem utilizar esse diretório, uma vez que ele é reservado ao uso privado dos administradores do sistema. Isto significa que você terá de dar uma lhadada no sistema de construção do seu programa, normalmente começando com o Makefile. Este é o script `make` (1) que você irá utilizar para automatizar a compilação do seu programa. Para maiores informações sobre Makefiles, leia ‘O arquivo ‘rules’’ on page 19.

Note que se o seu programa utiliza o GNU `automake` (1) e/ou `autoconf` (1), o código-fonte incluirá os arquivos `Makefile.am` e/ou `Makefile.in`, respectivamente, e você terá de modificar esses arquivos também. Isto acontece pois cada execução do `automake` irá reescrever o `makefile.in` com alguma informação gerada a partir do `Makefile.am`, e cada execução do `./configure` irá fazer o mesmo com os Makefiles, com informações do `Makefile.in`. Editando arquivos `Makefile.am` exigem algum conhecimento sobre o `automake`, que você pode adquirir no info do `automake`; ao passo que editar arquivos `Makefile.in` são mais ou menos a mesma coisa que editar Makefiles, tomando cuidado com as variáveis (ex: qualquer string entre ‘@’s, como `@CFLAGS@` ou `@LN_S@`, que são substituídos com a informação propriamente dita de cada execução do `./configure`).

Note também que foge ao nosso escopo aqui descrever *todos* os detalhes sobre correção de códigos, mas existem alguns problemas que normalmente são encontrados.

3.1 Instalação em um subdiretório

A maioria dos programas tem algum modo de ser instalados na estrutura de diretório existente do seu sistema, de modo que seus binários são incluídos no seu `$PATH` e sua documentação se encontra em lugares comuns. Se você fizer isso, entretanto, seu programa será instalado junto com tudo o mais que já esteja no seu sistema. Isso tornaria difícil para que as ferramentas de empacotamento descobrissem quais arquivos pertencem ou não ao seu pacote.

Portanto você precisa fazer algo mais: instalar o programa dentro num subdiretório temporário de onde as ferramentas de mantenedor irão trabalhar no pacote `.deb`. Tudo que estiver nesse

diretório irá ser instalado no sistema do usuário quando eles instalarem o pacote. A única diferença é que o `dpkg` vai estar instalando os arquivos no diretório raiz.

Este diretório temporário é normalmente criado dentro do seu diretório `debian/`, na árvore de diretórios e código-fonte descompactada. Ele é normalmente chamado `debian/nome_do_pacote`.

Tenha em mente que mesmo que você tenha que fazer o programa ser instalado em `debian/nome_do_pacote`, ele ainda precisa se comportar corretamente quando colocado no diretório raiz (ex: quando instalado do pacote `.deb`). Logo, você não deve deixar que o sistema de construção utilize strings como `/home/me/gentoo-0.9.12/usr/share/gentoo` nos arquivos do pacote.

Com programas que utilizam o GNU `autoconf`, isto será bastante fácil. A maioria destes programas tem `Makefiles` que são como padrão definidos para permitir que a instalação seja feita num diretório aleatório, tendo em mente que `/usr` (por exemplo) é o prefixo canônico. O `dh_make`, quando detectar que seu programa utiliza o `autoconf`, irá definir os comandos para fazer tudo isso automaticamente, de forma que você pode inclusive saltar a leitura desta seção, mas com outros programas, você provavelmente terá de analisar e editar os `Makefiles`.

Eis a parte relevante do `Makefile` do `gentoo`:

```
# Onde colocar o binário durante o 'make install'?
BIN      = /usr/local/bin

# Onde colocar os ícones durante o 'make install'?
ICONS    = /usr/local/share/gentoo
```

Nota-se que os arquivos estão definidos para serem instalados em `/usr/local`. Mude tais caminhos para:

```
# Onde colocar o binário durante o 'make install'?
BIN      = $(DESTDIR)/usr/bin

# Onde colocar os ícones durante o 'make install'?
ICONS    = $(DESTDIR)/usr/share/gentoo
```

Mas porque em tal diretório, e não algum outro? Porque os pacotes `debian` nunca instalam pacotes em `/usr/local` – essa árvore de diretórios é reservada ao uso do administrados do sistema. Tais arquivos, no `Debian`, devem ser colocados em `/usr`.

A localização mais exata para binários, ícones, documentação, etc, é especificada na Hierarquia Padrão de Sistemas de Arquivos (Filesystem Hierarchy Standard - FHS - veja `/usr/share/doc/debian-policy/fhs`). Eu recomendo que você procure e leia as seções que podem interessar ao seu pacote.

Então nós devemos instalar o binário em `/usr/bin` no lugar de `/usr/local/bin`, os manuais em `/usr/share/man/man1` ao invés de `/usr/local/man/man1`, etc. Note que, como o `gentoo` não

fornece um manual em seu Makefile, e o Debian-Policy exige que todo programa tenha um, nós faremos um, mais tarde, e o instalaremos em `/usr/share/man/man1`.

Alguns programas não usam variáveis no Makefile para definir caminhos como estes. Isto significa que você talvez tenha de editar alguns arquivos-fonte em C para que eles utilizem os caminhos adequados. Mas onde procurar, e exatamente o que procurar? Você pode resolver isso executando:

```
grep -nr -e 'usr/local/lib' --include='*.[c|h]' .
```

O Grep irá executar recursivamente por toda a árvore do código e dizer-lhe o nome e a linha do arquivo quando encontrar uma ocorrência.

Edite esses arquivos e substitua `/usr/local/*` com `usr/*` e pronto. Tome cuidado para não mexer no resto do código! :-)

Depois disso você deve encontrar o alvo de instalação (procure pela linha começa com `'install:'`, o que normalmente funciona) e renomeie todas as referências a diretórios que não aqueles definidos no topo do Makefile. Antes, o alvo de instalação do gentoo continha:

```
install:          gentoo
                  install ./gentoo $(BIN)
                  install icons/* $(ICONS)
                  install gentoorc-example $(HOME)/.gentoorc
```

Depois da mudança, passou a conter:

```
install:          gentoo-target
                  install -d $(BIN) $(ICONS) $(DESTDIR)/etc
                  install ./gentoo $(BIN)
                  install -m644 icons/* $(ICONS)
                  install -m644 gentoorc-example $(DESTDIR)/etc/gentoorc
```

Você certamente notou que agora tem um comando `install -d` antes dos outros comandos semelhantes. O Makefile original não tinha isso porque normalmente o diretório `/usr/local/bin` e outros diretórios já existiam no sistema onde alguém executasse o `'make install'`. Entretanto, como nós vamos instalar o programa em nosso próprio diretório vazio (ou não existente), nós teremos de criar cada um desses diretórios.

Nós podemos inclusive adicionar outras coisas no final da regra, como a instalação de documentação adicional, que os autores algumas vezes omitem:

```
install -d $(DESTDIR)/usr/share/doc/gentoo/html
cp -a docs/* $(DESTDIR)/usr/share/doc/gentoo/html
```

Um leitor atento notará que eu troquei 'gentoo' para 'gentoo-target' na linha de 'install:'. Isto é chamado de correção de bug não-relacionado. :-)

Sempre que você fizer mudanças que não são especificamente relacionado ao pacote Debian, certifique-se de manda-las para o mantenedor superior para que elas possam ser incluídas na próxima revisão do programa e serem úteis para todo mundo. Lembre-se também de fazer suas mudanças de modo não específico ao Debian ou Linux (ou até mesmo Unix!) antes de enviá-las – faça-as portáveis. Isto tornará suas correções mais fáceis de serem aplicadas.

