Empacotamento de Software

no

Debian GNU/Linux

4ª edição, revisada em abril de 2013

- João Eriberto Mota Filho -

http://debianet.com.br

Sumário

Introdução

O empacotamento Debian é uma tarefa complexa e precisa, que deve ser estudada e treinada exaustivamente para que haja pleno êxito.

Praticamente tudo poderá ser empacotado. Programas, documentações, scripts, arquivos de configuração etc. É possível, por exemplo, criar um pacote que fará a instalação e a configuração completa de um servidor automaticamente.

Esta apostila tem por objetivo auxiliar as pessoas que desejam aprender a criar pacotes Debian Binary (.deb). É importante ressaltar que este trabalho não substitui os documentos oficiais do Projeto Debian que versam sobre o assunto (Debian Policy Manual, Debian Developer's Reference e Guia do Novo Mantenedor Debian, todos disponíveis em http://www.debian.org/doc). Assim sendo, é extremamente recomendável completa a leitura dos citados documentos.

Ainda, este documento foi elaborado pela primeira vez em 2006 e não teria sido escrito sem a valiosa ajuda de dois grandes amigos: Goedson Teixeira Paixão e do Felipe Augusto van de Wiel (faw). Agradeço imensamente pela boa vontade em sempre ajudar e pela seriedade no trabalho realizado em prol do Debian GNU/Linux.

Esta apostila foi originalmente criada por João Eriberto Mota Filho, em 2006, em BrOffice.Org. Atualizações em 2007 e 2013, este último em LibreOffice.

"O Debian só fica

pronto quanto está pronto!"

Capítulo 1 - Definições Gerais

Este capítulo irá mostrar algumas definições importantes, que deverão ser conhecidas por quem mantém pacotes no Debian GNU/Linux.

1.1 As pessoas

- Empacotador: pessoa que mantém pacotes no Debian mas não pertence, oficialmente, ao Projeto Debian.
- Mantenedor Debian (Debian Maintainer): pessoa que mantém pacotes no Debian e pode realizar algumas pequenas ações dentro do Projeto Debian, como o upload dos seus pacotes. Referência: `http://wiki.debian.org/DebianMaintainer`.
- Desenvolvedor Debian (Debian Developer ou DD): pessoa que mantém pacotes no Debian e que pertence, oficialmente, ao Projeto Debian GNU/Linux. A sua admissão no projeto é feita considerando-se as suas contribuições, as indicações por parte de outros DDs e os resultados em provas teóricas e práticas. Referência: `http://wiki.debian.org/DebianDeveloper`.
- Sponsor: DD que faz o upload dos pacotes de um mantenedor.
- Upstream: pessoa que desenvolveu um determinado Software Livre que está disponível na Internet.
- Uploader: atualmente, utilizado para referir-se aos demais mantenedores de um pacote (o principal é o maintainer).
- Líder: DD eleito por outros DDs para liderar o Debian por 1 ano.
- FTP-Master: equipe que controla a entrada de pacotes no Debian. O seu site é `http://ftp-master.debian.org`.

1.2 Os mecanismos

- BTS (Bug Track System): sistema de tratamento de bugs. O seu site é `http://bugs.debian.org`.
- New queue: fila de entrada de pacotes inéditos. Depois de serem analisados pelo FTP-Master, os pacotes vão para a fila incoming. O seu site é `http://ftp-master.debian.org/new.html`.
- Incoming: fila de entrada de pacotes que não são inéditos. Nesta fila, os pacotes aguardam o dinstall. O seu site é `http://incoming.debian.org`.

- dinstall: processo cíclico que retira os pacotes existentes na incoming e os distribui nas diversas arquiteturas para a sua compilação e instalação. O tempo restante para o próximo dinstall poderá ser visto em `http://people.debian.org/~joerg/dinstall.html`.
- O debconf: sistema de instalação do Debian.
- DebHelper - Conjunto de programas utilizados no Debian para gerar arquivos `.deb`. Este é apenas um dos mecanismos possíveis para gerar pacotes para o Debian.

1.3 Geral

- WNPP - Work-Needing and Prospective Packages. É a lista de pacotes que necessitam de novos mantenedores, de ajuda ou de pacotes que estão em fase de criação. Disponível em `http://www.debian.org/devel/wnpp`.
- A DebConf: reunião anual dos desenvolvedores Debian. O seu site é `http://www.debconf.org`.

Capítulo 2 - Licenças em Software Livre

2.1 A importância da licença

Existem várias licenças no mundo do Software Livre. Ao utilizar ou empacotar um programa, é necessário ler a sua licença para saber se tal programa possui alguma restrição ao empacotamento, à redistribuição etc.

2.2 O que é uma licença?

Licença é um contrato entre o desenvolvedor e o usuário de um software. A partir do momento em que um usuário usa o software, ele está automaticamente aceitando o contrato (licença) daquele software. Isso faz com que a GPL, por exemplo, tenha validade no Brasil.

2.3 Qual a relação entre a licença e o copyright?

Copyright é um termo que refere-se ao direito autoral de algum trabalho. Um programa licenciado pela GNU GPL pode ser redistribuído, modificado etc. No entanto, o autor do trabalho original sempre deverá ser citado, pois o mesmo detém o direito autoral sobre esse trabalho.

2.4 O copyleft

O copyright pressupõe que nenhum trabalho poderá ser copiado ou redistribuído. Assim sendo, a FSF criou o copyleft, que quer dizer algo como “copie e modifique à vontade, mas não deixe de citar o autor original”.

As licenças GNU GPL e BSD modificada, por exemplo, possuem características de copyleft. Mas é importante ressaltar que mesmo que um software seja copyleft, não desaparecem os direitos autorais.

2.5 Algumas licenças

A seguir, alguns exemplos de licenças muito utilizadas e o seu respectivo site:

- GPL 3 (2007): `http://www.gnu.org/licenses/gpl.html`
- LGPL 3 (2007): `http://www.gnu.org/licenses/lgpl.html`
- MIT (1985): `http://www.opensource.org/licenses/mit-license.html`
- Creative Commons: `http://creativecommons.org/licenses`

Uma lista comentada de licenças poderá ser obtida em http://www.gnu.org/licenses/license-list.html. Cabe ressaltar que um software poderá estar licenciado por mais de uma licença (cada parte por um licença) ou por uma licença desenvolvida pelo próprio autor. Também é interessante ressaltar que quem escolhe a licença de um software é o seu autor. Um exemplo: a licença GNU GPL atual está na versão 7, desde 2007; no entanto, caso o autor de um software queira, poderá usar a versão 2.

2.6 A DFSG (Debian Free Software Guidelines)

A DFSG é uma norma do Debian que estabelece quando um software será considerado livre ou não, uma vez que o Debian somente aceita softwares totalmente livres na sua seção principal (Main). A DFSG está disponível em http://www.debian.org/social_contract e baseia-se nos seguintes tópicos: redistribuição livre, código fonte, trabalhos derivados, integridade do código fonte do autor, não à discriminação contra pessoas ou grupos, distribuição de licença, a licença não pode ser específica para o Debian e a licença não deve contaminar outros softwares. Ainda, a DFSG deixa claro que, dentre outras, a GPL, a BSD e a Artistic são licenças livres para o Debian.

Esclarecimentos importantes sobre diversas licenças poderão ser obtidos em http://wiki.debian.org/DFSGLicenses.

"Nossas prioridades são o usuário e o Software Livre."

Capítulo 3 - Filesystem Hierarchy Standard (FHS)

3.1 O que é a FHS?

A FHS é uma tentativa de padronização de diretórios em Unix e derivados. A FHS é um documento simples e fácil de ser consultado. Lá estão definidas algumas questões comuns, como a diferença entre o `/tmp` e o `/var/tmp`, por exemplo.

3.2 Onde encontro a FHS mais atual?

A FHS poderá ser encontrada em `http://www.pathname.com/fhs`.

3.3 Por que não devo contrariar a FHS ao empacotar para o Debian?

O Debian segue rigidamente a FHS e, ao contrariar a mesma, um pacote será marcado como tendo um bug do nível “serious” (qualquer violação da Debian Policy será marcada como serious).

Capítulo 4 - Criação e aplicação de patches

4.1 Patches

Patch é um conjunto de linhas de código que representa as diferenças entre dois arquivos ou diretórios não binários. Um patch é criado com o comando *diff* e aplicado com o comando *patch*. Uma aplicação usual para o patch é a sugestão para a correção de erros em códigos de programas alheios. Geralmente, fazemos o patch e enviamos para o programador para que ele receba apenas as alterações necessárias.

Considere o conteúdo do arquivo *1.txt*:

```
cebola
laranja
TOMATE
alho
```

Considere, agora, o conteúdo do arquivo *2.txt*:

```
cebola
laranja
tomate
alho
```

A seguir, o patch que deverá ser aplicado em *1.txt* para chegarmos a *2.txt*:

```
--- 1.txt       2007-06-14 09:08:50.000000000 -0300
+++ 2.txt       2007-06-14 09:09:10.000000000 -0300
@@ -1,4 +1,4 @@
 cebola
 laranja
-TOMATE
+tomate
 alho
```

Observe que as duas primeiras linhas referem-se aos arquivos envolvidos: *1.txt* (inicial) e *2.txt* (final). As linhas marcadas com "-" e "+" representam a(s) linha(s) subtraída(s) e a(s) linha(s) adicionada(s).

4.2 O comando diff

O comando diff é utilizado para mostrar a diferença entre dois arquivos ou diretórios. Essa diferença pode ser utilizada como um patch.

