

debian

Guia para Maintainers de Debian

Osamu Aoki e Américo Monteiro

8 de outubro de 2026

Guia para Maintainers de Debian
by Osamu Aoki e Américo Monteiro

É dada permissão, livre de custos, a qualquer pessoa que obtenha uma cópia deste software e ficheiros de documentação associados (o ”Software”). para lidar com o Software sem restrições, incluindo sem limitação o direito de usar, copiar, modificar, fundir, publicar, distribuir, sub-licenciar, e/ou vender cópias do Software, e de permitir pessoas a quem o Software é mobilado de o fazer, sujeito às seguintes condições:

O aviso de copyright em cima e este aviso de permissão deve ser incluído em todas as cópias ou porções substanciais do Software.

O SOFTWARE É FORNECIDO ”COMO ESTÁ”, SEM GARANTIA DE NENHUM TIPO, EXPRESSA OU IMPLICADA, INCLUINDO MAS NÃO LIMITADO ÀS GARANTIAS DE COMERCIALIZAÇÃO, FITNESS PARA UM OBJECTIVO PARTICULAR E NÃO VIOLAÇÃO. EM NENHUMA SITUAÇÃO DEVEM OS AUTORES NEM OS DETENTORES DO COPYRIGHT SEREM ACUSADOS DE NENHUMA QUEIXA, DANOS OU OUTRA RESPONSABILIDADE, SEJA NA ACÇÃO DO CONTRATO, ATO ILÍCITO OU OUTRO, QUE OCORRA, A PARTIR DE OU COM QUALQUER LIGAÇÃO AO SOFTWARE OU DO USO DE OUTRAS NEGOCIAÇÕES NO SOFTWARE.

Este guia foi criado usando os seguintes documentos anteriores como referência:

- “Making a Debian Package (AKA the Debmake Manual)”, copyright © 1997 Jaldhar Vyas.
- “The New-Maintainer’s Debian Packaging Howto”, copyright © 1997 Will Lowe.
- “Debian New Maintainers’ Guide”, copyright © 1998-2002 Josip Rodin, 2005-2017 Osamu Aoki, 2010 Craig Small, e 2010 Raphaël Hertzog.

A versão mais recente deste guia deverá estar disponível:

- no “pacote debmake-doc” e
- na “ página webGuia para Maintainers Debian”.

Índice

1 Prefácio **1**

2 Visão geral **3**

3 Pré-requisitos **5**
3.1 Pessoas em redor de Debian . . . 5
3.2 Como contribuir . . . 5
3.3 Dinâmicas sociais de Debian . . . 6
3.4 Lembretes técnicos . . . 6
3.5 Documentação de Debian . . . 7
3.6 Recursos de ajuda . . . 8
3.7 Situação de arquivo . . . 8
3.8 Abordagens de contribuição . . . 9
3.9 Contribuidor e maintainer novato . . . 10

4 Configurações de Ferramenta **12**
4.1 Configuração do Email . . . 12
4.2 Configuração do **mc** . . . 13
4.3 Configuração do **git** . . . 13
4.4 Configuração do **quilt** . . . 13
4.5 Configuração do **openPGP** . . . 14
4.6 Configuração do **devscripts** . . . 15
4.7 Configuração do **sbuild** . . . 15
4.8 Configuração chroot persistente . . . 17
4.9 Configuração do **gbp** . . . 18
4.10 Proxy HTTP . . . 18
4.11 Repositório Debian privado . . . 18
4.12 Máquinas virtuais . . . 18
4.13 Rede local com máquinas virtuais . . . 18

5 Empacotamento simples **19**
5.1 Empacotar o tarball . . . 19
5.2 O grande panorama . . . 19
5.3 O que é debmake? . . . 20
5.4 O que é debuild? . . . 21
5.5 Passo 1: Obter a fonte do autor . . . 21
5.6 Passo 2: Gerar ficheiros modelo com o **debmake** . . . 23
5.7 Passo 3: Modificação dos ficheiros modelo . . . 27
5.8 Passo 4: Compilar pacote com **debuild** . . . 29
5.9 Passo 3 (alternativos): Modificação da fonte do autor . . . 32
5.10 Patch por abordagem “**diff -u**” . . . 32
5.11 Patch por abordagem **dquilt** . . . 33
5.12 Patch por abordagem “**dpkg-source --auto-commit**” . . . 35

6 Bases para empacotamento **38**
6.1 Fluxo de trabalho de empacotamento . . . 38
6.2 Pacote **debhelper** . . . 40
6.3 Nome e versão do pacote . . . 41
6.4 Pacote Debian nativo . . . 42
6.5 Ficheiro **debian/rules** . . . 43
6.6 Ficheiro **debian/control** . . . 44
6.7 Ficheiro **debian/changelog** . . . 44
6.8 Ficheiro **debian/copyright** . . . 45
6.9 Ficheiros **debian/patches/*** . . . 45

6.10 Ficheiro **debian/source/include-binaries** . . . 46
6.11 Ficheiro **debian/watch** . . . 46
6.12 Ficheiro **debian/upstream/signing-key.asc** . . . 46
6.13 Ficheiro **debian/salsa-ci.yml** . . . 47
6.14 Outros ficheiros **debian/*** . . . 47

7 Qualidade de empacotamento **52**
7.1 Reformatar ficheiros **debian/*** com **wrap-and-sort** . . . 52
7.2 Validar ficheiros **debian/*** com o **debputy** . . . 52
7.3 Verificar o empacotamento com **cme** . . . 52

8 Higienização da fonte **53**
8.1 Corrige com **Files-Excluded** . . . 53
8.2 Corrigir com "**debian/rules clean**" . . . 53
8.3 Corrigir com **extend-diff-ignore** . . . 54
8.4 Corrigir com **tar-ignore** . . . 54
8.5 Corrigir com "**git clean -dfx**" . . . 55

9 Mais sobre empacotamento **56**
9.1 Personalização do pacote . . . 56
9.2 **debian/rules** personalizado . . . 56
9.3 Variáveis para **debian/rules** . . . 57
9.4 Novo lançamento do autor . . . 57
9.5 Gerir a lista de patch com **dquilt** . . . 58
9.6 Comandos de compilação . . . 58
9.7 Nota sobre o **sbuild** . . . 59
9.8 Casos especiais de compilação . . . 59
9.9 Enviar **orig.tar.xz** . . . 60
9.10 Envios saltados . . . 60
9.11 Relatórios de bug . . . 60

10 Empacotamento avançado **62**
10.1 Perspetiva histórica . . . 62
10.2 Tendências actuais . . . 62
10.3 Nota sobre sistema de compilação . . . 63
10.4 Integração contínua . . . 63
10.5 Bootstrapping . . . 63
10.6 Endurecimento de compilação . . . 64
10.7 Compilação reproduzível . . . 64
10.8 Substvar . . . 64
10.9 Pacote de biblioteca . . . 65
10.10 Multiarch . . . 66
10.11 Divisão de um pacote binário Debian . . . 66
10.12 Cenário de divisão de pacote e exemplos . . . 67
10.13 Caminho da biblioteca Multiarch . . . 67
10.14 Caminho do ficheiro de cabeçalho Multiarch . . . 68
10.15 Caminho do ficheiro ***.pc** Multiarch . . . 68
10.16 Símbolos de biblioteca . . . 68
10.17 Nome de pacote da biblioteca . . . 69
10.18 Transição de biblioteca . . . 70
10.19 binNMU seguro . . . 70
10.20 Informação de depuração . . . 71
10.21 Pacote **-dbgsym** . . . 71
10.22 **debconf** . . . 71

11 Empacotar com git **73**
11.1 Repositório Salsa . . . 74
11.2 Configuração da conta Salsa . . . 74
11.3 Serviço CI de Salsa . . . 74
11.4 Nomes de ramos: . . . 74
11.5 Repositório Git de patch não-aplicada . . . 75
11.6 Patch por abordagem “**gbp-pq**” . . . 75
11.7 Gerir a lista de patch com **gbp-pq** . . . 76
11.8 **gbp import-dscs --debsnap** . . . 76
11.9 Note sobre **gbp** . . . 77
11.10 O navegador do repositório Git . . . 77
11.11 Organização de histórico de submetidos Git . . . 77
11.12 Empacotamento Debian **Quasi-native** . . . 78
11.13 Repositório Git de patch aplicada . . . 78
11.14 Nota sobre **dgit** . . . 79

12 Dicas **81**
12.1 Compilar sob UTF-8 . . . 81
12.2 Conversão UTF-8 . . . 81
12.3 Dicas para Depuração . . . 81

13 Utilizações de Ferramenta **84**
13.1 **debdiff** . . . 84
13.2 **dget** . . . 84
13.3 **mk-origtargz** . . . 85
13.4 **origtargz** . . . 85
13.5 **git deborig** . . . 85
13.6 **dpkg-source -b** . . . 85
13.7 **dpkg-source -x** . . . 85
13.8 **debc** . . . 85
13.9 **bts** . . . 86
13.10 **dpkg-depcheck** . . . 86

14 Mais Exemplos **87**
14.1 Modelos Cherry-pick . . . 87
14.2 Nenhum Makefile (shell, CLI) . . . 89
14.3 Makefile (shell, CLI) . . . 96
14.4 pyproject.toml (Python3, CLI) . . . 99
14.5 Makefile (shell, GUI) . . . 105
14.6 pyproject.toml (Python3, GUI) . . . 108
14.7 Makefile (pacote singular-binário) . . . 111
14.8 Makefile.in + configure (pacote singular-binário) . . . 114
14.9 Autotools (pacote singular-binário) . . . 118
14.10 CMake (pacote singular-binário) . . . 122
14.11 Autotools (pacote multi-binário) . . . 126
14.12 CMake (pacote multi-binário) . . . 133
14.13 Internacionalização . . . 138
14.14 Detalhes . . . 144

15 manual do debmake(1) **145**
15.1 NOME . . . 145
15.2 RESUMO . . . 145
15.3 DESCRIÇÃO . . . 145
15.4 Argumentos posicionais . . . 146
15.5 Opções . . . 146
15.6 EXEMPLOS . . . 148
15.7 PACOTES DE AJUDA . . . 148
15.8 ADVERTÊNCIA . . . 148
15.9 DEBUG . . . 149

15.10AUTOR 149
15.11LICENÇA 149
15.12VEJA TAMBÉM 149

16 opções do debmake **150**
16.1 Opção curta (**-i**) 150
16.2 **debmake -b** 150
16.3 **debmake -B** 151
16.4 **debmake -x** 151

Resumo

Este “Guia para Maintainers Debian” (2026-10-08) guia tutorial descreve a compilação do pacote Debian para utilizadores Debian normais e futuros desenvolvedores usando o comando **debmake**.

Este guia foca-se no estilo de empacotamento moderno e vem com muitos exemplos simples.

- Empacotamento por script de shell POSIX
- Empacotamento por script de Python3
- C com Makefile/Autotools/CMake
- vários pacotes binários com biblioteca partilhada etc.

Este “Guia para Debian Maintainers” pode ser considerado o sucessor do “Debian New Maintainers’ Guide”.

Capítulo 1

Prefácio

Se você é um utilizador Debian já com alguma experiência [1], você pode ter encontrado as seguintes situações:

- Você deseja instalar um certo pacote de software que ainda não existe no arquivo Debian.
- Você deseja actualizar um pacote Debian com a versão mais recente do autor.
- Você deseja corrigir bugs num pacote Debian com algumas patches.

Se você quer criar um pacote Debian para preencher essas necessidades e partilhar o seu trabalho com a comunidade, você é a audiência alvo deste guia como um futuro maintainer Debian. [2] Bem vindo à comunidade Debian.

Debian tem muitas regras sociais e técnicas e convenções a seguir, pois é uma grande organização de voluntários com uma história rica. Debian tem também desenvolvido uma extensiva matriz de ferramentas de empacotamento e ferramentas de manutenção de arquivo para compilar conjuntos consistentes de pacotes binários que endereçam muitos objetivos técnicos:

- pacotes têm dependências de pacotes e patches declaradas claramente e compilam corretamente a partir de rascunho num ambiente de compilação limpo (“Secção 6.6”, “Secção 6.9”, “Secção 4.7”)
- compilação de pacotes contra muitas arquitecturas (“Secção 9.3”)
- compilações são reproduzíveis (“Secção 10.7”)
- a multi-arquitectura é suportada (“Secção 10.10”)
- é possível boot strapping para novas arquitecturas (“Secção 10.5”)
- compilações usam bandeiras de compilação específicas para endurecer segurança (“Secção 10.6”)
- pacotes são otimizados ao serem divididos em vários pacotes binários (“Secção 10.11”)
- nomes de bibliotecas e conteúdos são geridos para assegurar transições suaves nas actualizações (“Secção 10.18”)
- instalações usam correntemente perguntas interativas (quando as usam) (“Secção 10.22”)
- integração contínua é usada para garantir qualidade (“Secção 10.4”)
- …

Estes factores podem ser um pouco esmagadores para muitos novos futuros maintainers Debian. Este guia destina-se a fornecer pontos de entrada para os ajudar a começar. Ele cobre o seguinte:

- O que você deve saber antes de se envolver com Debian como um futuro responsável de manutenção (maintainer).

1Você precisa saber um pouco sobre programação de Unix mas não precisa ser muito experiente. Você pode aprender sobre manuseamento básico de um sistema Debian consultando “Debian Reference”. Também contém tópicos a aprender sobre programação de Unix.

2Se você não está interessado em partilhar o pacote Debian, você pode endereçar as suas necessidades locais ao compilar e instalar o pacote fonte de autor corrigido em **/usr/local/**.

- Como é criar um pacote Debian simples.
- Que tipo de regras existem para criar um pacote Debian.
- Dicas para criar o pacote Debian com mínimo esforço.
- Exemplos de criar pacotes Debian em cenários típicos.

O autor reconheceu as limitações ao atualizar o original “Guia de Novos Maintainers” com o pacote **dh-make** e decidiu criar uma ferramenta alternativa com documentação a acompanhar para endereçar os requerimentos modernos como o multi-arch. Isto resultou no pacote **debmake**, inicialmente lançado com versão 4.0 em 2013. a versão actual do **debmake** é 5.1.5. Vem com este “Guia para Maintainers Debian” atualizado no pacote **debmake-doc** (versão: 1.31). (Em 2016, o **dh-make** foi portado de Perl para Python com funcionalidades atualizadas.)

Muitas tarefas e dicas foram integradas no comando **debmake** permitindo que este guia seja conciso. Este guia também oferece muitos exemplos em empacotamento para você começar.

Cuidado



Demora muitas horas a criar e manter apropriadamente pacotes Debian. O maintainer Debian tem de ser **tanto competente como diligente tecnicamente** para aceitar este desafio.

Alguns tópicos importantes são explicados em detalhes. Enquanto alguns deles podem parecer irrelevantes para si, por favor seja paciente. Certos casos de canto são omitidos, e alguns tópicos são apenas cobertos através de referências externas. Estas são escolhas intencionais para manter este guia simples e passível de manter.

Capítulo 2

Visão geral

O empacotamento Debian de *package-1.0*.**tar.xz**, que contém uma fonte C simples seguindo os "Standards de Codificação de GNU" e "FHS", pode ser feito com o comando **debmake** como se segue.

```
[base_dir] $ tar --xz -xvf package-1.0.tar.xz
[base_dir] $ cd package-1.0
[package-1.0] $ debmake
   ... Make manual adjustments of generated configuration files
[package-1.0] $ debuild
```

Se forem saltados ajustes manuais nos ficheiros de configuração gerados, o pacote binário gerado fica com falta duma descrição de pacote significativa mas mesmo assim vai funcionar bem sob o comando **dpkg** para ser usado para a sua implantação local.

> Cuidado
>
> 
>
> O comando **debmake** apenas fornece ficheiros modelo decentes. Estes ficheiros modelo têm de ser ajustados manualmente à sua perfeição para obedecer com os requerimentos de qualidade estritos do arquivo Debian, se o pacote gerado destinar-se para consumo geral.

Se você é novato no empacotamento Debian, foque-se em compreender o processo global em vez de se preocupar com os detalhes.

Se você está familiarizado com empacotamento Debian, vai perceber que o **debmake** é semelhante ao comando **dh_make**. Isto porque o **debmake** foi desenhado para substituir a funcionalidade histórica fornecida pelo **dh_make**. 1

O comando **debmake** é desenhado com as seguintes características:

- estilo de empacotamento moderno
  - **debian/copyright**: de acordo com "DEP-5"
  - **debian/control**: suporte a **substvar**, suporte a **multiarch**, pacotes multi binário, …
  - **debian/rules**: sintaxe **dh**, opções de melhoramento de compilação, …
- flexibilidade
  - muitas opções (veja "Secção 16.2", "Capítulo 15", e "Capítulo 16")
- ações predefinidas sãs
  - executar sem-parar com resultados limpos
  - gerar o pacote multiarch, a menos que a opção **-m** seja especificada explicitamente.
  - gerar o pacote Debian não-nativo com o formato fonte Debian "**3.0 (quilt)**", a menos que a opção **-n** seja especificada explicitamente.

1Antes do **dh_make**, o comando **deb-make** era popular. O pacote actual **debmake** começa a sua versão a partir de **4.0** para evitar conflitos de versão com o pacote **debmake** obsoleto, o qual fornecia o comando "**deb-make**".

O comando **debmake** delega a maioria do trabalho pesado para os seus pacotes back-end: **debhelper**, **dpkg-dev**, **devscripts**, **sbuild**, **schroot**, **licensecheck**, **licenserecon**, etc.

Dica



Assegure-se que cita apropriadamente os argumentos das opções **-b**, **-f**, e **-w** para os proteger de interferências da shell.

Dica



O pacote Debian não-nativo é o pacote Debian normal.

Dica



O registo detalhado de todos os exemplos de compilação de pacote neste documento pode ser obtido seguindo as instruções em “Secção 14.14”.

Capítulo 3

Pré-requisitos

Aqui estão os pre-requisitos que você precisa compreender antes de se envolver com Debian.

3.1 Pessoas em redor de Debian

Existem vários tipos de pessoas a interagir em redor de Debian com diferentes papeis:

- **autor upstream**: a pessoa que fez o programa original.
- **upstream maintainer**: a pessoa que actualmente mantém o programa.
- **maintainer**: a pessoa que faz o pacote Debian do programa.
- **sponsor**: uma pessoa que ajuda os maintainers a enviar pacotes para o arquivo de pacotes oficial de Debian (após verificar o seu conteúdo).
- **mentor**: uma pessoa que ajuda maintainers novatos com empacotamento e etc.
- **Debian Developer** (DD): um membro do projeto Debian com totais direitos de envio para o arquivo de pacotes Debian oficial.
- **Debian Maintainer** (DM): uma pessoa com direitos de envio limitados para o arquivo de pacotes Debian oficial.

Por favor note que você não pode tornar-se um **Debian Developer** (DD) oficial da noite pró dia, pois isso requer mais do que habilidades técnicas. Não fique desencorajado por isto. Se o seu trabalho for útil para outros, você pode à mesma enviar o seu pacote seja como um **maintainer** através dum **sponsor** ou como um **Debian Maintainer**.

Por favor note que você não precisa de criar um pacote novo para se tornar num Debian Developer oficial. Contribuir para os pacotes existentes também pode fornecer um caminho para se tornar num Debian Developer oficial. Existem muitos pacotes à espera de bons maintainers (veja “”`Secção 3.8”`”).

3.2 Como contribuir

Por favor consulte o seguinte para aprender a como contribuir para Debian:

- “Como você pode ajudar Debian?” (oficial)
- “O Debian GNU/Linux FAQ, Capítulo 13 - Contribuir para o projecto Debian” (semi-oficial)
- “Debian Wiki, HelpDebian” (suplementar)
- “Sítio do Novo Membro Debian” (oficial)
- “Debian Mentors FAQ” (suplementar)

3.3 Dinâmicas sociais de Debian

Por favor compreenda as dinâmicas sociais de Debian para se preparar para interagir com Debian:

- Somos todos voluntários.
  - Você não pode impor tarefas aos outros.
  - Você deve ser auto-motivado a fazer as coisas.
- Cooperação amigável é a força motriz.
  - O seu contributo não deve sobre-esforçar os outros.
  - O seu contributo só tem valor quando os outros o apreciam.
- Debian não uma escola onde você obtêm atenção automática dos professores.
  - Você deve ser capaz de aprender muitas coisas independentemente.
  - A atenção de outros voluntários é um recurso escasso.
- Debian está em melhoria constante.
  - Espera-se que você crie pacotes de alta qualidade.
  - Você deve adaptar-se à mudança.

Como nós nos focamos apenas nos aspetos técnicos do empacotamento no resto deste guia, por favor consulte o seguinte para compreender as dinâmicas sociais de Debian:

- "Debian: 17 anos de Software Livre, "do-ocracy", e democracia" (Slides Introdutórios pelos ex-DPL)

3.4 Lembretes técnicos

Aqui estão alguns lembretes técnicos para ajudar outros maintainers a trabalhar no seu pacote facilmente e efectivamente, maximizando o resultado de Debian como um todo.

- Faça o seu pacote fácil de depurar.
  - Mantenha o seu pacote simples.
  - Não torne o seu pacote num engenho-excessivo.
- Mantenha o seu pacote bem documentado.
  - Use estilo de código legível.
  - Faça comentários no código.
  - Formate o código de modo consistente.
  - Manter o repositório git [1] do pacote.

> Nota
>
> 
>
> A depuração do software tende em consumir mais tempo que escrever o software inicial funcional.

Não é inteligente correr o seu sistema base sob a suite **unstable** mesmo para objetivos de desenvolvimento.

1A grande maioria de maintainers Debian usa **git** sobre outros sistemas VCS como os **hg**, **bzr**, etc.

- A criação e verificação de pacotes binário **deb** deve usar um mínimo de chroot **unstable** como descrito em “Secção 4.7”.
- Atividades de desenvolvimento de pacotes interativas básicas devem usar uma chroot **unstable** como descrito em “Secção 4.8”.

Nota



Atividades avançadas de desenvolvimento de pacotes tais como testes de sistemas Desktop completos, daemons de rede, e pacotes de instalador de sistemas, devem usar a suite **unstable** a correr sob “virtualização”.

3.5 Documentação de Debian

Por favor prepare-se para ler a parte pertinente da documentação Debian mais recente para gerar pacotes Debian perfeitos.

- “Manual de Política Debian”
  - As regras “é para seguir” oficiais (https://www.debian.org/doc/devel-manuals#policy)
- “Debian Developer’s Reference”
  - O documento “melhores práticas” oficial (https://www.debian.org/doc/devel-manuals#devref)
- “Guide for Debian Maintainers” — este guia
  - Um documento “referência tutorial” (https://www.debian.org/doc/devel-manuals#debmake-doc)

Todos estes documentos são publicados em https://www.debian.org usando as versões da suite **unstable** dos pacotes Debian correspondentes. Se você deseja ter acesso local a todos estes documentos a partir do seu sistema base, por favor considere usar técnicas como “apt-pinning” e “chroot”.

Se este guia contradizer a documentação Debian oficial, a documentação Debian oficial está correta. Por favor envie um relatório de bug sobre o pacote **debmake-doc** usando o comando **reportbug**.

Aqui estão documentos tutorial alternativos, que pode ler juntamente com este guia:

- “Tutorial de Empacotamento Debian”
  - https://www.debian.org/doc/devel-manuals#packaging-tutorial
  - https://packages.qa.debian.org/p/packaging-tutorial.html
- “Guia de Empacotamento de Ubuntu” (Ubuntu é baseado em Debian.)
  - http://packaging.ubuntu.com/html/
- “Guia de Novos Maintainers Debian” (antecessor deste manual, descontinuado)
  - https://www.debian.org/doc/devel-manuals#maint-guide
  - https://packages.qa.debian.org/m/maint-guide.html

Dica



Ao ler estes tutoriais, pode considerar usar o comando **debmake** em vez do comando **dh_make**.

3.6 Recursos de ajuda

Antes de decidir perguntar a sua questão num lugar público, por favor faça a sua parte ao ler a documentação relevante:

- informação do pacote disponível através dos comandos **aptitude**, **apt-cache**, e **dpkg**.
- ficheiro em **/usr/share/doc/***pacote* para todos os pacotes pertinentes.
- conteúdo de **man** *comando* para todos os comandos pertinentes.
- conteúdo de **info** *comando* para todos os comandos pertinentes.
- o conteúdo de "arquivo de lista de mail debian-mentors@lists.debian.org".
- o conteúdo de "arquivo de lista de mail debian-devel@lists.debian.org".

Você pode encontrar a sua desejada informação ao usar uma string de busca bem-formada como "keyword site:lists.debian.org" para limitar o domínio da busca do motor de busca web.

Criar um pequeno pacote de teste é uma boa maneira de aprender detalhes do empacotamento. Inspecionar pacotes bem mantidos existentes é a melhor maneira de aprender como as outras pessoas criam pacotes.

Se ainda tiver questões sobre empacotamento, você pode pergunta-las interativamente:

- lista de mail debian-mentors@lists.debian.org. (Esta lista de mail é para novatos.)
- lista de mail debian-devel@lists.debian.org. (Esta lista de mail é para experientes.)
- canal IRC tal como #debian-mentors.
- As equipas focam-se num conjunto específico de pacotes. (Lista completa em https://wiki.debian.org/-Teams)
- Listas de mail especificas de linguagem.
  - "debian-devel-{french,italian,portuguese,spanish}@lists.debian.org"
  - "debian-chinese-gb@lists.debian.org" (Esta lista de mail é para discussão geral em Chinês (Simplificado).)
  - "debian-devel@debian.or.jp"

Desenvolvedores Debian mais experientes irão ajuda-lo, se perguntar de modo apropriado após fazer os esforços requeridos.

Cuidado



O desenvolvimento Debian é um alvo em movimento. Alguma informação encontrada na web pode estar desatualizada, incorreta, ou não ser aplicável. Por favor use tal informação com cuidado.

3.7 Situação de arquivo

Por favor perceba a situação do arquivo Debian.

- Debian já tem pacotes para a maioria dos tipos de programas.
- O número de pacotes já existentes no arquivo Debian é várias dezenas de vezes maior que os maintainers ativos.
- Infelizmente, a alguns pacotes falta-lhes um nível apropriado de atenção por parte do maintainer.

Assim, contributos a pacotes que já existem no arquivo são muito mais apreciados (e mais prováveis de receber patrocínios para o envio) pelos outros maintainers.

Dica



O comando **wnpp-alert** do pacote **devscripts** pode procurar por pacotes instalados que estão para ser adotados ou órfãos.

Dica



O pacote **how-can-i-help** pode mostrar oportunidades para contribuir para Debian com base em pacotes instalados localmente.

3.8 Abordagens de contribuição

Aqui está o código pseudo-Python para as suas abordagens de contribuição a Debian com um **programa**:

```
if exist_in_debian(program):
  if is_team_maintained(program):
    join_team(program)
  if is_orphaned(program): # maintainer: Debian QA Group
    adopt_it(program)
  elif is_RFA(program): # Request for Adoption
    adopt_it(program)
  else:
    if need_help(program):
      contact_maintainer(program)
      triaging_bugs(program)
      preparing_QA_or_NMU_uploads(program)
    else:
      leave_it(program)
else: # new packages
  if not is_good_program(program):
    give_up_packaging(program)
  elif not is_distributable(program):
    give_up_packaging(program)
  else: # worth packaging
    if is_ITPed_by_others(program):
      if need_help(program):
        contact_ITPer_for_collaboration(program)
      else:
        leave_it_to_ITPer(program)
    else: # really new
      if is_applicable_team(program):
        join_team(program)
      if is_DFSG(program) and is_DFSG(dependency(program)):
        file_ITP(program, area="main") # This is Debian
      elif is_DFSG(program):
        file_ITP(program, area="contrib") # This is not Debian
      else: # non-DFSG
        file_ITP(program, area="non-free") # This is not Debian
      package_it_and_close_ITP(program)
```

Aqui:

- Para exist_in_debian(), e is_team_maintained(); veja:
  - o comando **aptitude**
  - página web "Pacotes Debian"
  - Página "Equipas" do Debian wiki
- Para is_orphaned(), is_RFA(), e is_ITPed_by_others(); veja:
  - O resultado do comando **wnpp-alert**.
  - "Pacotes Prospectivos e Com-Falta-de-Mão-de-Obra"
  - "Registos de relatórios de Bug Debian: Bugs em pseudo-pacote wnpp em unstable"
  - "Pacotes Debian que precisam de Amor"
  - "Navegar em bugs wnpp baseados em debtags"
- Para is_good_program(), veja:
  - O programa deve ser útil.
  - O programa não deve introduzir preocupações de segurança e manutenção no sistema Debian.
  - O programa precisa estar bem documentado e o seu código precisa ser compreensível (isto é, não ofuscado).
  - O autor do programa concorda com o empacotamento e é amistoso de Debian. 2
- Para is_it_DFSG(), e is_its_dependency_DFSG(); veja:
  - "Directrizes de Software Livre Debian" (DFSG).
- Para is_it_distributable(), veja:
  - O software tem de ter uma licença e esta deve permitir a sua distribuição.

Você ou precisa de preencher um **ITP** ou adotar um pacote para começar a trabalhar nele. Veja "Debian Developer's Reference":

- "5.1. Novos pacotes".
- "5.9. Mover, remover, renomear, orfanar, adotar, e reintroduzir pacotes".

3.9 Contribuidor e maintainer novato

O contribuidor e maintainer novato pode se questionar sobre o que aprender para começar o seu contributo a Debian. Aqui estão as minhas sugestões dependendo do seu objectivo.

- Empacotamento
  - Bases de **shell POSIX** e **make**.
  - Algum conhecimento rudimentar de **Perl** e **Python**.
- Tradução
  - Bases sobre como o sistema de tradução baseada em PO funciona.
- Documentação
  - Bases sobre marcações de texto (XML, ReST, Wiki, …).

O contribuidor e maintainer novato pode questionar onde começar a sua contribuição para Debian. Aqui estão algumas sugestões que dependem das suas habilidades:

- Habilidades **shell POSIX**, **Perl**, e **Python**:

2Isto não é um requerimento absoluto. O autor hostil por tornar-se num grande consumo de recursos para todos nós. O autor amigável pode ser consultado para resolver quaisquer problemas com o programa.

  - Enviar patches para o Instalador Debian.
  - Enviar patches para os scripts de ajuda em empacotamento Debian como os **devscripts**, **sbuild**, **schroot**, etc. mencionados neste documento.

- Habilidades **C** e **C++**:
  - Enviar patches para os pacotes com a prioridade **required** e **important**.
- Habilidades Não-Inglês:
  - Enviar patches para o ficheiro PO do Instalador Debian.
  - Enviar patches para ficheiros PO dos pacotes com a prioridade **required** e **important**.
- Habilidades de documentação:
  - Actualizar conteúdo da “Debian Wiki”.
  - Enviar patches para a “Documentação Debian” existente.

Esta atividades devem dar-lhe boa exposição às outras pessoas de Debian para estabilizar a sua credibilidade.

O maintainer novato deve evitar empacotar programas com exposição alta de segurança:

- programa **setuid** ou **setgid**
- programa **daemon**
- programa instalado nos directórios **/sbin/** ou **/usr/sbin/**

Quando você ganhar mais experiência em empacotamento, você vai ser capaz de empacotar tais programas.

Capítulo 4

Configurações de Ferramenta

O pacote **build-essential** tem de ser instalado no ambiente de compilação.

O pacote **devscripts** deve ser instalado no ambiente de desenvolvimento do maintainer.

É boa ideia instalar e configurar todos os conjuntos de pacotes populares mencionados neste capítulo. Isto permite-nos partilhar uma base comum de ambiente de trabalho, apesar de este não ser um requerimento absolutamente necessário.

Por favor considere também instalar as ferramentas mencionadas em "Overview of Debian Maintainer Tools" no "Debian Developer's Reference", como necessário.

> **Cuidado**
>
> 
>
> As configurações de ferramenta apresentadas aqui servem apenas de exemplo e podem não estar atualizadas com os pacotes mais recentes no sistema. O desenvolvimento Debian é um alvo em movimento. Por favor certifique-se de ler a documentação pertinente e atualize a configuração se necessário.