Note que você não tem de enviar os arquivos de debian/* ao mantenedor superior.

3.2 Mudando bibliotecas

Existe um outro problema comum: bibliotecas muitas vezes variam de plataforma para plataforma. Por exemplo, um Makefile faz referência a uma biblioteca que não existe em sistemas Debian. Neste caso, nós precisamos muda-la para uma biblioteca que existe no Debian, e serve pra mesma coisa.

Assim sendo, se existe alguma linha no Makefile do seu programa (ou Makefile.in) que tenha algo como isto (e o seu programa não compile):

```
LIBS = -lcurses -lalgumacoisa -lalgumaoutracoisa
```

Mude-a para isso, e provavelmente funcionará:

```
LIBS = -lncurses -lalgumacoisa -lalgumaoutracoisa
```

(o autor percebe que este não é um bom exemplo, considerando que nosso pacote libncurses agora cria um symlink para libcurses.so, mas ele não pode encontrar nada melhor. Sugestões são muito bem vindas :-)

Capítulo 4

Coisas necessárias no debian/

Agora existe um novo subdiretório no diretório-fonte do programa chamado 'debian'. Existem alguns arquivos nesse diretório que nós devemos editar para personalizar o comportamento do nosso pacote. Os mais importantes deles são 'control', 'changelog', 'copyright' e 'rules', que são necessários para todos os pacotes.

4.1 arquivo 'control'

Este arquivo contém vários valores que o `dpkg`, `dselect` e outras ferramentas de controle de pacotes irão utilizar para administrar o pacote.

Eis o arquivo de controle que o `dh_make` criou para nós:

```
1 Source: gentoo
2 Section: unknown
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (>> 3.0.0)
6 Standards-Version: 3.5.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Description: <insert up to 60 chars description>
12 <insert long description, indented with spaces>
```

(eu adicionei os números das linhas)

As linhas 1-6 são as informações de controle para o pacote.

A linha 1 é o nome do pacote.

A linha 2 é a seção da distribuição em que o pacote será incluído.

Como você deve ter notado, o Debian é dividido em seções: main (a seção de software livre), non-free (os softwares gratuitos, mas não-livres) e contrib (software livre que depende de software não-livre). Dentro dessas, existem subseções lógicas que descrevem brevemente o tipo de pacote guardam. Então temos 'admin' para ferramentas de administradores, 'base' para ferramentas básicas, 'devel' para ferramentas de programação, 'doc' para documentação, 'libs' para bibliotecas, 'mail' para clientes e daemons de email, 'net' para aplicativos e daemons de rede, 'x11' para programas gráficos que não se enquadram em outras categorias, e muitas outras.

Então vamos alterá-lo para x11. (um prefixo 'main/' é implícito, e logo podemos omiti-lo.)

A linha 3 descreve o quão importante que o usuário instale este pacote. Leia o Debian-Policy para orientação no preenchimento desta lacuna. A prioridade 'optional' normalmente caberá para novos pacotes.

Seção e prioridade são utilizados por frontends como o dselect quando eles ordenam e selecionam os pacotes padrão. Assim que você enviar o pacote para o Debian, o valor dessas duas lacunas poderá ser alterada pelos mantenedores do repositório, caso o qual você será notificado por email.

Como este pacote é de prioridade normal e não entra em conflito com nada, nós deixaremos a lacuna preenchida como 'optional'.

A linha 4 é o nome e endereço de email do mantenedor. Certifique-se que este campo contenha um cabeçalho 'To:' válido para um email, porque depois que você enviar o pacote, o sistema de procura de bugs utilizará ele para enviar emails de bugs para você. evite utilizar vírgulas, '&'s e parênteses.

A 5ª linha contém a lista dos pacotes necessários para construir o seu pacote. Alguns pacotes, como o gcc e o make são implícitos, veja o pacote build-essential para detalhes. Se algum compilador ou ferramenta não-padrão for necessária para construir o seu pacote, você deve adicioná-lo à linha 'Build-Depends'. Múltiplas entradas são separadas por vírgulas; leia a seguir para uma explicação das dependências binárias para saber mais sobre a sintaxe deste campo.

Você também pode ter Build-depends-Indep, Build-Conflicts e outros campos aqui. Estas informações serão utilizadas pelo software de criação automática de pacotes do Debian para criar os pacotes binários para outras plataformas de computadores. Veja o Debian-Policy para mais informações sobre as dependências de compilação e a Referência do Desenvolvedor para mais informações sobre outras plataformas (arquiteturas) e como portar software para elas.

Eis um hack que você pode utilizar para descobrir quais pacotes o seu pacotes precisa para ser compilado:

```
strace -f -o /tmp/log ./configure
# or make instead of ./configure, if the package doesn't use autoconf
for x in `dpkg -S $(grep open /tmp/log|\
    perl -pe 's!.* open\(\\"([^\"]*).*!$1!' |\
    grep "^/" | sort | uniq|\
```

```

        grep -v "^(/tmp\|/dev\|/proc\)" ) 2>/dev/null|\
        cut -f1 -d":" | sort | uniq`; \
do \
    echo -n "$x (>=" `dpkg -s $x|grep ^Version|cut -f2 -d":" `"), "; \
done

```

O gentoo precisa dos pacotes `xlibs-dev`, `libgtk1.2-dev` e `libglib1.2-dev` para ser compilado, portanto vamos adiciona-los após o pacote `debhelper`.

A linha 6 é a versão do padrão do Debian-Policy que este pacote segue. As verões do Debian-Policy que você lê enquanto cria seu pacote.

A linha 8 é o nome do seu pacote binário. Isto normalmente é a mesma coisa que o nome do pacote-fonte, mas não precisa ser necessariamente desta forma.

A linha 9 descreve as arquiteturas de CPU para que o pacote binário pode ser compilado. Nós deixaremos este como 'any' pois `dpkg-gencontrol(1)` preencherá este campo com o valor adequado para qualquer máquina em que este pacote seja compilado.

Se o seu pacote é independente de arquitetura (por exemplo, um script shell ou Perl, ou um documento), mude este para 'all', e leia mais tarde em 'O arquivo 'rules'' on page 19 sobre a utilização da regra 'binary-indep' ao invés da 'binary-arch' para construir o pacote.

A linha 10 mostra uma das funcionalidades mais poderosas do sistema de empacotamento do Debian. Os pacotes podem relacionar-se uns com os outros de várias formas. Além da 'Depends:', outros campos relacionais são 'Recommends:', 'Suggests:', 'Pre-Depends:', 'Conflicts:', 'Provides:', e 'Replaces:'.

As ferramentas de manutenção de pacotes normalmente se comportam da mesma maneira quando lidando com essas relações; se não, será explicado. (veja `dpkg(8)`, `dselect(8)`, `apt(8)`, `aptitude(1)` etc.)