Considere um determinado diretório. Caso você deseje alterar o conteúdo desse diretório e gerar um patch, utilize os seguintes procedimentos:

- Faça uma cópia do diretório, renomeando-o com um *.orig* no final. Exemplo:

```
# cp -a diretorio diretorio.orig
```

- Edite o conteúdo do "*diretorio*", deixando o "*diretorio.orig*" inalterado.
- Crie um arquivo diff, do diretório original para o modificado, com o comando:
  ```
  # diff –Naur diretorio.orig diretorio > diretorio.patch
  ```

  Para ver o que significam as opções *N*, *a*, *u* e *r*, use o comando # *man diff*.

- Dentro do arquivo *diretorio.patch*, com já foi dito, as linhas subtraídas iniciarão com um sinal "-" e as adicionadas com "+". As linhas sem sinal servem apenas para balizar o conteúdo do patch, mostrando o que havia antes e depois das modificações (3 linhas antes e 3 linhas depois, sempre que possível).

4.3 O comando patch

O comando *patch* é utilizado para aplicar um patch, gerado pelo comando *diff*, em algum arquivo ou diretório. A opção *-p* serve para dizer qual será o nível de diretórios utilizado no "patcheamento" (ver # *man patch*). Geralmente, o patch é colocado dentro do diretório do programa em questão (no 1º nível) e, nesse caso, é utilizada a opção *-p1* (para remover a barra inicial).

Exemplo de utilização do comando *patch* (o arquivo de patch deverá estar dentro do diretório do código fonte e o comando deverá ser emitido naquele ponto):

```
# patch –p1 < diretorio.patch
```

"Integridade do código fonte do autor."

Capítulo 5 - PGP e GnuPG

5.1 O PGP (Pretty Good Privacy)

O Pretty Good Privacy (Privacidade Muito Boa) é um sistema de criptografia assimétrica e assinatura digital desenvolvido em 1991 por Philip Zimmermann. No início, o PGP possuia restrições quanto ao uso. Assim sendo, o Projeto GNU desenvolveu o GnuPG (GNU Privacy Guard), que é totalmente compatível com o PGP. O GnuPG segue o protocolo OpenPGP, definido pela RFC 2440. O GnuPG será utilizado para assinar os pacotes criados para o Debian.

O site do PGP é o *http://www.pgp.com*. O site do GnuPG é o *http://www.gnupg.org*.

5.2 Instalação

O GnuPG poderá ser instalado com o comando # *apt-get install gnupg*.

5.3 O chaveiro digital

O chaveiro digital de um usuário é a reunião de todas as chaves de interesse de tal usuário. Esse chaveiro, geralmente, contém o par de chaves do usuário e as chaves públicas de outros usuários. O chaveiro digital de cada usuário fica armazenado em ~/*.gnupg* e pode ser visto com o comando *$ gpg --list-keys*.

5.4 A criação e a manipulação das chaves

A criação das chaves poderá ser feita com o comando *$ gpg --gen-key*. Para maiores informações sobre a criação e a manipulação de chaves PGP (GPG), consulte o site *http://www.eriberto.pro.br/wiki/index.php?title=Usando_o_GnuPG*.

5.5 Uso do PGP na debianização

Para assinar pacotes no Debian, o par de chaves PGP deverá estar instalado no chaveiro local. A assinatura se dará automaticamente no final do processo de empacotamento.

Capítulo 6 - Arquivos de instalação

6.1 Arquivos utilizados em instalações

No GNU/Linux, na maioria dos casos, os programas (principalmente os feitos em C) são compilados e instalados com a sequência ./*configure*, *make*, *make install*.

O *configure* é um arquivo que tem a finalidade de verificar se o sistema possui os requisitos mínimos para compilar o programa em questão e, muitas vezes, cria ou prepara o arquivo *Makefile*. O comando *make* irá procurar pelo arquivo *Makefile* para realizar a compilação do programa. O *make install* irá instalar o programa já compilado.

6.2 Os alvos do Makefile

O arquivo *Makefile* possui uma sintaxe própria e é capaz de realizar ações extremamente customizadas e poderosas. Apenas para ilustrar a sintaxe, as linhas que possuem comandos deverão iniciar com 2 tabs.

O *Makefile* trabalha com alvos. Os alvos são as subdivisões do arquivo e funcionam como subrotinas. Considere o alvo denominado “clean”. O comando # *make clean* irá executar todos os comandos existentes na subdivisão (alvo) que possui o nome *clean*.

É uma boa prática SEMPRE analisar o *Makefile* de um programa antes de iniciar a sua compilação, a fim de identificar as possibilidades para o comando *make*.

Os alvos mais utilizados por usuários são:

- *all*: realiza a compilação do programa. Caso o usuário digite somente # *make*, será considerado o comando # *make all*.
- *clean*: desfaz, total ou parcialmente, o processo de compilação, removendo os binários e objetos criados. Muitas vezes, é necessário utilizar # *make dist-clean* (quando existir) para um resultado mais eficiente.
- *dist-clean*: desfaz totalmente o processo de compilação, apagando todos os arquivos gerados. Funcionamento similar ao *clean*.
- *install*: instala o programa, copiando cada arquivo para o local adequado.
- *uninstall*: desinstala o programa, removendo todos os arquivos.

6.3 Exemplo simples de Makefile

A seguir, será mostrado o Makefile do programa HAPM, disponível em `http://hapm.sourceforge.net`.

```
# Makefile for hapm

CC              = gcc
SYSCONFDIR      = /etc/ha.d
INITDIR         = /etc/init.d
INSTALL         = cp
INSTALLDIR      = /usr/sbin
MANDIR          = /usr/share/man/man8
SRCS            = hapm.c

all: hapm

hapm:
        $(CC) $(SRCS) -o $@
        #
        # ------------------------------------
        # Run "# make install" to install hapm.
        # ------------------------------------
        #

install:
        @if test ! -e hapm; then \
                        echo -e '\nERROR: Run "# make" before "# make
install".\n'; \
                exit 1; \
        fi
        @if test  -e $(INSTALLDIR)/hapm; then \
                echo -e "\nERROR: hapm already installed. \
                         \nRun \"# make uninstall\" before reinstall.\n";
\
                 echo -e "(don't fotget to make a backup of the hapm.conf
file)" \
                exit 1; \
        fi
        @if test ! -d $(SYSCONFDIR); then \
                echo "creating $(SYSCONFDIR)"; \
                mkdir $(SYSCONFDIR); \
        fi

        $(INSTALL) hapm.conf $(SYSCONFDIR)/hapm.conf
        $(INSTALL) hapm $(INSTALLDIR)
        chmod 0755 $(INSTALLDIR)/hapm
        $(INSTALL) init.d/hapm $(INITDIR)/hapm
        chmod 0755 $(INITDIR)/hapm
        $(INSTALL) hapm.8 $(MANDIR)
        #
        # ---------------
        # hapm installed.
        # ---------------
        #

uninstall:
        @if test ! -e $(INSTALLDIR)/hapm; then \
```

```
                echo -e "\nERROR: hapm isn't installed.\n"; \
                exit 1; \
        fi
        rm -f $(INSTALLDIR)/hapm
        rm -f $(INITDIR)/hapm
        rm -rf $(SYSCONFDIR)/hapm.conf
        rm -f $(MANDIR)/hapm.8
        #
        # -----------------
        # hapm uninstalled.
        # -----------------
        #

clean:
        @if test -e hapm; then \
                rm -rf hapm; \
        fi
```

Dentro do *Makefile*, o caractere @ faz com que o comando não seja mostrado no momento da execução.

6.4 Como estudar Makefiles

Para entender melhor o *Makefile*, siga os seguintes procedimentos:

- Leia o manual oficial do *make*, disponível em `http://www.gnu.org/software/make/manual`.
- Estude os Makefiles de vários programas.
- Leia o site `http://www.opussoftware.com/tutorial/TutMakefile.htm`.

6.6 Automatizando a criação de projetos e instaladores

É possível utilizar ferramentas especiais para a criação automatizada de projetos e instaladores. São elas:

- *autoproject*: cria a estrutura básica de um projeto. Arquivos como changelog, licença etc. Detalhes em `http://directory.fsf.org/autoproject.html`.
- *autoconf*: cria o esqueleto do arquivo configure. Detalhes no site `http://www.gnu.org/software/autoconf`.
- *automake*: cria o esqueleto do arquivo Makefile. Detalhes no site `http://www.gnu.org/software/automake`.

O *autoconf* e o *automake* utilizam arquivos com regras para gerar os arquivos finais (*configure* e *Makefile*). Assim, a existência de arquivos com as extensões *.ac* (autoconf) e *.am* (automake) ou com o nome *aclocal.m4* denotam a utilização dos utilitários *autoconf* e *automake*.

O autoconf é extremamente versátil. O comando # *autoconf -h* mostrará diversas opções para a compilação e a instalação do programa em questão. Será possível, por exemplo, determinar a instalação em diretórios diferentes do previsto para o referido programa. **Isso será muito útil ao realizarmos o empacotamento no Debian GNU/Linux.**

Capítulo 7 - Pacotes no Debian

7.1 Releases Debian

O Debian, normalmente, disponibiliza as seguintes releases:

- unstable: contém os pacotes recém-chegados ao Debian. Não possui repositório de segurança, uma vez que as correções de segurança são feitas diretamente no repositório. É a release na qual os desenvolvedores mais trabalham.
- testing: contém pacotes oriundos da release unstable. Possui repositório de segurança. Dependendo do estágio, poderá ser utilizado em desktops.
- stable: contém pacotes na sua versão final, já sem bugs ou problemas de segurança. Próprio para produção e serviços críticos. Não recebe novos programas. Recebe apenas atualizações de segurança ou reparos de bugs críticos. Possui repositório de segurança.