4.1 Configuração do Email

Várias ferramentas de manutenção Debian reconhecem o seu endereço de email e nome a usar pelas variáveis de ambiente da shell **$DEBEMAIL** e **$DEBFULLNAME**.

Vamos configurar essas variáveis de ambiente ao adicionar as seguintes linhas ao **\~/.bashrc** 1.

Adicione ao ficheiro ~/.bashrc

```
DEBEMAIL="osamu@debian.org"
DEBFULLNAME="Osamu Aoki"
export DEBEMAIL DEBFULLNAME
```

> **Nota**
>
> 
>
> O que está em cima é para o autor deste manual. Os exemplos de configuração e operação apresentados neste manual usam essas definições de endereço de email e nome. Você tem de usar o seu endereço de email e seu nome para o seu sistema.

1Isto assume que você está a usar Bash como shell de login. Se está a usar outra shell de login como a shell Z, use os seus ficheiros de configuração correspondentes em vez de **\~/.bashrc**.

4.2 Configuração do mc

O comando **mc** oferece maneiras muito fáceis de gerir ficheiros. Ele pode abrir o ficheiro binário **deb** para verificar o seu conteúdo ao pressionar a tecla Enter sobre o ficheiro binário **deb**. Ele usa o comando **dpkg-deb** como seu back-end. Vamos configura-lo para usar **chdir** fácil como se segue.

Adicione ao ficheiro ~/.bashrc

```
# mc related
if [ -f /usr/lib/mc/mc.sh ]; then
  . /usr/lib/mc/mc.sh
fi
```

4.3 Configuração do git

Hoje em dia, o comando **git** e a ferramenta essencial para gerir a árvore fonte com histórico.

A configuração de utilizador global para o comando **git** como o seu nome e endereço de email pode ser definida em **~/.gitconfig** como se segue.

```
[~] $ git config --global user.name "Osamu Aoki"
[~] $ git config --global user.email osamu@debian.org
```

Se você está muito acostumado aos comandos do CVS ou Subversion, pode desejar definir vários nomes alternativos de comandos como se segue.

```
[~] $ git config --global alias.ci "commit -a"
[~] $ git config --global alias.co checkout
```

Pode verificar a sua configuração global como se segue.

```
[~] $ git config --global --list
```

Dica



É essencial usar algumas ferramentas GUI do git como **gitk** ou **gitg** para trabalha efectivamente com o histórico do repositório git.

4.4 Configuração do quilt

O comando **quilt** oferece um método básico de gravar modificações. Para o empacotamento Debian, deve ser personalizado para guardar modificações no directório **debian/patches/** em vez de no seu directório predefinido **patches/**.

De modo a evitar alterar o comportamento do próprio comando **quilt**, vamos criar um nome alternativo **dquilt** para o empacotamento Debian ao adicionar as seguintes linhas ao ficheiro **~/.bashrc**. A segunda linha fornece a mesma funcionalidade de completação da shell do comando **quilt** para o comando **dquilt**.

Adicione ao ficheiro ~/.bashrc

```
alias dquilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
. /usr/share/bash-completion/completions/quilt
complete -F _quilt_completion $_quilt_complete_opt dquilt
```

Depois vamos criar **~/.quiltrc-dpkg** como se segue.

```
d=.
while [ ! -d $d/debian -a `readlink -e $d` != / ];
    do d=$d/..; done
```

```
if [ -d $d/debian ] && [ -z $QUILT_PATCHES ]; then
    # if in Debian packaging tree with unset $QUILT_PATCHES
    QUILT_PATCHES="debian/patches"
    QUILT_PATCH_OPTS="--reject-format=unified"
    QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
    QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
    QUILT_COLORS="diff_hdr=1;32:diff_add=1;34:diff_rem=1;31:diff_hunk=1;33:"
    QUILT_COLORS="${QUILT_COLORS}diff_ctx=35:diff_cctx=33"
    if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi
```

Veja **quilt**(1) e “Como Sobreviver Com Muitas Patches ou Introdução ao Quilt (**quilt.html**)” sobre como usar o comando **quilt**.

Veja “Secção 5.9” para exemplos de utilização.

Note que o “gbp pq” é capaz de consumir **debian/patches** existentes, automatizar a actualização e modificação das patches, e exporta-las de volta para **debian/patches**, tudo sem usar o quilt nem a necessidade de prender ou configurar quilt.

4.5 Configuração do openPGP

O comando **gpg**(1), incluído no pacote **gnupg**, pode ser usado para gerar uma nova chave openPGP para a atividade de pacote Debian como se segue.

```
[~] $ gpg --generate-key
gpg (GnuPG) 2.4.7; Copyright (C) 2024 g10 Code GmbH
... [snip] ...
public and secret key created and signed.

pub   ed25519 2026-02-10 [SC] [expires: 2029-02-09]
      ABCDEF0123456789ABCDEF0123456789ABCDEF01
uid                      Osamu Aoki  <osamu@debian.org>
sub   cv25519 2026-02-10 [E] [expires: 2029-02-09]
```

Esta é uma chave openPGP baseada na Criptografia de curva elíptica (ECC).

> Nota
>
> 
>
> Se você já tem uma chave openPGP baseada no Sistema de criptografia RSA assinada por muitas outras, pode usa-la.

Pode editar e atualizar esta chave openPGP com:

```
[~] $ gpg --edit-key ABCDEF0123456789ABCDEF0123456789ABCDEF01
... [snip] ...
gpg> save
```

Por exemplo, a chave openPGP resultante poderá se parecer com:

```
[~] $ gpg --list-keys ABCDEF0123456789ABCDEF0123456789ABCDEF01
pub   ed25519 2026-02-10 [SC]
      ABCDEF0123456789ABCDEF0123456789ABCDEF01
uid           [ultimate] Osamu Aoki <osamu@debian.org>
sub   cv25519 2026-02-10 [E] [expires: 2030-02-10]
```

Nota



O termo “chave GPG” encontrado em muitos documentos de Debian pode ser considerado um sinónimo de “chave openPGP”.

Para mais sobre chaves OpenPGP, veja:

- “Criando uma nova chave GPG”
- “Debian wiki: Assinatura por chave”
- “Debian wiki: Usar sub-chaves OpenPGP no desenvolvimento de Debian”
- “Debian wiki: FixKeyWithSha1” — atualizar chaves OpenPGP antigas

4.6 Configuração do devscripts

O comando **debsign**, incluído no pacote **devscripts**, é usado para assinar o pacote Debian com a sua chave openPGP privada.

O comando **debuild**, incluído no pacote **devscripts**, compila o pacote binário e verifica-o com o comando **lintian**. É útil ter resultados detalhados gerados do comando **lintian**.

Você pode configurar estes no **~/.devscripts** como se segue.

```
DEBUILD_DPKG_BUILDPACKAGE_OPTS="-i -I -us -uc"
DEBUILD_LINTIAN_OPTS="-i -I --show-overrides"
DEBSIGN_KEYID="Your_openPGP_keyID"
```

As opções **-i** e **-I** em **DEBUILD_DPKG_BUILDPACKAGE_OPTS** para o comando **dpkg-source** ajuda na recompilação de pacotes Debian sem conteúdos estranhos (veja “Capítulo 8”).

4.7 Configuração do sbuild

O pacote **sbuild** fornece um ambiente de compilação (“chroot”) de sala limpa. Oferece isto eficientemente com a ajuda do **schroot** usando a funcionalidade bind-mount do kernel Linux moderno.

Como é o mesmo ambiente de compilação que a infraestrutura buildd de Debian, está sempre atualizado e vem completo de funcionalidades úteis.

Pode ser personalizado para oferecer as seguintes funcionalidades:

- O pacote **schroot** para ampliar a velocidade de criação da chroot.
- O pacote **lintian** para encontrar bugs no pacote.
- O pacote **piuparts** para encontrar bugs no pacote.
- O pacote **autopkgtest** para encontrar bugs no pacote.
- O pacote **ccache** para ampliar a velocidade do **gcc**. (opcional)
- O pacote **libeatmydata1** para ampliar a velocidade do **dpkg**. (opcional)
- O **make** paralelo para ampliar a velocidade da compilação. (opcional)

Vamos configurar o ambiente **sbuild** 2:

```
[~] $ sudo apt install sbuild piuparts autopkgtest lintian
[~] $ sudo apt install sbuild-debian-developer-setup
[~] $ sudo sbuild-debian-developer-setup -s unstable
```

Vamos atualizar a sua adesão aos grupos reiniciando todo o sistema 3:

2Tenha cuidado pois alguns HOWTOs antigos podem usar diferentes configurações de chroot.

3Sob alguns ambientes GUI Desktop modernos, “terminar e iniciar sessão usando a GUI” pode não atualizar o seu ambiente de sistema como a adesão a grupos.

```
[~] $ sudo shutdown -r now
```

Vamos verificar se a sua adesão a grupos inclui o **sbuild**:

```
[~] $ id
uid=1000(<yourname>) gid=1000(<yourname>) groups=...,132(sbuild)
```

Vamos criar o ficheiro de configuração **~/.local/sbuild/config.pl** alinhado com a prática Debian recente de "envio-apenas-fonte" como:

```
[~] $ cat >~/.local/sbuild/config.pl << 'EOF'
################################################################################
# PACKAGE BUILD RELATED (source-only-upload as default)
################################################################################
# -d
$distribution = 'unstable';
# -A
$build_arch_all = 1;
# -s
$build_source = 1;
# --source-only-changes
$source_only_changes = 1;
# -v
$verbose = 1;

################################################################################
# POST-BUILD RELATED (turn off functionality by setting variables to 0)
################################################################################
$run_lintian = 1;
$lintian_opts = ['-i', '-I'];
$run_piuparts = 1;
$piuparts_opts = ['--schroot', 'unstable-amd64-sbuild'];
$run_autopkgtest = 1;
$autopkgtest_root_args = '';
$autopkgtest_opts = [ '--', 'schroot', '%r-%a-sbuild' ];

################################################################################
# PERL MAGIC
################################################################################
1;
EOF
```

Nota



Existem alguns casos excecionais como NOVOS envios, envios com NOVOS pacotes binários, e envios de segurança onde não vai poder fazer envio-apenas-fonte mas será necessário enviar com pacotes binários. A configuração de cima precisa de ser ajustada para esses casos excecionais.

Dica



Poderá precisar de adicionar "**$chroot_mode = "schroot";**" a **~/.local/sbuild/config.pl** para **piuparts** se não funcionar bem sob **unshare**. Veja os bugs Debian: #1125784 e #1126127.

O documento seguinte assume que o **sbuild** está configurado deste modo.

Edite isto à sua necessidade. Os testes pós-compilação pode ser ligados e desligados ao atribuir 1 ou 0 às variáveis correspondentes,

> **Atenção**
>
> 
>
> A personalização opcional pode causar efeitos negativos. Em caso de dúvidas. desative-a.

> **Nota**
>
> 
>
> O **make** paralelo pode falhar para alguns pacotes existentes e pode tornar o relatório de compilação difícil de ler.

> **Dica**
>
> 
>
> Muitas dicas relacionadas com **sbuild** estão disponíveis em “Secção 9.7” e “https://wiki.debian.org/sbuild” .

4.8 Configuração chroot persistente

> **Nota**
>
> 
>
> O uso de sistema de ficheiros chroot copiado independente previne a contaminação do chroot fonte usado pelo **sbuild**.

Para compilar novos pacotes experimentais ou para depurar pacotes bugados, vamos configurar um chroot persistente dedicado “**source:unstable-amd64-desktop**” ao:

```
[~] $ sudo cp -a /srv/chroot/unstable-amd64-sbuild /srv/chroot/unstable-amd64- ↩
    desktop
[~] $ sudo tee /etc/schroot/chroot.d/unstable-amd64-desktop-XXXXXX << EOF
[unstable-amd64-desktop]
description=Debian sid/amd64 persistent chroot
groups=root,sbuild
root-groups=root,sbuild
profile=desktop
type=directory
directory=/srv/chroot/unstable-amd64-desktop
union-type=overlay
EOF
```

Aqui, o perfil **desktop** é usado em vez do perfil **sbuild**. Por favor certifique que ajusta **/etc/schroot/desktop/fstab** para tornar o pacote fonte acessível a partir do interior da chroot.

Você pode iniciar sessão nesta chroot “**source:unstable-amd64-desktop**” com:

```
[~] $ sudo schroot -c source:unstable-amd64-desktop
```

4.9 Configuração do gbp

O pacote **git-buildpackage** oferece o comando **gbp**(1). O seu ficheiro de configuração de utilizador é **~/.gbp.conf**.

```
# Configuration file for "gbp <command>"

[DEFAULT]
# the default build command:
builder = sbuild
# use pristine-tar:
pristine-tar = True
# Use color when on a terminal, alternatives: on/true, off/false or auto
color = auto
```

4.10 Proxy HTTP

Você deve configurar um proxy de cache HTTP local para poupar na largura de banda para o acesso ao repositório de pacotes Debian. Existem várias escolhas:

- O proxy de cache HTTP especializado usando o pacote **apt-cacher-ng**.
- O proxy de cache HTTP genérico (pacote **squid**) configurado pelo pacote **squid-deb-proxy**

De modo a usar este proxy HTTP sem ajustes manuais de configuração, é boa ideia instalar o pacote **auto-apt-proxy** ou **squid-deb-proxy-client** em todo lado.

4.11 Repositório Debian privado

Você pode definir um repositório de pacotes Debian privado com o pacote **reprepro**.

4.12 Máquinas virtuais

Para testar aplicações GUI, é boa ideia ter máquinas virtuais. Instale os pacotes **virt-manager** e **qemu-kvm**.

O uso de chroot e máquinas virtuais permite-nos não termos de actualizar o PC anfitrião para suite **unstable** mais recente.

4.13 Rede local com máquinas virtuais

De modo a aceder facilmente a máquinas virtuais numa rede local, é boa ideia configurar uma infraestrutura de descoberta de serviços DNS multicast ao instalar o **avahi-utils**.

Para todas as máquinas virtuais a correr e o PC anfitrião, nós podemos usar cada nome de máquina acrescentado com **.local** para o SSH para aceder a cada uma.

Capítulo 5

Empacotamento simples

Existe um velho ditado Latino que diz: “**Longum iter est per praecepta, breve et efficax per exempla**” (“É um longo caminho pelas regras, mas curto e eficiente com exemplos”).

5.1 Empacotar o tarball

Aqui está um exemplo de criar um pacote Debian simples a partir duma fonte C simples usando o **Makefile** como seu sistema de compilação.

Vamos assumir que o tarball do autor seja **debhello-0.0.tar.xz**.

Este tipo de fonte destina-se a ser instalado como ficheiro não-sistema como:

Bases para a instalação a partir do tarball do autor

```
[base_dir] $ tar --xz -xmf debhello-0.0.tar.xz
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ make
[debhello-0.0] $ make install
```

O empacotamento Debian requer alterar este processo “**make install**” para instalar ficheiros na localização imagem do sistema alvo em vez de na localização normal sob **/usr/local**.

Nota



Exemplos de criar um pacote Debian a partir de outros sistemas de compilação complicados estão descritos em “Capítulo 14”.

5.2 O grande panorama

O grande panorama para compilar um pacote Debian não-nativo singular a partir do tarball de autor **debhello-0.0.tar.xz** pode ser resumido como:

- O maintainer obtém o tarball do autor **debhello-0.0.tar.xz** e descompacta-o para o directório **debhello-0.0**.
- O comando **debmake** debianiza a árvore fonte do autor ao adicionar ficheiros modelo apenas no directório **debian**.
  - O link simbólico **debhello_0.0.orig.tar.xz** é criado apontando para o ficheiro **debhello-0.0.tar.xz**.
  - O maintainer personaliza os ficheiros modelo.
- O comando **debuild** compila o pacote binário a partir da árvore fonte debianizada.
  - É criado **debhello-0.0-1.debian.tar.xz** contendo o directório **debian**.

O grande panorama de compilação de pacote

```
[base_dir] $ tar --xz -xmf debhello-0.0.tar.xz
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ debmake
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
   ...
[debhello-0.0] $ ... manual customization of debian/* files
[debhello-0.0] $ debuild
   ...
```

Dica



O comando **debuild** neste e nos exemplos seguintes pode ser substituído por comandos equivalentes tais como o comando **sbuild**.

Dica



Se o tarball de autor estiver disponível em formato **.tar.xz** use-o em vez dos que têm formatos em **.tar.gz** e **.tar.bz2**. O formato de compressão **xz** oferece melhor compressão que a compressão do **gzip** e **bzip2**.

5.3 O que é debmake?

Nota



As atividades de empacotamento actuais são muitas vezes executadas manualmente sem se usar o **debmake** enquanto se referencia apenas pacotes existentes semelhantes e o “Manual de Política Debian”.

O comando **debmake** é o script de ajuda para o empacotamento Debian. (“Capítulo 15”)

- Cria bons ficheiros modelo para os pacotes Debian.
- Ele sempre define a maioria dos estados de opção e valores óbvios para predefinições razoáveis.
- Ele gera o tarball do autor e o seu link simbólico necessário se estes estiverem em falta.
- Ele não sobrescreve os ficheiros de configuração existentes no directório **debian/**.
- Ele suporta o pacote **multiarch**.
- Fornece textos de licença curtos extraídos como **debian/copyright** usando **licensecheck** para ajudar na revisão da licença.

Estas funcionalidades tornam o empacotamento Debian com **debmake** simples e moderno.

Em retrospetiva, Eu criei o **debmake** para simplificar esta documentação. Eu considero o **debmake** sendo mais ou menos um gerador de sessão de demonstração para objetivos de tutorial.

O comando **debmake** não é o único script ajudante para criar um pacote Debian. Se você está interessado em ferramentas de ajuda de empacotamento alternativas, por favor veja:

- Debian wiki: "AutomaticPackagingTools" — Comparação extensiva de scripts de ajuda no empacotamento
- Debian wiki: "CopyrightReviewTools" — Comparação extensiva de scripts de ajuda de revisão de copyright

5.4 O que é debuild?

Aqui está um sumário de comandos semelhantes ao comando **debuild**.

- O ficheiro **debian/rules** define como o pacote binário Debian é compilado.
- O comando **dpkg-buildpackage** é o comando oficial para compilar o pacote binário Debian. Para compilação binária normal, ele executa aproximadamente:
  - "**dpkg-source --before-build**" (aplica as patches Debian, a menos que estejam já aplicadas)
  - "**fakeroot debian/rules clean**"
  - "**dpkg-source --build**" (compila o pacote fonte Debian)
  - "**fakeroot debian/rules build**"
  - "**fakeroot debian/rules binary**"
  - "**dpkg-genbuildinfo**" (gera um ficheiro ***.buildinfo**)
  - "**dpkg-genchanges**" (gera um ficheiro ***.changes**)
  - "**fakeroot debian/rules clean**"
  - "**dpkg-source --after-build**" (retira as patches Debian, se elas foram aplicadas durante **--before-build**)
  - "**debsign**" (assina os ficheiros ***.dsc** e ***.changes**)
    - Se você seguiu "Secção 4.6" para definir as opções **-us** e **-uc**, este passo é saltado e você tem de correr o comando **debsign** manualmente.
- O comando **debuild** é um script invólucro do comando **dpkg-buildpackage** para compilar o pacote binário Debian sob as variáveis de ambiente apropriadas.
- O comando **sbuild** é um script invólucro para compilar o pacote binário Debian sob o ambiente chroot apropriado e com as variáveis de ambiente apropriadas.

Nota



Veja **dpkg-buildpackage**(1) para detalhes exatos.

5.5 Passo 1: Obter a fonte do autor

Vamos obter a fonte do autor.

Download debhello-0.0.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-0.0.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-0.0.tar.xz
[base_dir] $ tree
.
+-- debhello-0.0
|   +-- Makefile
|   +-- README.md
```

```
|   +-- src
|       +-- hello.c
+-- debhello-0.0.tar.xz

3 directories, 4 files
```

Aqui, a fonte C **hello.c** é uma muito simples.

hello.c

```
[base_dir] $ cat debhello-0.0/src/hello.c
#include <stdio.h>
int
main()
{
        printf("Hello, world!\n");
        return 0;
}
```

Aqui, o **Makefile** suporta "GNU Coding Standards" e "FHS". Notavelmente:

- compila binários honrando **$(CPPFLAGS)**, **$(CFLAGS)**, **$(LDFLAGS)**, etc.
- instala ficheiros com **$(DESTDIR)** definida para a imagem de sistema alvo
- instala ficheiros com **$(prefix)** definido, os quais podem ser sobrepostos para ser **/usr**

Makefile

```
[base_dir] $ cat debhello-0.0/Makefile
prefix = /usr/local

all: src/hello

src/hello: src/hello.c
        @echo "CFLAGS=$(CFLAGS)" | \
                fold -s -w 70 | \
                sed -e 's/^/# /'
        $(CC) $(CPPFLAGS) $(CFLAGS) $(LDCFLAGS) -o $@ $^

install: src/hello
        install -D src/hello \
                $(DESTDIR)$(prefix)/bin/hello

clean:
        -rm -f src/hello

distclean: clean

uninstall:
        -rm -f $(DESTDIR)$(prefix)/bin/hello

.PHONY: all install clean distclean uninstall
```

Nota



O **echo** da variável **$(CFLAGS)** é usado para verificar a definição apropriada da bandeira de compilação no exemplo seguinte:

5.6 Passo 2: Gerar ficheiros modelo com o debmake

Os resultados do comando **debmake** é muito detalhado e explica o que faz como se segue.

O resultado do comando debmake com a opção -x1

```
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ debmake -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-0.0] $ cd ..
I: Non-native Debian package pkg="debhello", ver="0.0", rev="1" method="dir_d...
I: already in the package-version form: "debhello-0.0"
I: [base_dir] $ ln -sf 'debhello-0.0.tar.xz' 'debhello_0.0.orig.tar.xz'
I: [base_dir] $ cd debhello-0.0
I: parsing option -b ""
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: build_type = make
I: ext_type = c                       1 files
I: ext_type = md                      1 files
I: creating debian/* files with "-x 1" option
I: [debhello-0.0] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
I: creating debian/source/format from extra0source_format
I: creating debian/README.Debian from extra1_README.Debian
I: creating debian/README.source from extra1_README.source
I: creating debian/clean from extra1_clean
I: creating debian/dirs from extra1_dirs
I: creating debian/docs from extra1_docs
I: creating debian/examples from extra1_examples
I: creating debian/gbp.conf from extra1_gbp.conf
I: creating debian/links from extra1_links
I: creating debian/manpages from extra1_manpages
I: creating debian/salsa-ci.yml from extra1_salsa-ci.yml
I: creating debian/watch from extra1nn_watch
I: creating debian/tests/control from extra1tests_control
I: creating debian/upstream/metadata from extra1upstream_metadata
I: creating debian/patches/series from extra1patches_series
I: creating debian/install from extra1bin_install
I: [debhello-0.0] $ wrap-and-sort -ast
I: debian/* may have a blank line at the top.
```

O comando **debmake** gera todos estes ficheiros modelo baseado nas opções da linha de comandos. Como nenhuma opção é especificada, o comando **debmake** escolhe valores predefinidos razoáveis por si:

- O nome do pacote fonte: **debhello**
- A versão do autor: **0.0**
- O nome do pacote binário: **debhello**
- A revisão Debian: **1**
- O tipo de pacote: **bin** (o pacote binário executável ELF)
- A opção **-x**: **-x1** (sem suportes para simplicidade de script de maintainer)

Nota



Aqui, o comando **debmake** é invocado com a opção **-x1** para manter este tutorial simples. O uso da opção predefinida **-x2** ou a mais extensiva **-x3** é altamente recomendado.

Vamos inspecionar os ficheiros modelo gerados.

A árvore fonte após a execução básica do debmake.

```
[debhello-0.0] $ cd ..
[base_dir] $ tree
.
+-- debhello-0.0
|   +-- Makefile
|   +-- README.md
|   +-- debian
|   |   +-- README.Debian
|   |   +-- README.source
|   |   +-- changelog
|   |   +-- clean
|   |   +-- control
|   |   +-- copyright
|   |   +-- dirs
|   |   +-- docs
|   |   +-- examples
|   |   +-- gbp.conf
|   |   +-- install
|   |   +-- links
|   |   +-- manpages
|   |   +-- patches
|   |   |   +-- series
|   |   +-- rules
|   |   +-- salsa-ci.yml
|   |   +-- source
|   |   |   +-- format
|   |   +-- tests
|   |   |   +-- control
|   |   +-- upstream
|   |   |   +-- metadata
|   |   +-- watch
|   +-- src
|       +-- hello.c
+-- debhello-0.0.tar.xz
+-- debhello_0.0.orig.tar.xz -> debhello-0.0.tar.xz

8 directories, 25 files
```

O ficheiro **debian/rules** é o script de compilação fornecido pelo maintainer do pacote. Aqui está o seu ficheiro modelo gerado pelo comando **debmake**.

debian/rules (ficheiro modelo):

```
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
```

```
#
# See FEATURE AREAS/ENVIRONMENT in dpkg-buildflags(1)
# Apply all hardening options
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
# Package maintainers to append CFLAGS
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
# Package maintainers to append LDFLAGS
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1
#
# With debhelper version 9 or newer, the dh command exports
# all buildflags.  So there is no need to include the
# /usr/share/dpkg/buildflags.mk file here if compat is 9 or newer.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@

# debmake generated override targets
# Use "make prefix=/usr" (override prefix=/usr/local in Makefile)
#override_dh_auto_install:
#       dh_auto_install -- prefix=/usr

# Do not install python .pyc .pyo if they exist
#override_dh_install:
#       dh_install --list-missing -X.pyc -X.pyo

# Multiarch package requires library files to be installed to
# /usr/lib/<triplet>/ .  If the build system does not support
# $(DEB_HOST_MULTIARCH), you may need to override some targets such as
# dh_auto_configure or dh_auto_install to use $(DEB_HOST_MULTIARCH) .
```

Isto é essencialmente o ficheiro **debian/rules** standard com o comando **dh**. (Existem alguns conteúdos comentados para você personalizar.)

O ficheiro **debian/control** fornece os meta-dados principais para o pacote Debian. Aqui está o seu ficheiro modelo gerado pelo comando **debmake**.

debian/control (ficheiro modelo):

```
[debhello-0.0] $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.3
```

```
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no
#Vcs-Git: https://salsa.debian.org/debian/debhello.git
#Vcs-Browser: https://salsa.debian.org/debian/<project_site>

Package: debhello
Section: unknown
Architecture: any
Multi-Arch: foreign
Depends:
 ${misc:Depends},
 ${shlibs:Depends},
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.
 .
 ===== This comes from the unmodified template file =====
 .
 Please edit this template file (debian/control) and other package files
 (debian/*) to make them meet all the requirements of the Debian Policy
 before uploading this package to the Debian archive.
 .
 See
  * https://www.debian.org/doc/manuals/developers-reference/best-pkging-pract...
  * https://www.debian.org/doc/manuals/debmake-doc/ch05.en.html#control
 .
 The synopsis description at the top should be about 60 characters and
 written as a phrase.  No extra capital letters or a final period.  No
 articles b''—b'' "a", "an", or "the".
 .
 The package description for general-purpose applications should be
 written for a less technical user.  This means that we should avoid
 jargon.  GNOME or KDE is fine but GTK+ is probably not.
 .
 Use the canonical forms of words:
  * Use X Window System, X11, or X; not X Windows, X-Windows, or X Window.
  * Use GTK+, not GTK or gtk.
  * Use GNOME, not Gnome.
  * Use PostScript, not Postscript or postscript.
```

Atenção



Se você deixar “**Section: unknown**” no ficheiro modelo **debian/control** não modificado, o erro do **lintian** pode fazer a compilação falhar.

Como este é o pacote executável binário ELF, o comando **debmake** define “**Architecture: any**” e “**Multi-Arch: foreign**”. Também, define parâmetros requeridos **substvar** como “**Depends: ${shlibs:Depends}, ${misc:Depends}**”. Estes estão explicados em “Capítulo 6”.

Nota



Por favor note que este ficheiro **debian/control** usa o estilo RFC-822 como documentado em “5.2 Ficheiros de controle de pacote fonte — debian/control” do “manual de Política Debian”. O uso da linha vazia e do espaço inicial tem significado.

O ficheiro **debian/copyright** fornece os dados de sumário de copyright do pacote Debian usando o

comando **licensecheck**.

debian/copyright (ficheiro modelo):

```
[debhello-0.0] $ cat debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: FIXME
Upstream-Contact: FIXME
Source: FIXME
Disclaimer: Autogenerated by licensecheck

Files: ./Makefile
 ./README.md
 ./src/hello.c
Copyright: NONE
License: UNKNOWN
 FIXME
```

5.7 Passo 3: Modificação dos ficheiros modelo

É requerida alguma modificação manual para fazer o pacote Debian apropriado como um maintainer.

De modo a instalar ficheiros como parte dos ficheiros do sistema, o valor **$(prefix)** de **/usr/local** em **Makefile** deve ser sobreposto para ser **/usr**. Isto pode ser acomodado no ficheiro **debian/rules** com o alvo **override_dh_auto_install** a definir "**prefix=/usr**".

debian/rules (versão do maintainer):

```
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-0.0] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
        dh $@

override_dh_auto_install:
        dh_auto_install -- prefix=/usr
```

Exportar a variável de ambiente **DH_VERBOSE** no ficheiro **debian/rules** como em cima força a ferramenta **debhelper** a fazer um relatório de compilação afinado.

Exportar **DEB_BUILD_MAINT_OPTION** como em cima define as opções de endurecimento descritas em "AREAS DE FUNCIONALIDADE/AMBIENTE" em **dpkg-buildflags**(1). 1

Exportar **DEB_CFLAGS_MAINT_APPEND** como em cima força o compilador C a emitir todos os avisos.

Exportar **DEB_LDFLAGS_MAINT_APPEND** como em cima força o vinculador a vincular apenas quando a biblioteca é mesmo necessária. 2

O comando **dh_auto_install** para o sistema de compilação baseado no Makefile essencialmente corre "**$(MAKE) install DESTDIR=debian/debhello**". A criação deste alvo **override_dh_auto_install** muda o seu comportamento para "**$(MAKE) install DESTDIR=debian/debhello prefix=/usr**".

Aqui estão as versões de maintainer dos ficheiros **debian/control** e **debian/copyright**.

debian/control (versão de maintainer):

1Isto é um cliché para forçar uma ligação de re-alocação só-leitura para o endurecimento e para prevenir o aviso do lintian "**W: debhello: hardening-no-relro usr/bin/hello**". Isto não é realmente preciso para este exemplo mas deve ser inofensivo. A ferramenta lintian parece produzir um aviso de falso positivo para este caso que não tem biblioteca vinculada.

2Isto é um cliché para prevenir a sobre-vinculação para casos de dependências complexas de biblioteca como os programas de Gnome. Isto não é realmente preciso para este exemplo simples mas deve ser inofensivo.