Eis o que as dependencias significam:

- **Depends:**
O pacote não será instalado a menos que os pacotes de que ele depende estejam instalados. Use isso se o seu programa não rodará de forma alguma (ou terá sérias implicações) a menos que um pacote em particular esteja presente.
- **Recommends:**
Os Frontends como o `dselect` ou o `aptitude` irão perguntar-lhe se você deseja instalar os pacotes recomendados junto com o seu pacote; o `dselect` vai inclusive insistir. O `dpkg` e o `apt-get` vão ignorar este campo. Use isso para pacotes que não são estritamente necessários, mas são tipicamente utilizados com o seu programa.
- **Suggests:**
Quando um usuário instala o seu programa, todos os frontends vão perguntar-lhe se ele deseja instalar os pacotes sugeridos. O `dpkg` e o `apt-get` não vão. use isso para pacotes que vão funcionar bem com o seu programa, mas não são de forma alguma necessários.

- **Pre-Depends:**
Este é mais forte que o 'Depends:'. O pacote não será instalado a menos que os pacotes de que ele "pre-depends" estejam instalados *e corretamente configurados* use isso **muito** pouco e somente após discutir isso na lista do debian-devel. Leia-se: não use isso. :-)
- **Conflicts:**
O pacote não será instalado até que todos os pacotes com que ele conflite sejam removidos. Use isso se o seu programa absolutamente não rodará ou causará sérios problemas se um pacote em particular esteja presente.
- **Provides:**
para alguns tipos de pacotes existem muitos nomes virtuais que podem ser definidos. Você pode obter a lista completa no arquivo `/usr/share/doc/debian-policy/virtual-package-name-list.txt.gz` . Use isso se o seu programa fornece uma função existente em um pacote virtual.
- **Replaces:**
Use isso quando o seu programa substitui arquivos de outro pacote, ou substitui completamente outro pacote (utilizado juntamente com 'Conflicts'). Arquivos de outros pacotes listados serão sobrescritos com os arquivos do seu pacote.

Todos estes campos tem uma sintaxe uniforme. Eles são uma lista de nomes de pacotes separados por vírgulas. Estes nomes de pacotes podem inclusive ser listas de nomes de pacotes alternativos, serados por simbolos de pipe (barras verticais '|').

Os campos podem restringir sua aplicabilidade a versões particulares de cada pacote listado. Essas versões são listadas entre parênteses após cada nome de pacote individualmente, e devem conter uma relação da seguinte lista seguida pelo número da versão. As relações permitidas são: <<, <=, =, >= e >> para anterior, anterior ou igual, exatamente igual, mais nova ou igual e igual, respectivamente. Por exemplo,

```
Depends: foo (>= 1.2), libbar1 (= 1.3.4)
Conflicts: baz
Recommends: libbaz4 (>> 4.0.7)
Suggests: quux
Replaces: quux (<< 5), quux-foo (<= 7.6)
```

Finalmente, a última funcionalidade que você precisa saber é `dh_shlibs:Depends`. Depois que seu pacote for compilado e instalado no diretório temporário, `dh_shlibdeps(1)` irá procurar por binários e bibliotecas nesse diretório, determinar as dependências de bibliotecas compartilhadas e detectar em quais pacotes elas estão, como `libc6` ou `xlib6g`. Ele irá passar a lista para o `dh_gencontrol(1)` que irá preencher o campo adequadamente, e você não terá de se preocupar mais com isso.

Dito tudo isto, podemos deixar a linha 'Depends:' exatamente como ela está agora, e inserir outra linha após ela com `Suggests: file`, pois o gentoo pode utilizar algumas funcionalidade fornecidas por este programa/pacote.

A linha 11 é uma breve descrição. A maioria das telas das pessoas são de 80 colunas de largura, então esta descrição não deve ser maior que 60 caracteres. Eu mudarei este campo para “mantenedor de arquivos com interface completamente configurável para X utilizando GTK+”.

A linha 12 é onde a descrição detalhada entra. Este deve ser um parágrafo que fornece maiores informações sobre o pacote. A coluna 1 de cada linha deve estar vazia. Não podem haver linhas em branco, mas você pode colocar um único ‘.’ (ponto) numa coluna para simular isto. Além disso, não pode haver mais de uma linha em branco após a descrição detalhada.

Finalmente, eis o arquivo de controle atualizado:

```
1 Source: gentoo
2 Section: x11
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (> 3.0.0), xlibs-dev, libgtk1.2-dev, libglib1.
6 Standards-Version: 3.5.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Suggests: file
12 Description: mantenedor de arquivos com interface completamente configur
13 o gentoo é um mantenedor de arquivos para Linux escrito totalmente em C
14 utiliza o toolkit GTK+ para todas as suas necessidades de interface. O
15 uma interface 100% configurável; sem necessidade de editar arquivos de
16 reiniciar o programa. O gentoo suporta a identificação do tipo de vários
17 arquivos (utilizando extensão, expressões regulares, ou o comando 'file
18 e pode exibir arquivos de diferentes tipos com diferentes cores e ícone
19 .
20 O gentoo adota alguns temas do clássico mantenedor de arquivos do Amiga
21 "Directory OPUS" (escrito por Jonathan Potter).
```

(Eu adicionei os números das linhas.)

4.2 O arquivo ‘copyright’

Este arquivo contém informação sobre os recursos superiores do pacote, informações de copyright e licença. Seu formato não é tratado no Debian-Policy, mas seu conteúdo é (seção 12.6 “Informações de Copyright”).

O dh_make criou o seguinte arquivo padrão:

```
1 This package was debianized by Josip Rodin <joy-mg@debian.org> on
2 Wed, 11 Nov 1998 21:02:14 +0100.
```

```
3
4 It was downloaded from <fill in ftp site>
5
6 Upstream Author(s): <put author(s) name and email here>
7
8 Copyright:
9
10 <Must follow here>
```

(Eu coloquei os números das linhas)

As coisas importantes a serem adicionadas a este arquivo são o lugar de onde você pegou o pacote, as informações de copyright e a licença do pacote. Você deve incluir a licença completa, a menos que seja um software livre de licença comum, como GNU GPL ou LPL, ou a licença artística do BSD, onde você pode simplesmente fazer referência à licença adequada ao diretório `/usr/share/common-licenses/` que existe em todo sistema Debian.

Eis como o arquivo de copyright do gentoo ficaria:

```
1 This package was debianized by Josip Rodin <joy-mg@debian.org> on
2 Wed, 11 Nov 1998 21:02:14 +0100.
3
4 It was downloaded from: ftp://ftp.obsession.se/gentoo/
5
6 Upstream author: Emil Brink <emil@obsession.se>
7
8 Este software é copyright (c) 1998-99 por Emil Brink, Obsession
9 Development.
10
11 Você é livre para distribuir este software sob os termos da
12 GNU General Public License.
13 Em sistemas Debian, o texto completo da GNU General Public
14 License pode ser encontrado em '/usr/share/common-licenses/GPL'
```

(Eu adicionei os números das linhas.)