7.2 Seções dos repositórios

Os repositórios são divididos em seções, sendo que a default nas instalações é a *main*. São as seguintes, a seções existentes:

- *main*: contém os pacotes principais da distribuição. É a seção default do arquivo */etc/apt/sources.list*. Todos os pacotes existentes tem que ser livres, de acordo com a DFSG.
- *contrib*: contém pacotes livres que dependem de outros não livres ou que necessitam de ferramentas não livres para serem compilados.
- *non-free*: contém pacotes com pequenas restrições na sua licença (quanto ao uso ou à redistribuição dos mesmos).

A seguir, um exemplo da sintaxe de chamada da seção principal de um repositório com a adição da seção *contrib* (dentro de */etc/apt/sources.list*):

```
deb http://ftp.us.debian.org/debian stable main contrib
```

7.3 Pacotes fonte

Os pacotes fonte dos arquivos .deb poderão ser baixados a partir do seguinte comando:

```
# apt-get source <pacote>
```

Após o download, o seguinte comando será utilizado, automaticamente, para gerar um diretório com o conteúdo “debianizado”:

```
# dpkg-source -x <pacote>.dsc
```

Para fazer o download de pacotes fonte é necessário ter um repositório fonte listado em */etc/apt/sources.list*. Os repositórios fonte são chamados com uma linha *deb-src*. Exemplo:

```
deb-src http://ftp.us.debian.org/debian etch main
```

7.4 Pacotes virtuais

Os pacotes virtuais não podem ser instalados. Eles servem para sanar dependências referentes a um grande número que pacotes que faça algo similar. Exemplo: caso um pacote necessite de um servidor http para funcionar, em vez de listarmos todos os http servers existentes nas dependências, poderemos utilizar o pacote virtual *httpd*. No entanto, pelo menos um servidor real deverá ser listado. Exemplo:

```
depends: lynx, apache2 | httpd
```

A linha anterior, responsável por listar as dependências de um pacote. diz o seguinte: “depende de *lynx* e de *apache2* ou de outro http server qualquer”. Caso haja algum servidor http instalado, a dependência *httpd* estará sanada. Caso não haja, o pacote *apache2* será instalado (uma vez que *httpd* é um pacote virtual e não pode ser instalado).

Os pacotes virtuais estão listados em `http://www.debian.org/doc/packaging-manuals/virtual-package-names-list.txt`.

Para saber quais pacotes são englobados por um pacote virtual, bastará executar:

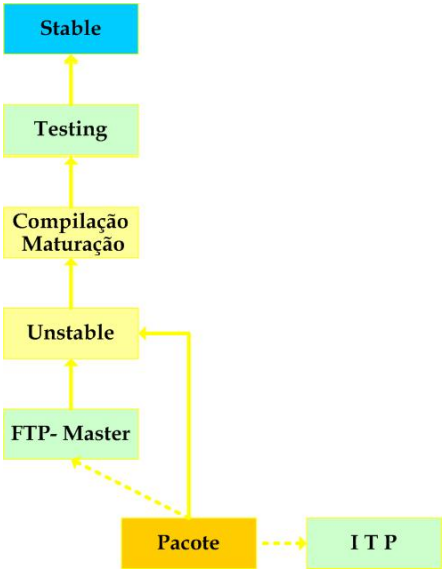
```
# apt-get install <nome do pacote virtual>
```

“O Debian permanecerá 100% livre.”

Capítulo 8 - Ciclo de vida do pacote

8.1 Ciclo de vida do pacote (versão resumida)

Todo pacote tem um ciclo de vida no Debian. Resumidamente, um pacote passa pelos seguintes estágios:

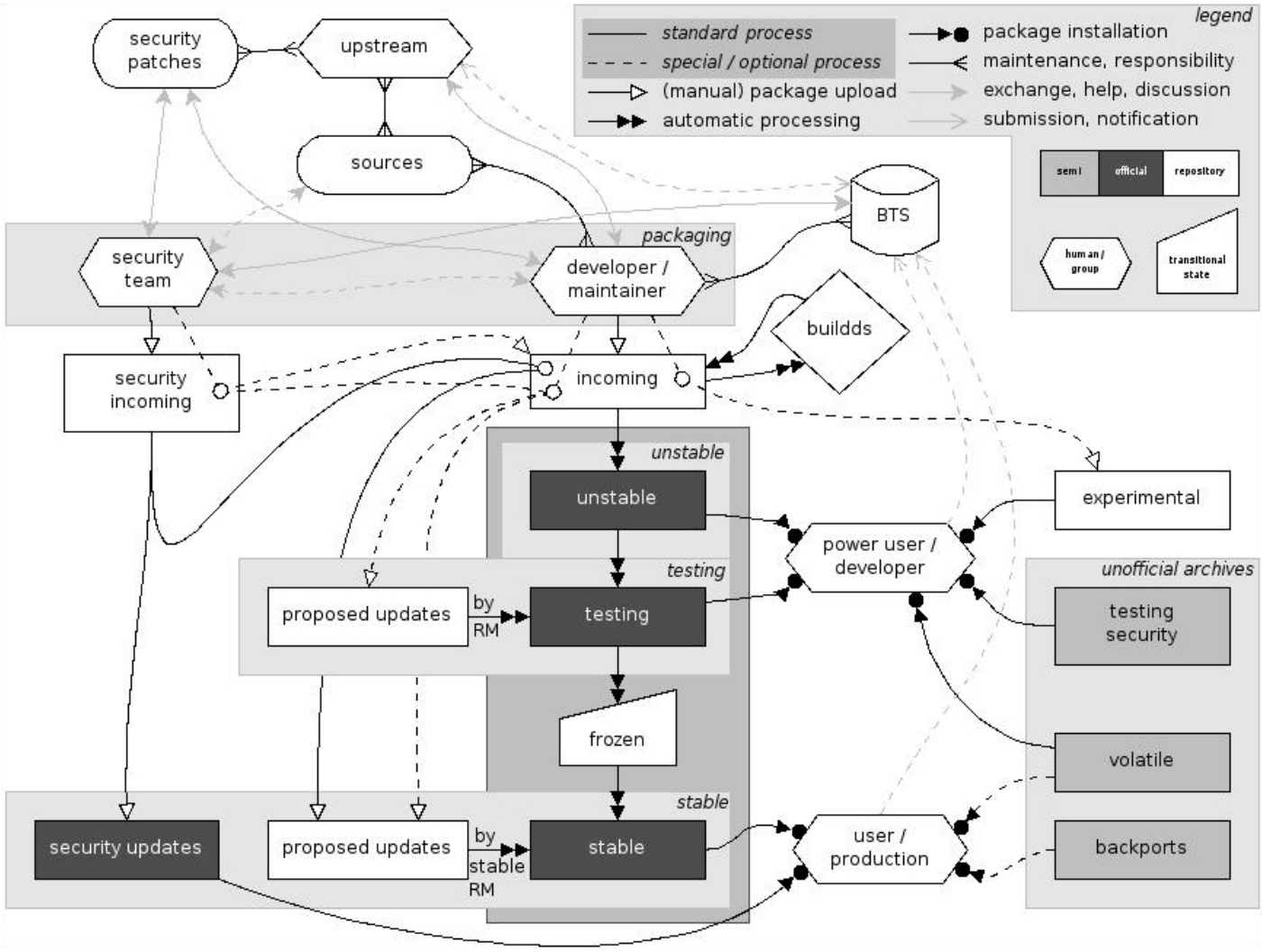


- Ao decidir por um novo empacotamento, o desenvolvedor gera um bug WNPP ITP (Work-Needing and Prospective Packages / Intent To Package).
- O primeiro upload do pacote segue para a New Queue para ser examinado e autorizado pelo FTP-Master. Os próximos uploads irão diretamente para a fila que leva à release unstable (incoming).
- Na unstable haverá a compilação em todas as arquiteturas. O pacote permanecerá alguns dias na unstable (de 2 a 10, dependendo da urgência) para poder ser testado por outros usuários.
- Após o tempo mínimo de permanência, caso não haja bugs RC (release critical), o pacote irá para a testing. Uma cópia do pacote permanece na unstable para manter a integridade da mesma.
- Na testing o pacote ficará esperando para, um dia, com todos os outros, ser promovido a stable. Nesse tempo de espera, uma nova versão do pacote, vinda da unstable, poderá substituir a versão atual.

8.2 Mapa geral

Um mapa mais amplo, feito no utilitário DIA, constantemente atualizado por Martin F. Krafft, está disponível em `http://people.debian.org/~madduck/talks/etch-delay_skycon_2007.02.18/package-cycle.png`.

A seguir, o mapa (atualizado em junho de 2007):



8.3 Mapa completo

Um mapa completo do ciclo de vida do pacote está disponível no endereço *http://eriberto.pro.br/debian*.

Capítulo 9 - Sites de interesse do desenvolvedor Debian

9.1 Canto dos desenvolvedores

O canto dos desenvolvedores é a página mais importante para quem quer manter pacotes .deb. O seu endereço é `http://www.debian.org/devel`. Essa página contém links para diversos documentos e recursos relevantes para desenvolvedores e mantenedores. Outra saída é entrar no site do Debian (`http://www.debian.org`) e clicar no link “Canto dos Desenvolvedores”.