```
[debhello-0.0] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-0.0] $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.3
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends:
 ${misc:Depends},
 ${shlibs:Depends},
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

debian/copyright (versão de maintainer):

```
[debhello-0.0] $ vim debian/copyright
 ... hack, hack, hack, ...
[debhello-0.0] $ cat debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: debhello
Upstream-Contact: Osamu Aoki <osamu@debian.org>
Source: https://salsa.debian.org/debian/debmake-doc

Files:     *
Copyright: 2015-2021 Osamu Aoki <osamu@debian.org>
License:   Expat
 Permission is hereby granted, free of charge, to any person obtaining a
 copy of this software and associated documentation files (the "Software"),
 to deal in the Software without restriction, including without limitation
 the rights to use, copy, modify, merge, publish, distribute, sublicense,
 and/or sell copies of the Software, and to permit persons to whom the
 Software is furnished to do so, subject to the following conditions:
 .
 The above copyright notice and this permission notice shall be included
 in all copies or substantial portions of the Software.
 .
 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Vamos remover os ficheiros modelo não usados e editar os restantes ficheiros modelo:

- **debian/README.source**
- **debian/patches/series** (Nenhuma patch do autor)
- **clean**
- **dirs**
- **install**

- **links**

Ficheiros modelo sob debian/. (v=0.0):

```
[debhello-0.0] $ rm -f debian/clean debian/dirs debian/install debian/links
[debhello-0.0] $ rm -f debian/README.source debian/source/*.ex
[debhello-0.0] $ rm -rf debian/patches
[debhello-0.0] $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- docs
+-- examples
+-- gbp.conf
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 14 files
```

Dica



Os ficheiros de configuração usados pelos comandos **dh**_* do pacote **debhelper** geralmente tratam # como o inicio de uma linha comentário.

5.8 Passo 4: Compilar pacote com debuild

Você pode criar um pacote Debian não-nativo usando o comando **debuild** ou os seus equivalentes (veja "Secção 5.4") nesta árvore fonte. O texto resultante do comando é muito detalhado e explica o que ele faz como se segue.

Compilando pacote com debuild

```
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ debuild
 dpkg-buildpackage -us -uc -ui -i
dpkg-buildpackage: info: source package debhello
dpkg-buildpackage: info: source version 0.0-1
dpkg-buildpackage: info: source distribution unstable
dpkg-buildpackage: info: source changed by Osamu Aoki <osamu@debian.org>
 dpkg-source -i --before-build .
dpkg-buildpackage: info: host architecture amd64
 debian/rules clean
dh clean
   dh_auto_clean
        make -j12 distclean
 ...
 debian/rules binary
dh binary
   dh_update_autotools_config
```

```
   dh_autoreconf
   dh_auto_configure
   dh_auto_build
        make -j12 INSTALL="install --strip-program=true"
make[1]: Entering directory '/path/to/base_dir/debhello-0.0'
# CFLAGS=-g -O2 -Werror=implicit-function-declaration
 ...
Finished running lintian.
```

Você pode verificar que **CFLAGS** é atualizado apropriadamente com **-Wall** e **-pedantic** pela variável **DEB_CFLAGS_MAINT_APPEND**.

O manual deve ser adicionado ao pacote como reportado pelo pacote **lintian**, como mostrado em exemplos posteriores (veja “Capítulo 14”). Vamos seguir por agora.

Vamos inspecionar o resultado.

Os ficheiros gerados de debhello versão 0.0 pelo comando debuild:

```
[debhello-0.0] $ cd ..
[base_dir] $ tree -FL 1
./
+-- debhello-0.0/
+-- debhello-0.0.tar.xz
+-- debhello-dbgsym_0.0-1_amd64.deb
+-- debhello_0.0-1.debian.tar.xz
+-- debhello_0.0-1.dsc
+-- debhello_0.0-1_amd64.build
+-- debhello_0.0-1_amd64.buildinfo
+-- debhello_0.0-1_amd64.changes
+-- debhello_0.0-1_amd64.deb
+-- debhello_0.0.orig.tar.xz -> debhello-0.0.tar.xz

2 directories, 9 files
```

Você vê todos os ficheiros gerados.

- O **debhello_0.0.orig.tar.xz** é um link simbólico para o tarball do autor.
- O **debhello_0.0-1.debian.tar.xz** contém os conteúdos gerados pelo maintainer.
- O **debhello_0.0-1.dsc** é o ficheiro de meta-dados para o pacote fonte Debian.
- O **debhello_0.0-1_amd64.deb** é o pacote binário Debian.
- O **debhello-dbgsym_0.0-1_amd64.deb** é o pacote binário de símbolos de depuração Debian. Veja “Secção 10.21”.
- O ficheiro **debhello_0.0-1_amd64.build** é o ficheiro de relatório de compilação.
- O ficheiro **debhello_0.0-1_amd64.buildinfo** é o ficheiro de meta-dados gerado pelo **dpkg-genbuildinfo**(1).
- O **debhello_0.0-1_amd64.changes** é o ficheiro de meta-dados para o pacote binário Debian.

O **debhello_0.0-1.debian.tar.xz** contém as alterações Debian feitas à fonte do autor como se segue.

O conteúdo do arquivo comprimido de debhello_0.0-1.debian.tar.xz:

```
[base_dir] $ tar --xz -tf debhello-0.0.tar.xz
debhello-0.0/
debhello-0.0/src/
debhello-0.0/src/hello.c
debhello-0.0/Makefile
debhello-0.0/README.md
[base_dir] $ tar --xz -tf debhello_0.0-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
```

```
debian/docs
debian/examples
debian/gbp.conf
debian/manpages
debian/rules
debian/salsa-ci.yml
debian/source/
debian/source/format
debian/tests/
debian/tests/control
debian/upstream/
debian/upstream/metadata
debian/watch
```

O **debhello_0.0-1_amd64.deb** contém os ficheiros binário a serem instalados no sistema alvo.

O **debhello-dbgsym_0.0-1_amd64.deb** contém os ficheiros de símbolos de depuração a serem instalados no sistema alvo.

O conteúdo de pacote binário de todos os pacotes binário:

```
[base_dir] $ dpkg -c debhello-dbgsym_0.0-1_amd64.deb
drwxr-xr-x root/root ...  ./
drwxr-xr-x root/root ...  ./usr/
drwxr-xr-x root/root ...  ./usr/lib/
drwxr-xr-x root/root ...  ./usr/lib/debug/
drwxr-xr-x root/root ...  ./usr/lib/debug/.build-id/
drwxr-xr-x root/root ...  ./usr/lib/debug/.build-id/46/
-rw-r--r-- root/root ...  ./usr/lib/debug/.build-id/46/b45380346979015bc7a113...
drwxr-xr-x root/root ...  ./usr/share/
drwxr-xr-x root/root ...  ./usr/share/doc/
lrwxrwxrwx root/root ...  ./usr/share/doc/debhello-dbgsym -> debhello
[base_dir] $ dpkg -c debhello_0.0-1_amd64.deb
drwxr-xr-x root/root ...  ./
drwxr-xr-x root/root ...  ./usr/
drwxr-xr-x root/root ...  ./usr/bin/
-rwxr-xr-x root/root ...  ./usr/bin/hello
drwxr-xr-x root/root ...  ./usr/share/
drwxr-xr-x root/root ...  ./usr/share/doc/
drwxr-xr-x root/root ...  ./usr/share/doc/debhello/
-rw-r--r-- root/root ...  ./usr/share/doc/debhello/README.Debian
-rw-r--r-- root/root ...  ./usr/share/doc/debhello/changelog.Debian.gz
-rw-r--r-- root/root ...  ./usr/share/doc/debhello/copyright
```

A lista de dependências gerada de todos os pacotes binário.

A lista de dependências gerada de todos os pacotes binários (v=0.0):

```
[debhello-0.0] $ dpkg -f debhello-dbgsym_0.0-1_amd64.deb pre-depends \
           depends recommends conflicts breaks
Depends: debhello (= 0.0-1)
[debhello-0.0] $ dpkg -f debhello_0.0-1_amd64.deb pre-depends \
           depends recommends conflicts breaks
Depends: libc6 (>= 2.34)
```

Cuidado



Muitos mais detalhes precisam ser observados antes de se enviar o pacote para o arquivo Debian.

Nota



Se forem saltados ajustes manuais nos ficheiros de configuração auto-gerados pelo comando **debmake**, o pacote binário gerado pode ficar com falta duma descrição de pacote significativa e alguns dos requerimentos de política podem ficar a faltar. Este pacote desleixado vai funcionar bem sob o comando **dpkg**, e pode ser suficiente bom para a sua implantação local.

5.9 Passo 3 (alternativos): Modificação da fonte do autor

O exemplo de cima não tocou na fonte do autor para criar o próprio pacote Debian. Uma abordagem alternativa como maintainer é modificar os ficheiros na fonte do autor. Por exemplo, o **Makefile** pode ser modificado para definir o valor **$(prefix)** para **/usr**.

Nota



O “Secção 5.7” em cima usando o ficheiro **debian/rules** é a melhor abordagem para empacotamento para este exemplo. Mas vamos continuar com estas abordagens alternativas como inclinação à experiência.

No seguinte, vamos considerar 3 variantes simples desta abordagem alternativa para gerar ficheiros **debian/patches/*** representando modificações à fonte do autor no formato fonte Debian “**3.0 (quilt)**”. Estas substituem “Secção 5.7” no exemplo passo-a-passo em cima:

- “Secção 5.10”
- “Secção 5.11”
- “Secção 5.12”

Por favor note que o ficheiro **debian/rules** usado para estes exemplos não tem o alvo **override_dh_auto_install** como se segue:

debian/rules (versão alternativa de maintainer):

```
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-0.0] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
        dh $@
```

5.10 Patch por abordagem “diff -u”

Aqui, o ficheiro patch **000-prefix-usr.patch** é criado usando o comando **diff**.

Patch por diff -u

```
[base_dir] $ cp -a debhello-0.0 debhello-0.0.orig
[debhello-0.0] $ vim debhello-0.0/Makefile
 ... hack, hack, hack, ...
[base_dir] $ diff -Nru debhello-0.0.orig debhello-0.0 >000-prefix-usr.patch
[base_dir] $ cat 000-prefix-usr.patch
```

```
diff -Nru debhello-0.0.orig/Makefile debhello-0.0/Makefile
--- debhello-0.0.orig/Makefile  2026-07-22 15:28:48.183844548 +0900
+++ debhello-0.0/Makefile       2026-07-22 15:28:48.271746924 +0900
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

 all: src/hello

[base_dir] $ rm -rf debhello-0.0
[base_dir] $ mv -f debhello-0.0.orig debhello-0.0
```

Por favor note que a árvore fonte do autor é restaurada ao seu estado original após de gerar um ficheiro patch **000-prefix-usr.patch**.

Este **000-prefix-usr.patch** é editado para estar em conformidade com DEP-3 e movido para a localização correta como em baixo.

000-prefix-usr.patch (DEP-3):

```
[debhello-0.0] $ echo '000-prefix-usr.patch' >debian/patches/series
[debhello-0.0] $ vim ../000-prefix-usr.patch
 ... hack, hack, hack, ...
[debhello-0.0] $ mv -f ../000-prefix-usr.patch debian/patches/000-prefix-usr....
[debhello-0.0] $ cat debian/patches/000-prefix-usr.patch
From: Osamu Aoki <osamu@debian.org>
Description: set prefix=/usr patch
diff -Nru debhello-0.0.orig/Makefile debhello-0.0/Makefile
--- debhello-0.0.orig/Makefile
+++ debhello-0.0/Makefile
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

 all: src/hello
```

Nota



Quando se gera o pacote fonte Debian por **dpkg-source** via **dpkg-buildpackage** no passo seguinte de “Secção 5.8”, o comando **dpkg-source** assume que nenhuma patch foi aplicada à fonte do autor, pois o **.pc/applied-patches** está em falta.

5.11 Patch por abordagem dquilt

Aqui, o ficheiro patch **000-prefix-usr.patch** é criado usando o comando **dquilt**.

dquilt é um invólucro simples do programa **quilt**. A sintaxe e função do comando **dquilt** é a mesma que do comando **quilt**(1), excepto para o facto que a patch gerada é guardada no directório **debian/patches/**.

Patch por dquilt

```
[debhello-0.0] $ dquilt new 000-prefix-usr.patch
Patch debian/patches/000-prefix-usr.patch is now on top
[debhello-0.0] $ dquilt add Makefile
File Makefile added to patch debian/patches/000-prefix-usr.patch
 ... hack, hack, hack, ...
[debhello-0.0] $ head -1 Makefile
prefix = /usr
[debhello-0.0] $ dquilt refresh
Refreshed patch debian/patches/000-prefix-usr.patch
[debhello-0.0] $ dquilt header -e --dep3
```

```
 ... edit the DEP-3 patch header with editor
[debhello-0.0] $ tree -a
.
+-- .pc
|   +-- .quilt_patches
|   +-- .quilt_series
|   +-- .version
|   +-- 000-prefix-usr.patch
|   |   +-- .timestamp
|   |   +-- Makefile
|   +-- applied-patches
+-- Makefile
+-- README.md
+-- debian
|   +-- README.Debian
|   +-- README.source
|   +-- changelog
|   +-- clean
|   +-- control
|   +-- copyright
|   +-- dirs
|   +-- docs
|   +-- examples
|   +-- gbp.conf
|   +-- install
|   +-- links
|   +-- manpages
|   +-- patches
|   |   +-- 000-prefix-usr.patch
|   |   +-- series
|   +-- rules
|   +-- salsa-ci.yml
|   +-- source
|   |   +-- format
|   +-- tests
|   |   +-- control
|   +-- upstream
|   |   +-- metadata
|   +-- watch
+-- src
    +-- hello.c

9 directories, 30 files
[debhello-0.0] $ cat debian/patches/series
000-prefix-usr.patch
[debhello-0.0] $ cat debian/patches/000-prefix-usr.patch
Description: set prefix=/usr patch
Author: Osamu Aoki <osamu@debian.org>
Index: debhello-0.0/Makefile
===================================================================
--- debhello-0.0.orig/Makefile
+++ debhello-0.0/Makefile
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

 all: src/hello
```

Aqui, **Makefile** na árvore fonte do autor não precisa de ser restaurada para o estado original para o empacotamento.

Nota



Quando se gera o pacote fonte Debian por **dpkg-source** via **dpkg-buildpackage** no passo seguinte de “Secção 5.8”, o comando **dpkg-source** assume que patches foram aplicadas à fonte do autor, pois o **.pc/applied-patches** existe.

A árvore fonte do autor pode ser restaurada para o estado original para o empacotamento.

A árvore fonte do autor (restaurada):

```
[debhello-0.0] $ dquilt pop -a
Removing patch debian/patches/000-prefix-usr.patch
Restoring Makefile

No patches applied
[debhello-0.0] $ head -1 Makefile
prefix = /usr/local
[debhello-0.0] $ tree -a .pc
.pc
+-- .quilt_patches
+-- .quilt_series
+-- .version

1 directory, 3 files
```

Aqui, **Makefile** é restaurado e o **.pc/applied-patches** está em falta.

5.12 Patch por abordagem “dpkg-source --auto-commit”

Aqui, o ficheiro patch não é criado neste passo mas os ficheiros fonte estão configurados para criar ficheiros **debian/patches/*** no passo seguinte de “Secção 5.8”.

Para isto, **debmake** tem de ser invocado sem a opção **-x1** para gerar ficheiros modelo normais usando em vez disso a opção predefinida **-x2**.

O resultado do comando debmake

```
[base_dir] $ cd debhello-0.0
[debhello-0.0] $ debmake
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
...
```

Vamos editar a fonte do autor.

Makefile modificado

```
[debhello-0.0] $ vim Makefile
 ... hack, hack, hack, ...
[debhello-0.0] $ head -n1 Makefile
prefix = /usr
```

Vamos editar **debian/source/options**:

debian/source/options para auto-commit

```
[debhello-0.0] $ mv debian/source/options.ex debian/source/options
[debhello-0.0] $ vim debian/source/options
 ... hack, hack, hack, ...
[debhello-0.0] $ cat debian/source/options
# == Patch applied strategy (merge) ==
#
# The source outside of debian/ directory is modified by maintainer and
# different from the upstream one:
#   * Workflow using dpkg-source commit (commit all to VCS after dpkg-source ...
#       https://www.debian.org/doc/manuals/debmake-doc/ch04.en.html#dpkg-sour...
```

```
#   * Workflow described in dgit-maint-merge(7)
#
single-debian-patch
auto-commit
```

Vamos editar o **debian/source/patch-header**:
debian/source/patch-header para auto-commit

```
[debhello-0.0] $ mv debian/source/patch-header.ex debian/source/patch-header
[debhello-0.0] $ vim debian/source/patch-header
 ... hack, hack, hack, ...
[debhello-0.0] $ cat debian/source/patch-header
Description: debian-changes
Author: Osamu Aoki <osamu@debian.org>
```

Vamos remover ficheiros **debian/patches/*** e outros ficheiros modelo não usados
Remove ficheiros modelo não utilizados.

```
[debhello-0.0] $ rm -f debian/clean debian/dirs debian/install debian/links
[debhello-0.0] $ rm -f debian/README.source debian/*.ex debian/source/*.ex
[debhello-0.0] $ rm -rf debian/patches
[debhello-0.0] $ tree debian
debian
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- docs
+-- examples
+-- gbp.conf
+-- manpages
+-- rules
+-- salsa-ci.yml
+-- source
|   +-- format
|   +-- options
|   +-- patch-header
+-- tests
|   +-- control
+-- upstream
|   +-- metadata
+-- watch

4 directories, 16 files
```

Não existem ficheiros **debian/patches/*** no final deste passo.

Nota



Quando se gera o pacote fonte Debian por **dpkg-source** via **dpkg-buildpackage** no passo seguinte de “Secção 5.8”, o comando **dpkg-source** usa opções especificadas em **debian/source/options** para auto-submeter a modificação aplicada à fonte do autor como **patches/debian-changes**.

Vamos inspecionar o pacote fonte Debian gerado após o passo seguinte “Secção 5.8” e extrair os ficheiros de **debhello-0.0.debian.tar.xz**.
Inspecione debhello-0.0.debian.tar.xz após debuild

```
[base_dir] $ tar --xz -xvf debhello_0.0-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
```

```
debian/copyright
debian/docs
debian/examples
debian/gbp.conf
debian/manpages
debian/patches/
debian/patches/debian-changes
debian/patches/series
debian/rules
debian/salsa-ci.yml
debian/source/
debian/source/format
debian/source/options
debian/source/patch-header
debian/tests/
debian/tests/control
debian/upstream/
debian/upstream/metadata
debian/watch
```

Vamos verificar os ficheiros gerados **debian/patches/***.
Inspecione debian/patches/* após debuild

```
[base_dir] $ cat debian/patches/series
debian-changes
[base_dir] $ cat debian/patches/debian-changes
Description: debian-changes
Author: Osamu Aoki <osamu@debian.org>

--- debhello-0.0.orig/Makefile
+++ debhello-0.0/Makefile
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

 all: src/hello
```

Confirma-se que o pacote fonte Debian **debhello-0.0.debian.tar.xz** é gerado apropriadamente com ficheiros **debian/patches/*** para a modificação Debian.

Capítulo 6

Bases para empacotamento

Aqui, é apresentada aqui uma visão geral extensa sem se usar operações VCS para as regras básicas do empacotamento Debian focando-se no pacote Debian não-nativo no formato "**3.0 (quilt)**".

Nota



Alguns detalhes são saltados intencionalmente para clarificar. Por favor leia os manuais do **dpkg-source**(1), **dpkg-buildpackage**(1), **dpkg**(1), **dpkg-deb**(1), **deb**(5), etc.

O pacote fonte Debian é um conjunto de ficheiros de entrada usados para compilar o pacote binário Debian e não é um ficheiro único.

O pacote binário Debian é um ficheiro de arquivo especial que contém um conjunto de dados binários instaláveis com a sua informação associada.

Um único pacote fonte Debian pode gerar múltiplos pacotes binário Debian definidos no ficheiro **debian/control**.

O pacote Debian não-nativo no formato fonte Debian "**3.0 (quilt)**" é o formato mais normal de pacote fonte Debian.

Nota



Existem muitos scripts invólucro. Use-os para simplificar o seu fluxo de trabalho mas certifique-se que compreende as suas bases internas.

6.1 Fluxo de trabalho de empacotamento

O fluxo de trabalho de empacotamento Debian para criar um pacote binário Debian involve gerar vários ficheiros nomeados especificamente (veja "Secção 6.3") como definido no "Manual de Política Debian". Este fluxo de trabalho pode ser resumido em 10 passos com alguma simplificação como se segue.

1. O tarball de autor é descarregado como ficheiro *pacote-versão***.tar.xz**.
2. O tarball de autor é descompactado para criar muitos ficheiros sob o directório *pacote-versão***/**.
3. O tarball de autor é copiado (ou ligado por link simbólico) para o nome de ficheiro particular *nome-pacote_versão***.orig.tar.xz**.
    - o caractere que separa *pacote* e *versão* é alterado de **-** (hífen) para _ (underscore)
    - é adicionado **.orig** à extensão do ficheiro.
4. Os ficheiros de especificação de pacote Debian são adicionados à fonte do autor sob o directório *pacote-versão***/debian/**.

- Ficheiros de especificação requeridos sob o directório **debian/**:

  debian/rules O script executável para compilar o pacote Debian (veja "Secção 6.5")

  debian/control O ficheiro de configuração do pacote que contém o nome do pacote fonte, as dependências de compilação da fonte, o nome do pacote binário, as dependências binárias, etc. (veja "Secção 6.6")

  debian/changelog O ficheiro de histórico de pacote Debian que define a versão de pacote do autor e a revisão Debian na sua primeira linha (veja "Secção 6.7")

  debian/copyright O resumo de copyright e licença (veja "Secção 6.8")

  debian/source/format Isto indica o formato desejado para **dpkg-source**(1) (veja Debian wiki: "DebSrc3.0")

- Ficheiros de especificação opcionais sob **debian/*** (veja "Secção 6.14"):
- Estes ficheiros têm de ser editados manualmente à sua perfeição de acordo com o "Manual Debian de Política Debian" e "Debian Developer's Reference".

5. O comando **dpkg-buildpackage** (geralmente a partir do seu invólucro **debuild** ou **sbuild**) é invocado no directório *pacote-versão***/** para criar a fonte Debian e os pacotes binário ao invocar o script **debian/rules**.
   - O directório actual é definido como: "**CURDIR=***/caminho/para/pacote-versão/*"
   - Criar o pacote fonte Debian no formato fonte Debian "**3.0 (quilt)**" usando o **dpkg-source**(1)
     - *pacote_versão*.**orig.tar.xz** (copia ou link simbólico para *pacote-versão*.**.tar.xz**)
     - *pacote_versão-revisão*.**debian.tar.xz** (tarball de **debian/** encontrado em *pacote-versão*/**/**)
     - *pacote_versão-revisão*.**dsc**
   - Compila a fonte usando "**debian/rules build**" na **$(DESTDIR)**
     - "**DESTDIR=debian/***pacote-binário***/**" para pacote binário único 1
     - "**DESTDIR=debian/tmp/**" (para pacote de múltiplos binários)
   - Cria o pacote binário Debian usando **dpkg-deb**(1), **dpkg-genbuildinfo**(1), e **dpkg-genchanges**(1).
     - *pacotebinário_versão-revisão_arch*.**deb**
     - … (Podem existir múltiplos ficheiros de pacote binário Debian.)
     - *pacote_versão-revisão_arch*.**changes**
     - *pacote_versão-revisão_arch*.**buildinfo**
6. Verifica a qualidade do pacote Debian com o comando **lintian**. (recomendado)
   - Segue as diretrizes de rejeição a partir de ftp-master.
     - "REJECT-FAQ"
     - "NOVA lista de verificação"
     - "Auto-rejeições do Lintian" ("lista de etiquetas do lintian")
7. Teste a fiabilidade do pacote binário Debian gerado manualmente ao instala-lo e ao correr os seus programas.
8. Após confirmar a fiabilidade, prepare os ficheiros para o envio apenas-fonte normal para o arquivo Debian.
9. Assine o ficheiro de pacote Debian com o comando **debsign** usando a sua chave openPGP privada.
   - Use "**debsign** *pacote_versão-revisão*_**source.changes**" (situação de envio apenas-fonte)
   - Use "**debsign** *pacote_versão-revisão_arquitectura***.changes**" (situação de envio fonte+binário)
10. Envie o conjunto de ficheiros de pacote Debian com o comando **dput** para o arquivo Debian.
    - Use "**dput** *pacote_versão-revisão*_**source.changes**" (envio apenas-fonte)

1Isto é a predefinição até ao **debhelper** v13. No **debhelper** v14, ele avisa sobre a mudança de predefinição. Após **debhelper** v15, irá mudar a predefinição para **DESTDIR=debian/tmp/** .

- Use “**dput** *pacote_versão-revisão_arch***.changes**” (envio fonte+binário)

O testar da compilação e confirmar da fiabilidade do pacote binário como em cima é obrigação moral como um desenvolvedor Debian diligente mas neste momento não há barreira física para as pessoas saltarem tais operações para o envio apenas-fonte.

Para o tarball de autor, o comando **debmake** ajuda-o até ao passo 4 no fluxo de trabalho de cima. Para o *pacote/* da árvore de trabalho do autor verificado, por ex por “**git clone https://github.com/-*nome-de-autoria/pacote*.git**” sem nenhum tarball de autor, o comando **debmake** invocado nele também o ajuda até ao passo 4. O comando **debmake** não sobrescreve nenhuns ficheiros de configuração existentes.

Aqui, por favor substitua cada parte do nome de ficheiro como:

- a parte *pacote* pelo nome de pacote fonte Debian
- a parte *pacotebinário* pelo nome do pacote binário Debian
- a parte *versão* pela versão do autor
- a parte *revisão* pela revisão Debian
- a parte *arch* pela arquitectura do pacote (ex., **amd64**)

A pratica Debian atual para envio de pacote Debian normal é:

- Usa o envio apenas-fonte se todos os pacotes binário gerados existirem no arquivo **sid** de Debian. Este é o caso mais usual.
- Usa o envio fonte+binário se qualquer um dos pacote gerados estiver a faltar no arquivo **sid** de Debian. (Isto involve o processo de NOVO PACOTE lidado manualmente pela equipa de gestão do arquivo.)

Veja também “Envios apenas-fonte”.

Dica



São praticadas muitas estratégias de gestão de patch e utilização de VCS para o empacotamento Debian. Você não precisa de as usar todas.

Dica



Existe documentação muito extensiva em “Capítulo 6. Melhores Práticas de Empacotamento” no “Debian Developer’s Reference”. Por favor leia-o.

6.2 Pacote debhelper

Apesar de se poder fazer um pacote Debian escrevendo um script **debian/rules** sem se usar o pacote **debhelper**, é impraticável fazer-lo. Existem demasiadas funcionalidades requeridas pela “Política Debian” moderna a serem observadas, como a aplicação de permissões de ficheiros apropriadas, uso do caminho apropriado de instalação de bibliotecas dependentes da arquitectura, inserção de scripts hook de instalação, geração do pacote de símbolos de depuração, geração da informação de dependências do pacote, geração dos ficheiros de informação do pacote, aplicação do registo temporal apropriado para compilação reproduzível, etc.

O pacote **Debhelper** fornece um conjunto de scripts úteis de modo a simplificar o fluxo de trabalho do empacotamento Debian e reduzir o fardo dos maintainers de pacotes. Quando usados de modo apropriado, eles irão ajudar os empacotadores a lidar e implementar funcionalidades requeridas pela “Política Debian” automaticamente.

O fluxo de trabalho de empacotamento Debian moderno pode ser organizado num fluxo de trabalho simples modular ao:

- usar o comando **dh** para invocar muitos scripts utilitários automaticamente a partir do pacote **debhelper**, e
- configurar o seu comportamento com ficheiros de configuração declarativos no directório **debian/**.

Você deve quase sempre usar o **debhelper** como sua dependência de compilação do pacote. Este documento também assume que você está a usar uma versão razoavelmente contemporânea do **debhelper** para lidar com os trabalhos de empacotamento nos seguintes contextos.

Nota



Para **debhelper** "compat >= 9", o comando **dh** exporta as bandeiras de compilação (**CFLAGS**, **CXXFLAGS**, **FFLAGS**, **CPPFLAGS** e **LDFLAGS**) com valores como retornados por **dpkg-buildflags** se não estiverem definidas previamente. (O comando **dh** chama **set_buildflags** definido no módulo **Debian::Debhelper::Dh_Lib**.)

Nota



O **debhelper**(1) muda o seu comportamento com o tempo. Por favor certifique-se de ler **debhelper-compat-upgrade-checklist**(7) para compreender a situação.

6.3 Nome e versão do pacote

Se a fonte do autor vem como **hello-0.9.12.tar.xz**, você pode tirar **hello** como o nome do pacote fonte do autor e **0.9.12** como a versão do autor.

Existem algumas limitações para quais caracteres podem ser usados como parte do pacote Debian. A limitação mais notável é a proibição de letras maiúsculas no nome do pacote. Aqui está um sumário como um conjunto de expressões regulares:

- Nome de pacote do autor (**-p**): `[-+.a-z0-9]{2,}`
- Nome de pacote binário (**-b**): `[-+.a-z0-9]{2,}`
- Versão do autor (**-u**): `[0-9][-+.:~a-z0-9A-Z]*`
- Revisão Debian (**-r**): `[0-9][+.~a-z0-9A-Z]*`

Veja a definição exacta em "Capítulo 5 - Ficheiros de controle e seus campos" no "Manual de Política Debian".

Você tem de ajustar o nome do pacote e a versão do autor de acordo com o empacotamento Debian.

De modo a se gerir efectivamente o nome do pacote e a informação da versão sob ferramentas populares como o comando **aptitude**, é boa ideia manter o comprimento do nome do pacote a ser igual ou menor que 30 caracteres, e o comprimento total da versão e revisão a ser igual ou menor que 14 caracteres. 2

De modo a evitar colisões de nomes, o nome de pacote binário visível ao utilizador não deve ser escolhido a partir de nenhumas palavras genéricas.

Se o autor não usar um esquema normal de indicação da versão como **2.30.32** mas usa algum tipo da data como **11Apr29**, uma string aleatória de código, ou um valor hash VCS como parte da a versão, certifique-se de a remover da versão de autor. Tal informação pode ser guardada no ficheiro

2Para mais de 90% dos pacotes, o nome do pacote é igual ou menos que 24 caracteres; a versão de autor é igual ou menos que 10 caracteres e a revisão Debian é igual ou menos que 3 caracteres.

debian/changelog. Se você precisa de inventar uma string de versão, use o formato **AAAAMMDD** tal como **20110429** como versão do autor. Isto assegura que o comando **dpkg** interpreta as versões posteriores corretamente como atualizações. Se você precisa de assegurar uma transição suave para um esquema normal de versão como **0.1** no futuro, então use o formato **0~AAMMDD** como ex **0~110429** como versão do autor.

As strings de versão podem ser comparadas usando o comando **dpkg** como se segue.

```
[~] $ dpkg --compare-versions ver1 op ver2
```

A regra de comparação de versão pode ser resumida a:

- As strings são comparadas desde o inicio para o fim.
- As letras são mais largas que os dígitos.
- Números são comparados como inteiros.
- As letras são comparadas em ordem de código ASCII.

Existem regras especiais para os caracteres ponto (**.**), mais (**+**), e til (**~**), como se segue.