4.3 O arquivo 'changelog'

Este arquivo é necessário, e tem um formato especial descrito no Debian-Policy seção 4.4 'debian/changelog'. Este formato é utilizado pelo dpkg e outros programas para obter o número da versão, revisão, distribuição e urgência do seu pacote.

Para você, também é importante, desde que é bom ter documentadas todas as mudanças que você fez. Este arquivo ajudará as pessoas que baixam o seu pacote a

ver se existem questões sobre o pacote que eles devam saber. Ele será salvo como `'/usr/share/doc/gentoo/changelog.Debian.gz'` no pacote binário.

O `dh_make` criou o seguinte arquivo padrão:

```
1 gentoo (0.9.12-1) unstable; urgency=low
2
3 * Initial Release.
4
5 -- Josip Rodin <joy-mg@debian.org> Wed, 11 Nov 1998 21:02:14 +0100
6
```

(Eu adicionei os números das linhas.)

A linha 1 é o nome do pacote, versão, distribuição e urgência. O nome deve ser o mesmo nome do pacote-fonte, a distribuição pode ser ou `'unstable'` (ou até mesmo `'experimental'`), e a urgência não deve ser mudada para nada acima de `'low'`. :-)

As linhas 3-5 são a entrada de log, onde você deve documentar as mudanças feitas nessa revisão do pacote (não as mudanças superiores - existe um arquivo especial para este propósito, criado pelos autores, que você instalará depois em `/usr/share/doc/gentoo/changelog.gz`). Novas linhas devem ser inseridas logo antes da linha mais alta que começa com um asterisco (*). Você pode fazê-lo com o `dch(1)`, ou manualmente com um editor de texto.

Você terá algo como isso, no final das contas: You will end up with something like this:

```
1 gentoo (0.9.12-1) unstable; urgency=low
2
3 * Initial Release.
4 * Este é o meu primeiro pacote Debian.
5 * Ajustado o Makefile para corrigir problemas de $DESTDIR.
6
7 -- Josip Rodin <joy-mg@debian.org> Wed, 11 Nov 1998 21:02:14 +0100
8
```

(Eu inseri os números das linhas.)

Você pode ler mais sobre a atualização do arquivo de changelog mais tarde em `'Atualizando o pacote'` on page 39.

4.4 O arquivo `'rules'`

Nós precisamos dar uma olhada nas regras exatas que o `dpkg-buildpackage(1)` irá seguir para criar efetivamente o pacote. Este arquivo é, na verdade, outro Makefile, mas diferente do fornecido pelo autor do código-fonte. Diferentemente dos outros arquivos no debian, este arquivo tem de ser executável.

Todo arquivo 'rules', como qualquer outro Makefile, consiste em várias regras que especificam como tratar do código-fonte. Cada regra consistem em alvos, nomes de arquivos ou nomes de ações que devem ser seguidas (ex: 'build:' ou 'install:'). Regras que você quer que sejam invocadas como argumentos de linhas de comando (por exemplo, './debian/rules build' ou 'make -f rules install'). Depois do nome do alvo, você pode listar as dependências, programas ou arquivos de que a regra depende. Depois disso, podem haver quantos comandos forem necessários, indentados com <tab>. Uma nova regra começa com a declaração do alvo na primeira coluna. Linhas vazias e começadas com '#s (hashs) são tratadas como comentários e são ignoradas.

Você provavelmente está confuso agora, mas isso tudo ficará mais claro com a análise do arquivo 'rules' que o dh_make nos dá como padrão. Você deve também ler a info do 'make' para maiores informações.

A parte importante de saber sobre o arquivo 'rules' criado pelo dh_make, é que ele é somente uma sugestão. Ele funcionará para pacotes simples, mas para os mais complicados, não tenha medo de adicionar ou retirar coisas para satisfazer às suas necessidades. A única coisa que não deve mudar são os nomes das regras, pois todas as ferramentas usam estes nomes, que são exigidos no Debian-Policy.

Eis (aproximadamente) como o debian/rules padrão que o dh_make gerou para nós é:

```
1  #!/usr/bin/make -f
2  # -*- makefile -*-
3  # Sample debian/rules that uses debhelper.
4  # This file was originally written by Joey Hess and Craig Small.
5  # As a special exception, when this file is copied by dh-make into a
6  # dh-make output file, you may use that output file without restricti
7  # This special exception was added by Craig Small in version 0.37 of
8  # Uncomment this to turn on verbose mode.
9  #export DH_VERBOSE=1
10 configure: configure-stamp
11 configure-stamp:
12     dh_testdir
13     # Add here commands to configure the package.
14     touch configure-stamp
15 build: build-stamp
16 build-stamp: configure-stamp
17     dh_testdir
18     # Add here commands to compile the package.
19     $(MAKE)
20     #docbook-to-man debian/testpack.sgml > testpack.1
21     touch $@
22 clean:
23     dh_testdir
24     dh_testroot
25     rm -f build-stamp configure-stamp
```

```
26         # Add here commands to clean up after the build process.
27         $(MAKE) clean
28         dh_clean
29 install: build
30         dh_testdir
31         dh_testroot
32         dh_clean -k
33         dh_installdirs
34         # Add here commands to install the package into debian/testpa
35         $(MAKE) DESTDIR=$(CURDIR)/debian/testpack install
36 # Build architecture-independent files here.
37 binary-indep: build install
38 # We have nothing to do by default.
39 # Build architecture-dependent files here.
40 binary-arch: build install
41         dh_testdir
42         dh_testroot
43         dh_installchangelogs
44         dh_installdocs
45         dh_installexamples
46 #         dh_install
47 #         dh_installdata
48 #         dh_installdebconf
49 #         dh_installogrotate
50 #         dh_installemacs
51 #         dh_installpam
52 #         dh_installdata
53 #         dh_python
54 #         dh_installdata
55 #         dh_installdata
56 #         dh_installdata
57         dh_installman
58         dh_link
59         dh_strip
60         dh_compress
61         dh_fixperms
62 #         dh_perl
63 #         dh_makeshlibs
64         dh_installdeb
65         dh_shlibdeps
66         dh_gencontrol
67         dh_md5sums
68         dh_builddeb
69 binary: binary-indep binary-arch
70 .PHONY: build clean binary-indep binary-arch binary install configure
```

(Eu adicionei os números das linhas.)

Você provavelmente já tem intimidade com linhas como a linha 1 de scripts shell e Perl. Ela diz ao sistema operacional que esse arquivo é para ser processado com o `/usr/bin/make`.

O significado das variáveis `DH_*` mencionadas nas linhas 8 e 9 deve ser evidente para a breve descrição. Para maiores informações sobre o `DH_COMPAT` leia a seção 'Níveis de compatibilidade do Debhelper' no manual `debhelper(1)`.

As linhas 11-16 são o esqueleto do suporte para parâmetros do `DEB_BUILD_OPTIONS`, descritos no Debian-Policy seção 10.1 'Binários'. Basicamente, essas coisas controlam se os binários devem ser construídos com a tabela de símbolos, ou se eles devem ser retirados na instalação. Novamente, este é somente um esqueleto, uma dica de que você deve fazê-lo. Você deve conferir como os sistemas de construção superiores tratam da inclusão da tabela de símbolos e da sua retirada na instalação, ou implementar isso você mesmo.