9.2 Documentação oficial

O site `http://www.debian.org/doc` possui uma vasta documentação para desenvolvedores e usuários. Em “Manuais” é possível encontrar a subseção “Manuais do desenvolvedor”. Nesta seção, destacam-se os seguintes manuais:

- *Manual de Políticas Debian*: descreve as normas que levam à homologação do empacotamento.

  > O Manual de Política Debian possui algumas documentações suplementares como, por exemplo, a Política Java e as Políticas PHP. Há links para todos esses documentos em *http://www.debian.org/devel*.

- Referência do Desenvolvedor Debian: recomendações e ferramentas para desenvolvedores.
- Guia dos Novos Mantenedores Debian: passo-a-passo inicial para novos mantenedores.

Além disso, o Debian segue a FHS, disponível em `http://www.pathname.com/fhs`.

9.3 Segurança

O Manual de Segurança Debian trata do assunto com foco para usuários e desenvolvedores. Ele pode ser obtido em `http://www.debian.org/doc/user-manuals#securing`.

Outro endereço que deverá ser estudado pelo desenvolvedor `http://security.debian.org`.

9.4 Listas de discussão

O site `http://lists.debian.org` reúne as listas de discussão do Debian. Há várias listas voltadas para o desenvolvedor. Destacam-se:

- debian-devel
- debian-devel-portuguese
- debian-announce
- debian-news
- debian-mentors

> As listas também estão disponíveis via newsgroup por intermédio do servidor *news.gmane.org*.

9.5 Controle de pacotes

Os endereços a seguir são voltados para o controle dos pacotes no Debian:

- FTP-Master: `http://ftp-master.debian.org`
- Quality Assurance: `http://qa.debian.org`
- Package Track System (PTS): `http://packages.qa.debian.org`
- Busca por pacotes: `http://packages.debian.org`
- Bug track System (BTS): `http://bugs.debian.org`
- WNPP: `http://www.debian.org/devel/wnpp`
- Processamento de bugs: `http://www.debian.org/Bugs/Developer`
- Dinstall: `http://people.debian.org/~joerg/dinstall.html`
- Listas de pacotes: `http://packages.debian.org/<release>`

Capítulo 10 - Sistema de bugs no Debian (BTS)

10.1 O BTS (Bug Track System)

O BTS é responsável por controlar todos os bugs no Debian. A busca por um determinado bug, dentro do BTS, pode ser feita de várias formas diferentes. O site do BTS é `http://bugs.debian.org`.

10.2 reportbug

Sem dúvida, a melhor forma de reportar um bug é utilizando a ferramenta *reportbug*. Quando um bug é reportado, o máximo de informações relevantes deve ser fornecido. Assim, é recomendável executar a ferramenta *reportbug* na máquina que apresentou o problema para que alguns dados do hardware e de pacotes instalados sejam enviados juntamente com o bug.

O *reportbug* poderá ser instalado com o comando `# apt-get install reportbug`.

10.3 Comandos a serem emitidos para o servidor de bugs

É possível controlar o servidor de bugs via e-mail. Para isso, consulte os seguintes endereços:

- `http://www.debian.org/Bugs/server-control`
- `http://www.debian.org/Bugs/server-request`
- `http://www.debian.org/Bugs/server-refcard`

10.4 Manipulação e níveis de severidade em bugs

Um excelente manual para aprender como classificar e manipular bugs está disponível em `http://www.debian.org/Bugs/Developer`. Esse é um site de leitura obrigatória sempre que houver a necessidade de tratar um bug.

"Nós não esconderemos problemas."

Capítulo 11 - Preliminares no empacotamento

11.1 O pacote já existe?

Uma medida preliminar no empacotamento é verificar se o pacote já existe oficialmente dentro do Debian. Para isso, os seguintes endereços deverão ser consultados (todos poderão ser acessados a partir dos links `As pessoas` e `Pacotes que precisam de ajuda`, no Canto dos Desenvolvedores):

- Pacotes ativos: `http://www.debian.org/devel/people`
- Pacotes sendo processados: `http://www.debian.org/devel/wnpp/being_packaged`
- Pacotes órfãos: `http://www.debian.org/devel/wnpp/rfa_bypackage`
- Pacotes em adoção: `http://www.debian.org/devel/wnpp/being_adopted`
- Pacotes contra-indicados: `http://www.debian.org/devel/wnpp/unable-to-package`

Você também pode ajudar alguém que está empacotando algo mas não consegue fazê-lo (ou mantê-lo) sozinho. Com isso, você se tornará co-mantenedor do pacote. Neste caso, veja o link: `http://www.debian.org/devel/wnpp/help_requested`. Caso o pacote esteja órfão, você poderá adotá-lo.

11.2 Eu conseguirei empacotar?

Não é preciso dizer que você deverá ter a certeza de que conseguirá empacotar o programa antes de fazê-lo. Assim sendo, dentro do Debian Unstable, tente instalar o programa de forma convencional (./*configure*, *make*, *make install* etc). Isso poderá ser feito em uma jaula Sid (veja o item 11.9). Se conseguir, você estará apto a empacotar tal programa.

11.3 O autor é meu amigo?

A próxima fase será buscar um contato com o autor e lhe dizer sobre a sua intenção em empacotar o programa. Ele não precisa concordar com você. No entanto, se ele não for seu amigo e você resolver empacotar, todas as soluções de bugs e falhas de segurança cairão sobre a sua pessoa. Você se considera apto a fazer isso sozinho? De qualquer forma, você nunca estará sozinho. Nas listas *debian-devel* e *debian-devel-portuguese* você encontrará ajuda sempre que precisar. Mas não queira que os outros façam tudo para você!

11.4 Verificação das falhas básicas de segurança

As falhas básicas de segurança em códigos C poderão ser analisadas com os programas *flawfinder*, *rats*, *pscan* e *splint*. Esses são os programas utilizados pelos robôs que atuam no Debian.

11.5 Eu preciso de um sponsor!

Se você não for um desenvolvedor Debian, não terá acesso aos servidores e precisará de um sponsor. O sponsor é um desenvolvedor que se prontifica a analisar o seu trabalho e a fazer uploads para você. Você poderá conseguir um, facilmente, nas listas *debian-mentors* e *debian-devel-portuguese*.

Inicialmente, disponibilize o seu pacote no site `http://mentors.debian.net`. A seguir, anuncie o programa nas duas listas para atrair a atenção de algum sponsor. O próprio site Mentors proporá um modelo de post para a lista.

> Para fazer upload no site Mentors, você precisará de uma chave GPG cadastrada.

Uma outra solução é entrar para um time do Debian. Por exemplo: caso você deseje empacotar uma fonte TrueType, você poderá solicitar a entrada no time pkg-fonts. Coloque o seu pacote no site Mentors.net e solicite a entrada no time por e-mail ou chat, indicando que você já tem um pacote que precisa de upload. A relação de times está em `http://wiki.debian.org/Teams#Packaging_teams`.

11.6 Vou empacotar: Intent To Package (ITP)

Você decidiu que vai empacotar, pois reúne as condições citadas nos itens anteriores. Então, você deve declarar isso ao mundo Debian. Isso é um ITP e deve ser declarado por intermédio da abertura de um bug. Emita os seguintes comandos:

```
# apt-get install reportbug
$ export DEBFULLNAME="Seu nome completo e sem acentos"
$ export DEBEMAIL="seu e-mail"
$ reportbug -b
```

> Você poderá inserir as linhas *export* no arquivo */etc/bash.bashrc* (se a máquina for somente sua) ou no *~/.bashrc* (somente para o seu usuário), para não ter que digitar os dados toda vez que for empacotar. Ainda, se desejar utilizar outro editor de textos dentro do *reportbug* (o padrão é o nano), utilize: *export EDITOR=nome_do_editor*.

Procure nunca mais mudar a forma como entrou o seu nome e e-mail para não aparecer com duplicidade na relação de mantenedores (`http://www.debian.org/devel/people`). Utilize a opção standard. Quando for perguntado, pela primeira vez, pelo nome do pacote, digite “wnpp”, sem as aspas. Depois, escolha a opção ITP. Você terá que inserir uma descrição sumária e uma descrição completa do pacote. Na descrição sumária use até 80 caracteres e não repita o nome do pacote. Seja conciso e preciso. Lembre-se que o *apt-cache* pesquisa os pacotes pelas suas descrições sumária e completa.

Existem outras opções interessantes além do ITP, como a RFA, que permite a adoção de pacotes órfãos. Veja mais detalhes sobre o assunto em *http://www.debian.org/devel/wnpp*.

Exemplo de uma descrição sumária ruim:

```
HAPM is a light and fast high availability port monitor and works with
Heartbeat daemon to check TCP/UDP ports
```

A descrição sumária anterior, além de citar o nome do pacote, ficou muito longa. Exemplo de uma boa descrição sumária:

```
light and fast high availability port monitor
```

Quer fazer um bom empacotamento? Estude os pacotes de outras pessoas para ver como elas fizeram o que você quer fazer! Para isso: # *apt-get source* <*nome_do_pacote*>.

Quando chegar o momento de fazer a descrição completa, você será redirecionado para um editor de textos. No fim, verfique se tudo está certo, salve e saia. Um e-mail com os dados do pacote será enviado para o sistema de bugs do Debian. Em pouco tempo, o BTS irá responder ao seu e-mail, enviando um número de bug (ticket). Guarde esse número pois ele será muito importante.