```
0.0 < 0.5 < 0.10 < 0.99 < 1 < 1.0~rc1 < 1.0 < 1.0+b1 < 1.0+nmu1 < 1.1 < 2.0
```

Um caso complicado ocorre quando o autor lança **hello-0.9.12-ReleaseCandidate-99.tar.xz** como o pré-lançamento de **hello-0.9.12.tar.xz**. Você pode assegurar que a atualização de pacote Debian vai funcionar bem ao renomear a fonte do autor para **hello-0.9.12~rc99.tar.xz**.

6.4 Pacote Debian nativo

O pacote Debian não-nativo no formato " "**3.0 (quilt)**" é o formato de pacote fonte Debian mais normal, O ficheiro **debian/source/format** deve ter "**3.0 (quilt)**" nele como descrito em **dpkg-source**(1). O fluxo de trabalho em cima e os exemplos de empacotamento seguintes usam sempre este formato.

O pacote Debian nativo é o formato de pacote binário Debian raro. Só pode ser usando quando o pacote é útil e valoroso apenas para Debian. Assim , o seu uso é geralmente desencorajado.

> Cuidado
>
> 
>
> Um pacote Debian nativo é muitas vezes compilado acidentalmente quando o seu tarball de autor não é acessível a partir do comando **dpkg-buildpackage** com o seu nome correto *pacote_versão*.**orig.tar.xz** . Este é um engano típico de novato causado por criar um nome de link simbólico com "**-**" em vez do correto "_".

Um pacote nativo Debian não tem separação entre o **código do autor** e as **modificações Debian** e consistem apenas do seguinte:

- *pacote_versão*.**tar.xz** (copia ou link simbólico para *pacote-versão*.**tar.xz** com ficheiros **debian/***.)
- *pacote_versão*.**dsc**

Se você precisa criar um pacote Debian nativo, crie-o no formato fonte Debian "**3.0 (native)**" usando o **dpkg-source**(1).

> Dica
>
> 
>
> Não há necessidade de criar o tarball antes se for usado o formato de pacote nativo Debian. O ficheiro **debian/source/format** deve ter "**3.0 (native)**" nele como descrito em **dpkg-source**(1) e o ficheiro **debian/source/format** deve ter a versão sem a revisão Debian (**1.0** em vez de **1.0-1**). Depois, o tarball que contém é gerado quando "**dpkg-source -b**" for invocado na árvore fonte.

6.5 Ficheiro debian/rules

O ficheiro **debian/rules** é o script executável que redireciona o sistema de compilação do autor para instalar ficheiros em **$(DESTDIR)** e cria o ficheiro arquivo dos ficheiros gerados como o ficheiro **deb**. O ficheiro **deb** é usado para a distribuição binária e instalado no sistema usando o comando **dpkg**.

O ficheiro **debian/rules** compatível com a política Debian e que suporta todos os alvos requeridos pode ser escrito tão simplesmente como 3:

debian/rules simples:

```
#!/usr/bin/make -f
#export DH_VERBOSE = 1

%:
  dh $@
```

O comando **dh** funciona como o sequenciador para chamar todos os comandos "**dh** *target*" requeridos no momento certo. [4]

- **dh clean** : limpa ficheiros na árvore fonte.
- **dh build** : compila a árvore fonte
- **dh build-arch** : compila a árvore fonte para pacotes dependentes da arquitectura
- **dh build-indep** : compila a árvore fonte para pacotes independentes da arquitectura
- **dh install** : instala os ficheiros binário em **$(DESTDIR)**
- **dh install-arch** : instala os ficheiros binário em **$(DESTDIR)** para pacotes dependentes da arquitectura
- **dh install-indep** : instala os ficheiros binário em **$(DESTDIR)** para pacotes independentes da arquitectura
- **dh binary** : gera o ficheiro **deb**
- **dh binary-arch** : gera o ficheiro **deb** para pacotes dependentes de arquitectura
- **dh binary-indep** : gera o ficheiro **deb** para pacotes independentes da arquitectura

Aqui, o caminho **$(DESTDIR)** depende do tipo de compilação.

- "**DESTDIR=debian/***pacote-binário***/**" para pacote binário único [5]
- "**DESTDIR=debian/tmp/**" (para pacote de múltiplos binários)

Veja "Secção 9.2" e "Secção 9.3" para personalização.

> Dica
>
> 
>
> Definir "**export DH_VERBOSE = 1**" faz escrever cada comando que modifica ficheiros no sistema de compilação. Também ativa registos de compilação detalhados para alguns sistemas de compilação.

3O comando **debmake** gera um ficheiro **debian/rules** um pouco mais complicado. Mas está a parte central.

4Esta simplicidade está disponível desde a versão 7 do pacote **debhelper**. Este guia assume o uso de **debhelper** versão 14 ou mais recente.

5Isto é a predefinição até ao **debhelper** v13. No **debhelper** v14, ele avisa sobre a mudança de predefinição. Após **debhelper** v15, irá mudar a predefinição para **DESTDIR=debian/tmp/** .

6.6 Ficheiro debian/control

O ficheiro **debian/control** consiste de blocos de metadados separados por linhas vazias. Cada bloco de metadados define o seguinte por esta ordem:

- meta-dados para os pacotes fonte Debian
- meta-dados para os pacotes binário Debian

Veja "Capítulo 5 - Ficheiros control e seus campos" do "Manual de Política Debian" para a definição de cada campo de metadados.

Nota



O comando **debmake** define o ficheiro **debian/control** com "**Build-Depends: debhelper-compat (= 14)**" para definir o nível de compatibilidade do **debhelper**.

Dica



Se um pacote existente tem um nível de compatibilidade do **debhelper** inferior a 14, é provavelmente hora de actualizar o seu empacotamento.

6.7 Ficheiro debian/changelog

O ficheiro **debian/changelog** regista o histórico do pacote Debian.

- Edite este ficheiro usando o comando **debchange** (nome alternativo **dch**).
- A primeira linha define a versão de pacote do autor e a revisão Debian.
- Documente as alterações num estilo específico, formal, e conciso.
  - Se a modificação do maintainer Debian corrigir bugs reportados, adicione "**Closes:** *#<número_bug>*" para fechar esses bugs.
- Mesmo que seja você próprio a enviar o seu pacote, você tem de documentar todas as alterações não-triviais visíveis-ao-utilizador, tais como:
  - Correções de bugs relacionada com segurança.
  - Alterações à interface de utilizador.
- Se você está a pedir a um patrocinador para o enviar, documente as alterações de forma mais compreensiva, incluindo todas as relacionadas com o empacotamento, para ajudar na revisão do seu pacote.
  - O patrocinador não deve ter de adivinhar o seu raciocínio por de trás das alterações ao pacote.
  - Lembre-se que o tempo do patrocinador é valioso.

Após terminar o seu empacotamento e verificar a sua qualidade, execute o comando "**dch -r**" e guarde o ficheiro **debian/changelog** finalizado com a suite normalmente definida para **unstable**. [6] Se você está a empacotar para backports, security updates, LTS, etc., então use os nomes de distribuição apropriados.

O comando **debmake** cria o ficheiro modelo inicial com a versão do pacote de autor e a revisão Debian. A distribuição é definida para **UNRELEASED** para prevenir envios acidentais para o arquivo

[6] Se você está a usar o editor **vim**, certifique-se de guardar isto com o comando "**:wq**".

Debian.

Dica



A string de data usada no ficheiro **debian/changelog** pode ser gerada manualmente pelo comando "**LC_ALL=C date -R**".

Dica



Use uma entrada no **debian/changelog** com uma string de versão como **1.0.1-1~rc1** quando experimentar. Depois, consolide tais entradas **changelog** numa entrada única para o pacote oficial.

O ficheiro **debian/changelog** é instalado no directório **/usr/share/doc/***pacotebinário* como **changelog.Debian.gz** pelo comando **dh_installchangelogs**.

O registo de alterações do autor é instalado no directório **/usr/share/doc/***pacotebinário* como **changelog.gz**.

O registo de alterações do autor é descoberto automaticamente pelo **dh_installchangelogs** usando a correspondência não sensível a maiúsculas minúsculas do seu nome de ficheiro para **changelog**, **changes**, **changelog.txt**, **changes.txt**, **history**, **history.txt**, ou **changelog.md** e procurado nos directórios **./ doc/** ou **docs/**.

6.8 Ficheiro debian/copyright

Debian leva o copyright e a licença muito a sério. O "Manual de Política Debian" requer um resumo deles no ficheiro **debian/copyright** do pacote.

- "12.5. Informação de Copyright"
- "2.3. Considerações de Copyright"
- "Informação de Licença"

O comando **debmake** cria o ficheiro modelo inicial **debian/copyright** usando o comando **licensecheck**(1).

6.9 Ficheiros debian/patches/*

Como demonstrado em "Secção 5.9", o directório **debian/patches/** contém

- ficheiros *nome-ficheiro-patch***.patch** que fornecem patches **-p1** e
- o ficheiro **series** o qual define como estas patches são aplicadas.

Veja como estes ficheiros são usados em:

- "Secção 13.6" para compilar o pacote fonte Debian
- "Secção 13.7" para extrair ficheiros fonte a partir do pacote fonte Debian

Nota



Os textos de cabeçalho destas patches devem estar em conformidade com "DEP-3".

Nota



Se você quer usar ferramentas VCS como **git**, **gbp** e **dgit** para criar e gerir estas patches após aprender as bases aqui, por favor depois consulte “Capítulo 11”.

6.10 Ficheiro debian/source/include-binaries

O comando “**dpkg-source --commit**” funciona como o **dquilt** mas tem uma vantagem sobre o comando **dquilt**. O comando **dquilt** não consegue lidar com ficheiros binário modificados porque eles não são representados num diff. Também, o adicionar de ficheiros binário sob o directório **debian/** é normalmente rejeitado pelo **dpkg-source**. Ao listar estes ficheiros binário em **debian/source/include-binaries**, o maintainer pode incluir estes ficheiros binário no pacote fonte Debian gerado pelo **dpkg-source**.

6.11 Ficheiro debian/watch

Nota



Este ficheiro é para ser usado pelo pacote não-nativo Debian.

O comando **uscan**(1) descarrega a versão de autor mais recente usando o ficheiro **debian/watch**. Ex.:

Ficheiro debian/watch básico:

```
version=4
https://ftp.gnu.org/gnu/hello/ @PACKAGE@@ANY_VERSION@@ARCHIVE_EXT@
```

O comando **uscan** pode verificar a autenticidade do tarball de autor com configuração opcional (veja “Secção 6.12”).

Veja **uscan**(1), “Secção 9.4”, “Secção 8.1”, e “Secção 11.7” para mais.

6.12 Ficheiro debian/upstream/signing-key.asc

Alguns pacotes estão assinados com uma chave openPGP e a sua autenticidade pode ser verificada usando a sua chave openPGP pública.

Por exemplo, “GNU hello” pode ser descarregado via HTTP de https://ftp.gnu.org/gnu/hello/. Existem conjuntos de ficheiros:

- **hello-*versão*.tar.xz** (fonte de autor)
- **hello-*versão*.tar.xz.sig** (assinatura desanexada)

Vamos pegar no conjunto da versão mais recente.

Descarrega o tarball de autor e a sua assinatura.

```
[base_dir] $ wget https://ftp.gnu.org/gnu/hello/hello-2.9.tar.xz
 ...
[base_dir] $ wget https://ftp.gnu.org/gnu/hello/hello-2.9.tar.xz.sig
 ...
[base_dir] $ gpg --verify hello-2.9.tar.xz.sig
gpg: Signature made Thu 10 Oct 2013 08:49:23 AM JST using DSA key ID 80EE4A00
gpg: Can't check signature: public key not found
```

Se você conhece a chave openPGP pública do maintainer do autor a partir da lista de mail, use-a como o ficheiro **debian/upstream/signing-key.asc**. Caso contrário, use o servidor de chave hkp e verifique-a via seu web de confiança.

Descarregue a chave openPGP pública para o autor

```
[base_dir] $ gpg --keyserver hkp://keys.gnupg.net --recv-key 80EE4A00
gpg: requesting key 80EE4A00 from hkp server keys.gnupg.net
gpg: key 80EE4A00: public key "Reuben Thomas <rrt@sc3d.org>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1
[base_dir] $ gpg --verify hello-2.9.tar.xz.sig
gpg: Signature made Thu 10 Oct 2013 08:49:23 AM JST using DSA key ID 80EE4A00
gpg: Good signature from "Reuben Thomas <rrt@sc3d.org>"
  ...
Primary key fingerprint: 9297 8852 A62F A5E2 85B2  A174 6808 9F73 80EE 4A00
```

Dica

Se o seu ambiente de rede bloquear o acesso ao porto HKP **11371**, então use "**hkp://keyserver.ubuntu.com:80**".

Após confirmar que o ID de chave **80EE4A00** é de confiança, descarregue a sua chave pública para o ficheiro **debian/upstream/signing-key.asc**.

Defina a chave openPGP pública para debian/upstream/signing-key.asc

```
[base_dir] $ gpg --armor --export 80EE4A00 >debian/upstream/signing-key.asc
```

Com o ficheiro **debian/upstream/signing-key.asc** em cima e o ficheiro **debian/watch** seguinte, o comando **uscan** consegue verificar a autenticidade do tarball de autor após a sua descarga. Ex.:

Ficheiro debian/watch melhorado com suporte a openPGP:

```
version=4
opts="pgpsigurlmangle=s/$/.sig/" \
https://ftp.gnu.org/gnu/hello/ @PACKAGE@@ANY_VERSION@@ARCHIVE_EXT@
```

6.13 Ficheiro debian/salsa-ci.yml

Instale o ficheiro de configuração Salsa CI. Veja "Secção 11.3".

6.14 Outros ficheiros debian/*

Podem ser adicionados ficheiros de configuração opcionais sob o directório **debian/**. A maioria deles são comandos de controle **dh**_* oferecidos pelo pacote **debhelper** mas existem alguns para os comandos **dpkg-source**, **lintian** e **gbp**.

Dica

Mesmo uma fonte de autor sem o seu sistema de compilação pode ser empacotada apenas usando estes ficheiros. Veja "Secção 14.2" como um exemplo.

A lista alfabética de ficheiros de configuração opcionais notáveis **debian/***pacotebinário***.*** listados em baixo fornece meios muito poderosos de definir o caminho de instalação dos ficheiros. Por favor note:

- A notação de superscript "**-x[01234]**" que aparece na seguinte lista indica o valor mínimo para a opção **-x** do **debmake** que gera o ficheiro modelo associado. Veja "Secção 16.4" ou **debmake**(1) para detalhes.
- Para um pacote binário único, a parte "*pacote-binário*" do nome de ficheiro na lista pode ser removida.
- Para um pacote multi-binário, um ficheiro de configuração onde falta a parte "*pacote-binário*" no nome de ficheiro é aplicado ao primeiro pacote binário listado em **debian/control**.
- Quando existem muitos pacotes binários, a suas configurações podem ser especificadas independentemente ao prefixar o seu nome aos seus nomes de ficheiros de configuração tal como "*pacote-1***.install**", "*pacote-2***.install**", etc.
- Alguns ficheiros de configuração modelo podem não ser criados pelo comando **debmake**. Em tais casos, você precisa de cria-los com um editor.
- Alguns ficheiros modelo de configuração gerados pelo comando **debmake** com um sufixo **.ex** extra precisam de ser ativados ao remover esse sufixo.
- O comando **debmake -B** adiciona ficheiros modelo com um sufixo extra **.ex** a todos os ficheiros modelo existentes sem o **.ex** e eles precisam de ser ativados ao remover esse sufixo.
- Ficheiros modelo de configuração não usados gerados pelo comando **debmake** devem ser removidos.
- Copie ficheiros modelo de configuração como necessário para que os nomes de ficheiros correspondam aos seus nomes de pacote binário pertinentes.

pacotebinário*.bug-control** [-x2] instalado como **usr/share/bug/pacotebinário***/control** em *pacotebinário*. Veja "Secção 9.11".

pacotebinário*.bug-presubj** [-x2] instalado como **usr/share/bug/pacotebinário***/presubj** em *pacotebinário*. Veja "Secção 9.11".

pacotebinário*.bug-script** [-x2] instalado como **usr/share/bug/pacotebinário* ou **usr/share/bug/***pacotebinário***/sc** em *pacotebinário*. Veja "Secção 9.11".

***pacotebinário*.bash-completion** Lista scripts de completação **bash** a serem instalados.

O pacote **bash-completion** é requerido para ambos ambientes de compilação e utilizador.

Veja **dh_bash-completion**(1).

clean [-x1] Lista ficheiros que devem ser removidos mas não são limpos pelo comando **dh_auto_clean**.

Veja **dh_auto_clean**(1) e **dh_clean**(1).

compat [-x4] Define o nível de compatibilidade do **debhelper**. (descontinuado)

Use "**Build-Depends: debhelper-compat (= 14)**" em **debian/control** para especificar o nível de compatibilidade e remova **debian/compat**.

Veja "**NÍVEIS DE COMPATIBILIDADE**" no **debhelper**(7).

***pacote-binário*.conffiles** [-x3] Este ficheiro opcional é instalado no directório **DEBIAN** dentro do pacote binário enquanto o suplementa com todos os ficheiros de configuração auto-detetados pelo **debhelper**.

Este ficheiro é principalmente útil para se usar entradas "especiais" como a funcionalidade remove-on-upgrade do **dpkg**(1).

Se o programa que você está a empacotar requerer que cada utilizador modifique os ficheiros de configuração no directório **/etc**, existem duas maneiras populares de fazer com que não sejam ficheiros de configuração, mantendo o comando **dpkg** contente e caladinho.

- Cria um link simbólico sob o directório **/etc** a apontar para um ficheiro sob o directório **/var** gerado pelos scripts do maintainer.
- Cria um ficheiro gerado pelos scripts do maintainer sob o directório **/etc**.

Veja **dh_installdeb**(1).

***pacote-binário*.config** Este é o script **config** do **debconf** usado para perguntar quaisquer questões necessárias para configurar o pacote. Veja "Secção 10.22".

pacote-binário*.cron.hourly** -x3 Instalado no ficheiro **etc/cron/hourly/pacote-binário* em *pacote-binário*.

Veja **dh_installcron**(1) e **cron**(8).

pacote-binário*.cron.daily** -x3 Instalado no ficheiro **etc/cron/daily/pacote-binário* em *pacote-binário*.

Veja **dh_installcron**(1) e **cron**(8).

pacote-binário*.cron.weekly** -x3 Instalado no ficheiro **etc/cron/weekly/pacote-binário* em *pacote-binário*.

Veja **dh_installcron**(1) e **cron**(8).

***pacote-binário*.cron.monthly** -x3 Instalado no ficheiro *etc/cron/monthly/**pacote-binário* em *pacote-binário*.

Veja **dh_installcron**(1) e **cron**(8).

pacote-binário*.cron.d** -x3 Instalado no ficheiro **etc/cron.d/pacote-binário* em *pacote-binário*.

Veja **dh_installcron**(1), **cron**(8), e **crontab**(5).

pacote-binário*.default** -x3 Se isto existir, é instalado em **etc/default/pacote-binário* em *pacote-binário*.

Veja **dh_installinit**(1).

***pacote-binário*.dirs** -x1 Lista directórios a serem criados em *pacote-binário*.

Veja **dh_installdirs**(1).

Geralmente, isto não é necessário pois todos os comandos **dh_install*** criam os directórios requeridos automaticamente. Use isto apenas se encontrar problemas.

***pacote-binário*.doc-base** -x1 Instalado como o ficheiro de controle do **doc-base** em *pacote-binário*.

Veja **dh_installdocs**(1) e "Manual Debian doc-base (**doc-base.html**)" fornecido pelo pacote **doc-base**.

***pacote-binário*.docs** -x1 Lista ficheiros de documentação a serem instalados em *pacote-binário*.

Veja **dh_installdocs**(1).

pacotebinário*.emacsen-compat** Instalado em **usr/lib/emacsen-common/packages/compat/pacote-binário* em *pacote-binário*.

Veja **dh_installemacsen**(1).

pacote-binário*.emacsen-install** -x3 Instalado em **usr/lib/emacsen-common/packages/install/pacote-binário* em *pacote-binário*.

Veja **dh_installemacsen**(1).

pacote-binário*.emacsen-remove** -x3 Instalado em **usr/lib/emacsen-common/packages/remove/pacote-binário* em *pacote-binário*.

Veja **dh_installemacsen**(1).

pacote-binário*.emacsen-startup** -x3 Instalado em **usr/lib/emacsen-common/packages/startup/pacote-binário* em *pacote-binário*.

Veja **dh_installemacsen**(1).

pacote-binário*.examples** -x1 Lista ficheiros ou directórios exemplo a serem instalados em **usr/share/doc/paco*
*binário***examples/** em *pacote-binário*.

Veja **dh_installexamples**(1).

gbp.conf -x1 Se isto existir, funciona como ficheiro de configuração para o comando **gbp**.

Veja **gbp.conf**(5), **gbp**(1), e **git-buildpackage**(1).

***pacote-binário*.info** -x1 Lista ficheiros de informação a serem instalados em *pacote-binário*.

Veja **dh_installinfo**(1).

pacote-binário*.init** -x4 Instalado em **etc/init.d/pacote-binário* em *pacote-binário*. (descontinuado)

Veja **dh_installinit**(1).

***pacote-binário*.install** -x1 Lista ficheiros que devem ser instalados mas não são instalados pelo comando **dh_auto_install**.

Veja **dh_install**(1) e **dh_auto_install**(1).

***pacote-binário*.links** -x1 Lista pares de ficheiros de fonte e destino para serem ligados por link simbólico. Cada par deve ser posto na sua própria linha, com a fonte e destino separados por espaço em branco.

Veja **dh_link**(1).

pacote-binário*.lintian-overrides** -x2 Instalado em **usr/share/lintian/overrides/pacote-binário* no directório de compilação do pacote. Este ficheiro é usado para suprimir diagnósticos do **lintian** erróneos.

Veja **dh_lintian**(1), **lintian**(1) e "Manual do Utilizador de Lintian".

pacote-binário*.maintscript** -x2 Se este ficheiro opcional existir, o **debhelper** usa-o como modelo para gerar ficheiros **DEBIAN/pacotebinário***.{pre,post}{inst,rm}** dentro do pacote binário enquanto adiciona "**-- "$@"**" ao comando **dpkg-maintscript-helper**(1).

Veja **dh_installdeb**(1) e "Capítulo 6 - Scripts de maintainer de pacote e procedimento de instalação" no "Manual de Política Debian".

manpage.* -x2 Estes são ficheiros modelo de manuais gerados pelo comando **debmake**. Por favor renomeie estes para nomes de ficheiro apropriados e atualize o seu conteúdo.

A Política Debian requer que cada programa, utilitário e função deve ter um manual associado incluído no mesmo pacote. Os manuais são escritos em **nroff**(1). Se você é novato a fazer um manual, use **manpage.asciidoc** -x3 ou **manpage.1** -x3 como ponto de partida.

***pacote-binário*.manpages** -x1 Lista os manuais a serem instalados.

Veja **dh_installman**(1).

***pacote-binário*.menu (descontinuado, não mais instalado)** tech-ctte #741573 decidiu "Debian deve usar ficheiros **.desktop** como apropriado".

Ficheiro de menu Debian instalado em **usr/share/menu/***pacote-binário* em *pacote-binário*.

Veja **menufile**(5) para o seu formato. Veja **dh_installmenu**(1).

NOVIDADES Instalado em **usr/share/doc/***pacote-binário***/NEWS.Debian**.

Veja **dh_installchangelogs**(1).

patches/* Coleção de ficheiros patch **-p1** que são aplicados à fonte do autor antes de se compilar o pacote.

Nenhuns ficheiros patch são gerados pelo comando **debmake**.

Veja **dpkg-source**(1), "Secção 4.4" e "Secção 5.9".

patches/series -x1 A sequência de aplicação dos ficheiros patch **patches/***.

***pacotebinário*.preinst** -x3**, *pacotebinário*.postinst** -x3**, *pacotebinário*.prerm** -x3**, *pacotebinário*.postrm** -x3 Se estes ficheiros opcionais existirem, os ficheiros correspondentes são instalados no directório **DEBIAN** dentro do pacote binário após enriquecido pelo **debhelper**. Caso contrário, estes ficheiros no directório **DEBIAN** dentro do pacote binário são gerados pelo **debhelper**.

Sempre que possível, em vez disto deve ser usando um mais simples *pacotebinário***.maintscript**.

Veja **dh_installdeb**(1) e "Capítulo 6 - Scripts de maintainer de pacote e procedimento de instalação" no "Manual de Política Debian".

Veja também **debconf-devel**(7) e "3.9.1 Perguntando nos scripts do maintainer" no "Manual de Política Debian".

README.Debian -x1 Instalado no primeiro pacote binário listado no ficheiro **debian/control** como **usr/share/doc/***pacote-binário***/README.Debian**.

Este ficheiro fornece a informação especifica ao pacote Debian.

Veja **dh_installdocs**(1).

README.source -x1 Instalado no primeiro pacote binário listado no ficheiro **debian/control** como **usr/share/doc/***pacote-binário***/README.source**.

Se correr "**dpkg-source -x**" num pacote fonte não produz a fonte do pacote, pronto para editar, e permite-nos fazer alterações e correr **dpkg-buildpackage** para produzir um pacote modificado sem tomar nenhum passo adicional, é recomendado criar este ficheiro.

Veja "Manual de Política Debian secção 4.14".

pacote-binário*.service** -x3 Se isto existir, será instalado em **lib/systemd/system/pacote-binário***.service** em *pacote-binário*.

Veja **dh_systemd_enable**(1), **dh_systemd_start**(1), e **dh_installinit**(1).

source/format [-x1] O formato de pacote Debian.

- Use "**3.0 (quilt)**" para tornar este pacote não-nativo (popular)
- Use "**3.0 (native)**" para tornar este pacote nativo.

Veja "FORMATOS DE PACOTE FONTE" em **dpkg-source**(1).

source/lintian-overrides [-x2] Este ficheiro não é instalado, mas é sondado pelo comando **lintian** para fornecer sobreposições para o pacote fonte.

Veja **dh_lintian**(1) e **lintian**(1).

source/local-options e source/local-patch-header [-x4]

> Nota
>
> 
>
> Estes ficheiros não são compatíveis com o fluxo de trabalho do **dgit**. Veja "Secção 11.14".

Não há nenhuma razão para usar estes com a versão atual do **dpkg-source**(1).

source/options [-x2] O comando **dpkg-source** usa este conteúdo como suas opções. Isto é usado tipicamente com "Secção 11.13" e as opções podem ser:

- **auto-commit**
- **single-debian-patch**

Isto é incluído no pacote fonte gerado.

Veja "FORMATOS DE FICHEIRO" em **dpkg-source**(1).

source/patch-header [-x2] Texto de formato livre que é colocado no topo da patch automática gerada.

Isto está incluído no pacote fonte gerado e destina-se a ser submetido para o "`Secção 11.13`.".

Veja "FORMATOS DE FICHEIRO" em **dpkg-source**(1).

***pacote-binário*.symbols** [-x1] Os ficheiros de símbolos, se presentes, são passados ao comando **dpkg-gensymbols** para serem processados e instalados.

Veja **dh_makeshlibs**(1) e "Secção 10.16"..

***pacotebinário*.templates** Isto é o ficheiro de **modelos debconf** usado para fazer perguntas necessárias para configurar o pacote. Veja "Secção 10.22".

tests/control [-x1] Isto é o ficheiro de meta-dados de teste estilo-RFC822 definido em DEP-8. Veja **autopkgtest**(1) e "Secção 10.4".

A FAZER Instalado no primeiro pacote binário listado no ficheiro **debian/control** como **usr/share/doc/***pacote-binário***/TODO.Debian**.

Veja **dh_installdocs**(1).

pacote-binário*.tmpfile** [-x3] Se isto existir, será instalado em **usr/lib/tmpfiles.d/pacote-binário***.conf** em *pacote-binário*.

Veja **dh_systemd_enable**(1), **dh_systemd_start**(1), e **dh_installinit**(1).

***pacote-binário*.upstart** [-x4] Se isto existir, será instalado em **etc/init/package.conf** no directório de compilação do pacotes. (descontinuado)

Veja **dh_installinit**(1).

upstream/metadata [-x1] Metadados por-pacote legível-por-máquina acerca do autor (DEP-12). Veja "Upstream MEtadata GAthered com YAml (UMEGAYA)".

Capítulo 7

Qualidade de empacotamento

A qualidade do empacotamento Debian pode ser melhorada ao se usar ferramentas de teste.

- **lintian**(1)
- **piuparts**(1)
- **autopkgtest**(1)

Se você seguir “Capítulo 4”, estas são executadas automaticamente. Espera-se você corrija todos os avisos.

7.1 Reformatar ficheiros debian/* com wrap-and-sort

É uma boa ideia reformatar os ficheiros **debian/*** de forma consistente usando o comando **wrap-and-sort**(1) do pacote **devscripts**.

Reformatar ficheiros debian/*

```
[debhello-0.0] $ wrap-and-sort -vast
```

7.2 Validar ficheiros debian/* com o debputy

A nova ferramenta debputy [1] inclui sub-comandos par validar (e corrigir) a maioria dos ficheiros em **debian/***.

Verifica a correção dos ficheiros em debian/*

```
[debhello-0.0] $ debputy lint --spellcheck
```

Formata ficheiros debian/control e debian/tests/control

```
[debhello-0.0] $ debputy reformat --style black
```

Usar o comando “**debputy reformat**” torna obsoleto usar “**wrap-and-sort -vast**”.

A ferramenta debputy também inclui um servidor de linguagem. Você pode configurar para obter feedback em tempo real enquanto edita ficheiros **debian/*** com qualquer editor moderno que suporte Language Server Protocol.

7.3 Verificar o empacotamento com cme

É uma boa ideia verificar os ficheiros de configuração do **dpkg** usando o comando **cme**(1) no pacote **cme**. Isto é usado pelo DFSG, Licenciamento & Equipa de Novos Pacotes.

Verifica se está correto usar em cme/*

```
[debhello-0.0] $ cme fix --verbose dpkg
```

1O objectivo principal da ferramenta debputy é oferecer um novo paradigma de compilação de pacotes Debian. Este novo paradigma está para lá do escopo deste tutorial.

Capítulo 8

Higienização da fonte

Existem alguns casos que requerem higienizar a fonte para prevenir a contaminação do pacote fonte Debian gerado.

- Conteúdos não compatíveis com DFSG na fonte do autor.
  - Debian leva a sério a liberdade do software e adere a DFSG.
- Conteúdos estranhos auto-gerados na fonte do autor.
  - Pacotes Debian devem recompilar estes sob o sistema mais recente.
- Conteúdo VCS estranho na fonte do autor.
  - As opções **-i** e **-I** definidas em “Secção 4.6” para o comando **dpkg-source**(1) devem evitar isto.
    * A opção **-i** destina-se a pacotes Debian não nativos.
    * A opção **-I** destina-se a pacotes Debian nativos.

Existem vários métodos de evitar a inclusão de conteúdos indesejáveis.

8.1 Corrige com Files-Excluded

Este método é apropriado para evitar conteúdos não compatíveis com https://www.debian.org/social_contract.html#gu no tarball fonte do autor.

- Lista os ficheiros a serem removidos na estrofe **Files-Excluded** do ficheiro **debian/copyright**.
- Lista o URL para descarregar o tarball do autor no ficheiro **debian/watch**.
- Corre o comando **uscan** para descarregar o novo tarball de autor.
  - Em alternativa, use o comando “**gbp import-orig --uscan --pristine-tar**”.
- **mk-origtargz** invocado de **uscan** remove ficheiros excluídos do tarball do autor e re-empacota-o como um tarball limpo.
- O tarball resultante tem o número de versão com um sufixo adicional **+dfsg**.

Veja “**EXEMPLOS DE FICHEIROS COPYRIGHT**” em **mk-origtargz**(1).

8.2 Corrigir com “debian/rules clean”

Este método é apropriado para evitar ficheiros auto-gerados ao remove-los no alvo ”**debian/rules clean**”.

Nota



O alvo "**debian/rules clean**" é chamado antes do comando "**dpkg-source --build**" pelo comando **dpkg-buildpackage**. O comando "**dpkg-source --build**" ignora os ficheiros removidos.

8.3 Corrigir com extend-diff-ignore

Isto é para o pacote não-nativo Debian.

O problema dos diffs estranhos pode ser resolvido ao ignorar as alterações feitas a partes específicas da árvore fonte. Isto é feito ao adicionar a linha "**extend-diff-ignore=...**" no ficheiro **debian/source/options**.

debian/source/options para excluir os ficheiros config.sub, config.guess e Makefile:

```
# Don't store changes on autogenerated files
extend-diff-ignore = "(^|/)(config\.sub|config\.guess|Makefile)$"
```

Nota



Esta abordagem funciona sempre, mesmo quando você não consegue remover o ficheiro. Isto poupa-o de ter de fazer uma salvaguarda do ficheiro não-modificado apenas para o restaurar antes da próxima compilação.

Dica



Se você usar o ficheiro **debian/source/ local-options**, você pode esconder esta definição do pacote fonte gerado. Isto pode ser útil quando ficheiros VCS locais não-standard interferem com o seu empacotamento.

8.4 Corrigir com tar-ignore

Isto é para o pacote nativo Debian.

Você pode excluir certos ficheiros na árvore fonte do tarball gerado ao ajustar o globo de ficheiros. Adicione as linhas "**tar-ignore=...**" nos ficheiros **debian/source/options** ou **debian/source/local-options**.

Nota



Por exemplo, se o pacote fonte de um pacote nativo precisar de ficheiros com a extensão **.o** como uma parte dos dados de teste, a definição em "Secção 4.6" pode ser demasiado agressiva. Você pode contornar isto ao largar a opção **-I** para **DEBUILD_DPKG_BUILDPACKAGE_OPTS** em "Secção 4.6" e adicionar as linhas "**tar-ignore=...**" no ficheiro **debian/source/local-options** para cada pacote.

8.5 Corrigir com “git clean -dfx”

O problema do conteúdo estranho na segunda compilação pode ser evitado ao restaurar a árvore fonte. Isto é feito ao submeter a árvore fonte para o repositório Git antes da primeira compilação.

Você pode restaurar a árvore fonte antes da segunda compilação de pacote. Por exemplo:

```
[debhello] $ git reset --hard
[debhello] $ git clean -dfx
```

Isto funciona porque o comando **dpkg-source** ignora os conteúdos dos ficheiros VCS típicos na árvore fonte, como especificado pela definição **DEBUILD_DPKG_BUILDPACKAGE_OPTS** em “Secção 4.6”.

Dica



Se a árvore fonte não for gerida por um VCS, corra ”**git init; git add -A .; git commit**” antes da primeira compilação.

Capítulo 9

Mais sobre empacotamento

Vamos explorar mais fundamentos do empacotamento Debian.

9.1 Personalização do pacote

Todos os dados de personalização para o pacote fonte Debian residem no directório **debian/** como apresentado em “Secção 5.7”:

- O sistema de compilação de pacote Debian pode ser personalizado através do ficheiro **debian/rules** (veja “Secção 9.2”).
- O caminho de instalação do pacote Debian etc. pode ser personalizado através da adição de ficheiros de configuração como *pacote***.install** e *pacote***.docs** no directório **debian/** para os comandos **dh**_* do pacote **debhelper** (veja “Secção 6.14”).

Quando estes não são suficientes para fazer um bom pacote Debian, ficheiros patches **-p1** de **debian/patches/*** são implantados para modificar a fonte do autor. Estes são aplicados uma sequência definida no ficheiro **debian/patches/series** antes de compilar o pacote como apresentado em “Secção 5.9”.

Você deve endereçar a causa raiz do problema de empacotamento Debian pela maneira menos invasiva possível. Esta abordagem irá fazer o pacote gerado mais robusto para atualizações futuras.

Nota



Se a patch que endereça a causa raiz é útil para o projeto do autor, envie-a para o maintainer do autor.

9.2 debian/rules personalizado

A personalização flexível do Secção 6.5 é alcançado ao adicionar alvos **override_dh_*** apropriados e suas regras.

Quando uma operação especial é requerida para um certo comando **dh**_*foo* invocado pelo comando **dh**, a sua execução automática pode ser sobreposta ao adicionar o alvo makefile **override_dh**_*foo* no ficheiro **debian/rules**.

O processo de compilação pode ser personalizado via interface fornecida pelo autor como argumentos para os comandos do sistema de compilação de fonte de autor, tais como:

- **configure**,
- **Makefile**,
- “**python -m build**”, ou

- **Build.PL**.

Neste caso, você deve adicionar o alvo **override_dh_auto_build** com "**dh_auto_build --** *argumentos*". Isto assegura que os *argumentos* são passados para o sistema de compilação após os parâmetros predefinidos que o **dh_auto_build** geralmente passa.

Dica



Evite executar os comandos crus do sistema de compilação diretamente se eles forem suportados pelo comando **dh_auto_build**.

Veja:

- "Secção 5.7" para exemplo básico.
- "Secção 10.3" para ser lembrado do desafio que envolve o sistema de compilação subjacente.
- "Secção 10.10" para personalização multiarch.
- "Secção 10.6" para personalização de endurecimento.

9.3 Variáveis para debian/rules

Algumas definições de variável úteis para personalizar o **debian/rules** podem ser encontradas em ficheiros sob **/usr/share/dpkg/**. A notar:

pkg-info.mk Define variáveis **DEB_SOURCE**, **DEB_VERSION**, **DEB_VERSION_EPOCH_UPSTREAM**, **DEB_VERSION_UPSTREAM_REVISION**, **DEB_VERSION_UPSTREAM**, e **DEB_DISTRIBUTION** obtidas de **dpkg-parsechangelog**(1). (úteis para suporte a backport etc...)

vendor.mk Define variáveis **DEB_VENDOR** e **DEB_PARENT_VENDOR**; e macro **dpkg_vendor_derives_from** obtidas de **dpkg-vendor**(1). (úteis para suporte a fornecedor (Debian, Ubuntu, …).)

architecture.mk Veja variáveis **DEB_HOST**_* e **DEB_BUILD**_* obtidas de **dpkg-architecture**(1).

buildflags.mk Define as bandeiras de compilação **CFLAGS**, **CPPFLAGS**, **CXXFLAGS**, **OBJCFLAGS**, **OBJCXXFLAGS**, **GCJFLAGS**, **FFLAGS**, **FCFLAGS**, e **LDFLAGS** obtidas de **dpkg-buildflags**(1).

Por exemplo, você pode adicionar uma opção extra ao **CONFIGURE_FLAGS** para alvo **linux-any** de arquitecturas ao adicionar o seguinte ao **debian/rules**:

```
DEB_HOST_ARCH_OS ?= $(shell dpkg-architecture -qDEB_HOST_ARCH_OS)
 ...
ifeq ($(DEB_HOST_ARCH_OS),linux)
CONFIGURE_FLAGS += --enable-wayland
endif
```

Veja "Secção 10.10", **dpkg-architecture**(1) e **dpkg-buildflags**(1).

9.4 Novo lançamento do autor

Quando um novo tarball de lançamento de autor **debhello-***novaversão***.tar.xz** é lançado, o pacote fonte Debian pode ser atualizado ao invocar comandos na árvore fonte antiga como:

```
[debhello-0.0] $ uscan
 ... debhello-newversion.tar.xz downloaded
[debhello-0.0] $ uupdate -v newversion ../debhello-newversion.tar.xz
```

- O ficheiro **debian/watch** na árvore fonte antiga tem de ser um válido.

- Isto cria um link simbólico **../debhello**_*novaversão*.**orig.tar.xz** a apontar para **../debhello-***novaversão***.tar.xz**.
- Ficheiros são extraídos de **../debhello-***novaversão***.tar.xz**para **../debhello-***novaversão***/**
- Ficheiros são copiados de **../debhello-***versãoantiga***/debian/** para **../debhello-***novaversão***/debian/** .

Após o de cima, você deve refrescar os ficheiros **debian/patches/*** (veja "Secção 9.5") e atualizar **debian/changelog** com o comando **dch**(1).

Quando "**debian uupdate**" é especificado no final da linha no ficheiro **debian/watch**, o **uscan** executa automaticamente **uupdate**(1) após descarregar o tarball.

9.5 Gerir a lista de patch com dquilt

Você pode adicionar, retirar, e refrescar ficheiros **debian/patches/**com **dquilt** para gerir a lista de patch.

- **Adiciona** uma nova patch **debian/patches/***nomebug.patch* registando a modificação à fonte do autor no ficheiro *ficheiro_buggy* como:

```
[debhello-0.0] $ dquilt push -a
[debhello-0.0] $ dquilt new bugname.patch
[debhello-0.0] $ dquilt add buggy_file
[debhello-0.0] $ vim buggy_file
  ...
[debhello-0.0] $ dquilt refresh
[debhello-0.0] $ dquilt header -e
[debhello-0.0] $ dquilt pop -a
```

- **Drop** (== desactiva) um caminho existente
  - Comenta linha pertinente em **debian/patches/series**
  - Apagar o próprio caminho (opcional)
- **Refresca** ficheiros **debian/patches/*** para fazer o "**dpkg-source -b**" funcionar como esperado após atualizar um pacote Debian para o novo lançamento do autor.

```
[debhello-0.0] $ uscan; uupdate  # updating to the new upstream release
[debhello-0.0] $ while dquilt push; do dquilt refresh ; done
[debhello-0.0] $ dquilt pop -a
```

  - Se, em cima, forem encontrados conflitos com "**dquilt push**", resolva-os e depois corra "**dquilt refresh**" manualmente para cada um deles.

9.6 Comandos de compilação

Aqui vamos recapitular os comandos de compilação de pacotes de baixo nível disponíveis. Existem muitas maneiras de se fazer a mesma coisa.

- **dpkg-buildpackage** = núcleo da ferramenta de compilação de pacote
- **debuild** = **dpkg-buildpackage** + **lintian** (compilação sob as variáveis de ambiente higienizadas)
- **schroot** = núcleo da ferramenta de ambiente chroot de Debian
- **sbuild** = **dpkg-buildpackage** em **schroot** personalizado (compilar na chroot)

9.7 Nota sobre o sbuild

O comando **sbuild**(1) é um script invólucro do **dpkg-buildpackage** o qual compila pacotes binário Debian num ambiente chroot gerido pelo comando **schroot**(1). Por exemplo, compilar para a suite Debian **unstable** pode ser feito assim:

```
[debhello-0.0] $ sudo sbuild -d unstable
```

Em terminologia do **schroot**(1), isto compila um pacote Debian num limpo e efémero chroot "**chroot:unstable-amd64-sbuild**" começando como uma cópia do mínimo persistente limpo chroot "**source:unstable-amd64-sbuild**".

Este ambiente de compilação foi configurado como descrito em "Secção 4.7" com "**sbuild-debian-developer-setup** -s unstable" o que essencialmente fez o seguinte:

```
[~] $ sudo mkdir -p /srv/chroot/dist-amd64-sbuild
[~] $ sudo sbuild-createchroot unstable /srv/chroot/unstable-amd64-sbuild http:// ↩
    deb.debian.org/debian
[~] $ sudo usermod -a -G sbuild <your_user_name>
[~] $ sudo shutdown -r now
 ... reboot
```

A configuração **schroot**(1) para **unstable-amd64-sbuild** foi gerada em **/etc/schroot/chroot.d/unstable-amd64-sbuild**.*$suffix* :

```
[unstable-amd64-sbuild]
description=Debian sid/amd64 autobuilder
groups=root,sbuild
root-groups=root,sbuild
profile=sbuild
type=directory
directory=/srv/chroot/unstable-amd64-sbuild
union-type=overlay
```

Aqui:

- O perfil definido no directório **/etc/schroot/sbuild/** é usado para configurar o ambiente chroot.
- **/srv/chroot/unstable-amd64-sbuild** directório que contém o sistema de ficheiros da chroot.
- **/etc/sbuild/unstable-amd64-sbuild** é ligado por link simbólico a **/srv/chroot/unstable-amd64-sbuild** .

Você pode atualizar esta chroot fonte "**source:unstable-amd64-sbuild**" fazendo:

```
[~] $ sudo sbuild-update -udcar unstable
```

Você pode iniciar sessão nesta chroot fonte "**source:unstable-amd64-sbuild**" assim:

```
[~] $ sudo sbuild-shell unstable
```

Dica



Se o seu sistema de ficheiros chroot fonte tem falta de pacotes como **libeatmydata1**, **ccache**, e **lintian**, para as suas necessidades, você pode querer instalar estes ao iniciar sessão nele.

9.8 Casos especiais de compilação

O ficheiro **orig.tar.xz** pode precisar de ser enviado para uma revisão Debian diferente de **0** ou **1** sob alguns casos excecionais (ex. para um envio de segurança).

Quando um pacote essencial se torna não-essencial (ex, **adduser**), você precisa de remove-lo manualmente do ambiente chroot existente para o seu uso pelo **piuparts**.

9.9 Enviar orig.tar.xz

Quando você faz o primeiro envio do pacote para o arquivo, você tem de incluir a fonte **orig.tar.xz** original, também.

Se o número de revisão Debian do pacote for um de **1** ou **0**, isto é a predefinição. Caso contrário, você tem de fornecer a opção do **dpkg-buildpackage -sa** ao comando **dpkg-buildpackage**.

- **dpkg-buildpackage -sa**
- **debuild -sa**
- **sbuild --debbuildopts=-sa**
- **gbp buildpackage -sa**

Dica



Por outro lado, a opção **-sd** irá forçar a exclusão da fonte **orig.tar.xz** original.

Dica



Envios de segurança requerem incluir o ficheiro **orig.tar.xz**.

9.10 Envios saltados

Se você criou múltiplas entradas no **debian/changelog** enquanto saltou envios, você tem de criar um ficheiro ***_.changes** apropriado que inclui todas as alterações desde o último envio. Isto pode ser feito ao especificar a opção do dpkg-buildpackage **-v** com a última versão enviada, ex., *1.2*.

- **dpkg-buildpackage -v***1.2*
- **debuild -v***1.2*
- **sbuild --debbuildopts=-v***1.2*
- **gbp buildpackage -v***1.2*

9.11 Relatórios de bug

O comando **reportbug**(1) usado para o relatório de bug de *pacote-binário* pode ser personalizado pelos ficheiros em **usr/share/bug/***pacote-binário***/**.

O comando **dh_bugfiles** instala estes ficheiros a partir dos ficheiros modelo no directório **debian/**.

- **debian/***pacote-binário***.bug-control** → **usr/share/bug/***pacote-binário***/control**
  - Este ficheiro contém algumas direções tais como redirecionar o relatório de bug para outro pacote.
- **debian/***pacote-binário***.bug-presubj** → **usr/share/bug/***pacote-binário***/presubj**
  - Este ficheiro é mostrado ao utilizador pelo comando **reportbug**.
- **debian/***pacote-binário***.bug-script** → **usr/share/bug/***pacote-binário* ou **usr/share/bug/***pacote-binário***/script**

– O comando **reportbug** corre este script para gerar um ficheiro modelo para o relatório de bug.

Veja **dh_bugfiles**(1) e “Funcionalidades do reportbug para Desenvolvedores (**README.developers**)”

Dica



Se você está sempre a lembrar o relatador de bug de algo ou a pergunta sobre a sua situação, use estes ficheiros para o automatizar.

Capítulo 10

Empacotamento avançado

Vamos descrever tópicos avançados no empacotamento Debian.

10.1 Perspetiva histórica

Vou sobre-simplificar a perspetiva histórica das práticas de empacotamento Debian focando o empacotamento não-nativo.

Debian começou nos anos 1990 quando os pacotes de autor estavam disponíveis a partir de sites FTP públicos como o Sunsite. Nesses primeiros dias, o empacotamento Debian usada o formato fonte Debian actualmente conhecido como formato fonte Debian "**1.0**":

- O pacote fonte Debian contém um conjunto de ficheiros para o pacote fonte Debian.
  - *pacote_versão*.**orig.tar.xz** : link simbólico ou cópia do ficheiro de lançamento do autor.
  - *pacote_versão-revisão*.**diff.gz** : "**Uma grande patch**" para modificações Debian.
  - *pacote_versão-revisão*.**dsc**: descrição do pacote.
- Várias abordagens de contorno como o **dpatch**, **dbs**, ou **cdbs** foram implantadas para gerir patches de múltiplos tópicos.

O formato fonte Debian moderno "**3.0 (quilt)**" foi inventado cerca de 2008 (veja "ProjectsDebSrc3.0"):

- O pacote fonte Debian contém um conjunto de ficheiros para o pacote fonte Debian.
  - *pacote_versão*.**orig.tar.?z** : link simbólico ou cópia do ficheiro de lançamento do autor.
  - *pacote_versão-revisão*.**debian.tar.?z** : tarball de **debian/** para modificações Debian.
    - O ficheiro **debian/source/format** contém "**3.0 (quilt)**".
    - As patches de tópico múltiplo opcionais estão guardadas no directório **debian/patches/**.
  - *pacote_versão-revisão*.**dsc**: descrição do pacote.
- A abordagem standard para gerir patches de tópico múltiplo usando **quilt**(1) é implantada para o formato fonte Debian "**3.0 (quilt)**".

A maioria dos pacotes Debian adotaram os formatos fonte Debian "**3.0 (quilt)**" e "**3.0 (native)**".

Agora, o **git**(1) é popular entre autores e desenvolvedores Debian. O **git** e as suas ferramentas associadas são parte importante do fluxo de trabalho de empacotamento moderno de Debian. Este fluxo de trabalho moderno que envolve **git** será mencionado mais tarde em "Capítulo 11".

10.2 Tendências actuais

As práticas de empacotamento Debian actuais e suas tendências são um alvo em movimento. Veja:

- "Tendências Debian" — Dicas para "Standard de facto" das práticas Debian.
  - Sistemas de compilação: **dh**

  - Formato fonte Debian: "**3.0 (quilt)**"
  - VCS: **git**
  - Alojamento VCS: salsa
  - Rules-Requires-Root: adotado, fakeroot
  - Formato de Copyright: DEP-5

- "**debhelper-compat-upgrade-checklist**(7)" — Lista de verificação de actualização para o **debhelper**
- "DEP - Propostas de Melhoramento Debian"— Propostas formais para melhorar Debian

Você pode também procurar os dados do código fonte inteiro de Debian por si mesmo.

- "Fontes Debian" — ferramenta de busca de código
  - "Busca de Código Debian" — página wiki que descreve a sua utilização
- "Busca de Código Debian" — outra ferramenta de busca de código

10.3 Nota sobre sistema de compilação

Pode ser encontrados ficheiros auto-gerados do sistema de compilação no tarball do autor. Estes devem ser re-gerados quando o pacote Debian é compilado. Ex.:

- "**dh $@ --with autoreconf**" deve ser usado no **debian/rules** se forem usados Autotools (**autoconf** + **automake**).

Alguns sistemas de compilação modernos podem ser capazes de descarregar códigos fonte e ficheiros binários requeridos a partir de máquinas remotas arbitrárias para satisfazer requerimentos de compilação. Não use esta funcionalidade de download. É requerido que o pacote Debian oficial seja compilado apenas com os pacotes listados em **Build-Depends:** do ficheiro **debian/control**.

10.4 Integração contínua

O comando **dh_auto_test**(1) é um comando **debhelper** que tenta correr automaticamente a suite de teste fornecida pelo desenvolvedor autor durante o processo de compilação do pacote Debian.

O comando **autopkgtest**(1) pode ser usado após o processo de compilação do pacote Debian. Ele testa os pacotes binário Debian gerados num ambiente virtual usando o ficheiro de metadados estilo RFC822 **debian/tests/control** como integração contínua (CI). Veja:

- Documentos no directório **/usr/share/doc/autopkgtest/**
- "4. autopkgtest: Teste automático para pacotes" do "Guia de Empacotamento Ubuntu"

Existem várias outras ferramentas CI em Debian para você explorar.

- O Salsa oferece "Secção 11.3".
- O pacote **debci**: Plataforma CI no topo do pacote **autopkgtest**
- O pacote **jenkins**: plataforma CI genérica

10.5 Bootstrapping

Debian preocupa-se com suporte a novos portes ou sabores. Os novos portes ou sabores requerem a operação bootstrapping para a compilação-cruzada do sistema inicial mínimo de compilação-nativa. De modo a evitar ciclos infinitos de dependências-de-compilação durante o bootstrapping, as dependências de compilação precisam ser reduzidas usando a variável de ambiente **DEB_BUILD_PROFILES**.

Veja Debian wiki: “BuildProfileSpec”.

Dica



Se um pacote núcleo **foo** tiver dependências de compilação num pacote **bar** com cadeias de dependência de compilação funda mas **bar** é apenas usado no alvo **test** em **foo**, você pode marcar seguramente o **bar** com **<!nocheck>** em **Build-depends** de **foo** para evitar ciclos na compilação.

10.6 Endurecimento de compilação

O suporte de endurecimento de compilação lançado para Debian **jessie** (8.0) obriga a que prestemos atenção extra ao empacotamento.

Você deve ler as seguintes referências em detalhe.

- Debian wiki: “Endurecimento”
- Debian wiki: “Endurecimento Passo a Passo”

O comando **debmake** adiciona comentários modelo ao ficheiro **debian/rules** como necessário para **DEB_BUILD_MAINT_OPTIONS**, **DEB_CFLAGS_MAINT_APPEND**, e **DEB_LDFLAGS_MAINT_APPEND** (veja “Capítulo 5” e **dpkg-buildflags**(1)).

10.7 Compilação reproduzível

Aqui estão algumas recomendações para obter um resultado de compilação reproduzível.

- Não embuta o selo temporal baseado na hora do sistema.
- Não embute o caminho do ficheiro do ambiente de compilação.
- Use “**dh $@**” no **debian/rules** para aceder às funcionalidades mais recentes do **debhelper**.
- Exporta o ambiente de compilação como “**LC_ALL=C.UTF-8**” (veja “Secção 12.1”).
- Defina o selo temporal usado na fonte do autor a partir do valor da variável de ambiente fornecida pelo debhelper **$SOURCE_DATE_EPOCH**.
- Leia mais em “ReproducibleBuilds”.
  - “Como Fazer Compilações Reproduzíveis”.
  - “Proposta de Selos Temporais para Compilações Reproduzíveis”.

Compilações reproduzíveis são importantes para garantia de segurança e qualidade. Elas permitem verificações independentes de que nenhumas vulnerabilidades ou backdoors foram introduzidas durante o processo de compilação.

O ficheiro de controle *nome-fonte_versão-fonte_arch***.buildinfo** gerado pelo **dpkg-genbuildinfo**(1) guarda o ambiente de compilação. Veja **deb-buildinfo**(5)

10.8 Substvar

O ficheiro **debian/control** também define a dependência do pacote no qual o “mecanismo de substituição de variáveis” (substvar) pode ser usada para libertar os maintainers de pacotes de tarefas de seguir a maioria dos casos de dependências simples de pacotes. Veja **deb-substvars**(5).

O comando **debmake** suporta os seguintes substvars:

- **${misc:Depends}** para todos os pacotes binário

- **${misc:Pre-Depends}** para todos os pacotes multiarch
- **${shlibs:Depends}** para todos os pacotes binário executáveis e bibliotecas
- **${python:Depends}** Para todos os pacotes Python
- **${python3:Depends}** Para todos os pacotes Python3
- **${perl:Depends}** para todos os pacotes Perl
- **${ruby:Depends}** para todos os pacotes Ruby

Para a biblioteca partilhada, as bibliotecas requeridas encontradas simplesmente por "**objdump -p** */caminho/para/programa* **| grep NEEDED**" são cobertas pelo substvar **shlib**.

Para Python e outros interpretes, os modules requeridos encontrados simplesmente procurando por linhas com "**import**", "**use**", "**require**", etc., são cobertos pelos substvars correspondentes.

Para outros programas que não implantam os seus próprios substvars, o substvar **misc** cobre as suas dependências.

Para programas de shell POSIX, não existe maneira fácil de identificar a dependência e nenhum substvar cobre a sua dependência.

Para bibliotecas e módulos requeridos via mecanismo de carga dinâmico incluindo o mecanismo "GObject introspection", não existe maneira fácil de identificar a dependência e nenhum substvar cobre a sua dependência.

10.9 Pacote de biblioteca

Empacotar software biblioteca requer que você execute muito mais trabalho que o normal. Aqui estão alguns lembretes para empacotamento de software biblioteca:

- O pacote binário de biblioteca tem de ser nomeado como em "Secção 10.17".
- Debian envia bibliotecas partilhadas como a **/usr/lib/<triplet>/lib***foo-0.1***.so.***1.0.0* (veja "Secção 10.10").
- Debian encoraja o uso de símbolos versionados na biblioteca partilhada (veja "Secção 10.16").
- Debian não envia ficheiros de arquivo de biblioteca libtool ***.la**.
- Debian desencoraja o uso e envio de ficheiros de biblioteca estática ***.a**.

Antes de empacotar software de biblioteca partilhada, veja:

- "Capítulo 8 - Bibliotecas partilhadas" no "Manual de Política Debian"
- "10.2 Bibliotecas" no "Manual de Política Debian"
- "6.7.2. Bibliotecas" no "Debian Developer's Reference"

Para o estudo de fundo histórico, veja:

- "Escapar ao Inferno das Dependências" 1
  - Isto encoraja a termos símbolos versionados na biblioteca partilhada.
- "Guia de Empacotamento de Biblioteca Debian" 2
  - Por favor leia também o tema de discussão que se seguiu ao seu anúncio.

1Este documento foi escrito antes da introdução do ficheiro **symbols**.

2A forte preferência é usar nomes de pacotes **-dev** versionados SONAME sobre o nome de pacote **-dev** singular em "Capítulo 6. Desenvolvimento de pacotes (-DEV)", o que não parece ser partilhado pelo anterior ftp-master (Steve Langasek). Este documento foi escrito antes da introdução do sistema **multiarch** e do ficheiro **symbols**.

10.10 Multiarch

Suporte Multiarch para instalação de arquitectura-cruzada de pacotes binários (particularmente **i386** e **amd64**, mas também outras combinações) nos pacotes **dpkg** e **apt** introduzidos em Debian **wheezy** (7, Maio 2013), obriga a que tenhamos atenção extra no empacotamento.

Você deve ler as seguintes referências em detalhe.

- Ubuntu wiki (autor)
  - "Especificações de Multi Arquitectura"
- Debian wiki (situação de Debian)
  - "Suporte a multi-arquitectura Debian"
  - "Implementação/Multiarch"

O multiarch é activado ao usar o valor **<triplet>** como **i386-linux-gnu** e **x86_64-linux-gnu** no caminho de instalação de várias bibliotecas como **/usr/lib/<triplet>/**, etc...

- O valor **<triplet>** requerido internamente pelo script **debhelper** é definido implicitamente neles próprios. O maintainer não tem de se preocupar.
- O valor **<triplet>** usado nos scripts alvo **override_dh_*** têm de ser explicitamente definidos no ficheiro **debian/rules** pelo maintainer. O valor **<triplet>** é guardado na variável **$(DEB_HOST_MULTIARCH)** no seguinte exemplo de trecho do **debian/rules**:

```
DEB_HOST_MULTIARCH = $(shell dpkg-architecture -qDEB_HOST_MULTIARCH)
...
override_dh_install:
  mkdir -p package1/lib/$(DEB_HOST_MULTIARCH)
  cp -dR tmp/lib/. package1/lib/$(DEB_HOST_MULTIARCH)
```

Veja:

- "Secção 9.3"
- "Secção 16.2"
- "Secção 10.12"
- "**dpkg-architecture**(1) manpage"

10.11 Divisão de um pacote binário Debian

Para sistemas de compilação bem comportados, a divisão de um pacote binário Debian em pacotes menores pode ser realizada como se segue.

- Criar entradas em pacote binário para todos os pacotes binário no ficheiro **debian/control**.
- Listar todos os caminhos de ficheiro (relativos a **debian/tmp**) nos ficheiros **debian/***pacotebinário***.install** correspondentes.

Por favor verifique os exemplos neste guia:

- "Secção 14.11" (baseado em Autotools)
- "Secção 14.12" (baseado em CMake)

Um método intuitivo e flexível de criar o ficheiro modelo inicial **debian/control** definindo a divisão dos pacotes binário Debian com a opção **-b**. Veja "Secção 16.2".

10.12 Cenário de divisão de pacote e exemplos

Aqui estão alguns cenários de divisão de pacote multiarch típicos para os seguintes exemplos de fonte de autor usando o comando **debmake**:

- uma fonte de biblioteca *libfoo-1.0***.tar.xz**
- uma fonte de ferramenta *bar-1.0***.tar.xz** escrita numa linguagem compilada
- uma fonte de ferramenta *baz-1.0***.tar.xz** escrita numa linguagem interpretada.

pacote-binário	***tipo***	**Architecture:**	**Multi-Arch:**	**Conteúdo do pacote**
lib*foo1*	**lib***	any	same	a biblioteca partilhada, co-instalável
lib*foo***-dev**	**dev***	any	same	os ficheiros de cabeçalho da biblioteca partilhada etc., co-instaláveis
lib*foo***-tools**	**bin***	any	foreign	os programas de suporte de tempo-de-execução, não co-instaláveis
lib*foo***-doc**	**doc***	all	foreign	os ficheiros de documentação da biblioteca partilhada
bar	**bin***	any	foreign	os ficheiros do programa compilado, não co-instaláveis
*bar***-doc**	**doc***	all	foreign	os ficheiros de documentação para o programa
baz	**script**	all	foreign	os ficheiros de programa interpretados

10.13 Caminho da biblioteca Multiarch

A política Debian requer que se cumpra com "Filesystem Hierarchy Standard (FHS), versão 3.0", com as excepções anotadas em "Estrutura de Sistema de Ficheiros".

A excepção mais notável é o uso de **/usr/lib/<triplet>/** em vez de **/usr/lib<qual>/** (ex., **/lib32/** e **/lib64/**) para suportar uma biblioteca multiarch.

Tabela 10.2 As opções do caminho da biblioteca multiarch

Caminho clássico	**caminho multiarch i386**	**caminho multiarch amd64**
/lib/	**/lib/i386-linux-gnu/**	**/lib/x86_64-linux-gnu/**
/usr/lib/	**/usr/lib/i386-linux-gnu/**	**/usr/lib/x86_64-linux-gnu/**

Para pacotes baseados em Autotools sob o pacote **debhelper** (compat>=9), esta definição de caminho é cuidada automaticamente pelo comando **dh_auto_configure**.

Para outros pacote com sistemas de compilação não-suportados, você tem de ajustar manualmente o caminho de instalação como se segue.

- Se "**./configure**" for usado no alvo **override_dh_auto_configure** em **debian/rules**, certifique-se de o substituir por "**dh_auto_configure --**" enquanto redefine o alvo do caminho de instalação de **/usr/lib/** para **/usr/lib/$(DEB_HOST_MULTIARCH)/**.
- Substitua todas as ocorrências de **/usr/lib/** por **/usr/lib/*/** nos ficheiros **debian/***foo***.install**.

Todos os ficheiros instalados em simultâneo como o pacote multiarch para o mesmo caminho de ficheiro devem ter exatamente o mesmo conteúdo de ficheiro. Você tem de ter cuidado com as diferenças geradas pela ordem de bytes de dados e pelo algoritmo de compressão.

Os ficheiros de biblioteca partilhada nos caminhos predefinidos **/usr/lib/** e **/usr/lib/<triplet>/** são carregados automaticamente.

Para ficheiros de biblioteca partilhada noutro caminho, a opção GCC **-l** tem de ser definida pelo comando **pkg-config** para fazer com que elas carreguem de maneira apropriada.

10.14 Caminho do ficheiro de cabeçalho Multiarch

GCC inclui ambos **/usr/include/** e **/usr/include/<triplet>/** por predefinição no sistema Debian multiarch.

Se o ficheiro cabeçalho não estiver nesses caminhos, a opção GCC **-I** tem de ser definida pelo comando **pkg-config** para fazer "**#include <***foo.h***>**" funcionar de maneira apropriada.

Tabela 10.3 As opções do caminho do ficheiro de cabeçalho multiarch

Caminho clássico	caminho multiarch i386	caminho multiarch amd64
/usr/include/	**/usr/include/i386-linux-gnu/**	**/usr/include/x86_64-linux-gnu/**
/usr/include/*nomepacote*	**/usr/include/i386-linux-gnu/***nomepacote***/**	**/usr/include/x86_64-linux-gnu/***nomepacote***/**
	/usr/lib/i386-linux-gnu/*nomepacote***/**	**/usr/lib/x86_64-linux-gnu/***nomepacote***/**

O uso do caminho **/usr/lib/<triplet>/***nomepacote***/** para os ficheiros biblioteca permite ao maintainer o autor usar o mesmo script de instalação para o sistema multi-arch com **/usr/lib/<triplet>** e o sistema bi-arch com **/usr/lib<qual>/**. 3

O uso de caminho de ficheiro que contém *nomepacote* permite ter mais de 2 bibliotecas de desenvolvimento instaladas simultaneamente num sistema.

10.15 Caminho do ficheiro *.pc Multiarch

O programa **pkg-config** é usado para obter informação acerca de bibliotecas instaladas no sistema. Ele guarda os seus parâmetros de configuração no ficheiro ***.pc** e é usado para definir as opções **-I** e **-l** para o GCC.

Tabela 10.4 As opções de caminho do ficheiro ***.pc**

Caminho clássico	caminho multiarch i386	caminho multiarch amd64
/usr/lib/pkgconfig/	**/usr/lib/i386-linux-gnu/pkgconfig/**	**/usr/lib/x86_64-linux-gnu/pkgconfig/**

10.16 Símbolos de biblioteca

O suporte de símbolos em **dpkg** introduzido em Debian **lenny** (5.0, Maio 2009) ajuda-nos a gerir a compatibilidade ABI com versões anteriores do pacote biblioteca com o pacote com o mesmo nome. O ficheiro **DEBIAN/symbols** no pacote binário fornece a versão mínima associada a cada símbolo.

Um método sobre-simplificado para o empacotamento de biblioteca é como se segue:

- Extrair o ficheiro **DEBIAN/symbols** antigo do pacote binário imediatamente anterior com o comando "**dpkg-deb -e**".
  - Alternativamente, o comando **mc** pode ser usado para extrair o ficheiro **DEBIAN/symbols**.
- Copie-o para ficheiro **debian/***pacote-binário***.symbols**.
  - Se este for o primeiro pacote, use antes um ficheiro de conteúdo vazio.
- Compilar o pacote binário
  - Se o comando **dpkg-gensymbols** avisar acerca de alguns novos símbolos:
    - Extrair o ficheiro **DEBIAN/symbols** atualizado com o comando "**dpkg-deb -e**".
    - Corte a revisão Debian tal como **-1** nele.

3Este caminho é compatível com. "Standard de Hierarquia de Sistema de Ficheiros: /usr/lib : Bibliotecas para programação e pacotes" declara "As aplicações podem usar um único sub-directório sob **/usr/lib**. Se uma aplicação usar um sub-directório, todos os dados independentes-de-arquitectura exclusivamente usados pela aplicação têm de ser colocados dentro desse sub-directório."

    * Copie-o para ficheiro **debian/***pacote-binário***.symbols**.
    * Re-compilação do pacote binário.
  - Se o comando **dpkg-gensymbols** não avisar acerca de novos símbolos:
    * Você terminou com o empacotamento de biblioteca.

Para detalhes, você deve ler as seguintes referências primárias.

- "8.6.3 O sistema de símbolos" do "Manual de Política Debian"
- "manual do **dh_makeshlibs**(1)"
- "manual do **dpkg-gensymbols**(1)"
- "manual do **dpkg-shlibdeps**(1)"
- "manual do **deb-symbols**(5)"

Você deve também verificar:

- Debian wiki: "UsarFicheirosSymbols"
- Debian wiki: "Projectos/DpkgShlibdepsMelhorados"
- Equipa kde Debian: "Trabalhar com ficheiros de símbolos"
- "Secção 14.11"
- "Secção 14.12"

Dica



Para bibliotecas C++ e outros casos onde acompanhar os símbolos é problemático, então siga "8.6.4 O sistema shlibs" do "Manual de Política Debian". Por favor certifique-se de apagar o ficheiro vazio **debian/***pacotebinário***.symbols** gerado pelo comando **debmake**. Para este caso, é usado o ficheiro **DEBIAN/shlibs**.

10.17 Nome de pacote da biblioteca

Vamos considerar que o tarball fonte do autor da biblioteca **lib***foo* é actualizada de **lib***foo-7.0***.tar.xz** para **lib***foo-8.0***.tar.xz** com um novo SONAME de versão maior que afecta outros pacotes.

O pacote biblioteca binário tem de ser renomeado de **lib***foo7* para **lib***foo8* para manter o sistema da suite **unstable** a funcionar para todos os pacotes dependentes após o envio do pacote baseado em **lib***foo-8.0***.tar.xz**.

Atenção



Se o pacote biblioteca binário não for renomeado, muitos pacotes dependentes na suite **unstable** ficam quebrados logo após o envio da biblioteca mesmo que seja requisitado um envio binNMU. O binNMU pode não acontecer imediatamente após o envio devido a várias razões.

O pacote **-dev** tem de seguir uma das seguintes regras de nomeação:

- Use o nome de pacote **não-versionado -dev**: **lib***foo***-dev**
  - Isto é o típico para pacotes biblioteca leaf.
  - Apenas uma versão do pacote fonte de biblioteca é permitida no arquivo.

    * O pacote da biblioteca associada precisa ser renomeado de **lib***foo7* para **lib***foo8* para prevenir quebra de dependência na suite **unstable** durante a transição da biblioteca.
  - Esta abordagem deve ser usada se o binNMU simples resolver a dependência da biblioteca rapidamente para todos os pacotes afectados. (Mudança de ABI ao abandonar o descontinuado API enquanto se mantém a API ativa não modificada.)
  - Esta abordagem pode ainda ser boa ideia se atualizações de código manuais, etc. possam ser coordenadas e tratáveis dentro de pacote limitados. (mudança de API)

- Use os nomes de pacote **versionados -dev**: **lib***foo7***-dev** e **lib***foo8***-dev**
  - Isto é típico para muitos pacotes biblioteca principais.
  - São permitidas em simultâneo duas versões de pacotes fonte de biblioteca no arquivo.
    * Faz todos os pacotes dependentes depender de **lib***foo***-dev**.
    * Faz ambos **lib***foo7***-dev** e **lib***foo8***-dev** fornecerem **lib***foo***-dev**.
    * O pacote fonte precisa de ser renomeado como **lib***foo7-7.0***.tar.xz** e **lib***foo8-8.0***.tar.xz** respetivamente de **lib***foo-?.0***.tar.xz**.
    * O caminho do ficheiro de instalação especifico do pacote incluindo **lib***foo7* e **lib***foo8* respetivamente para ficheiros cabeçalho etc. precisa de ser escolhido para os tornar co-instaláveis.
  - Não utilize esta abordagem pesada, se possível.
  - Esta abordagem deve ser usada se a actualização de múltiplos pacotes dependentes requererem actualizações de código manuais, etc. (Mudança de API). Caso contrário, os pacotes afectados tornam-se RC buggy com FTBFS (Fails To Build From Source).

Dica



Se o esquema de codificação de dados alterar (ex. latin1 para utf-8), é preciso tomar a mesma alteração na API.

Veja “Secção 10.9”.

10.18 Transição de biblioteca

Quando você empacota uma nova versão de pacote biblioteca que afecta outros pacotes, você tem de preencher um relatório de bug de transição contra o pseudo pacote **release.debian.org** usando o comando **reportbug** com o ficheiro ben e esperar a aprovação do seu envio pela Equipa de Lançamento.

Lançamento de equipa tem o “seguidor de transição”. Veja “Transições”.

Cuidado



Por favor certifique-se de renomear pacotes binários como em “Secção 10.17”.

10.19 binNMU seguro

Um “binNMU” é um envio apenas-binário não-maintainer executado para transições de biblioteca etc. Num envio binNMU, apenas os pacotes “**Architecture: any**” são recompilados com um número de versão em sufixo (ex. a versão 2.3.4-3 irá tornar-se 2.3.4-3+b1). Os pacotes “**Architecture: all**” não são compilados.

A dependência definida no ficheiro **debian/control** entre pacotes binário do mesmo pacote fonte deve ser segura para o binNMU. Isto precisa de atenção se existirem ambos pacotes "**Architecture: any**" e "**Architecture: all**" envolvidos nisto.

- pacote "**Architecture: any**": depende de pacote "**Architecture: any**" *foo*
  - **Depends:** *foo* **(= ${binary:Version})**
- pacote "**Architecture: any**": depende de pacote "**Architecture: all**" *bar*
  - **Depends:** *bar* **(= ${source:Version})**
- pacote "**Architecture: all**": depende de pacote "**Architecture: any**" *baz*
  - **Depends:** *baz* **(>= ${source:Version})**, *baz* **(<< ${source:Version}.0~)**

10.20 Informação de depuração

O pacote Debian é compilado com a informação de depuração mas empacotado no pacote binário após despir a informação de depuração como requerido por "Capítulo 10 - Ficheiros" do "Manual de Política Debian".

Veja

- "6.7.9. Melhores práticas para pacotes de depuração" do "Debian Developer's Reference".
- "18.2 Informação de Depuração em Ficheiros Separados" do " Depurar com o gdb"
- "manual do **dh_strip**(1)"
- "manual do **strip**(1)"
- "manual do **readelf**(1)"
- "manual do **objcopy**(1)"
- Debian wiki: "DebugPackage"
- Debian wiki: "AutomaticDebugPackages"
- postado no debian-devel de Debian: "Estado em pacotes de depuração automáticos" (2015-08-15)

10.21 Pacote -dbgsym

A informação de depuração é automaticamente empacotada separadamente como o pacote de depuração usando o comando **dh_strip** com o seu comportamento predefinido. O nome de tal pacote de depuração normalmente tem o sufixo **-dbgsym**.

- O ficheiro **debian/rules** não deve conter explicitamente **dh_strip**.
- Define o **Build-Depends** para **debhelper-compat (>=14)** enquanto remove **Build-Depends** para **debhelper** em **debian/control**.

10.22 debconf

O pacote **debconf** permite-nos configurar pacotes durante a sua instalação de 2 maneiras principais:

- não-interativamente a partir de pré-preenchimento do **debian-installer**.
- interativamente a partir da interface de menu (**dialog**, **gnome**, **kde**, …)
  - a instalação do pacote: invocado pelo comando **dpkg**
  - o pacote instalado: invocado pelo comando **dpkg-reconfigure**

Todas as interações do utilizador para a instalação do pacote tem de ser lidada por este sistema **debconf** usando os seguintes ficheiros.

- **debian/***pacote-binário***.config**
  - Este é o script **config** do **debconf** usado para perguntar quaisquer questões necessárias para configurar o pacote.
- **debian/***pacote-binário***.template**
  - Este é o ficheiro de **modelos debconf** usado para perguntar quaisquer questões necessárias para configurar o pacote.

Estes ficheiros **debconf** são chamados por scripts de configuração do pacote no pacote binário Debian.

- **DEBIAN/***pacote-binário***.preinst**
- **DEBIAN/***pacote-binário***.prerm**
- **DEBIAN/***pacote-binário***.postinst**
- **DEBIAN/***pacote-binário***.postrm**

Veja **dh_installdebconf**(1), **debconf**(7), **debconf-devel**(7) e “3.9.1 Perguntando nos scripts do maintainer” no “Manual de Políticas Debian”.

Capítulo 11

Empacotar com git

Até ao “Capítulo 10”, nós concentrava-nos em operações de empacotamento sem usar Git ou qualquer outro VCS. Estas operações de empacotamento tradicionais eram baseadas no tarball lançado pelo autor como mencionado em “Secção 10.1”.

Atualmente, o comando **git**(1) é a plataforma de-facto par a ferramenta VCS e é a parte essencial para ambos desenvolvimento do autor e atividades de empacotamento Debian. (Veja Debian wiki “Empacotamento git Debian formatos de ramo de maintainer e fluxos de trabalho” para fluxos de trabalho VCS existentes.)

Nota



Como o pacote fonte Debian não-nativo usa “**diff -u**” como sua tecnologia backend para a modificação do maintainer, não pode representar modificações que envolvam links simbólicos, permissões de ficheiros, nem dados binários (Março 2022 discussão em debian-devel@l.d.o). Por favor evite fazer tais modificações de maintainer mesmo sabendo que estas podem ser gravadas no repositório Git.

Como os fluxos de trabalho VCS são um tópico complicado e existem muitos estilos práticos, aqui vou apenas tocar nalguns pontos chave com informação mínima.

Salsa é o serviço de repositório Git remoto com as ferramentas associadas. Oferece a plataforma de colaboração para atividades de empacotamento Debian usando uma instância de aplicação GitLab. Veja:

- “Secção 11.1”
- “Secção 11.2”
- “Secção 11.3”

Existem 2 estilos nomes de ramo para o repositório Git usado para o empacotamento. Veja “Secção 11.4”.

Existem 2 estilos de utilização principais para o repositório Git para o empacotamento. Veja:

- “Secção 11.5”
- “Secção 11.13”

Existem 2 ferramentas notáveis de empacotamento Debian para o repositório Git para o empacotamento.

- **gbp**(1) e os seus sub-comandos:
  - Esta é uma ferramenta desenhada para trabalhar principalmente com “Secção 11.5”.
  - Veja “Secção 11.9”.
- **dgit**(1) e os seus sub-comandos:

  - Esta é uma ferramenta desenhada para trabalhar principalmente com “Secção 11.13”.
  - Isto contém uma ferramenta para enviar pacotes Debian usando o servidor **dgit**.
  - Veja “Secção 11.14”.

11.1 Repositório Salsa

É altamente desejável hospedar o pacote de código fonte Debian em Salsa. Mais de 90% de todos os pacotes de código fonte Debian estão hospedados Salsa. 1

O repositório VCS exacto que hospeda um pacote de código fonte Debian pode ser identificado por um campo de metadados `Vcs-*` que pode ser visualizado com o comando `apt-cache showsrc <nome-pacote>`.

11.2 Configuração da conta Salsa

Após assinar uma conta no Salsa, certifique-se que as seguintes páginas têm o mesmo endereço de e-mail e as chaves openPGP que você configurou para usar com Debian, assim como a sua chave SSH:

- https://salsa.debian.org/-/profile/emails
- https://salsa.debian.org/-/user_settings/gpg_keys
- https://salsa.debian.org/-/user_settings/ssh_keys

11.3 Serviço CI de Salsa

Salsa corre o serviço Salsa CI como uma instância de GitLab CI para “Secção 10.4”.

Para cada instância de “**git push**”, testa quais testes de mímica podem correr no pacote Debian oficial ao definir o ficheiro de configuração do Salsa CI “Secção 6.13” como:

```
---
include:
  - https://salsa.debian.org/salsa-ci-team/pipeline/raw/master/recipes/debian.yml

# Customizations here
```

11.4 Nomes de ramos:

O repositório Git para o empacotamento Debian deve ter pelo menos 2 ramos:

- **debian-branch** para manter a cabeça de empacotamento Debian actual.
  - estilo antigo: **master** (ou **debian**, **main**, …)
  - DEP-14 estilo: **debian/latest**
- **upstream-branch** para manter os lançamentos do autor no formato importado.
  - estilo antigo: **upstream**
  - DEP-14 estilo: **upstream/latest**

1O uso de git.debian.org ou alioth.debian.org está agora descontinuado.

Neste tutorial, são usados nomes de ramo no estilo antigo nos exemplos para simplicidade.

Nota



Este **upstream-branch** pode precisar de ser criado usando tarball lançado pelo autor independente repositório Git do autor pois tende a conter ficheiros gerados automaticamente.

O conteúdo do repositório Git do autor pode co-existir no repositório Git local para o empacotamento Debian ao adicionar uma cópia dele. Ex.:

```
[debhello] $ git remote add upstreamvcs <url-upstream-git-repo>
[debhello] $ git fetch upstreamvcs master:upstream-master
```

Isto permite selecção fácil a partir do repositório Git do autor para correção de bugs.

11.5 Repositório Git de patch não-aplicada

O repositório Git de patch não-aplicada pode ser resumido a:

- Esta parece ser a prática tradicional até 2024.
- A árvore fonte corresponde ao conteúdo extraído por "**dpkg-source -x --skip-patches**" do pacote fonte Debian.
  - A fonte do autor é gravada no repositório Git sem alterações.
  - Os conteúdos modificados pelo maintainer estão confinados dentro do directório **debian/***.
  - Modificações do maintainer à fonte do autor são gravadas em ficheiros **debian/patches/*** para o formato de fonte Debian "**3.0 (quilt)**".
- Este estilo de repositório é útil para todas as variantes dos fluxos de trabalho tradicionais e fluxo de trabalho baseado no **gbp**:
  - "Secção 5.7" (nenhum patch)
  - "Secção 5.10"
    - Os ficheiros **debian/patches/*** também podem ser gerados usando "**git format-patch**", "**git diff**", ou "**gitk**" a partir de commits **git** no meio do ramo de modificação do maintainer ou a partir de commits do autor ainda não lançados.
  - "Secção 5.11" incluindo o último passo "**dquilt pop -a**"
  - "Secção 11.6"
- Use scripts de ajuda como o **dquilt**(1) e **gbp-pq**(1) para gerir dados nos ficheiros **debian/patches/***.
  - Adicione a linha **.pc** ao ficheiro **~/.gitignore** se for usado o **dquilt**.
- Use "**dpkg-source -b**" para compilar o pacote fonte Debian.
- Use **dput**(1) para enviar o pacote fonte Debian.

11.6 Patch por abordagem "gbp-pq"

Para "Secção 11.5", você pode gerar ficheiros **debian/patches/*** usando o comando **gbp-pq**(1) a partir de commits **git** feitos para o ramo **patch-queue**.

Ao contrário do **dquilt** que oferece funcionalidade semelhante como visto "Secção 5.11" e "Secção 9.5", o **gbp-pq** não usa ficheiros **.pc/*** para seguir o estado da patch, mas em vez disso o **gbp-pq** utiliza ramos temporários no git.

11.7 Gerir a lista de patch com gbp-pq

Você pode adicionar, retirar, e refrescar ficheiros **debian/patches/*** com o **gbp-pq** para gerir a lista das patch.

Se o pacote for gerido em "Secção 11.5" usando o pacote **git-buildpackage**, você pode revisar a fonte do autor para corrigir bug como maintainer e lançar uma nova revisão Debian usando **gbp pq**.

- **Adiciona** uma nova patch que grava a modificação à fonte do autor no ficheiro *buggy_file* como:

```
[debhello] $ git checkout master
[debhello] $ gbp pq import
gbp:info: ... imported on 'patch-queue/master
[debhello] $ vim buggy_file
  ...
[debhello] $ git add buggy_file
[debhello] $ git commit
[debhello] $ gbp pq export
gbp:info: On 'patch-queue/master', switching to 'master'
gbp:info: Generating patches from git (master..patch-queue/master)
[debhello] $ git add debian/patches/*
[debhello] $ dch -i
[debhello] $ git commit -a -m "Closes: #<bug_number>"
[debhello] $ git tag debian/<version>-<rev>
```

- **Drop** (== desactiva) um caminho existente
  - Comenta linha pertinente em **debian/patches/series**
  - Apagar o próprio caminho (opcional)
- **Refresca** ficheiros **debian/patches/*** para fazer o "**dpkg-source -b**" funcionar como esperado após atualizar um pacote Debian para o novo lançamento do autor.

```
[debhello] $ git checkout master
[debhello] $ gbp pq --force import # ensure patch-queue/master branch
gbp:info: ... imported on 'patch-queue/master
[debhello] $ git checkout master
[debhello] $ gbp import-orig --pristine-tar --uscan
  ...
gbp:info: Successfully imported version ?.?.? of ../packagename_?.?.?.orig. ↩
    tar.xz
[debhello] $ gbp pq rebase
 ... resolve conflicts and commit to patch-queue/master branch
[debhello] $ gbp pq export
gbp:info: On 'patch-queue/master', switching to 'master'
gbp:info: Generating patches from git (master..patch-queue/master)
[debhello] $ git add debian/patches
[debhello] $ git commit -m "Update patches"
[debhello] $ dch -v <newversion>-1
[debhello] $ git commit -a -m "release <newversion>-1"
[debhello] $ git tag debian/<newversion>-1
```

11.8 gbp import-dscs --debsnap

Para pacotes fonte Debian chamados "*<pacote-fonte>*" guardados no arquivo snapshot.debian.org, pode ser gerado um repositório git inicial gerido em "Secção 11.5" com todo o histórico de versão Debian como se segue.

```
[debhello] $ gbp import-dscs --debsnap --pristine-tar <source-package>
```

11.9 Note sobre gbp

O comando **gbp** é fornecido pelo pacote **git-buildpackage**.

- Este comando é desenhado para gerir os conteúdos de "Secção 11.5" de modo eficiente.
- Use "**gbp import-orig**" para importar o novo tar de autor para o repositório git.
  - A opção "**--pristine-tar**" para o comando "**git import-orig**" permite armazenar o tarball do autor no mesmo repositório git.
  - A opção "**--uscan**" como último argumento do comando "**gbp import-orig**" permite descarregar e submeter o novo tarball do autor para o repositório git.
- Use "**gbp import-dsc**" para importar o pacote fonte Debian anterior para o repositório git.
- Use "**gbp dch**" para gerar o changelog Debian a partir das mensagens cometidas ao git.
- Use "**gbp buildpackage**" para compilar o pacote binário Debian a partir do repositório git.
  - O pacote **sbuild** pode ser usado como seu backend de compilação chroot limpo seja por configuração ou adicionando "**--git-builder='sbuild -A -s --source-only-changes -v -d unstable'**"
- Use "**gbp pull**" para actualizar os ramos **debian**, **upstream** e **pristine-tar** em segurança a partir do repositório remoto.
- Use "**gbp pq**" para gerir patches de quilt sem usar o comando **dquilt**.
- Use "**gbp clone** *URL_REPOSITÓRIO*" para clonar e configurar ramos de acompanhamento para **debian**, **upstream** e **pristine-tar**.

A gestão de histórico do pacote com o pacote **git-buildpackage** está a tornar-se na prática standard para muitos maintainers Debian. Veja mais em:

- "compilar Pacotes Debian com git-buildpackage"
- "4 dicas para manter um pacote fonte Debian "3.0 (quilt)" num VCS"
- A documentação da prática de empacotamento do **systemd** em "Compilar a partir da fonte".
- O fluxo de trabalho mencionado em **dgit-maint-gbp**(7) o qual permite usar este **gbp** com **dgit**

11.10 O navegador do repositório Git

O comando **gitk** no pacote **gitk** mostra as alterações num repositório ou num conjunto selecionado de submetidos. Isto inclui visualizar o gráfico de submetidos, mostrar informação relacionada com cada submissão, e os ficheiros nas árvores de cada revisão.

Este comando **gitk** também fornece uma UI mito intuitiva para muitas operações complexas do comando "**git**" tais como "**git checkout** …", "**git reset*** …", "**git diff** …", etc..

11.11 Organização de histórico de submetidos Git

Quando o seu histórico de submetidos Git local fica entrelaçado, você precisa de o organizar antes de o lançar para o público.

O processo de organização mais simples é esmagar todas as alterações numa única submissão usando o "**git rebase -i**" de maneira interativa.

Mas isto pode criar um submetido enorme com ficheiros tais como ficheiros auto-gerados não destinados a serem submetidos. Você pode **despejar** tais ficheiros na submissão usando "**git rm** *qualquer-ficheiro*" e "**git commit --amend**". Isto pode ser bastante complexo.

Este processo de **despejo** complexo pode ser facilitado ao usar o comando "**git-ime**" no pacote **imediff**. Ele divide automaticamente um único submetido com muitos ficheiros em vários submetidos

envolvendo apenas mudanças em ficheiro único. Agora pode despejar tais ficheiros usando o "**git rebase -i**" em modo interativo.

Dica



O "**git-ime**" a operar num submetido de alterações em ficheiro único, divide-o em vários submetidos de alterações de linha usando o **imediff** de maneira interativa. Invocar isto com a opção **--auto** vai automatizar a operação de divisão do submetido. Veja **git-ime**(1) e **imediff**(1).

11.12 Empacotamento Debian Quasi-native

O esquema de empacotamento **quase-nativo** empacota uma fonte sem o real tarball de autor usando o formato de pacote **não-nativo**.

Dica



Algumas pessoas promovem este esquema de empacotamento **quase-nativo** mesmo para programas escritos apenas para Debian pois ajuda a facilitar a comunicação com as distribuições baseadas como a Ubuntu para correções de bugs e etc.

Este esquema de empacotamento **quase-nativo** envolve 2 passos de preparação:

- Organiza a sua árvore fonte quase como o pacote Debian **nativo** (veja "Secção 6.4") com ficheiros **debian/*** com algumas exceções:
  - Faz **debian/source/format** conter "**3.0 (quilt)**" em vez de "**3.0 (native)**" .
  - Faz **debian/changelog** conter *versão-revisão* em vez de *versão* .
- Gera o tarball de autor em falta de preferência sem ficheiros **debian/***.
  - Para o formato fonte Debian "**3.0 (quilt)**", a remoção de ficheiros sob o directório **debian/** é uma acção opcional.

O resto é o mesmo como o fluxo de trabalho de empacotamento **não-nativo** como escrito em "Secção 6.1".

Apesar de isto poder ser feito de muitas maneiras, você pode usar o repositório Git e "**git deborig**" como:

```
[~] $ cd /path/to/debhello
[debhello] $ dch -r
  ... set its <version>-<revision>, e.g., 1.0-1
[debhello] $ git tag -s debian/1.0-1
[debhello] $ git rm -rf debian
[debhello] $ git tag -s upstream/1.0
[debhello] $ git commit -m upstream/1.0
[debhello] $ git reset --hard HEAD^
[debhello] $ git deborig
[debhello] $ sbuild
```

11.13 Repositório Git de patch aplicada

Nota



O foco deste tutorial de introdução "Guia para Maintainers Debian" não é o repositório Git de patch aplicada o qual é uma nova tendência iniciada pelo promotor do comando **dgit**. Assim aqui só é dada explicação mínima.

O repositório Git de patch aplicada pode ser resumido a:

- A árvore fonte corresponde aos conteúdos extraídos pelo "**dpkg-source -x**" do pacote fonte Debian.
  - A árvore fonte é compilável e o mesmo que os maintainers NMU veem.
  - A fonte é guardada no repositório Git com as alterações de maintainer incluindo o directório **debian/**.
  - Modificações do maintainer ao código do autor são também gravados em ficheiros **debian/patches/*** para o formato fonte Debian "**3.0 (quilt)**".

11.14 Nota sobre dgit

O comando **dgit** é fornecido pelo pacote **dgit**.

- Este comando permite aceder ao repositório de pacotes Debian como se fosse um repositório **git** remoto.
- Este comando oferece ferramentas para gerir atividades de empacotamento Debian usando principalmente "Secção 11.13".
  - Não há mais operações complicadas para gerir ficheiros de patch no directório **debian/patches/**.
- Use "**dgit build-source**" ou "**dgit sbuild**" para compilar o pacote Debian apenas-fonte ou fonte+binário.
- Use "**dgit push-source**" ou "**dgit push-build**" para enviar o pacote Debian apenas-fonte ou fonte+binário via servidor **dgit**.
- Use **git-deborig**(1) para produzir um *pacote***.orig.tar.xz** Debian a partir da versão de autor em **debian/changelog**.

Dica



O servidor **dgit** é navegável no sítio https://browse.dgit.debian.org/.

Nota



De modo a manter a árvore de trabalho compatível com **dgit**, apague **debian/source/local-options** e **debian/source/local-patch-header** se estes existirem.

Dicas para estilos de fluxo de trabalho:

- Fluxo de trabalho do **dgit-maint-merge**(7).
  - Use isto para o pacote Debian não nativo sem patches de tópico granular gravadas no pacote fonte Debian.

    - * Suficiente bom para pacotes apenas com modificações triviais para o autor.
    - * Escolha apenas para pacotes com historial de modificações entrelaçadas para o autor.
  - Adiciona linhas **auto-commit** e **single-debian-patch** no ficheiro **debian/source/options**
    - * Nenhuma patches de tópico granular gravadas dentro do pacote fonte Debian.
  - Use "**git checkout upstream; git pull**" para puxar o novo commit do autor e use "**git checkout master ; git merge <nova-versão-tag>**" para o fundir com o ramo **master**.
  - Veja "Secção 5.12" para exemplo.

- Fluxo de trabalho do **dgit-maint-debrebase**(7).
  - Use isto para o pacote Debian não nativo com patches de tópico granular gravadas no pacote fonte Debian
  - Use o comando **git-debrebase**(1) para manter a série de alterações Debian feitas à fonte do autor.

- Fluxo de trabalho do **dgit-maint-native**(7)
  - Use isto para o pacote Debian nativo no repositório Git Debian. (Nenhumas alterações do maintainer)

- Fluxo de trabalho do **dgit-maint-gbp**(7)
  - Use isto para o pacote Debian não nativo que usa o formato de fonte "**3.0 (quilt)**" com o seu repositório Git Debian o qual tem usado **gbp-buildpackage**(1) com "Secção 11.5".

Este autor gosta deste novo comando **dgit** e começou a usa-lo com os fluxos de trabalho **dgit-maint-merge**(7) e **dgit-maint-native**(7). Assim os tópicos em volta de **dgit** estão para lá deste documento tutorial para os cobrir em profundidade. Por favor comece a ler os manuais relevantes mais recentes e os recursos do autor:

- "dgit: usar o arquivo Debian como um git remoto (2015)"
- "tag2upload (2023)"

Capítulo 12

Dicas

Por favor leia também as páginas perspicazes ligadas em “Notas sobre Debian” por Russ Allbery (muito tempo desenvolvedor Debian) que tem as melhores práticas para tópicos avançados de empacotamento.

12.1 Compilar sob UTF-8

O locale predefinido do ambiente de compilação é **C**.

Alguns programas como a função **read** do Python3 mudam o seu comportamento dependendo do locale.

Adicionar o seguinte código ao ficheiro **debian/rules** assegura a compilação do programa sob o locale **C.UTF-8**.

```
LC_ALL := C.UTF-8
export LC_ALL
```

12.2 Conversão UTF-8

Se os documentos do autor estão codificados em esquemas de codificação antigos, converte-los para UTF-8 é uma boa ideia.

Use o comando **iconv** do pacote **libc-bin** para converter a codificação de ficheiros de texto simples.

```
[debhello] $ iconv -f latin1 -t utf8 foo_in.txt > foo_out.txt
```

Use **w3m**(1) para converter de ficheiros HTML para ficheiros de texto simples UTF-8. Quando você faz isto, certifique-se de executar sob locale UTF-8.

```
[debhello] $ LC_ALL=C.UTF-8 w3m -o display_charset=UTF-8 \
        -cols 70 -dump -no-graph -T text/html \
        < foo_in.html > foo_out.txt
```

Corra estes scripts no alvo **override_dh_*** do ficheiro **debian/rules**.

12.3 Dicas para Depuração

Quando você de defronta com problemas de compilação ou despejos de núcleo dos programas binário gerados, você tem que resolve-los você próprio. Isso é **depuração** (debug).

Este é um tópico muito profundo para se descrever aqui. Assim, vamos apenas listar alguns ponteiros e dicas para algumas ferramentas de depuração típicas.

- Wikipedia: “core dump”
  - “**man core**”
  - Atualize o ficheiro “**/etc/security/limits.conf**” para incluir o seguinte:

```
* soft core unlimited
```

  - "**ulimit -c unlimited**" in **~/.bashrc**
  - "**ulimit -a**" para verificar
  - Pressione **Ctrl-** ou "**kill -ABRT** 'PID'" para criar um ficheiro de despejo de núcleo
- **gdb** - O GNU Debugger
  - "**info gdb**"
  - "Debugging with GDB" em **/usr/share/doc/gdb-doc/html/gdb/index.html**
- **strace** - Rastreio a chamadas e sinais do sistema
  - Use o script **strace-graph** encontrado em **/usr/share/doc/strace/examples/** para criar uma bonita vista em árvore
  - "**man strace**"
- **ltrace** - Rastreio a chamadas de biblioteca
  - "**man ltrace**"
- "**sh -n** *script.sh*" - Verificação de sintaxe de um script Shell
- "**sh -x** *script.sh*" - Rastreio a um script Shell
- "**python3 -m py_compile** *script.py*" - Verificação de sintaxe de um script Python
- "**python3 -mtrace --trace** *script.py*" - Rastreio a um script Python
- "**perl -I ../libpath -c** *script.pl*" - Verificação de sintaxe de um script Perl
- "**perl -d:Trace** *script.pl*" - Rastreio a um script Perl
  - Instale o pacote **libterm-readline-gnu-perl** ou o seu equivalente para adicionar capacidade de edição de linhas com suporte de histórico.
- **lsof** - Lista ficheiros abertos pelos processos
  - "**man lsof**"

Dica



O comando **script** grava resultados de consola.

Dica



Os comandos **screen** e **tmux** usados com o comando **ssh** oferecem terminais de ligação remota seguros e robustos.

Dica



Um ambiente Python- e Shell-like REPL (=READ + EVAL + PRINT + LOOP) para Perl é oferecido pelo comando **reply** do pacote (novo) **libreply-perl** e o comando **re.pl** do pacote (velho) **libdevel-repl-perl**.

Dica



Os comandos **rlwrap** e **rlfe** adicionam capacidades de edição de linhas com suporte de histórico a quaisquer comandos interativos. Ex. “**rlwrap dash -i**” .

Capítulo 13

Utilizações de Ferramenta

Aqui estão algumas dicas notáveis em redor do empacotamento Debian.

> Nota
>
> 
>
> As descrições nesta secção são intencionalmente breves. Os futuros maintainers são fortemente encorajados a procurar e a ler toda a documentação relevante associada a estes comandos.

> Nota
>
> 
>
> Os exemplos aqui usam a compressão **gz**. Pode ser usada a compressão **xz** em vez desta.

13.1 debdiff

Você pode comparar o conteúdo de ficheiros em dois pacotes Debian fonte com o comando **debdiff**.