Normalmente, você poderá usar o `gcc` para compilar com `'-g'` usando a variável `CFLAGS` – se este é o caso do seu pacote, estenda a variável `concatenando CFLAGS="$ (CFLAGS) "` à invocação do `$(MAKE)` na regra de construção (veja abaixo). Alternativamente, se o seu pacote usa um script de configuração do `autoconf`, você pode passá-lo *concatenando antes* do string acima na invocação do `./configure` na regra de construção.

Para a remoção, os programas são normalmente configurados para se instalarem sem a remoção da tabela de símbolos, e normalmente sem uma opção para mudar isso. Felizmente, você ainda tem o `dh_strip(1)` que irá detectar quando o flag `DEB_BUILD_OPTIONS=nostrip` está definido e sair silenciosamente.

As linhas 18-26 descrevem a regra `'build'` (e sua regra-filho `'build-stamp'`), que executa o `make` com a o Makefile próprio do programa para compila-lo. Vamos falar sobre o comentado exemplo `docbook-to-man` mais tarde em `'manpage.1.ex, manpage.sgml.ex'` on page 28.

A regra `'clean'`, como especificado nas linhas 28-36, limpa qualquer binário não mais necessário ou coisas geradas automaticamente, deixadas após a construção do pacote. Esta regra deve funcionar todas as vezes (mesmo quando a árvore do código-fonte *is cleaned up!*), portanto use as opções para forçar (ex: para o `rm`, é `'-f'`), ou faça o `make` ignorar valores de retorno (falhas) utilizando um `'-'` na frente do nome do comando.

O processo de instalação, a regra `'install'`, começa na linha 38. Basicamente, ela executa a regra `'install'` do Makefile do programa, mas instala ele no diretório `$(CURDIR)/debian/gentoo` - por isso especificamos `$(DESTDIR)` como a raiz do diretório de instalação no Makefile do `gentoo`.

Como os comentários sugerem, a regra `'binary-indep'`, na linha 48, é usada para construir pacotes independentes de arquitetura. Como nós não temos nenhum, nada será feito aqui.

Na próxima regra - `'binary-arch'`, nas linhas 52-79, nas quais nós executamos muitos utilitários pequenos do pacote `debhelper` para executar várias operações nos arquivos do seu pacote para fazer o pacote concordante com o Debian-Policy.

Se o seu pacote é um `'Architecture: all'`, você precisa incluir todos os comandos para construir o pacote na regra `'binary-indep'`, e deixar a regra `'binary-arch'` vazia.

Os nomes dos programas do debhelper começam com 'dh_', e o resto é a descrição do que o utilitário faz em particular. É bastante auto-explicativo, mas eis algumas explicações adicionais:

- `dh_testdir(1)` confere se você está no diretório correto (ex: o diretório superior do código-fonte),
- `dh_testroot(1)` confere se você tem permissões de root necessárias para os alvos 'binary-arch', 'binary-indep' e 'clean',
- `dh_installman(1)` copiará os manuais para os devidos lugares no diretório de destino. Você só precisa dizer onde eles estão, relativamente ao diretório superior do código-fonte,
- `dh_strip(1)` retira as tabelas de símbolos dos arquivos executáveis e bibliotecas, para torná-los menores.
- `dh_compress(1)` compacta os manuais e documentações maiores que 4kb com o `gzip(1)`,
- `dh_installdeb(1)` copia os arquivos relacionados com o pacote (ex: os scripts de mantenedor) para o diretório `debian/gentoo/DEBIAN`,
- `dh_shlibdeps(1)` calcula as dependências de bibliotecas compartilhadas das bibliotecas e arquivos executáveis.
- `dh_gencontrol(1)` instala uma versão com ajuste-fino do arquivo de controle em `debian/gentoo/DEBIAN`,
- `dh_md5sums(1)` gera um MD5 checksum para todos os arquivos do pacote.

Para informações mais completas sobre o que todos esses scripts `dh_*` fazem, e quais são suas opções, por favor leia seus respectivos manuais. Existem alguns outros (possivelmente muito úteis) scripts `dh_*` que não são mencionados aqui. Se você precisar deles, leia a documentação do debhelper.

A seção `binary-arch` é onde você realmente deve comentar ou remover quaisquer linhas que chamem funcionalidades que você não precisa. Para o `gentoo`, eu comentarei as linhas sobre exemplos, `cron`, `init`, `man` e `info`, simplesmente porque o `gentoo` não precisa delas. Além disso, na linha 68, eu irei substituir 'Changelog' com 'FIXES', porque esse é o nome real do arquivo de changelog superior.

As últimas duas linhas (junto com quaisquer outras não explicadas) são somente algumas coisas mais ou menos necessárias, considerando que você pode ler o manual do `make`, e o Debian-Policy. Por enquanto, elas não são importantes de se conhecer.

Capítulo 5

Outros arquivo no debian/

Você pode notar que existem vários outros arquivos no subdiretório `debian/`, a maioria deles com o sufixo `.ex`, o que significa que eles são exemplos. Dê uma olhada em todos eles. Se você deseja, ou precisa, utilizar alguma de suas funcionalidades,

- olhe também sua documentação (dica: leia o `Debian-Policy`),
- se necessário, modifique os arquivos para satisfazer suas necessidades,
- renomeie-os para remover o sufixo `.ex`, se eles o tiverem,
- modifique o arquivo `rules`, se necessário.

Alguns desses arquivos, os mais utilizados, são explicados nas seções seguintes:

5.1 README.Debian

Quaiquer outros detalhes ou discrepâncias entre o pacote original e a sua versão debianizada deve ser documentada aqui.

O `dh_make` criou um arquivo padrão, como seguinte:

```
gentoo for Debian  
-----
```

```
<notas possíveis acerca deste pacote - se nenhuma, apague este arquivo.>
```

```
-- Josip Rodin <joy-mg@debian.org>, Wed, 11 Nov 1998 21:02:14 +0100
```

Como não temos nada para colocar aqui, apagaremos o arquivo.

5.2 `conffiles.ex`

Uma das coisas mais irritantes em software é quando você gasta muito tempo e esforço personalizando o seu programa, e uma nova versão dele desfaz todas as suas mudanças. O Debian resolve esse problema marcando os arquivos de configurações de forma que, quando você atualiza um pacote, você será perguntado se quer manter sua configuração antiga ou não.

O modo de fazer isso num pacote é colocando o caminho completo de cada arquivo de configuração (normalmente em `/etc`), um por linha, num arquivo chamado `conffiles`. O gentoo tem um arquivo de configuração, `/etc/gentoorc`, e nós vamos adicioná-lo no arquivo `conffiles`.

Se o seu programa utiliza arquivos de configuração mas também os sobrescreve por conta própria, é melhor não marcá-los nos 'conffiles' pois senão o `dpkg` irá perguntar aos usuários se eles querem verificar as mudanças o tempo todo.

Se o programa que você está empacotando precisa que todo usuário modifique o arquivo para funcionar, considere também não marcar o arquivo no 'conffiles'.

Você pode utilizar exemplos de arquivos de configuração dos 'scripts de mantenedor'. Para maiores informações veja 'postinst.ex, preinst.ex, postrm.ex, prerm.ex' on page 30.