11.7 O nome do pacote

Talvez você tenha que mudar o nome do pacote, caso ele seja muito genérico. Exemplo: você quer empacotar um contador de acessos web, feito em perl, muito simples de utilizar. Digamos que o nome dele seja *counter*. É um nome muito genérico e já poderá haver outro pacote com o mesmo nome. Neste caso, você deverá mudar o nome para algo como *perlcounter* ou *swc* (simple web counter).

11.8 O nome do pacote fonte

O Debian segue a convenção que estabelece o formato do nome dos pacotes fonte. O nome do pacote fonte sempre deverá ser *nome-versão.tar.gz* (ou .Z ou .bz2). Exemplo: *meuprograma-3.2.4.tar.gz*.

11.9 A criação do ambiente de desenvolvimento

Há um guia de empacotamento disponível em `http://debianet.com.br`. A sequência de ações para a criação do ambiente de desenvolvimento de pacotes (jaula Debian) está descrito, passo a passo em tal guia.

> Todo pacote, para ser inserido no Debian, deverá ser criado na Release Unstable (Sid). Para isso, deveremos criar uma jaula Sid para podermos trabalhar em um ambiente limpo.

11.10 Backup do ambiente de desenvolvimento (jaula)

Uma vez montada a jaula, você poderá optar por guardar uma cópia comprimida da mesma, evitando a necessidade de recriá-la, caso necessite de uma nova jaula. Os procedimentos estão no guia mostrado no item anterior.

> Tenha muito cuidado com a jaula comprimida, pois a mesma conterá a sua chave privada PGP (GPG). Não permita que alguém, além de você, utilize essa jaula!

Capítulo 12 - O processo de empacotamento

12.1 O processo

O processo de empacotamento é a etapa mais difícil. Várias verificações deverão ser feitas, após o empacotamento, para certificar-se de que nada está errado.

É importante citar que quase tudo acontecerá dentro do diretório *debian*, que será criado dentro da jaula.

12.2 A fase inicial

Inicialmente, deveremos preparar o código-fonte para ser empacotado. Como já foi dito anteriormente, o pacote fonte deverá ter a seguinte formação no nome: *programa-versão.tar.gz*. Caso isso não esteja ocorrendo, recrie o pacote fonte já com o nome correto.

> Para corrigir o nome do pacote fonte, descomprima-o, altere o nome do diretório e recomprima-o.

Para preparar o código-fonte, siga os procedimentos:

- Copie o código-fonte do programa para dentro da jaula Sid.
- Enjaule-se na jaula Sid.
- Crie um diretório chamado *pkg-nomedoprograma*.
- Mova o código fonte para o diretório criado.
- Descompacte o código-fonte.
- Dentro do diretório do código-fonte, execute:

```
# dh_make –f ../nome_do_fonte.tar.gz –c <licença>
```

> O parâmetro “-c <licença>” não é obrigatório. Veja o # *man dh_make* para saber as licenças possíveis.

Note que dentro do código-fonte foi criado um diretório *debian*. Entre nesse diretório e veja os arquivos lá existentes. É dentro desse diretório que será criada uma estrutura, com o nome do programa, após a sua compilação, que conterá todos os arquivos a serem instalados (já compilados!).

12.3 A configuração dos arquivos de debianização

Dentro do diretório *debian* será necessário configurar vários arquivos. Todos os arquivos exemplo terminam com .ex. Renomeie-os para que os mesmos tenham efeito (remover o .ex). Os arquivos criados são os seguintes (em negrito os de existência obrigatória):

- **changelog**: conterá o changelog do mantenedor Debian. O changelog do mantenedor lista as alterações que foram feitas para que o pacote pudesse ser aceito pelo Debian, incluindo as correções de segurança ou bug feitas pelo upstream (autor). O changelog é responsável por fechar bugs (inclusive ITP!). Para isso, utiliza a expressão "closes". Veja um exemplo:

```
gajim (0.10.1-6) unstable; urgency=low

  * fix LDFLAGS problem. Closes: #384439

 -- Yann Le Boulanger <asterix@lagaule.org>  Mon, 24 Jul 2006 14:45:34

gajim (0.10.1-5) unstable; urgency=low

  * Add dependance on python2.4. Closes: #379662

 -- Yann Le Boulanger <asterix@lagaule.org>  Mon, 19 Jul 2006 21:30:08

gajim (0.10.1-4) unstable; urgency=low

  * Fix warning at installation. Closes: #366725
  * Fix pt_BR translation.

 -- Yann Le Boulanger <asterix@lagaule.org>  Mon, 19 Jul 2006 21:30:080
```

> Yann Le Boulanger <asterix@lagaule.org> é o mantenedor oficial do pacote gajim.

- **compat**: esse aquivo identifica a versão do DebHelper utilizado e não deve ser modificado, a não ser em futuras revisões do pacote, se for o caso.
- **control**: esse arquivo é um dos principais elementos de controle. Contém as descrições curta e completa, as dependências para a compilação, as dependências para a instalação, o nome do mantenedor etc.

> Todos os dados inseridos em todos os arquivos *control* dos pacotes instalados estão disponíveis em */var/lib/apt/lists*. Com um grep, será possível estudar como cada mantenedor implementou algum dado nesse arquivo.

- **copyright**: contém a licença do programa e do pacote Debian. Observe que todo código-fonte sempre deverá trazer consigo a licença utilizada.

- cron.d: contém os agendamentos de tarefas a serem inseridos no crontab. Poderá ser removido.

> Talvez você prefira fazer scripts e inseri-los dentro de */etc/cron.(hourly|daily|monthly|weekly)*, em vez dede utilizar o *crontab*. Penso que seja a melhor opção, uma vez que o administrador do sistema poderá apagar o *crontab* do root a qualquer instante.

- dirs: contém os diretórios que deverão ser criados para que o pacote possa ser instalado (colocar apenas os diretórios que o Makefile original do programa não criar).
- docs: contém a relação de documentos, oriundos do código-fonte do upstream, que serão copiados para */usr/share/doc/nome_do_programa*.
- emacsen-install: usado apenas com emacs. Remover caso não use emacs.
- emacsen-remove: usado apenas com emacs. Remover caso não use emacs.
- emacsen-startup: usado apenas com emacs. Remover caso não use emacs.
- init.d: um esqueleto de arquivo do */etc/init.d*, caso o programa não possua um e necessite do mesmo. Se for o caso, remova.
- manpage.1: um esqueleto de manpage nível 1, caso o programa não possua manpage e necessite de uma (CGI, por exemplo, não necessita). Se for o caso, remova.

> Para entender os níveis de manuais on-line, consulte a Seção 5.8 do Capítulo 5 do Guia do Novo Mantenedor Debian (*http://www.debian.org/doc/manuals/maint-guide*) ou o manual on-line do comando *man* (# *man man*). Caso você precise fazer uma manpage, o programa gmanedit, para ambiente gráfico, poderá ser útil.

- manpage.sgml: idem ao anterior em formato sgml.
- manpage.xml: idem ao anterior em formato xml.
- menu: insere a aplicação em menus nos ambientes gráfico e shell (se for o caso deste último). Remova se não for necessário.
- postinst: executa ações depois que o pacote for instalado. Remova se não for necessário.
- postrm: executa ações depois que o pacote for removido. Remova se não for necessário.
- preinst: executa ações antes da instalação do pacote. Remova se não for necessário.
- prerm: executa ações antes da remoção do pacote. Remova se não for necessário.

- README.Debian: contém instruções específicas para a utilização do pacote dentro do Debian. Só deverá ser criado caso o README do upstream (autor) não seja suficientemente esclarecedor ou caso você tenha feito mudanças que irão confundir o usuário final.
- **rules**: o arquivo rules funciona como um Makefile dentro do diretório *debian*. Possui funções especiais do tipo *dh*, que podem ser detalhadas nos seus respectivos manuais on-line. Há um resumo de cada função *dh* no Apêndice A.
- <programa>-default: esqueleto de um arquivo de opções que poderá ser instalado em */etc/default*. Remova se não for necessário.
- <programa>.doc-base: utilizado para coordenar manuais que não sejam do tipo man ou info. Chamadas a arquivos pdf, por exemplo, poderão ser inseridos aqui. Remova se não for necessário.
- watch: permite que o programa *uscan* monitore o site do programa fonte em busca de uma nova versão do mesmo. Remova se não for necessário.

Para obter melhores explicações sobre os arquivos do diretório debian, consulte o Guia do Novo Mantenedor Debian, existente em `http://www.debian.org/doc/manuals/maint-guide`.

12.4 O arquivo control

O arquivo control contém as informações essenciais do pacote. A seguir, dois exemplos de control.

Exemplo 1: pacote phpwebcounter

```
Source: phpwebcounter
Section: web
Priority: optional
Maintainer: Joao Eriberto Mota Filho <eriberto@eriberto.pro.br>
Build-Depends: debhelper (>= 5), dpatch
Standards-Version: 3.8.0
Vcs-Svn: svn://svn.debian.org/svn/debian-br-team/packages/phpwebcounter
Vcs-Browser: http://svn.debian.org/wsvn/debian-br-team/packages/phpwebcounter
Homepage: http://phpwebcounter.sourceforge.net

Package: phpwebcounter
Architecture: all
Depends: php5
Recommends: phpwebcounter-extra
Description: simple and light web hit counter
 PHP script to show the number of hits in web pages using ASCII text or PNG
 images. You can use your custom PNG images. PHP Web Counter is a very
 easy-to-configure application.
 .
 It isn't a CGI program and don't require special things to work.
 .
 Demo site: http://phpwebcounter.sf.net
```

Exemplo 2: pacote jp2a

```
Source: jp2a
Section: graphics
Priority: optional
Maintainer: Joao Eriberto Mota Filho <eriberto@eriberto.pro.br>
Build-Depends: debhelper (>= 7), autotools-dev, libjpeg-dev, libcurl4-gnutls-dev, libncurses5-
dev, dpatch
Standards-Version: 3.8.3
Vcs-Svn: svn://svn.debian.org/svn/debian-br-team/packages/jp2a
Vcs-Browser: http://svn.debian.org/wsvn/debian-br-team/packages/jp2a
Homepage: http://csl.sublevel3.org/jp2a

Package: jp2a
Architecture: any
Depends: ${shlibs:Depends}
Description: converts jpg images to ascii
 Small utility that converts JPG images to ASCII using libjpeg.
 jp2a is very flexible. It can use ANSI colors and html in output.
 .
 Screenshot: http://www.eriberto.pro.br/debian/screenshots/jp2a.jpg
```

Note que a descrição completa tem as linhas iniciadas por espaço e que cada linha em branco recebe um caractere ponto. Para a instalação, o programa phpwebcounter depende do php5.