```
[base_dir] $ debdiff old-package.dsc new-package.dsc
```

Você também pode comparar listas de ficheiros em dois conjuntos de pacotes Debian binário com o comando **debdiff**.

```
[base_dir] $ debdiff old-package.changes new-package.changes
```

Estes são úteis para identificar o que foi mudado nos pacotes fonte e para verificar por alterações inadvertidas feitas quando se actualiza pacotes binário, tais como ficheiros não intencionalmente mal colocados ou removidos.

Debian agora força o envio apenas-fonte quando se desenvolve pacotes. Assim podem existir 2 ficheiros diferentes ***.changes**:

- *pacote_versão-revisão*_**source.changes** para o envio normal apenas-fonte
- *package_version-revision_arch***.changes** para o envio fonte+binário

13.2 dget

Você pode descarregar o conjunto de ficheiros para o pacote fonte Debian com o comando **dget**.

```
[base_dir] $ dget https://www.example.org/path/to/package_version-rev.dsc
```

13.3 mk-origtargz

Você pode tornar o tarball de autor **../foo-***nova-versão***.tar.[xg]z** acessível a partir da árvore fonte Debian como **../foo**_*nova-versão***.orig.tar.[xg]z**. Este comando é útil para renomear e ligar por link simbólico o tarball do autor à convenção de nomes esperada em Debian.

13.4 origtargz

Você pode ir buscar o tarball original pré-existente de um pacote Debian a partir de várias fonte, e desempacota-lo com o comando **origtargz**.

Isto é basicamente para revisões **-2**, **-3**, ….

Nota



Quando o tarball do autor está em falta. o **debmake** produz automaticamente o tarball requerido. Esta é uma funcionalidade conveniente e suficientemente boa para fazer um pacote Debian privado. Mas quando se faz um pacote Debian para o repositório oficial de Debian, temos de usar exatamente o mesmo tarball de autor que a revisão **-1**. Para tais casos, deve ser usado **origtargz**.

13.5 git deborig

Se o projeto do autor estiver hospedado num repositório Git sem um lançamento oficial do tarball, você pode gerar o seu tarball original a partir do repositório **git** para ser usado pelo pacote fonte Debian. Execute “git deborig” a partir da raiz da árvore fonte copiada (checked-out).

Isto é basicamente para revisões **-1**.

13.6 dpkg-source -b

O comando “**dpkg-source -b**” empacota a árvore fonte do autor no pacote fonte Debian.

Ele espera uma série de patches no directório **debian/patches/** e a sua sequência de aplicação em **debian/patches/series**.

É compatível com operações **dquilt** (veja “Secção 4.4”) e entende o estado de aplicação da patch a partir da existência de **.pc/applied-patches**.

O comando **dpkg-buildpackage** invoca “**dpkg-source -b**”.

13.7 dpkg-source -x

O comando “**dpkg-source -x**” extrai a árvore fonte e aplica as patches no directório **debian/patches/** usando a sequência especificada em **debian/patches/series** na árvore fonte do autor. Também adiciona **.pc/applied-patches**. (Veja “Secção 11.13”.)

O comando “**dpkg-source -x --skip-patches**” extrai apenas a árvore fonte. Não adiciona **.pc/applied-patches**. (Veja “Secção 11.5”.)

Ambas árvores fonte extraídas estão prontas para compilar pacotes binário Debian com **dpkg-buildpackage**, **dbuild**, **sbuild**, etc..

13.8 debc

Você deve instalar os pacotes gerados com o comando **debc** para os testar localmente.

```
[base_dir] $ debc package_version-rev_arch.changes
```

13.9 bts

Após enviar o pacote, você irá receber relatórios de bug. É um dever importante de um maintainer de pacote gerir estes bugs apropriadamente, como descrito em "5.8. Lidar com bugs" do "Debian Developer's Reference".

O comando **bts** é uma ferramenta útil para gerir bugs no "Debian Bug Tracking System".

```
[~] $ bts severity 123123 wishlist , tags -1 pending
```

13.10 dpkg-depcheck

Pode usar **dpkg-depcheck**(1) para obter uma boa primeira aproximação à linha **Build-Depends** necessária por um pacote Debian.

```
[foo-1.0] $ dpkg-depcheck -b debian/rules build
```

Capítulo 14

Mais Exemplos

Há um velho ditado Latino que diz: “**fabricando fit faber**” (“a prática leva à perfeição”).

É altamente recomendado praticar e experimentar com todos os aspetos de empacotamento Debian com pacotes simples. Este capítulo fornece-lhe muitos casos de autor para você praticar.

Isto deve também servir como exemplos de introdução a muitos tópicos de programação.

- Programar na shell de POSIX, Python3, e C.
- Método para criar um lançador de programa de ambiente GUI com gráficos de ícones.
- Conversão de uma comando de CLI para GUI.
- Conversão de um programa para usar **gettext** para internacionalização e localização: fontes shell POSIX e C.
- Visão geral de muitos sistema de compilação: Makefile, Python, Autotools, e CMake.

Por favor note que Debian leva algumas coisas a sério:

- Free software (a.k.a. Software Livre)
- Estabilidade e segurança do SO
- SO universal realizado via:
  - escolha livre para fontes de autor,
  - escolha livre para arquitecturas de CPU, e
  - escolha livre para linguagens de Interface de Utilizador.

O exemplo de empacotamento típico apresentado em “Capítulo 5” é o pré-requisito para este capítulo.

Alguns detalhes foram deixados vagos intencionalmente nas secções seguintes. Por favor tente ler a documentação pertinente e pratique para os descobrir você próprio.

Dica



A melhor fonte para exemplos de empacotamento é o próprio arquivo Debian actual. Por favor use o serviço ““Debian Code Search”” para encontrar exemplos pertinentes.

14.1 Modelos Cherry-pick

Aqui está um exemplo de criar um pacote Debian simples a partir de uma fonte de conteúdo zero num directório vazio.

Isto é uma boa maneira de obter todos os ficheiros modelo sem criar desordem na árvore fonte do autor em que está a trabalhar.

Vamos assumir que este directório vazio seja **debhello-0.1**.

```
[base_dir] $ mkdir debhello-0.1
[base_dir] $ tree
.
+-- debhello-0.1

2 directories, 0 files
```

Vamos gerar a quantidade máxima de ficheiros modelo.

Vamos também usar as opções "**-p** *debhello* **-t -x3 -u** *0.1* **-r** *1*" para criar o tarball de autor em falta com as opções opcionais **-x3**, e **-t**.

```
[base_dir] $ cd debhello-0.1
[debhello-0.1] $ debmake -p debhello -x3 -t -T -u 0.1 -r 1
I: debmake (version: 5.1.5)
 ...
```

Vamos inspecionar os ficheiros modelo gerados.

```
[debhello-0.1] $ cd ..
[base_dir] $ tree
.
+-- debhello-0.1
|   +-- debian
|       +-- README.Debian
|       +-- README.source
|       +-- bug-control.ex
|       +-- bug-presubj.ex
|       +-- bug-script.ex
|       +-- changelog
|       +-- clean
|       +-- conffiles.ex
|       +-- control
|       +-- copyright
|       +-- cron.d.ex
|       +-- cron.daily.ex
|       +-- cron.hourly.ex
|       +-- cron.monthly.ex
|       +-- cron.weekly.ex
|       +-- default.ex
|       +-- dirs
|       +-- doc-base.ex
|       +-- docs
|       +-- emacsen-install.ex
|       +-- emacsen-remove.ex
|       +-- emacsen-startup.ex
|       +-- examples
|       +-- gbp.conf
|       +-- info.ex
|       +-- install
|       +-- links
|       +-- lintian-overrides.ex
|       +-- maintscript.ex
|       +-- manpage.1.ex
|       +-- manpage.asciidoc.ex
|       +-- manpage.md.ex
|       +-- manpage.sgml.ex
|       +-- manpage.xml.ex
|       +-- manpages
|       +-- patches
|       |   +-- series
|       +-- postinst.ex
|       +-- postrm.ex
|       +-- preinst.ex
|       +-- prerm.ex
```

```
|       +-- rules
|       +-- salsa-ci.yml
|       +-- service.ex
|       +-- source
|       |   +-- format
|       |   +-- lintian-overrides.ex
|       |   +-- options.ex
|       |   +-- patch-header.ex
|       +-- tests
|       |   +-- control
|       +-- tmpfile.ex
|       +-- upstream
|       |   +-- metadata
|       +-- watch
+-- debhello-0.1.tar.xz
+-- debhello_0.1.orig.tar.xz -> debhello-0.1.tar.xz

7 directories, 53 files
```

Agora você pode copiar qualquer destes ficheiros modelo gerados no directório *debhello-0.1/***debian/** para o seu pacote como necessário enquanto os renomeia se necessário.

14.2 Nenhum Makefile (shell, CLI)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa CLI de shell POSIX sem o seu sistema de compilação.

Vamos assumir que o tarball do autor seja **debhello-0.2.tar.xz**.

Este tipo de fonte não tem meios automatizados e os ficheiros têm de ser instalados manualmente. Por exemplo:

```
[base_dir] $ tar --xz -xmf debhello-0.2.tar.xz
[base_dir] $ cd debhello-0.2
[debhello-0.2] $ sudo cp scripts/hello /bin/hello
 ...
```

Vamos obter esta fonte como ficheiro tar a partir de um sítio remoto e fazer dela o pacote Debian.

Download debhello-0.2.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-0.2.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-0.2.tar.xz
[base_dir] $ tree
.
+-- debhello-0.2
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-0.2.tar.xz

5 directories, 6 files
```

Aqui, o script de shell POSIX **hello** é um muito simples.

hello (v=0.2)

```
[base_dir] $ cat debhello-0.2/scripts/hello
#!/bin/sh -e
echo "Hello from the shell!"
echo ""
```

```
echo -n "Type Enter to exit this program: "
read X
```

Aqui, **hello.desktop** suporta a “Especificação de Entrada Desktop”.

hello.desktop (v=0.2)

```
[base_dir] $ cat debhello-0.2/data/hello.desktop
[Desktop Entry]
Name=Hello
Name[fr]=Bonjour
Comment=Greetings
Comment[fr]=Salutations
Type=Application
Keywords=hello
Exec=hello
Terminal=true
Icon=hello.png
Categories=Utility;
```

Aqui, **hello.png** é o ficheiro de ícone gráfico.

Vamos empacotar isto com o comando **debmake**. Aqui, a opção **-b’:sh’** é usada para especificar que o pacote binário gerado é um script de shell.

```
[base_dir] $ cd debhello-0.2
[debhello-0.2] $ debmake -b':sh' -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-0.2] $ cd ..
I: Non-native Debian package pkg="debhello", ver="0.2", rev="1" method="dir_d...
I: already in the package-version form: "debhello-0.2"
I: [base_dir] $ ln -sf 'debhello-0.2.tar.xz' 'debhello_0.2.orig.tar.xz'
I: [base_dir] $ cd debhello-0.2
I: parsing option -b ":sh"
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: build_type = Unknown
I: ext_type = 1                       1 files
I: ext_type = desktop                 1 files
I: ext_type = md                      1 files
I: creating debian/* files with "-x 1" option
I: [debhello-0.2] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
I: creating debian/source/format from extra0source_format
 ...
```

Vamos inspecionar os ficheiros modelo notáveis gerados.

A árvore fonte após a execução debmake básica. (v=0.2)

```
[debhello-0.2] $ cd ..
[base_dir] $ tree
.
+-- debhello-0.2
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- debian
|   |   +-- README.Debian
|   |   +-- README.source
|   |   +-- changelog
|   |   +-- clean
|   |   +-- control
```

```
|   |   +-- copyright
|   |   +-- dirs
|   |   +-- docs
|   |   +-- examples
|   |   +-- gbp.conf
|   |   +-- install
|   |   +-- links
|   |   +-- manpages
|   |   +-- patches
|   |   |   +-- series
|   |   +-- rules
|   |   +-- salsa-ci.yml
|   |   +-- source
|   |   |   +-- format
|   |   +-- tests
|   |   |   +-- control
|   |   +-- upstream
|   |   |   +-- metadata
|   |   +-- watch
|   +-- man
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-0.2.tar.xz
+-- debhello_0.2.orig.tar.xz -> debhello-0.2.tar.xz

10 directories, 27 files
```

debian/rules (ficheiro modelo, v=0.2):

```
[base_dir] $ cd debhello-0.2
[debhello-0.2] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@
```

```
# debmake generated override targets
```

Isto é essencialmente o ficheiro **debian/rules** standard com o comando **dh**. Como isto é o pacote script, este ficheiro modelo **debian/rules** não tem conteúdos relacionados com bandeira de compilação.

debian/control (ficheiro modelo, v=0.2):

```
[debhello-0.2] $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.3
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no
#Vcs-Git: https://salsa.debian.org/debian/debhello.git
#Vcs-Browser: https://salsa.debian.org/debian/<project_site>

Package: debhello
Section: unknown
Architecture: all
Multi-Arch: foreign
Depends:
 ${misc:Depends},
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.
 .
 ===== This comes from the unmodified template file =====
 .
 Please edit this template file (debian/control) and other package files
 (debian/*) to make them meet all the requirements of the Debian Policy
 before uploading this package to the Debian archive.
 .
 See
  * https://www.debian.org/doc/manuals/developers-reference/best-pkging-pract...
  * https://www.debian.org/doc/manuals/debmake-doc/ch05.en.html#control
 .
 The synopsis description at the top should be about 60 characters and
 written as a phrase.  No extra capital letters or a final period.  No
 articles b''—b'' "a", "an", or "the".
 .
 The package description for general-purpose applications should be
 written for a less technical user.  This means that we should avoid
 jargon.  GNOME or KDE is fine but GTK+ is probably not.
 .
 Use the canonical forms of words:
  * Use X Window System, X11, or X; not X Windows, X-Windows, or X Window.
  * Use GTK+, not GTK or gtk.
  * Use GNOME, not Gnome.
  * Use PostScript, not Postscript or postscript.
```

Como este é o pacote script shell, o comando **debmake** define "**Architecture: all**" e "**Multi-Arch: foreign**". Também, define parâmetros **substvar** requeridos como "**Depends: ${misc:Depends}**". Isto está explicado em "Capítulo 6".

Como esta fonte de autor não tem o **Makefile** de autor, essa funcionalidade tem de ser fornecida pelo maintainer. Esta fonte de autor contém apenas um ficheiro script e ficheiros de dados e nenhuns ficheiros de fonte C; o processo **build** pode ser saltado mas o processo **install** precisa de ser implementado. Para este caso, isto é conseguido de forma limpa ao adicionar os ficheiros **debian/install** e **debian/manpages** sem se complicar o ficheiro **debian/rules**.

Vamos criar este pacote Debian melhor sendo o maintainer.

debian/rules (versão do maintainer, v=0.2):

```
[base_dir] $ cd debhello-0.2
[debhello-0.2] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-0.2] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
        dh $@
```

debian/control (versão do maintainer, v=0.2):

```
[debhello-0.2] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-0.2] $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.3
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends:
 ${misc:Depends},
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

Atenção



Se você deixar “**Section: unknown**” no ficheiro modelo **debian/control** não modificado, o erro do **lintian** pode causar uma falha de compilação.

debian/install (versão do maintainer, v=0.2):

```
[debhello-0.2] $ vim debian/install
 ... hack, hack, hack, ...
[debhello-0.2] $ cat debian/install
data/hello.desktop usr/share/applications
data/hello.png usr/share/pixmaps
scripts/hello usr/bin
```

debian/manpages (versão do maintainer, v=0.2):

```
 $ vim debian/manpages
 ... hack, hack, hack, ...
[debhello-0.2] $ cat debian/manpages
man/hello.1
```

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

Ficheiros modelo sob debian/. (v=0.2):

```
[debhello-0.2] $ rm -f debian/clean debian/dirs debian/links
[debhello-0.2] $ rm -f debian/README.source debian/source/*.ex
```

```
[debhello-0.2] $ rm -rf debian/patches
[debhello-0.2] $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- docs
+-- examples
+-- gbp.conf
+-- install
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 15 files
```

Você pode criar um pacote Debian não-nativo usando o comando **debuild** (ou os seus equivalentes) nesta árvore fonte. O texto resultante do comando é muito detalhado e explica o que ele faz como se segue.

```
[base_dir] $ cd debhello-0.2
[debhello-0.2] $ debuild
 dpkg-buildpackage -us -uc -ui -i
dpkg-buildpackage: info: source package debhello
dpkg-buildpackage: info: source version 0.2-1
dpkg-buildpackage: info: source distribution unstable
dpkg-buildpackage: info: source changed by Osamu Aoki <osamu@debian.org>
 dpkg-source -i --before-build .
dpkg-buildpackage: info: host architecture amd64
 debian/rules clean
dh clean
   dh_clean
        rm -f debian/debhelper-build-stamp
 ...
 debian/rules binary
dh binary
   dh_update_autotools_config
   dh_autoreconf
   create-stamp debian/debhelper-build-stamp
   dh_prep
        rm -f -- debian/debhello.substvars
        rm -fr -- debian/.debhelper/generated/debhello/ debian/debhello/ debi...
   dh_auto_install --destdir=debian/debhello/
 ...
Finished running lintian.
```

Vamos inspecionar o resultado.

Os ficheiros gerados de debhello versão 0.2 pelo comando debuild:

```
[debhello-0.2] $ cd ..
[base_dir] $ tree -FL 1
./
+-- debhello-0.2/
+-- debhello-0.2.tar.xz
+-- debhello_0.2-1.debian.tar.xz
+-- debhello_0.2-1.dsc
+-- debhello_0.2-1_all.deb
```

```
+-- debhello_0.2-1_amd64.build
+-- debhello_0.2-1_amd64.buildinfo
+-- debhello_0.2-1_amd64.changes
+-- debhello_0.2.orig.tar.xz -> debhello-0.2.tar.xz

2 directories, 8 files
```

Você vê todos os ficheiros gerados.

- O ficheiro **debhello_0.2.orig.tar.xz** é um link simbólico para o tarball do autor.
- O ficheiro **debhello_0.2-1.debian.tar.xz** contém os conteúdos gerados pelo maintainer.
- O ficheiro **debhello_0.2-1.dsc** é um ficheiro de meta-dados para o pacote fonte Debian.
- O ficheiro **debhello_0.2-1_all.deb** é o pacote binário Debian.
- O ficheiro **debhello_0.2-1_amd64.build** é o ficheiro de relatório de compilação.
- O ficheiro **debhello_0.2-1_amd64.buildinfo** é o ficheiro de meta-dados gerado pelo **dpkg-genbuildinfo**(1).
- O ficheiro **debhello_0.2-1_amd64.changes** é o ficheiro de meta-dados para o pacote binário Debian.

O ficheiro **debhello_0.2-1.debian.tar.xz** contém as alterações Debian à fonte do autor como se segue:

O conteúdo de arquivo comprimido de debhello_0.2-1.debian.tar.xz:

```
[base_dir] $ tar --xz -tf debhello-0.2.tar.xz
debhello-0.2/
debhello-0.2/data/
debhello-0.2/data/hello.desktop
debhello-0.2/data/hello.png
debhello-0.2/man/
debhello-0.2/man/hello.1
debhello-0.2/scripts/
debhello-0.2/scripts/hello
debhello-0.2/README.md
[base_dir] $ tar --xz -tf debhello_0.2-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
debian/docs
debian/examples
debian/gbp.conf
debian/install
debian/manpages
debian/rules
debian/salsa-ci.yml
debian/source/
debian/source/format
debian/tests/
debian/tests/control
debian/upstream/
debian/upstream/metadata
debian/watch
```

O ficheiro **debhello_0.2-1_amd64.deb** contém os ficheiros a serem instalados como se segue.

O conteúdo de pacote binário de debhello_0.2-1_all.deb:

```
[base_dir] $ dpkg -c debhello_0.2-1_all.deb
drwxr-xr-x root/root ...  ./
drwxr-xr-x root/root ...  ./usr/
drwxr-xr-x root/root ...  ./usr/bin/
```

```
-rwxr-xr-x root/root ...  ./usr/bin/hello
drwxr-xr-x root/root ...  ./usr/share/
drwxr-xr-x root/root ...  ./usr/share/applications/
-rw-r--r-- root/root ...  ./usr/share/applications/hello.desktop
drwxr-xr-x root/root ...  ./usr/share/doc/
drwxr-xr-x root/root ...  ./usr/share/doc/debhello/
-rw-r--r-- root/root ...  ./usr/share/doc/debhello/README.Debian
-rw-r--r-- root/root ...  ./usr/share/doc/debhello/changelog.Debian.gz
-rw-r--r-- root/root ...  ./usr/share/doc/debhello/copyright
drwxr-xr-x root/root ...  ./usr/share/man/
drwxr-xr-x root/root ...  ./usr/share/man/man1/
-rw-r--r-- root/root ...  ./usr/share/man/man1/hello.1.gz
drwxr-xr-x root/root ...  ./usr/share/pixmaps/
-rw-r--r-- root/root ...  ./usr/share/pixmaps/hello.png
```

Aqui está a lista de dependências gerada de **debhello_0.2-1_all.deb**.

A lista de dependências gerada de debhello_0.2-1_all.deb:

```
[debhello-0.2] $ dpkg -f debhello_0.2-1_all.deb pre-depends \
            depends recommends conflicts breaks
```

(Nenhum pacote de dependência extra requerido pois este é um programa de shell POSIX.)

> Nota
>
> 
>
> Se você deseja substituir o ficheiro PNG fornecido pelo autor **data/hello.png** por um fornecido pelo maintainer **debian/hello.png**, editar **debian/install** não é suficiente. Quando você adicionar **debian/hello.png**, você precisa de adicionar uma linha “include-binaries” ao **debian/source/options** pois PNG é um ficheiro binário. Veja **dpkg-source**(1).

/tep200.slog/ vim:set filetype=asciidoc:

14.3 Makefile (shell, CLI)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa CLI de shell POSIX usando o **Makefile** como seu sistema de compilação.

Vamos assumir que o tarball de autor seja **debhello-1.0.tar.xz**.

Este tipo de fonte destina-se a ser instalado como ficheiro não-sistema como:

```
[base_dir] $ tar --xz -xmf debhello-1.0.tar.xz
[base_dir] $ cd debhello-1.0
[debhello-1.0] $ make install
```

O empacotamento Debian requer alterar este processo “**make install**” para instalar ficheiros na localização imagem do sistema alvo em vez de na localização normal sob **/usr/local**.

Vamos obter a fonte e criar o pacote Debian.

Download debhello-1.0.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.0.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.0.tar.xz
[base_dir] $ tree
.
+-- debhello-1.0
|   +-- Makefile
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
```

```
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-1.0.tar.xz

5 directories, 7 files
```

Aqui, o **Makefile** usa **$(DESTDIR)** e **$(prefix)** apropriadamente. Todos os outros ficheiros são o mesmo que em “Secção 14.2” e a maioria das atividades de empacotamento são as mesmas.

Makefile (v=1.0)

```
[base_dir] $ cat debhello-1.0/Makefile
prefix = /usr/local

all:
        : # do nothing

install:
        install -D scripts/hello \
                $(DESTDIR)$(prefix)/bin/hello
        install -m 644 -D data/hello.desktop \
                $(DESTDIR)$(prefix)/share/applications/hello.desktop
        install -m 644 -D data/hello.png \
                $(DESTDIR)$(prefix)/share/pixmaps/hello.png
        install -m 644 -D man/hello.1 \
                $(DESTDIR)$(prefix)/share/man/man1/hello.1

clean:
        : # do nothing

distclean: clean

uninstall:
        -rm -f $(DESTDIR)$(prefix)/bin/hello
        -rm -f $(DESTDIR)$(prefix)/share/applications/hello.desktop
        -rm -f $(DESTDIR)$(prefix)/share/pixmaps/hello.png
        -rm -f $(DESTDIR)$(prefix)/share/man/man1/hello.1

.PHONY: all install clean distclean uninstall
```

Vamos empacotar isto com o comando **debmake**. Aqui, a opção **-b’:sh’** é usada para especificar que o pacote binário gerado é um script de shell.

```
[base_dir] $ cd debhello-1.0
[debhello-1.0] $ debmake -b':sh' -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.0] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.0", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.0"
I: [base_dir] $ ln -sf 'debhello-1.0.tar.xz' 'debhello_1.0.orig.tar.xz'
I: [base_dir] $ cd debhello-1.0
I: parsing option -b ":sh"
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: build_type = make
I: ext_type = 1                       1 files
I: ext_type = desktop                 1 files
I: ext_type = md                      1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.0] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
```

```
I: creating debian/source/format from extra0source_format
 ...
```

Vamos inspecionar os ficheiros modelo notáveis gerados.
debian/rules (ficheiro modelo, v=1.0):

```
[base_dir] $ cd debhello-1.0
[debhello-1.0] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@

# debmake generated override targets
# Use "make prefix=/usr" (override prefix=/usr/local in Makefile)
#override_dh_auto_install:
#       dh_auto_install -- prefix=/usr

# Do not install python .pyc .pyo if they exist
#override_dh_install:
#       dh_install --list-missing -X.pyc -X.pyo
```

Vamos criar este pacote Debian melhor sendo o maintainer.
debian/rules (versão do maintainer, v=1.0):

```
[base_dir] $ cd debhello-1.0
[debhello-1.0] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-1.0] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
        dh $@

override_dh_auto_install:
```

```
        dh_auto_install -- prefix=/usr
```

Como esta fonte de autor tem o **Makefile** de autor apropriado, não é preciso criar os ficheiros **debian/install** e **debian/manpages**.

O ficheiro **debian/control** é exactamente o mesmo que aquele em “Secção 14.2”.

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

Ficheiros modelo sob debian/. (v=1.0):

```
[debhello-1.0] $ rm -f debian/clean debian/dirs debian/install debian/links
[debhello-1.0] $ rm -f debian/README.source debian/source/*.ex
[debhello-1.0] $ rm -rf debian/patches
[debhello-1.0] $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- docs
+-- examples
+-- gbp.conf
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 14 files
```

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 14.2”.

14.4 pyproject.toml (Python3, CLI)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa CLI de Python3 usando **pyproject.toml**.

Vamos obter a fonte e criar o pacote Debian.

Download debhello-1.1.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.1.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.1.tar.xz
[base_dir] $ tree
.
+-- debhello-1.1
|   +-- LICENSE
|   +-- MANIFEST.in
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- manpages
|   |   +-- hello.1
|   +-- pyproject.toml
|   +-- src
|       +-- debhello
|           +-- __init__.py
|           +-- main.py
+-- debhello-1.1.tar.xz
```

```
6 directories, 10 files
```

Aqui, o conteúdo desta árvore fonte **debhello** como se segue.

pyproject.toml (v=1.1) — configuração PEP 517

```
[base_dir] $ cat debhello-1.1/pyproject.toml
[build-system]
requires = ["setuptools >= 61.0"]  # REQUIRED if [build-system] table is used...
build-backend = "setuptools.build_meta"  # If not defined, then legacy behavi...

[project]
name = "debhello"
version = "1.1.0"
description = "Hello Python (CLI)"
readme = {file = "README.md", content-type = "text/markdown"}
requires-python = ">=3.12"
license = "MIT"
keywords = ["debhello"]
authors = [
  {name = "Osamu Aoki", email = "osamu@debian.org" },
]
maintainers = [
  {name = "Osamu Aoki", email = "osamu@debian.org" },
]
classifiers = [
  "Development Status :: 5 - Production/Stable",
  "Intended Audience :: Developers",
        "Topic :: System :: Archiving :: Packaging",
  "Programming Language :: Python :: 3",
  "Programming Language :: Python :: 3.12",
  "Programming Language :: Python :: 3 :: Only",
  # Others
  "Operating System :: POSIX :: Linux",
  "Natural Language :: English",
]
[project.urls]
"Homepage" = "https://salsa.debian.org/debian/debmake"
"Bug Reports" = "https://salsa.debian.org/debian/debmake/issues"
"Source" = "https://salsa.debian.org/debian/debmake"
[project.scripts]
hello = "debhello.main:main"
[tool.setuptools]
package-dir = {"" = "src"}
packages = ["debhello"]
include-package-data = true
```

MANIFEST.in (v=1.1) — para tar-ball.

```
[base_dir] $ cat debhello-1.1/MANIFEST.in
include data/*
include manpages/*
```

src/debhello/__init__.py (v=1.1)

```
[base_dir] $ cat debhello-1.1/src/debhello/__init__.py
"""
debhello program (CLI)
"""
```

src/debhello/main.py (v=1.1) — ponto de entrada de comando

```
[base_dir] $ cat debhello-1.1/src/debhello/main.py
"""
debhello program
"""
```

```
import sys

__version__ = '1.1.0'

def main():  # needed for console script
    print(' ========== Hello Python3 ==========')
    print('argv = {}'.format(sys.argv))
    print('version = {}'.format(debhello.__version__))
    return

if __name__ == "__main__":
    sys.exit(main())
```

Vamos empacotar isto usando o comando **debmake**. Aqui, a opção **-b':py3'** é usada para especificar o pacote binário gerado que contém o script Python3 e os ficheiros do módulo.