Se o seu programa não tem arquivos de configurações, você pode apagar o arquivo `conffiles` do diretório `debian/`.

5.3 `cron.d.ex`

Se o seu pacote precisa agendar tarefas regularmente para funcionar adequadamente, você pode usar este arquivo para definir isso.

Note que isso não inclui rotação de logs; para isso, veja `dh_installlogrotate(1)` e `logrotate(8)`.

Se não, remova este arquivo.

5.4 `dirs`

Este arquivo especifica os diretórios que nós precisamos mas o processo de instalação normal (`make install`), por algum motivo, não cria.

Por padrão, ele é como seguinte:

```
usr/bin
usr/sbin
```

Note que a barra precedente não é incluída. Nós normalmente teríamos alterado para ficar como seguinte:

```
usr/bin
usr/man/man1
```

mas estes diretórios já são criados no Makefile, e portanto não precisamos deste arquivo e vamos então apagá-lo.

5.5 docs

Este arquivo especifica os nomes dos arquivos de documentação que o `dh_installdocs` irá instalar no diretório temporário para nós.

Por padrão, ele incluirá todos os arquivos existentes no topo do diretório-fonte que são chamados 'BUGS', 'README*', 'TODO', etc.

Para o gentoo, eu também incluí mais coisas:

```
BUGS
CONFIG-CHANGES
CREDITS
ONEWS
README
README.gtkrc
TODO
```

Nós podemos também remover este arquivo e listas estes arquivo no comando `dh_installdocs` do arquivo `rules`, como seguinte:

```
dh_installdocs BUGS CONFIG-CHANGES CREDITS ONEWS README \
               README.gtkrc TODO
```

Entretando, infelizmente você pode não ter arquivos deste tipo no código-fonte do seu pacote. Neste caso, você pode remover este arquivo. Mas não remova a invocação do `dh_installdocs` do arquivo `rules` pois ele é utilizado para instalar o arquivo de `copyright` e outras coisas.

5.6 emacs-*.ex

Se o seu pacote fornece arquivos do Emacs que pode ser 'bytecompilados' no momento da instalação do pacote, você pode utilizar estes arquivos para definir isso.

Eles são instalados no diretório temporário pelo `dh_installemacsen(1)`, portanto não se esqueça de descomentar essa linha no arquivo `rules` se você quer utilizar isso.

Se você não precisa destes, remova-os.

5.7 `init.d.ex`

Se o seu pacote é um daemon que precisa ser executado na inicialização do sistema, você obviamente desobedeceu minha recomendação inicial, não é mesmo? :-)

Isto é meramente um esqueleto genérico para um script `/etc/init.d/`, e você provavelmente terá de editá-lo muito. Ele é instalado no diretório temporário pelo `dh_installinit(1)`.

Se você não precisa disto, remova o arquivo.

5.8 `manpage.1.ex`, `manpage.sgml.ex`

Seu(s) programa(s) devem ter manuais. Se não têm, cada um destes arquivos são modelos que você pode preencher.

Manuais são normalmente escritos em `nroff(1)`. O exemplo `manpage.1.ex` é escrito em `nroff` também. Você o manual `man(7)` para uma breve descrição de como editar um destes arquivos.

Se, por outro lado, você prefere escrever em `sgml` ao invés de `nroff`, você pode usar o modelo `manpage.sgml.ex`. Se você for fazer isso, você tem de:

- instalar o pacote `docbook-to-man`
- adicionar o `docbook-to-man` á linha `Build-Depends` do arquivo `control`
- remover a o comentário da invocação do `docbook-to-man` na regra 'build' do seu arquivo `rules`

E lembre-se de renomear o arquivo para algo como `gentoo.sgml!`

O nome do arquivo final do manual deve incluir o nome do programa que está documentando, e portanto nós iremos renomeá-lo de 'manpage' para 'gentoo'. O nome do arquivo também inclui '.1' como o primeiro sufixo, o que significa que é um manual para um comando de usuário. Certifique-se de verificar que seção é, de fato, a correta. Eis uma pequena lista de seções de manuais.

Section	Description	Notes
1	Comandos de usuário	Comando executáveis ou scripts.
2	Chamadas de sistema	Funções fornecidas pelo kernel.
3	Chamadas de biblioteca	Funções dentro de bibliotecas do sistema.
4	Arquivos especiais	Normalmente encontrados em <code>/dev</code>
5	Formatos de arquivo	Ex: o formato do <code>/etc/passwd</code>
6	Jogos	Ou outros programas frívolos
7	Pacotes de Macro	Como macros.
8	Administração de sistema	Programas típicamente executados como <code>root</code>
9	Rotinas de Kernel	Chamadas internas e não-padronizadas.

Então o manual do gentoo deve ser chamado `gentoo.1`. Não existia um manual `gentoo.1` no fonte original, então eu o escrevi utilizando a informação do exemplo e da documentação.

5.9 menu.ex

Usuários de sistemas X-Window normalmente tem um manipulador de janelas com um menu que pode ser personalizado para executar programas. Se eles tiverem o pacote do Debian `menu`, um conjunto de menus para programa no sistema será criado para eles.

Eis o arquivo `menu.ex` padrão que o `dh_make` criou:

```
?package (gentoo) : needs=X11|text|vc|wm section=Apps/see-menu-manual\
  title="gentoo" command="/usr/bin/gentoo"
```

O primeiro campo após o caractere de dois-pontos é `'needs'`, e especifica que tipo de interface o programa precisa. Mude isto para uma das alternativas listadas, como `'text'` ou `'X11'`.

O seguinte é `'section'`, onde as entradas de menu e submenus devem aparecer. A lista atual de seções está em: `/usr/share/doc/debian-policy/menu-policy.html/ch2.html#s2.1`

O campo `'title'` é o nome do programa. Você pode começar este em caixa alta, se quiser, desde que seja curto.

Finalmente, o campo `'command'` é o comando que executa o programa.

Agora nós iremos mudar a entrada de menu para o seguinte:

```
?package (gentoo) : needs=X11 section=Apps/Tools title="Gentoo" command="gentoo"
```

Você pode também adicionar outros campos como `'logtitle'`, `'icon'`, `'hints'`, etc. Veja `menufile(5)`, `update-menus(1)` e `/usr/share/doc/debian-policy/menu-policy.html/` para maiores informações.

5.10 watch.ex

Este arquivo é utilizado para configurar os programas `uscan(1)` e `uupdate(1)` (no pacote `devscripts`). Estes são utilizados para monitorar o site de onde você pegou o código-fonte original.

Eis o que eu coloco nele:

```
# Arquivo de controle de monitoramento para o uscan
# Site                Diretório          Padrão                Versão
ftp.obsession.se     /gentoo                gentoo-(.*)\.tar\.gz
```

Dica: conecte à internet, e tente rodar o `'uscan'` no diretório do programa assim que você criar o arquivo. E leia os manuais! :)

5.11 ex.package.doc-base

Se o seu pacote tem outra documentação fora o manual e info, você deve utilizar o arquivo 'doc-base' para registrá-lo, de forma que o usuário possa encontrá-la com, por exemplo, `dhhelp(1)`, `dwww(1)` ou `doccentral(1)`.