Alguns campos merecem especial atenção para que não sejam preenchidos de forma errada. São eles:

- *Section*: lista a subdivisão a ser adotada dentro do repositório. Uma lista de seções de repositório possíveis poderá ser obtida na Seção 2.4 do Capítulo 2 da Debian Policy (`http://www.debian.org/doc/debian-policy`). Em `http://packages.debian.org/unstable` será possível obter uma rápida explicação sobre cada uma das seções de repositório.
- *Priority*: descreve a prioridade a ser adotada pelo pacote. Isso servirá para fazer com que pacotes de correção de segurança, por exemplo, cheguem mais rapidamente ao repositório. Uma explanação sobre cada uma das prioridades poderá ser encontrada na Seção 2.5 do Capítulo 2 da Debian Policy (`http://www.debian.org/doc/debian-policy`).
- *Maintainer*: é mantenedor principal do pacote. Este campo só poderá receber um nome e um e-mail. Os co-mantenedores deverão ser inseridos em *Uploaders*.
- *Uploaders*: dentre outras funções, lista os co-mantenedores do pacote. Este campo só existirá ser houver co-mantenedores.
- *Build-Depends*: lista as dependências necessárias para que possa ocorrer a compilação. Geralmente, essas dependências, quando se tratando de libraries, terão a extensão *-dev* no seu nome. Este campo só existirá ser houver dependências para a compilação.

- *Vcs-Svn*: refere-se ao servidor SVN que hospeda o diretório *debian* de um pacote. Este não é um campo obrigatório.
- *Depends*: lista as dependências para a instalação do pacote pelo usuário final. O caractere vírgula funciona como aditivo, enquanto o pipe funciona como "ou lógico". Exemplo: *x, y, z | k | h, w*. Ou seja: depende de x + y + w + um dos três outros citados (z, k ou h). Poderão ser utilizados pacotes virtuais, desde que haja pelo menos um real fazendo "ou lógico" antes do virtual. Este campo só existirá ser houver dependências para a instalação.

> Para entender melhor os parâmetros do arquivo *control*, veja o Capítulo 5 do Debian Policy Manual (Control files and their fields), em *http://www.debian.org/doc/debian-policy*. No referido link, há campos que não foram citados neste capítulo.

12.5 O arquivo rules

O arquivo *rules* funciona exatamente como um Makefile. Ele também executa o Makefile original do programa, caso haja, tornando-o um sub-processo. Assim sendo, o *rules* poderá passar parâmetros para o Makefile original. Isso é muito utilizado quando o Makefile original está instalando arquivos fora do diretório *debian/nome_do_pacote* (que é o diretório que será utilizado como estrutura virtual temporária para a criação do *.deb*). O *rules* poderá determinar um caminho de instalação em uma variável (*DESTDIR*, geralmente). Assim, o Makefile, alterado por um patch, poderá corrigir o destino de cada pacote com o uso dessa variável, instalando os arquivos em *debian/nome_do_pacote*. O rules também poderá servir como Makefile, caso o programa original não possua um. Um exemplo típico disso são as CGIs em Perl, códigos PHP etc.

Lembre-se que, se o código-fonte utilizar *autoconf*, será possível executar o comando # *configure --help* e determinar diretórios de destino corretamente, sem a necessidade de implementar patches (veja o Capítulo 15).

12.6 A técnica de criação do pacote

Para criar um pacote, siga os passos existentes no item 2. do Guia de Empacotamento de Software no Debian GNU/Linux, disponível em `http://debianet.com.br`.

12.7 O resultado da criação do pacote

Uma vez executado o comando *debuild*, serão criados alguns arquivos um diretório acima do diretório do código-fonte. São os seguintes, os referidos arquivos:

- *.build*: Debian Build. Contém um log de todo o empacotamento. As mensagens mostradas são as mesmas que aparecem na tela durante a execução do comando *debuild*.
- *.changes*: Debian Changes. Contém um controle de alterações e a relação de hashes dos arquivos criados.
- *.deb*: Debian binary. Pacote contendo binários e toda a estrutura do elemento que foi empacotado, prontos para a instalação.
- *.debian.tar.gz*: contém todas as modificações aplicadas sobre o código-fonte original para que se chegue aos arquivos existentes dentro do *.deb*, que serão instalados no sistema operacional.
- *.dsc*: Debian Source Control. O *.dsc* controla o código-fonte do programa e possui dados retirados do arquivo control.
- *.orig.tar.gz*: pacote contendo o código-fonte original do upstream (autor).

> O Capítulo 5 da Debian Policy, disponível em *http://www.debian.org/doc/debian-policy*, descreve os campos que aparecem em alguns dos arquivos citados.

12.8 Alvos no debuild

A função básica do *debuild*, dentre outras ações, é a de executar o arquivo *rules* (que é um Makefile).

> **IMPORTANTE:** O comando *debuild* deverá ser executado no diretório que contém o código-fonte do upstream ou no diretório *debian*, existente dentro do anteriormente citado. Quando executado dentro do diretório *debian*, o comando *debuild* segue imediatamente para o diretório do código-fonte do upstream. Considere esta relevante informação quando inserir comandos como *cp*, *mv* e *rm* no *debian/rules*. Ainda o *debuild* cria, no início da sua execução, a variável *CURDIR*, que representa o resultado do comando # *pwd* dentro do diretório do código-fonte do upstream.

12.9 O que é o .deb afinal?

Bem, agora que o pacote está criado, fica mais fácil entendermos o que é o *.deb*. O *.deb* é um arquivo compactado com o utilitário *ar*. Escolha um pacote *.deb* qualquer e execute:

```
# ar -x <pacote>.deb
```

Após a descompactação, surgirão três arquivos, a saber:

- *control.tar.gz*: contém os arquivos necessários para a instalação do *.deb*, como o arquivo *control* e os (*pre*|*post*)(*inst*|*rm*) etc.
- *data.tar.gz*: contém os binários. São os arquivos que serão instalados no Debian.
- *debian-binary*: contém a versão Debian do binário. É um arquivo de controle.

Caso os arquivos estejam dentro de um diretório, será possível reconstituir o pacote com o comando # *dpkg -b <diretório>*. Ainda, o comando # *dpkg -I <pacote.deb>* poderá ser utilizado para ver as informações internas do pacote. A sua estrutura interna poderá ser vista com o comando # *dpkg -c <pacote.deb>*.

12.10 O comando dpkg-source

O comando *dpkg-source* poderá ser utilizado para reconstruir um diretório de debianização. Para isso serão necessários os arquivos *.debian.tar.gz*, *.dsc* e *.orig.tar.gz*. Para reconstruir o diretório, utilize o comando:

```
# dpkg-source -x <pacote>.dsc
```

12.11 Alguns diretórios e arquivos importantes

Como já foi dito, uma das melhores formas de aprender a empacotar é verificar como outras pessoas fizeram os seus pacotes. Assim, considere os seguintes dados:

- Todos os arquivos *control* existentes estarão dentro dos arquivos existentes em */var/lib/apt/lists.*
- Todos os arquivos de override (vide Capítulo 15), poderão ser encontrados dentro de */usr/share/lintian/overrides*.
- Todos os arquivos *preinst, postinst*, *prerm* e *postrm* poderão ser vistos em */var/lib/dpkg/info*.

"Nós iremos retribuir à comunidade do software livre."

Capítulo 13 - A verificação do empacotamento

13.1 As verificações

Após construir o pacote, é importante realizar checagens para buscar erros e dependências não resolvidas. Várias ferramentas poderão auxiliar nesse trabalho. As principais (mais utilizadas) são o *lintian*, e o *cowbuilder* (um sistema baseado no famoso *pbuilder*).

13.2 Regra número 1: Não deixe lixo!

Não devemos deixar lixo dentro do código-fonte original e dentro do diretório *debian*. Assim sendo, faça o seguinte:

- Execute os comandos *dh_auto_clean* e *dh_clean* e verifique se o código-fonte do diretório do upstream foi totalmente limpo. Caso contrário, modifique o alvo *clean* do arquivo *rules* para que o mesmo faça a limpeza. Nos *rules* mais novos, será necessário criar toda a estrutura do alvo. Essa estrutura poderá ser capturada na tela gerada pelo comando *debuild*.
- Remova todos comentários desnecessários de arquivos do diretório *debian*. O arquivo *copyright*, por exemplo, originalmente possui muitos comentários. Não remova os comentários iniciais, de direitos autorias do *rules*.
- Remova os arquivos existentes no diretório *debian* e que não estiverem sendo utilizados.
- Siga as recomendações de limpeza existentes no item 2. do Guia de Empacotamento de Software no Debian GNU/Linux, disponível em `http://debianet.com.br`.

13.3 lintian

O *lintian* já é conhecido por nós, pois foi utilizado com o *debuild*. É interessante relembrar que ele pode ser executado contra os arquivos *.changes*, *.deb* e *.dsc* e que com a chave *-i* ele mostra detalhes sobre os erros encontrados. Com *-I*, ele mostra as mensagens menos graves (informativas). Então, seria uma boa ideia executar *-iI*.

13.4 cowbuilder

O *cowbuilder* cria uma jaula limpa e tenta construir e instalar o pacote *.deb* nessa jaula, sanando as dependências de acordo com o prescrito no arquivo *control*. Ele é essencial para sabermos se alguma dependência de construção está faltando no campo `Build-Depends` do arquivo `debian/control`, uma vez que a jaula poderia esta suja ou viciada.

Para utilizar o *cowbuilder*, siga os passos existentes no item 3. do Guia de Empacotamento de Software no Debian GNU/Linux, disponível em `http://debianet.com.br`.

13.5 piuparts

O *piuparts* funciona de forma similar ao *cowbuilder*. No entanto, o *cowbuilder* testa a construção do pacote em jaula limpa, enquanto o *piuparts* testa a instalação do pacote em jaula limpa. Para utilizá-lo, siga os seguintes procedimentos:

```
# apt-get install piuparts
# piuparts <pacote>.deb
```

Capítulo 14 - Revisão Debian e nova versão do programa

14.1 Revisão Debian

A revisão Debian caracteriza-se pela alteração do pacote contruído anteriormente. Não será utilizada uma nova versão de código criado pelo upstream (autor). A revisão Debian tem por objetivo corrigir erros no empacotamento ou bugs simples que não exijam uma nova versão do programa e sim do pacote. Para fazer uma revisão Debian, siga os passos:

- Dentro do diretório *debian*, execute o comando:
  ```
  # dch -i
  ```
- A seguir, edite o arquivo *changelog*, inserindo a descrição do bug e a sua solução, seguido de *Closes* (se for o caso) para fechar o bug (se houver algum no BTS) no upload do pacote. Exemplo:
  ```
  * Fix symbols files for non any-amd64 architectures, avoiding FTBFS (Closes: #630207)
  ```
- Reconstrua o pacote com *debuild*.

> Para mais detalhes, consulte existentes na Seção 9.1 do Capítulo 9 do Guia do Novo Mantenedor Debian (*http://www.debian.org/doc/manuals/maint-guide*).

14.2 Nova versão do programa

Quando uma nova versão do programa que você empacotou for lançada, execute os seguintes passos, para gerar um pacote referente a essa nova versão:

- Enjaule-se na jaula de desenvolvimento.
- Faça download do novo código-fonte (utilize *wget*).
- No mesmo diretório no qual fez o download, execute:
  ```
  # apt-get install liburi-perl
  # apt-get source <pacote>
  # cd <diretório_do_código_do_pacote>
  # uupdate -u ../<pacote-versãonova.tar.gz
  ```

Um novo diretório será criado já com o novo código-fonte e com o diretório *debian*.

Capítulo 15 - Ações especiais para o empacotamento

15.1 Patches

Pode ser que haja a necessidade de alterações para que o código-fonte do autor funcione corretamente no Debian. Como o código-fonte do autor não poderá ser alterado, haverá a necessidade de aplicar patches em tempo de compilação. Para isso, siga os procedimentos descritos nos itens 7 e 8 do Guia de Empacotamento de Software no Debian GNU/Linux, disponível em `http://debianet.com.br`.

15.2 Overrides

Overrides são utilizados para "calar" o *lintian*. A técnica de override só deve ser utilizada se você tiver a certeza do que está fazendo. Um exemplo clássico: digamos que um dos seus pacotes necessite criar um diretório vazio, que será populado no futuro, depois que a execução do programa ocorrer. Em consequência, o lintian irá alarmar da seguinte forma:

```
I: pacote_teste: package-contains-empty-directory usr/lib/pacote-teste/
```

Nesse caso, teremos que aplicar um override. Os overrides ficam localizados em */usr/share/lintian/overrides*. Aproveite para verificar o conteúdo de alguns deles.

Para criar um override, faça o seguinte:

- Crie um arquivo *debian/<pacote>.lintian-overrides*. Exemplo: *debian/pacote-teste.lintian-overrides*.
- Crie uma linha de comentário sobre o problema e, abaixo dela, insira a exata mensagem que o *lintian* apresenta, sem incluir o primeiro caractere representativo de nível de problema (E:, I: ou W:). Exemplo:

```
pacote_teste: package-contains-empty-directory usr/lib/pacote-teste/
```

> Em arquivos *debian/rules* no padrão antigo, será necessário inserir *dh_lintian* no bloco final.

15.3 A força dos DHs

Os DHs aparecem no fim do arquivo *rules* (padrão antigo) ou são gerados, automaticamente, após a execução do comando *debuild* (padrão novo). Além de realizar verificações, poderão ser utilizados para automatizar diversas tarefas. Um exemplo clássico: digamos que você esteja empacotando um programa servidor de páginas. Esse programa, após ser incluído em */etc/init.d*, precisará entrar nos runlevels adequados. Isso, geralmente, será

feito por um script *postinst*. Então, deverá haver também um script *prerm* que será responsável por parar o programa antes da sua remoção e por retirar os links nos runlevels. O *dh_installinit* poderá fazer tudo isso automaticamente para você. Batará criar um arquivo *debian/nome_do_pacote.init*, caso o upstream não tenha fornecido um init. Assim, o *dh_installinit* atuará e criará os comandos necessários dentro dos arquivos *postinst* e *prerm*.

> Lembre-se de que todo comando dh_ possui um manual on-line. Ainda, você poderá ver como outras pessoas fizeram os seus pacotes, bastando emitir o comando # *apt-get source <pacote>*. Assim, você aprenderá com trabalhos já aceitos pelo Debian.

No Apêndice A há uma relação com a função básica de cada *dh*.

15.4 Testando dependências

Fatalmente, será necessário instalar algumas dependências na tentativa de compilar o pacote. O grande problema é que isso irá "contaminar" a jaula. Essa jaula poderá ficar comprometida para um futuro empacotamento, forçando a reconstrução da mesma. Então, é interessante ter uma outra jaula destinada apenas a testes de dependências de programas que devam ser compilados.

15.5 Alterando os caminhos de instalação nos Makefiles

Constantemente, precisaremos alterar o caminho de instalação default nos Makefiles para que os mesmos coloquem os arquivos resultantes da compilação dentro de *debian/<nome_do_pacote>*. Essa é uma tarefa simples, uma vez que os Makefiles são executados a partir do *rules*, sendo um sub-processo deste.

> Se o programa a ser empacotado utilizar *autoconf*, a operação será mais fácil ainda, pois haverá como definir diretórios de instalação e outros detalhes importantes. Para saber como, bastará executar # *./configure -h*. Outra alternativa bem mais moderna é a utilização dos DHs dh_auto_build e dh_auto_install. (Dados de abril de 2013): Os pacotes sxiv e lighttpd utilizam esse método mais moderno de instalação. Já os pacotes phpwebcounter e boa utilizam os métodos antigos de *debian/rules*. Detalhes, inclusive sobre os alvos possíveis e sobre o método mais moderno, poderão ser vistos em *http://www.debian.org/doc/manuals/maint-guide/dreq.en.html#rules*. É interessante citar que você poderá baixar uma versão do Guia dos Novos Mantenedores em PDF em *http://www.debian.org/doc/devel-manuals#maint-guide*.

Inicialmente, deveremos criar uma variável, que geralmente chamaremos de *DESTDIR*, no comando *$(MAKE) install* existente no alvo *install* em *debian/rules*, imediatamente antes de executarmos o *Makefile* que acompanha o código-fonte. Exemplo:

```
install:
dh_testdir
dh_testroot
dh_clean -k
dh_installdirs
$(MAKE) install DESTDIR=$(CURDIR)/debian/pacote-teste
```

> *$(CURDIR)* refere-se ao diretório atual. O valor de tal variável é determinado no momento no qual executamos o comando # *debuild*. Assim, geralmente, teremos *DESTDIR=$(CURDIR)/debian/<pacote>*.

O próximo passo será fazer com que o Makefile utilize essa variável. Nesse caso, estaremos lidando com o código-fonte do autor, o que irá requerer um patch. A seguir um exemplo de como fazer o Makefile original do programa usar a variável criada no rules.

Antes:

```
# Makefile.in for hapm

CC                = gcc
SYSCONFDIR        = /etc/ha.d
INITDIR           = /etc/init.d
INSTALL           = cp
INSTALLDIR        = /usr/sbin
MANDIR            = /usr/share/man/man8
CFLAGS            = -O2 -g -w
...
```

Depois:

```
# Makefile.in for hapm

CC                = gcc
SYSCONFDIR        = $(DESTDIR)/etc/ha.d
INITDIR           = $(DESTDIR)/etc/init.d
INSTALL           = cp
INSTALLDIR        = $(DESTDIR)/usr/sbin
MANDIR            = $(DESTDIR)/usr/share/man/man8
CFLAGS            = -O2 -g -w
...
```

Cabe ressaltar que, no HAPM, a operação foi facilitada pelo uso de variáveis por parte do upstream (desenvolvedor). Se não existissem essas variáveis, teríamos que alterar várias linhas ao longo do Makefile.