Isto normalmente inclui arquivos HTML, PS ou PDF, distribuídos em `/usr/share/doc/`
`/nome_do_pacote/`

Eis como ficaria o arquivo do doc-base do gentoo:

```
Document: gentoo
Title: Manual do Gentoo
Author: Emil Brink
Abstract: Este manual descreve o que o Gentoo é, e como pode ser utilizado.
Section: Apps/Tools

Format: HTML
Index: /usr/share/doc/gentoo/html/index.html
Files: /usr/share/doc/gentoo/html/*.html
```

Para maiores informações sobre esse formato, veja `install-docs(8)` e o manual do doc-base, em `/usr/share/doc/doc-base/doc-base.html/`.

5.12 postinst.ex, preinst.ex, postrm.ex, prerm.ex

Estes arquivos são chamados scripts de manutenção. Eles são scripts que são postos na área de controle do pacote e executados pelo `dpkg` quando seu pacote é instalado, atualizado ou removido.

Por enquanto, você deve evitar qualquer edição manual dos scripts de manutenção se você puder, pois eles tendem a ficar complexos. Para maiores informações leia o capítulo 6 do Debian-Policy e analise estes arquivos de exemplo fornecidos pelo `dh_make`.

Capítulo 6

Construindo o pacote

Agora devemos estar prontos para construir o pacote.

6.1 Reconstrução completa

Mude para o diretório principal do programa e execute o seguinte comando:

```
dpkg-buildpackage -rfakeroot
```

Isto fará tudo para você. Ele irá:

- limpar a árvore do código-fonte (`debian/rules clean`), utilizando o `fakeroot`
- construir o pacote-fonte (`dpkg-source -b`)
- compilar o programa (`debian/rules build`)
- construir o pacote-binários (`debian/rules binary`), utilizando `fakeroot`
- validar o código-fonte arquivo `.dsc`, utilizando `gnupg`
- criar e validar o arquivo `.changes` para ser enviado, utilizando `dpkg-genchanges` e `gnupg`

A única entrada sua que será necessária será a sua frase-senha de GPG, duas vezes.

Feito tudo isso, você verá os seguintes arquivos no diretório acima (`~/debian/`):

- `gentoo_0.9.12.orig.tar.gz`

Este é o código-fonte original, compactado, meramente renomeado para o acima de forma a seguir o padrão Debian. Note que este foi criado utilizando a opção `'-f'` no `dh_make` quando nós o executamos inicialmente.

- *gentoo_0.9.12-1.dsc*

Este é o resumo do conteúdo do código-fonte. O arquivo é gerado do seu arquivo 'control', e será utilizado quando o código-fonte for desempacotado com o `dpkg-source(1)`. Este arquivo é validado com o PGP, de forma que as pessoas podem ter certeza que ele é realmente o seu.

- *gentoo_0.9.12-1.diff.gz*

Este arquivo compactado contém cada uma das modificações que você fez no código-fonte, no formato conhecido como "diff unificado". Ele é feito e utilizado pelo `dpkg-source(1)`. **ATENÇÃO:** se você não nomear o pacote compactado (tarball) original como `nomedopacote_versao.orig.tar.gz`, o `dpkg-source` irá falhar na geração do arquivo `.diff.gz`!

Se alguém quiser recriar o seu pacote desde o início, eles podem facilmente fazê-lo utilizando os três arquivos acima. O procedimento de extração é trivial: copie os três arquivos em algum lugar e execute `dpkg-source -x gentoo_0.9.12-1.dsc`.

- *gentoo_0.9.12-1_i386.deb*

Este é o seu pacote binário completo. Você pode utilizar o `dpkg` para instalá-lo e removê-lo como qualquer outro pacote.

- *gentoo_0.9.12-1_i386.changes*

Este arquivo descreve todas as modificações feitas na revisão atual do pacote, e é utilizado pelo programa de manutenção do arquivo FTP da Debian para instalar os pacotes binário e fonte. Ele é parcialmente gerado a partir do arquivo 'changelog' e do arquivo '.dsc'. Este arquivo é validado utilizando PGP, de forma que as pessoas podem ter certeza que ele é realmente seu.

A medida que você for trabalhando no pacote, o seu comportamento irá mudar e novas funcionalidades serão adicionadas. As pessoas que baixarem o seu pacote poderão olhar neste arquivo e rapidamente ver o que foi alterado. Os programas de manutenção do arquivo da Debian também irão enviar o conteúdo deste arquivo para a lista `debian-devel-changes`.

Os longos strings numéricos nos arquivos `.dsc` e `.changes` são as validações MD5 para os arquivos mencionados. Uma pessoa que baixar os seus arquivos podem testá-los com `md5sum(1)` e se os números não coincidirem, eles saberão que o arquivo está corrompido ou que alguém mexeu nele sem autorização.

6.2 Reconstrução rápida

Quando o pacote for grande, você pode não querer reconstruí-lo completamente toda vez que você mudar alguma coisa no arquivo `debian/rules`. Para fins de testes, você pode criar um arquivo `.deb` sem ter de reconstruir todo o código e pacote-fonte da seguinte forma:

```
fakeroot debian/rules binary
```

Assim que você terminar suas modificações, lembre-se de reconstruir o pacote como descrito anteriormente, o procedimento correto. Você pode não conseguir enviá-lo corretamente se você simplesmente enviar os arquivos .deb construídos dessa forma.

Capítulo 7

Procurando por erros no pacote

Execute o `lintian(1)` no seu arquivo `.changes`; este programa irá verificar o pacote procurando por uma série de erros comuns de empacotamento. O comando é:

```
lintian -i gentoo_0.9.12-1_i386.changes
```

Obviamente você deve substituir o nome do arquivo `.changes` pelo gerado para o seu pacote. Se forem mostrados erros (linhas começadas com E:), leia a descrição (as linhas N:), corrija os erros, e reconstrua o pacote como descrito em ‘Reconstrução completa’ on page 31. Se aparecerem linhas começadas com W:, são advertências, e você deve “afinar” o seu pacote ou certificar-se que as advertências são desnecessárias (e fazer objeções no Lintian; veja a documentação para mais detalhes.).

Note que você pode construir o pacote com o `dpkg-buildpackage` e executar o `lintian` ao mesmo tempo com um só comando com `debuild(1)`.

Olhe dentro do pacote utilizando um manipulador de arquivos como o `mc(1)`, ou `desempacote-o` num diretório temporário utilizando o `dpkg-deb(1)`. Procure por arquivos desnecessários, tanto no pacote binário quanto no fonte. Muitas vezes o lixo não é eliminado corretamente; ajuste o seu arquivo `rules` para compensar isso. Dicas: `'zgrep ^+++ ../gentoo_0.9.12-1.dif.gz'` lhe dará uma lista de modificações/adições nos arquivos-fonte, e `'dpkg-deb -c gentoo_0.9.12-1_i386.deb'` irá listar os arquivos no pacote binário.

Instale o pacote e teste-o, ex: utilizando o comando `debi(1)` como `root`. Tente instalar e executa-lo em outras máquinas diferentes e observe com atenção as advertências e/ou erros durante a instalação e execução.