> Lembre-se: ao alterar o Makefile, você estará manipulando o código-fonte do autor. Assim sendo, utilize um patch.

"Integridade do código fonte do autor."

Capítulo 16 - Considerações finais

16.1 Boas condutas para a criação de pacotes

- Conheça e utilize, regularmente, o programa a ser empacotado.
- Após empacotar o programa, passe a utilizá-lo.
- Teste o código em C com o *flawfinder*, o *pscan*, o *rats* e o *splint*.
- Dê uma "passada de olhos" no código-fonte para visualizar a sua estrutura.
- Contacte o desenvolvedor do programa (upstream) e converse sobre a sua intenção de empacotar o programa dele. Um bom relacionamento com o upstream é essencial para que o seu pacote seja eternamente mantido no Debian. Não se trata de um passo obrigatório, mas recomendado.
- Conheça a Debian Policy (leia-a toda ao menos uma vez).
- Conheça o Guia do Novo Mantenedor (leia-o todo ao menos uma vez).
- Conheça o Developer's Reference (leia-o todo ao menos uma vez).
- Não modifique diretamente o código-fonte original; use patches.
- Não poupe esforços para fazer o melhor.
- Siga a Política Debian.
- Abra outros pacotes para ver como outras pessoas fizeram o que você quer fazer.
- Só contrarie o *lintian* se tiver a certeza do que está fazendo. Se for o caso, utilize overrides.
- Analise o pacote no dia seguinte. Não o envie no mesmo dia. Você vai achar erros.
- Teste o pacote em ambiente limpo (*cowbuilder* e piuparts são opções) e, se puder, em ambiente de produção.

16.2 A criação de um repositório local simples

Para criar um repositório local bem simples, siga os seguintes passos:

- Instale um servidor de páginas.
- Crie o diretório */var/www/debian/binary*.
- Copie os arquivos *.deb* para */var/www/debian*.

- Dentro de */var/www/debian*, execute:

```
# dpkg-scanpackages binary /dev/null | gzip -9c > binary/Packages.gz
```

> Este procedimento será necessário sempre que for alterado o conteúdo do repositório.

- Caso deseje adicionar o código-fonte do pacote no repositório, copie-o para */var/www/debian* e execute:

```
# dpkg-scansources binary | gzip -9c > binary/Sources.gz
```

- Nas máquinas clientes, insira os repositórios da seguinte forma (arquivo */etc/apt/sources.list*):

```
deb http://<servidor>/debian/ binary/
deb-src http://<servidor>/debian/ binary/
```

16.3 A criação de um repositório local padrão Debian

Caso deseje um sistema de repositórios idêntico ao que o Debian utiliza, siga os passos do seguinte tutorial:

```
http://www.debian-administration.org/articles/286
```

Há mais alguns tutoriais interessantes:

```
http://wiki.debian.org/SettingUpSignedAptRepositoryWithReprepro
http://www.jejik.com/articles/2006/09/setting_up_and_managing_an_apt_repo
sitory_with_reprepro/
https://wikitech.wikimedia.org/wiki/Reprepro
```

"Não à discriminação contra pessoas ou grupos."

Referências bibliográficas e Bibliografia

1. Referências bibliográficas

Creative Commons. **Creative Commons Licenses.** Disponível em http://creativecommons.org/licenses. Acesso em 07 abr. 13.

Debian.org. **Debian - O sistema operacional universal.** Disponível em http://www.debian.org. Acesso em 07 abr. 13.

Freestandards.org. **Filesystem Hierarchy Standard.** Disponível em http://www.pathname.com/fhs. Acesso em 07 abr. 13.

GNU.org. **GNU General Public License.** Disponível em http://www.gnu.org/copyleft/gpl.html. Acesso em 07 abr. 13.

_____. **Licenses.** Disponível em http://www.gnu.org/licenses. Acesso em 07 abr. 13.

GnuPG.org. **The GNU Privacy Guard.** Disponível em http://www.gnupg.org. Acesso em 07 abr. 13.

KEMP, Steve. **Setting up your own APT repository with upload support.** Disponível em http://www.debian-administration.org/articles/286. Acesso em 28 set. 07.

O'Reilly. **The Artistic License.** Disponível em http://www.perl.com/pub/a/language/misc/Artistic.html. Acesso em 07 abr. 13.

OpenSource.org. **The approved licenses.** Disponível em http://opensource.org/licenses. Acesso em 07 abr. 13.

2. Bibliografia

Free Software Foundation. **Autoproject.** Disponível em http://directory.fsf.org/wiki/Autoproject. Acesso em 07 abr. 13.

GNU.org. **GNU Make Manual.** Disponível em http://www.gnu.org/software/make/manual. Acesso em 07 abr. 13.

_____. **Introduction to Autoconf.** Disponível em http://www.gnu.org/software/autoconf. Acesso em 07 abr. 13.

_____. **Introduction to Automake.** Disponível em http://www.gnu.org/software/automake. Acesso em 07 abr. 13.

KEREN, Guy. **Automating Program Compilation - Writing Makefiles.** Disponível em http://users.actcom.co.il/~choo/lupg/tutorials/writing-makefiles/writing-makefiles.html. Acesso em 07 abr. 13.

LEE, Clemens. **Debian Binary Package Building HOWTO.** Disponível em http://tldp.org/HOWTO/html_single/Debian-Binary-Package-Building-HOWTO. Acesso em 07 abr. 13.

MOTA FILHO, João Eriberto. **Descobrindo o Linux**. Livro, 3ª edição. Novatec Editora.

_____. **Usando o GnuPG**. Disponível em http://bit.ly/GNUPG. Acesso em 07 abr. 13.

Opus Software. **The Makefile.** Disponível em http://www.opussoftware.com/tutorial/TutMakefile.htm. Acesso em 07 abr. 13.

PGPI.org. **Freeware PGP versions.** Disponível em http://www.pgpi.org/products/pgp/versions/freeware. Acesso em 07 abr. 13.

Unicamp. **Pretty Good Privacy (PGP).** Disponível em http://www.dca.fee.unicamp.br/pgp/pgp.shtml. Acesso em 07 abr. 13.

Wikipedia.org. **GNU Privacy Guard.** Disponível em http://en.wikipedia.org/wiki/GNU_Privacy_Guard. Acesso em 07 abr. 13.

Apêndice A - Resumo das funções dh

dh_auto_build	automatically builds a package
dh_auto_clean	automatically cleans up after a build
dh_auto_configure	automatically configure a package prior to building
dh_auto_install	automatically runs make install or similar
dh_auto_test	automatically runs a package's test suites
dh_bugfiles	install bug reporting customization files into package build directories
dh_builddeb	build Debian binary packages
dh_clean	clean up package build directories
dh_compress	compress files and fix symlinks in package build directories
dh_desktop	deprecated no-op
dh_fixperms	fix permissions of files in package build directories
dh_gconf	install GConf defaults files and register schemas
dh_gencontrol	generate and install control file
dh_icons	Update Freedesktop icon caches
dh_install	install files into package build directories
dh_installcatalogs	install and register SGML Catalogs
dh_installchangelogs	install changelogs into package build directories
dh_installcron	install cron scripts into etc/cron.*
dh_installdeb	install files into the DEBIAN directory
dh_installdebconf	install files used by debconf in package build directories
dh_installdirs	create subdirectories in package build directories
dh_installdocs	install documentation into package build directories
dh_installemacsen	register an Emacs add on package
dh_installexamples	install example files into package build directories
dh_installgsettings	install GSettings overrides and set dependencies
dh_installifupdown	install if-up and if-down hooks
dh_installinfo	install info files
dh_installinit	install init scripts and/or upstart jobs into package build directories
dh_installlogcheck	install logcheck rulefiles into etc/logcheck/
dh_installlogrotate	install logrotate config files
dh_installman	install man pages into package build directories
dh_installmanpages	old-style man page installer (deprecated)
dh_installmenu	install Debian menu files into package build directories
dh_installmime	install mime files into package build directories
dh_installmodules	register modules with modutils
dh_installpam	install pam support files

dh_installppp	install ppp ip-up and ip-down files
dh_installudev	install udev rules files
dh_installwm	register a window manager
dh_installxfonts	register X fonts
dh_installxmlcatalogs	install and register XML catalog files
dh_link	create symlinks in package build directories
dh_lintian	install lintian override files into package build directories
dh_listpackages	list binary packages debhelper will act on
dh_make	prepare Debian packaging for an original source archive
dh_makeshlibs	automatically create shlibs file and call dpkg-gensymbols
dh_md5sums	generate DEBIAN/md5sums file
dh_movefiles	move files out of debian/tmp into subpackages
dh_perl	calculates Perl dependencies and cleans up after MakeMaker
dh_prep	perform cleanups in preparation for building a binary package
dh_python	calculates Python dependencies and adds postinst and prerm Python scripts (deprecated)
dh_python2	calculates Python dependencies, adds maintainer scripts to byte compile files, etc.
dh_quilt_patch	apply patches listed in debian/patches/series
dh_quilt_unpatch	unapply patches listed in debian/patches/series
dh_scrollkeeper	deprecated no-op
dh_shlibdeps	calculate shared library dependencies
dh_strip	strip executables, shared libraries, and some static libraries
dh_suidregister	suid registration program (deprecated)
dh_testdir	test directory before building Debian package
dh_testroot	ensure that a package is built as root
dh_ucf	register configuration files with ucf
dh_undocumented	undocumented.7 symlink program (deprecated no-op)
dh_usrlocal	migrate usr/local directories to maintainer scripts

Obs: listagem obtida com o comando # *apropos dh_*.