Capítulo 8

Enviando o pacote

Agora que você já testou completamente o seu novo pacote, você está pronto para começar a preencher o formulário de novo mantenedor do Debian, descrito em <http://www.debian.org/devel/join/newmaint>

Assim que você se tornar um desenvolvedor oficial, você precisará enviar o pacote para o arquivo da Debian. Você pode fazer isso manualmente, mas é mais fácil usar as ferramentas de automatização fornecidos, como o `dupload(1)` ou `dput(1)`. Vamos descrever como é feito com o `dupload`.

Primeiro você tem de configurar o `dupload`. Você pode tanto editar o arquivo de configuração global `/etc/dupload.conf`, ou ter o seu próprio arquivo `~/dupload.conf` com as configurações que você deseja personalizar. Coloque algo como o seguinte neste arquivo:

```
package config;

$default_host = "ftp-master";

$cfg{"ftp-master"}{"login"} = "seunomedeusuariodebian";

$cfg{"non-us"}{"login"} = "seunomedeusuariodebian";

1;
```

Obviamente você deve substituir minhas configurações pessoais pelas suas, e ler o manual `dupload.conf(5)` para entender o que cada uma dessas opções significa.

A opção `$default_host` é a que precisa de mais atenção – ela determina qual file de upload será usada como padrão. “ftp-master” é a primária, mas é possível que você queira usar outra mais rápida. Para mais informações sobre files de upload, leia a Referência do Desenvolvedor (Developer’s Reference), na seção “Uploading a package”, em `/usr/share/doc/developers-reference/developers-reference.html/ch-upload.en.html#s-uploading`

Conecte então ao seu provedor de internet e execute o comando:

```
dupload gentoo_0.9.12-1_i386.changes
```

O `dupload` verifica se as validações MD5 dos arquivos coincidem com aquelas do arquivo `.changes`, e irá advertí-lo a reconstruir o pacote como descrito em ‘Reconstrução completa’ on page 31, caso não coincidam, para que o pacote possa ser devidamente enviado.

Se você enviar para o “ftp-master”, o `dupload` irá pedir pela sua senha nas máquinas da Debian, e então irá enviar os pacotes.

Capítulo 9

Atualizando o pacote

9.1 Nova revisão Debian

Digamos que um relatório de bug foi preenchido contra o seu pacote, #54321, e descreve um problema que você pode solucionar. Para criar uma nova revisão de pacote Debian você precisa:

- Corrigir o problema no pacote-fonte, é claro.
- Adicionar uma nova revisão no topo do arquivo de changelog, por exemplo com `"dch -i"`, ou explicitamente com `"dch -v <versão>-<revisão>"` e então inserir os comentários utilizando o seu editor de textos favorito.

Dica: Como obter a data no formato adequado? Use `'822-date'`, ou `'date -R'`.

- Inclua uma breve descrição do bug e sua solução na entrada do changelog, seguido por `"Closes: #54321"`. Dessa forma, o relatório de bug será "automagicamente" fechado pelo software de manutenção do arquivo no momento em que seu pacote for aceito no arquivo Debian.
- Repita o que você fez em 'Reconstrução completa' on page 31, 'Procurando por erros no pacote' on page 35, e 'Enviando o pacote' on page 37. A diferença é que, desta vez, o arquivo-fonte não será incluído, uma vez que ele não mudou e já existe no arquivo da Debian.

9.2 Nova distribuição do programa

Agora vamos considerar uma situação diferente, um pouco mais complicada - uma nova versão do software foi lançada, e obviamente você a quer empacotada. Você precisa fazer o seguinte:

- Baixar o novo código-fonte e colocar o arquivo compactado (tarball) (ex: chamado 'gentoo-0.9.13.tar.gz') no diretório acima da árvore do código fonte antigo (ex: ~/debian/).
- Entre no diretório do código-fonte antigo e execute:

```
uupdate -u gentoo-0.9.13.tar.gz
```

Obviamente, substitua esse nome de arquivo com o nome do seu novo código-fonte do programa. O `uupdate(1)` irá renomear o arquivo devidamente, tentar aplicar todas as diferenças do arquivo `.diff.gz` antigo e atualizar o novo arquivo `debian/changelog`.

- Mude para o diretório `'../gentoo-0.9.14'`, a nova árvore de código-fonte, e repita o que você fez em 'Reconstrução completa' on page 31, 'Procurando por erros no pacote' on page 35, e 'Enviando o pacote' on page 37.

Note que se você definiu um arquivo `'debian/watch'` como descrito em `'watch.ex'` on page 29, você pode executar o `uscan(1)` para "automagicamente" procurar por novos códigos-fonte, baixar eles, e executar o `uupdate`.

9.3 Verificando atualizações de pacotes

Quando você constrói uma nova versão de um pacote, você deve fazer o seguinte para verificar que o pacote pode ser atualizado de forma segura:

- atualizar o pacote antigo
- reinstalar o pacote antigo, e então removê-lo
- instalar o novo pacote
- removê-lo e o reinstalar novamente,
- executar um "purge" nele.

Tenha em mente que se o seu pacote já foi previamente distribuído no Debian, as pessoas irão atualizar para o seu pacote a partir da última versão do Debian com frequência. Lembre-se de testar as atualizações a partir dessas versões também.

Capítulo 10

Onde pedir ajuda

Antes de decidir pedir ajuda em algum lugar público, por favor leia toda a documentação disponível. Isso inclui a os arquivos em `/usr/share/doc/dpkg`, `/usr/share/doc/debian`, `/usr/share/doc/package/*`, os manuais e infos de todos os programas mencionados nesse documento.

Se você tem perguntas sobre o empacotamento e não pode encontrar respostas na documentação, você pode fazê-las na lista Debian Mentors em `<debian-mentors@lists.debian.org>`. Os desenvolvedores Debian mais experientes ficarão felizes em poder ajudar, mas leia a documentação antes de fazer perguntas!

Veja <http://lists.debian.org/debian-mentors/> para maiores informações sobre essa lista.

Quando você receber relatórios de bugs (sim, relatórios de bugs de verdade!), você saberá que é hora de entrar no Debian Bug Tracking System (<http://www.debian.org/Bugs/>) e ler a documentação de lá, para poder lidar com os relatórios eficientemente. Eu recomendo fortemente que você leia a Referência do Desenvolvedor (Developers Reference) no capítulo “Tratando de Bugs” (Handling Bugs”, em `/usr/share/doc/developers-reference/developers-reference.html/ch-bug-handling.en.html`

Se você ainda tem perguntas, faça-as na lista do Debian Developers em `<debian-devel@lists.debian.org>`. Veja <http://lists.debian.org/debian-devel/> para maiores informações sobre esta lista.

Mesmo se você fez um bom trabalho, é hora de começar a rezar. Porque? Porque em poucas horas (ou dias) os usuários de todo o mundo começarão a usar o seu pacote, e se você cometeu algum erro grave será bombardeado por milhares de usuários Debian irritados... Brincadeira! :-)

Relaxe e se prepare para relatórios de bugs, pois ainda tem muito trabalho a ser feito antes que seu pacote esteja completamente de acordo com a política do Debian (novamente, leia a *documentação de verdade* para detalhes.). Boa sorte!