```
[base_dir] $ cd debhello-1.1
[debhello-1.1] $ debmake -b':py3' -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.1] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.1", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.1"
I: [base_dir] $ ln -sf 'debhello-1.1.tar.xz' 'debhello_1.1.orig.tar.xz'
I: [base_dir] $ cd debhello-1.1
I: parsing option -b ":py3"
I: binary package=debhello Type=python3 / Arch=all M-A=foreign
W: setuptools build system.
I: build_type = Python (pyproject.toml: PEP-518, PEP-621, PEP-660)
I: ext_type = python3                2 files
I: ext_type = 1                      1 files
I: ext_type = desktop                1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.1] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
 ...
```

Vamos inspecionar os ficheiros modelo notáveis gerados.

debian/rules (ficheiro modelo, v=1.1):

```
[base_dir] $ cd debhello-1.1
[debhello-1.1] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
```

```
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@ --with python3 --buildsystem=pybuild

# debmake generated override targets
# Too complicated to provide examples here.
#
# Check situation of Python on Debian
#     https://wiki.debian.org/Python
#
#     https://wiki.debian.org/Python/TransitionToDHPython2
#     https://wiki.debian.org/Python/Pybuild
#     https://wiki.debian.org/Python/LibraryStyleGuide
#
# If a module package doesn't use distutils or setuptools but uses flit
# you need flit plugin.  See pybuild(1).
#
# Pure PEP-517 based build with "python3 -m build ..." is supported.
#
# To update the upstream source to support python3, see
#     https://wiki.python.org/moin/Python2orPython3
#     https://wiki.python.org/moin/PortingToPy3k/BilingualQuickRef
```

Isto é essencialmente o ficheiro **debian/rules** standard com o comando **dh**.

O uso da opção "**--with python3**" invoca **dh_python3** para calcular as dependências Python, adicionar scripts de maintainer a ficheiros compilados a byte, etc. Veja **dh_python3**(1).

O uso da opção "**--buildsystem=pybuild**" invoca vários sistemas de compilação para as versões Python requeridas de modo a compilar módulos e extensões. Veja **pybuild**(1).

debian/control (ficheiro modelo, v=1.1):

```
[debhello-1.1] $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 dh-python,
 pybuild-plugin-pyproject,
 python3-all,
 python3-setuptools,
Standards-Version: 4.7.3
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no
#Vcs-Git: https://salsa.debian.org/debian/debhello.git
#Vcs-Browser: https://salsa.debian.org/debian/<project_site>

Package: debhello
Section: unknown
Architecture: all
Multi-Arch: foreign
Depends:
```

```
 ${misc:Depends},
 ${python3:Depends},
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.
 .
 ===== This comes from the unmodified template file =====
 .
 Please edit this template file (debian/control) and other package files
 (debian/*) to make them meet all the requirements of the Debian Policy
 before uploading this package to the Debian archive.
 .
 See
  * https://www.debian.org/doc/manuals/developers-reference/best-pkging-pract...
  * https://www.debian.org/doc/manuals/debmake-doc/ch05.en.html#control
 .
 The synopsis description at the top should be about 60 characters and
 written as a phrase.  No extra capital letters or a final period.  No
 articles b''—b'' "a", "an", or "the".
 .
 The package description for general-purpose applications should be
 written for a less technical user.  This means that we should avoid
 jargon.  GNOME or KDE is fine but GTK+ is probably not.
 .
 Use the canonical forms of words:
  * Use X Window System, X11, or X; not X Windows, X-Windows, or X Window.
  * Use GTK+, not GTK or gtk.
  * Use GNOME, not Gnome.
  * Use PostScript, not Postscript or postscript.
```

Como este é o pacote Python3, o comando **debmake** define "**Architecture: all**" e "**Multi-Arch: foreign**". Também, define parâmetros **substvar** requeridos como "**Depends: ${python3:Depends}, ${misc:Depends}**". Isto está explicado em "Capítulo 6".

Vamos criar este pacote Debian melhor sendo o maintainer.

debian/rules (versão do maintainer, v=1.1):

```
[base_dir] $ cd debhello-1.1
[debhello-1.1] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-1.1] $ cat debian/rules
#!/usr/bin/make -f
export PYBUILD_NAME=debhello
export PYBUILD_VERBOSE=1
export DH_VERBOSE=1

%:
        dh $@ --with python3 --buildsystem=pybuild
```

debian/control (versão do maintainer, v=1.1):

```
[debhello-1.1] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-1.1] $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 pybuild-plugin-pyproject,
 python3-all,
Standards-Version: 4.7.3
Rules-Requires-Root: no
Vcs-Browser: https://salsa.debian.org/debian/debmake-doc
Vcs-Git: https://salsa.debian.org/debian/debmake-doc.git
```

```
Homepage: https://salsa.debian.org/debian/debmake-doc

Package: debhello
Architecture: all
Depends:
 ${misc:Depends},
 ${python3:Depends},
Description: Simple packaging example for debmake
 This is an example package to demonstrate Debian packaging using
 the debmake command.
 .
 The generated Debian package uses the dh command offered by the
 debhelper package and the dpkg source format `3.0 (quilt)'.
```

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

Este comando **debhello** vem com o manual fornecido-pelo-autor e o ficheiro desktop mas o **pyproject.toml** do autor não os instala. Assim você precisa de actualizar **debian/install** e **debian/manpages** como se segue:

debian/install (versão do maintainer, v=1.1):

```
[debhello-1.1] $ vim debian/copyright
 ... hack, hack, hack, ...
[debhello-1.1] $ cat debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: debhello
Upstream-Contact: Osamu Aoki <osamu@debian.org>
Source: https://salsa.debian.org/debian/debmake-doc

Files:     *
Copyright: 2015-2024 Osamu Aoki <osamu@debian.org>
License:   Expat
 Permission is hereby granted, free of charge, to any person obtaining a
 copy of this software and associated documentation files (the "Software"),
 to deal in the Software without restriction, including without limitation
 the rights to use, copy, modify, merge, publish, distribute, sublicense,
 and/or sell copies of the Software, and to permit persons to whom the
 Software is furnished to do so, subject to the following conditions:
 .
 The above copyright notice and this permission notice shall be included
 in all copies or substantial portions of the Software.
 .
 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

debian/manpages (versão do maintainer, v=1.1):

```
[debhello-1.1] $ vim debian/install
 ... hack, hack, hack, ...
[debhello-1.1] $ cat debian/install
data/hello.desktop usr/share/applications
data/hello.png usr/share/pixmaps
```

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 14.3”.

Ficheiros modelo sob debian/. (v=1.1):

```
[debhello-1.1] $ rm -f debian/clean debian/dirs debian/links
[debhello-1.1] $ rm -f debian/README.source debian/source/*.ex
[debhello-1.1] $ rm -rf debian/patches
[debhello-1.1] $ tree -F debian
```

```
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- docs
+-- examples
+-- gbp.conf
+-- install
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 15 files
```

Aqui está a lista de dependências gerada de **debhello_1.1-1_all.deb**.
A lista de dependências gerada de debhello_1.1-1_all.deb:

```
[debhello-1.1] $ dpkg -f debhello_1.1-1_all.deb pre-depends \
           depends recommends conflicts breaks
Depends: python3:any
```

14.5 Makefile (shell, GUI)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa GUI de shell POSIX usando o **Makefile** como seu sistema de compilação.

Esta fonte de autor é baseada no “Secção 14.3” com suporte GUI avançado.
Vamos assumir que o tarball de autor seja **debhello-1.2.tar.xz**.
Vamos obter a fonte e criar o pacote Debian.
Download debhello-1.2.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.2.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.2.tar.xz
[base_dir] $ tree
.
+-- debhello-1.2
|   +-- Makefile
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-1.2.tar.xz

5 directories, 7 files
```

Aqui, o **hello** foi reescrito para usar o comando **zenity** para tornar isto num programa GUI GTK+.
hello (v=1.2)

```
[base_dir] $ cat debhello-1.2/scripts/hello
#!/bin/sh -e
zenity --info --title "hello" --text "Hello from the shell!"
```

Aqui, o ficheiro desktop foi atualizado para ser **Terminal=false** como um programa GUI.
hello.desktop (v=1.2)

```
[base_dir] $ cat debhello-1.2/data/hello.desktop
[Desktop Entry]
Name=Hello
Name[fr]=Bonjour
Comment=Greetings
Comment[fr]=Salutations
Type=Application
Keywords=hello
Exec=hello
Terminal=false
Icon=hello.png
Categories=Utility;
```

Todos os outros ficheiros são o mesmo que em “Secção 14.3”.
Vamos empacotar isto com o comando **debmake**. Aqui, a opção “**-b':sh'**” é usada para especificar que o pacote binário gerado é um script de shell.

```
[base_dir] $ cd debhello-1.2
[debhello-1.2] $ debmake -b':sh' -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.2] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.2", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.2"
I: [base_dir] $ ln -sf 'debhello-1.2.tar.xz' 'debhello_1.2.orig.tar.xz'
I: [base_dir] $ cd debhello-1.2
I: parsing option -b ":sh"
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: build_type = make
I: ext_type = 1                       1 files
I: ext_type = desktop                 1 files
I: ext_type = md                      1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.2] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
I: creating debian/source/format from extra0source_format
 ...
```

Vamos inspecionar os ficheiros modelo notáveis gerados.
debian/control (ficheiro modelo, v=1.2):

```
[debhello-1.2] $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.3
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no
#Vcs-Git: https://salsa.debian.org/debian/debhello.git
#Vcs-Browser: https://salsa.debian.org/debian/<project_site>

Package: debhello
Section: unknown
Architecture: all
Multi-Arch: foreign
```

```
Depends:
 ${misc:Depends},
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.
 .
 ===== This comes from the unmodified template file =====
 .
 Please edit this template file (debian/control) and other package files
 (debian/*) to make them meet all the requirements of the Debian Policy
 before uploading this package to the Debian archive.
 .
 See
  * https://www.debian.org/doc/manuals/developers-reference/best-pkging-pract...
  * https://www.debian.org/doc/manuals/debmake-doc/ch05.en.html#control
 .
 The synopsis description at the top should be about 60 characters and
 written as a phrase.  No extra capital letters or a final period.  No
 articles b''—b'' "a", "an", or "the".
 .
 The package description for general-purpose applications should be
 written for a less technical user.  This means that we should avoid
 jargon.  GNOME or KDE is fine but GTK+ is probably not.
 .
 Use the canonical forms of words:
  * Use X Window System, X11, or X; not X Windows, X-Windows, or X Window.
  * Use GTK+, not GTK or gtk.
  * Use GNOME, not Gnome.
  * Use PostScript, not Postscript or postscript.
```

Vamos criar este pacote Debian melhor sendo o maintainer.

debian/control (versão do maintainer, v=1.2):

```
[debhello-1.2] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-1.2] $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.3
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends:
 zenity,
 ${misc:Depends},
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

Por favor note a dependência adicionada manualmente **zenity**.

O ficheiro **debian/rules** é exatamente o mesmo que aquele em “Secção 14.3”.

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

Ficheiros modelo sob debian/. (v=1.2):

```
[debhello-1.2] $ rm -f debian/clean debian/dirs debian/install debian/links
[debhello-1.2] $ rm -f debian/README.source debian/source/*.ex
[debhello-1.2] $ rm -rf debian/patches
```

```
[debhello-1.2] $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- docs
+-- examples
+-- gbp.conf
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 14 files
```

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 14.3”.
Aqui está a lista de dependências gerada de **debhello_1.2-1_all.deb**.
A lista de dependências gerada de debhello_1.2-1_all.deb:

```
[debhello-1.2] $ dpkg -f debhello_1.2-1_all.deb pre-depends \
            depends recommends conflicts breaks
Depends: zenity
```

14.6 pyproject.toml (Python3, GUI)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa GUI de Python3 usando **pyproject.toml**.
Vamos assumir que o tarball do autor seja **debhello-1.3.tar.xz**.
Vamos obter a fonte e criar o pacote Debian.
Download debhello-1.3.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.3.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.3.tar.xz
[base_dir] $ tree
.
+-- debhello-1.3
|   +-- LICENSE
|   +-- MANIFEST.in
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- manpages
|   |   +-- hello.1
|   +-- pyproject.toml
|   +-- src
|       +-- debhello
|           +-- __init__.py
|           +-- main.py
+-- debhello-1.3.tar.xz

6 directories, 10 files
```

Aqui, o conteúdo desta árvore fonte **debhello** como se segue.
pyproject.toml (v=1.3) — configuração PEP 517

```
[base_dir] $ cat debhello-1.3/pyproject.toml
[build-system]
requires = ["setuptools >= 61.0"]  # REQUIRED if [build-system] table is used...
build-backend = "setuptools.build_meta"  # If not defined, then legacy behavi...

[project]
name = "debhello"
version = "1.3.0"
description = "Hello Python (GUI)"
readme = {file = "README.md", content-type = "text/markdown"}
requires-python = ">=3.12"
license = "MIT"
keywords = ["debhello"]
authors = [
  {name = "Osamu Aoki", email = "osamu@debian.org" },
]
maintainers = [
  {name = "Osamu Aoki", email = "osamu@debian.org" },
]
classifiers = [
  "Development Status :: 5 - Production/Stable",
  "Intended Audience :: Developers",
        "Topic :: System :: Archiving :: Packaging",
  "Programming Language :: Python :: 3",
  "Programming Language :: Python :: 3.12",
  "Programming Language :: Python :: 3 :: Only",
  # Others
  "Operating System :: POSIX :: Linux",
  "Natural Language :: English",
]
[project.urls]
"Homepage" = "https://salsa.debian.org/debian/debmake"
"Bug Reports" = "https://salsa.debian.org/debian/debmake/issues"
"Source" = "https://salsa.debian.org/debian/debmake"
[project.scripts]
hello = "debhello.main:main"
[tool.setuptools]
package-dir = {"" = "src"}
packages = ["debhello"]
include-package-data = true
```

MANIFEST.in (v=1.3) — para tar-ball.

```
[base_dir] $ cat debhello-1.3/MANIFEST.in
include data/*
include manpages/*
```

src/debhello/__init__.py (v=1.3)

```
[base_dir] $ cat debhello-1.3/src/debhello/__init__.py
"""
debhello program (GUI)
"""
```

src/debhello/main.py (v=1.3) — ponto de entrada de comando

```
[base_dir] $ cat debhello-1.3/src/debhello/main.py
#!/usr/bin/python3
from gi.repository import Gtk

__version__ = '1.3.0'

class TopWindow(Gtk.Window):

    def __init__(self):
```

```
        Gtk.Window.__init__(self)
        self.title = "Hello World!"
        self.counter = 0
        self.border_width = 10
        self.set_default_size(400, 100)
        self.set_position(Gtk.WindowPosition.CENTER)
        self.button = Gtk.Button(label="Click me!")
        self.button.connect("clicked", self.on_button_clicked)
        self.add(self.button)
        self.connect("delete-event", self.on_window_destroy)

    def on_window_destroy(self, *args):
        Gtk.main_quit(*args)

    def on_button_clicked(self, widget):
        self.counter += 1
        widget.set_label("Hello, World!\nClick count = %i" % self.counter)

def main():
    window = TopWindow()
    window.show_all()
    Gtk.main()

if __name__ == '__main__':
    main()
```

Vamos empacotar isto com o comando **debmake**. Aqui, a opção **-b':py3'** é usada para especificar que o pacote binário gerado que contém o script Python3 e os ficheiros do módulo.

```
[base_dir] $ cd debhello-1.3
[debhello-1.3] $ debmake -b':py3' -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.3] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.3", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.3"
I: [base_dir] $ ln -sf 'debhello-1.3.tar.xz' 'debhello_1.3.orig.tar.xz'
I: [base_dir] $ cd debhello-1.3
I: parsing option -b ":py3"
I: binary package=debhello Type=python3 / Arch=all M-A=foreign
W: setuptools build system.
I: build_type = Python (pyproject.toml: PEP-518, PEP-621, PEP-660)
I: ext_type = python3                2 files
I: ext_type = 1                      1 files
I: ext_type = desktop                1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.3] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
 ...
```

O resultado é praticamente o mesmo que em "Secção 14.4".

Vamos criar este pacote Debian melhor sendo o maintainer.

debian/rules (versão do maintainer, v=1.3):

```
[base_dir] $ cd debhello-1.3
[debhello-1.3] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-1.3] $ cat debian/rules
#!/usr/bin/make -f
export PYBUILD_NAME=debhello
export PYBUILD_VERBOSE=1
```

```
export DH_VERBOSE=1

%:
        dh $@ --with python3 --buildsystem=pybuild
```

debian/control (versão do maintainer, v=1.3):

```
[debhello-1.3] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-1.3] $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 pybuild-plugin-pyproject,
 python3-all,
Standards-Version: 4.7.3
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends:
 gir1.2-gtk-3.0,
 python3-gi,
 ${misc:Depends},
 ${python3:Depends},
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

Por favor note as dependências adicionadas manualmente **python3-gi** e **gir1.2-gtk-3.0**.

O resto das atividades de empacotamento são praticamente o mesmo que em <pyproject>>.

Aqui está a lista de dependências gerada de **debhello_1.3-1_all.deb**.

A lista de dependências gerada de debhello_1.3-1_all.deb:

```
[debhello-1.3] $ dpkg -f debhello_1.3-1_all.deb pre-depends \
            depends recommends conflicts breaks
Depends: gir1.2-gtk-3.0, python3-gi, python3:any
```

14.7 Makefile (pacote singular-binário)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa fonte C simples usando o **Makefile** como seu sistema de compilação.

Este é um exemplo de fonte de autor avançada para “Capítulo 5”. Isto vem com o manual, o ficheiro desktop, e o ícone de desktop. Isto também de liga a uma biblioteca externa **libm** para ser um exemplo mais prático.

Vamos assumir que o tarball de autor seja **debhello-1.4.tar.xz**.

Este tipo de fonte destina-se a ser instalado como ficheiro não-sistema como:

```
[base_dir] $ tar --xz -xmf debhello-1.4.tar.xz
[base_dir] $ cd debhello-1.4
[debhello-1.4] $ make
[debhello-1.4] $ make install
```

O empacotamento Debian requer mudar este processo “**make install**” para instalar ficheiros na localização da imagem de sistema alvo em vez da localização normal sob **/usr/local**.

Vamos obter a fonte e criar o pacote Debian.

Download debhello-1.4.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.4.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.4.tar.xz
[base_dir] $ tree
.
+-- debhello-1.4
|   +-- LICENSE
|   +-- Makefile
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- hello.1
|   +-- src
|       +-- config.h
|       +-- hello.c
+-- debhello-1.4.tar.xz

5 directories, 9 files
```

Aqui, os conteúdos desta fonte são como se segue.

src/hello.c (v=1.4):

```
[base_dir] $ cat debhello-1.4/src/hello.c
#include "config.h"
#include <math.h>
#include <stdio.h>
int
main()
{
        printf("Hello, I am " PACKAGE_AUTHOR "!\n");
        printf("4.0 * atan(1.0) = %10f8\n", 4.0*atan(1.0));
        return 0;
}
```

src/config.h (v=1.4):

```
[base_dir] $ cat debhello-1.4/Makefile
prefix = /usr/local

all: src/hello

src/hello: src/hello.c
        $(CC) $(CPPFLAGS) $(CFLAGS) $(LDFLAGS) -o $@ $^ -lm

install: src/hello
        install -D src/hello \
                $(DESTDIR)$(prefix)/bin/hello
        install -m 644 -D data/hello.desktop \
                $(DESTDIR)$(prefix)/share/applications/hello.desktop
        install -m 644 -D data/hello.png \
                $(DESTDIR)$(prefix)/share/pixmaps/hello.png
        install -m 644 -D man/hello.1 \
                $(DESTDIR)$(prefix)/share/man/man1/hello.1

clean:
        -rm -f src/hello

distclean: clean

uninstall:
        -rm -f $(DESTDIR)$(prefix)/bin/hello
        -rm -f $(DESTDIR)$(prefix)/share/applications/hello.desktop
```

```
        -rm -f $(DESTDIR)$(prefix)/share/pixmaps/hello.png
        -rm -f $(DESTDIR)$(prefix)/share/man/man1/hello.1

.PHONY: all install clean distclean uninstall
```

Makefile (v=1.4):

```
[base_dir] $ cat debhello-1.4/src/config.h
#define PACKAGE_AUTHOR "Osamu Aoki"
```

Por favor note que este **Makefile** tem o alvo **install** apropriado para o manual, o ficheiro desktop, e o ícone desktop.

Vamos empacotar isto com o comando **debmake**.

```
[base_dir] $ cd debhello-1.4
[debhello-1.4] $ debmake -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.4] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.4", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.4"
I: [base_dir] $ ln -sf 'debhello-1.4.tar.xz' 'debhello_1.4.orig.tar.xz'
I: [base_dir] $ cd debhello-1.4
I: parsing option -b ""
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: build_type = make
I: ext_type = c                       2 files
I: ext_type = 1                       1 files
I: ext_type = desktop                 1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.4] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
I: creating debian/source/format from extra0source_format
 ...
```

O resultado é praticamente o mesmo que em “Secção 5.6”.

Vamos tornar este pacote Debian, que é praticamente o mesmo que em “Secção 5.7”, melhor senso o maintainer.

Se a variável de ambiente **DEB_BUILD_MAINT_OPTIONS** não estiver exportada em **debian/rules**, o lintian avisa “W: debhello: hardening-no-relro usr/bin/hello” para a ligação de **libm**.

O ficheiro **debian/control** faz exatamente o mesmo que aquele em “Secção 5.7”, pois a biblioteca **libm** está sempre disponível como parte de **libc6** (Priority: required).

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

Ficheiros modelo sob debian/. (v=1.4):

```
[debhello-1.4] $ rm -f debian/clean debian/dirs debian/links
[debhello-1.4] $ rm -f debian/README.source debian/source/*.ex
[debhello-1.4] $ rm -rf debian/patches
[debhello-1.4] $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- docs
+-- examples
+-- gbp.conf
+-- install
+-- manpages
+-- rules*
```

```
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 15 files
```

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 5.8”.

Aqui está a lista de dependências gerada de todos os pacotes binários.

A lista de dependências gerada de todos os pacotes binários (v=1.4):

```
[debhello-1.4] $ dpkg -f debhello-dbgsym_1.4-1_amd64.deb pre-depends \
           depends recommends conflicts breaks
Depends: debhello (= 1.4-1)
[debhello-1.4] $ dpkg -f debhello_1.4-1_amd64.deb pre-depends \
           depends recommends conflicts breaks
Depends: libc6 (>= 2.34)
```

14.8 Makefile.in + configure (pacote singular-binário)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa fonte C simples usando **Makefile.in** e **configure** como seu sistema de compilação.

Este é um exemplo de fonte de autor avançado para “Secção 14.7”. Isto também liga a uma biblioteca externa, **libm**, e esta fonte é configurável usando argumentos ao script **configure**. o qual gera os ficheiros **Makefile** e **src/config.h**.

Vamos assumir que o tarball do autor seja **debhello-1.5.tar.xz**.

Este tipo de fonte destina-se a ser instalado como um ficheiro não-sistema, por exemplo, como:

```
[base_dir] $ tar --xz -xmf debhello-1.5.tar.xz
[base_dir] $ cd debhello-1.5
[debhello-1.5] $ ./configure --with-math
[debhello-1.5] $ make
[debhello-1.5] $ make install
```

Vamos obter a fonte e criar o pacote Debian.

Download debhello-1.5.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.5.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.5.tar.xz
[base_dir] $ tree
.
+-- debhello-1.5
|   +-- LICENSE
|   +-- Makefile.in
|   +-- README.md
|   +-- configure
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- hello.1
|   +-- src
|       +-- hello.c
+-- debhello-1.5.tar.xz

5 directories, 9 files
```

Aqui, os conteúdos desta fonte são como se segue.

src/hello.c (v=1.5):

```
[base_dir] $ cat debhello-1.5/src/hello.c
#include "config.h"
#ifdef WITH_MATH
#  include <math.h>
#endif
#include <stdio.h>
int
main()
{
        printf("Hello, I am " PACKAGE_AUTHOR "!\n");
#ifdef WITH_MATH
        printf("4.0 * atan(1.0) = %10f8\n", 4.0*atan(1.0));
#else
        printf("I can't do MATH!\n");
#endif
        return 0;
}
```

Makefile.in (v=1.5):

```
[base_dir] $ cat debhello-1.5/Makefile.in
prefix = @prefix@

all: src/hello

src/hello: src/hello.c
        $(CC) @VERBOSE@ \
                $(CPPFLAGS) \
                $(CFLAGS) \
                $(LDFLAGS) \
                -o $@ $^ \
                @LINKLIB@

install: src/hello
        install -D src/hello \
                $(DESTDIR)$(prefix)/bin/hello
        install -m 644 -D data/hello.desktop \
                $(DESTDIR)$(prefix)/share/applications/hello.desktop
        install -m 644 -D data/hello.png \
                $(DESTDIR)$(prefix)/share/pixmaps/hello.png
        install -m 644 -D man/hello.1 \
                $(DESTDIR)$(prefix)/share/man/man1/hello.1

clean:
        -rm -f src/hello

distclean: clean

uninstall:
        -rm -f $(DESTDIR)$(prefix)/bin/hello
        -rm -f $(DESTDIR)$(prefix)/share/applications/hello.desktop
        -rm -f $(DESTDIR)$(prefix)/share/pixmaps/hello.png
        -rm -f $(DESTDIR)$(prefix)/share/man/man1/hello.1

.PHONY: all install clean distclean uninstall
```

configure (v=1.5):

```
[base_dir] $ cat debhello-1.5/configure
#!/bin/sh -e
# default values
PREFIX="/usr/local"
VERBOSE=""
```

```
WITH_MATH="0"
LINKLIB=""
PACKAGE_AUTHOR="John Doe"

# parse arguments
while [ "${1}" != "" ]; do
  VAR="${1%=*}" # Drop suffix =*
  VAL="${1#*=}" # Drop prefix *=
  case "${VAR}" in
  --prefix)
    PREFIX="${VAL}"
    ;;
  --verbose|-v)
    VERBOSE="-v"
    ;;
  --with-math)
    WITH_MATH="1"
    LINKLIB="-lm"
    ;;
  --author)
    PACKAGE_AUTHOR="${VAL}"
    ;;
  *)
    echo "W: Unknown argument: ${1}"
  esac
  shift
done

# setup configured Makefile and src/config.h
sed -e "s,@prefix@,${PREFIX}," \
    -e "s,@VERBOSE@,${VERBOSE}," \
    -e "s,@LINKLIB@,${LINKLIB}," \
    <Makefile.in >Makefile
if [ "${WITH_MATH}" = 1 ]; then
echo "#define WITH_MATH" >src/config.h
else
echo "/* not defined: WITH_MATH */" >src/config.h
fi
echo "#define PACKAGE_AUTHOR \"${PACKAGE_AUTHOR}\"" >>src/config.h
```

Por favor note que o comando **configure** substitui strings com @**...**@ em **Makefile.in** para produzir **Makefile** e cria **src/config.h**.

Vamos empacotar isto com o comando **debmake**.

```
[base_dir] $ cd debhello-1.5
[debhello-1.5] $ debmake -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.5] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.5", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.5"
I: [base_dir] $ ln -sf 'debhello-1.5.tar.xz' 'debhello_1.5.orig.tar.xz'
I: [base_dir] $ cd debhello-1.5
I: parsing option -b ""
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: build_type = configure
I: ext_type = c                       1 files
I: ext_type = 1                       1 files
I: ext_type = desktop                 1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.5] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
```

```
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
I: creating debian/source/format from extra0source_format
 ...
```

O resultado é semelhante a “Secção 5.6” mas não exactamente o mesmo.
Vamos inspecionar os ficheiros modelo notáveis gerados.
debian/rules (ficheiro modelo, v=1.5):

```
[base_dir] $ cd debhello-1.5
[debhello-1.5] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# See FEATURE AREAS/ENVIRONMENT in dpkg-buildflags(1)
# Apply all hardening options
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
# Package maintainers to append CFLAGS
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
# Package maintainers to append LDFLAGS
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1
#
# With debhelper version 9 or newer, the dh command exports
# all buildflags.  So there is no need to include the
# /usr/share/dpkg/buildflags.mk file here if compat is 9 or newer.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@

# debmake generated override targets
# Multiarch package requires library files to be installed to
# /usr/lib/<triplet>/ .  If the build system does not support
# $(DEB_HOST_MULTIARCH), you may need to override some targets such as
# dh_auto_configure or dh_auto_install to use $(DEB_HOST_MULTIARCH) .
```

Vamos criar este pacote Debian melhor sendo o maintainer.
debian/rules (versão do maintainer, v=1.5):

```
[base_dir] $ cd debhello-1.5
[debhello-1.5] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-1.5] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
        dh $@

override_dh_auto_configure:
        dh_auto_configure -- \
                --with-math \
                --author="Osamu Aoki"
```

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 5.8”.

14.9 Autotools (pacote singular-binário)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa fonte C simples usando Autotools = Autoconf e Automake (**Makefile.am** e **configure.ac**) como seu sistema de compilação.

Esta fonte geralmente vem também com os ficheiros de autor auto-gerados **Makefile.in** e **configure** . Esta fonte pode ser empacotada usando estes ficheiros como em “Secção 14.8” com a ajuda do pacote **autotools-dev**.

A melhor alternativa é regenerar esses ficheiros usando os pacotes Autoconf 3 Automake mais recentes se o **Makefile.am** e o **configure.ac** fornecidos pelo autor forem compatíveis com a versão mais recente. Isto é vantajoso para portar para novas arquitecturas de CPU, etc. Isto pode ser automatizado ao se usar a opção “**--with autoreconf**” para o comando **dh**.

Vamos assumir que o tarball do autor seja **debhello-1.6.tar.xz**.

Este tipo de fonte destina-se a ser instalado como um ficheiro não-sistema, por exemplo, como:

```
[base_dir] $ tar --xz -xmf debhello-1.6.tar.xz
[base_dir] $ cd debhello-1.6
[debhello-1.6] $ autoreconf -ivf # optional
[debhello-1.6] $ ./configure --with-math
[debhello-1.6] $ make
[debhello-1.6] $ make install
```

Vamos obter a fonte e criar o pacote Debian.

Download debhello-1.6.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.6.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.6.tar.xz
[base_dir] $ tree
.
+-- debhello-1.6
|   +-- LICENSE
|   +-- Makefile.am
|   +-- README.md
|   +-- configure.ac
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- Makefile.am
```

```
|   |   +-- hello.1
|   +-- src
|       +-- Makefile.am
|       +-- hello.c
+-- debhello-1.6.tar.xz

5 directories, 11 files
```

Aqui, os conteúdos desta fonte são como se segue.

src/hello.c (v=1.6):

```
[base_dir] $ cat debhello-1.6/src/hello.c
#include "config.h"
#ifdef WITH_MATH
#  include <math.h>
#endif
#include <stdio.h>
int
main()
{
        printf("Hello, I am " PACKAGE_AUTHOR "!\n");
#ifdef WITH_MATH
        printf("4.0 * atan(1.0) = %10f8\n", 4.0*atan(1.0));
#else
        printf("I can't do MATH!\n");
#endif
        return 0;
}
```

Makefile.am (v=1.6):

```
[base_dir] $ cat debhello-1.6/Makefile.am
SUBDIRS = src man
[base_dir] $ cat debhello-1.6/man/Makefile.am
dist_man_MANS = hello.1
[base_dir] $ cat debhello-1.6/src/Makefile.am
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

configure.ac (v=1.6):

```
[base_dir] $ cat debhello-1.6/configure.ac
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.
AC_PREREQ([2.69])
AC_INIT([debhello],[2.1],[foo@example.org])
AC_CONFIG_SRCDIR([src/hello.c])
AC_CONFIG_HEADERS([config.h])
echo "Standard customization chores"
AC_CONFIG_AUX_DIR([build-aux])
AM_INIT_AUTOMAKE([foreign])
# Add #define PACKAGE_AUTHOR ... in config.h with a comment
AC_DEFINE(PACKAGE_AUTHOR, ["Osamu Aoki"], [Define PACKAGE_AUTHOR])
echo "Add --with-math option functionality to ./configure"
AC_ARG_WITH([math],
  [AS_HELP_STRING([--with-math],
    [compile with math library  @<:@default=yes@:>@])],
  [],
  [with_math="yes"]
  )
echo "==== withval   := \"$withval\""
echo "==== with_math := \"$with_math\""
# m4sh if-else construct
AS_IF([test "x$with_math" != "xno"],[
  echo "==== Check include: math.h"
  AC_CHECK_HEADER(math.h,[],[
```

```
    AC_MSG_ERROR([Couldn't find math.h.] )
  ])
  echo "==== Check library: libm"
  AC_SEARCH_LIBS(atan, [m])
  #AC_CHECK_LIB(m, atan)
  echo "==== Build with LIBS := \"$LIBS\""
  AC_DEFINE(WITH_MATH, [1], [Build with the math library])
],[
  echo "==== Skip building with math.h."
  AH_TEMPLATE(WITH_MATH, [Build without the math library])
])
# Checks for programs.
AC_PROG_CC
AC_CONFIG_FILES([Makefile
                 man/Makefile
                 src/Makefile])
AC_OUTPUT
```

Dica



Sem um nível de rigor “**foreign**” especificado em **AM_INIT_AUTOMAKE()** como em cima, o **automake** predefine para o nível de rigor de “**gnu**” requerendo vários ficheiros no directório de nível de topo. Veja “3.2 Strictness” no documento do **automake**.

Vamos empacotar isto com o comando **debmake**.

```
[base_dir] $ cd debhello-1.6
[debhello-1.6] $ debmake -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.6] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.6", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.6"
I: [base_dir] $ ln -sf 'debhello-1.6.tar.xz' 'debhello_1.6.orig.tar.xz'
I: [base_dir] $ cd debhello-1.6
I: parsing option -b ""
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: build_type = Autotools with autoreconf
I: ext_type = am                      3 files
I: ext_type = c                       1 files
I: ext_type = 1                       1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.6] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
I: creating debian/source/format from extra0source_format
 ...
```

O resultado é semelhante a “Secção 14.8” mas não exactamente o mesmo.
Vamos inspecionar os ficheiros modelo notáveis gerados.
debian/rules (ficheiro modelo, v=1.6):

```
[base_dir] $ cd debhello-1.6
[debhello-1.6] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
```

```
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# See FEATURE AREAS/ENVIRONMENT in dpkg-buildflags(1)
# Apply all hardening options
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
# Package maintainers to append CFLAGS
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
# Package maintainers to append LDFLAGS
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1
#
# With debhelper version 9 or newer, the dh command exports
# all buildflags.  So there is no need to include the
# /usr/share/dpkg/buildflags.mk file here if compat is 9 or newer.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@ --with autoreconf

# debmake generated override targets
# Set options for ./configure
#CONFIGURE_FLAGS = <options for ./configure>
#overrride_dh_configure:
#       dh_configure -- $(CONFIGURE_FLAGS)
#
# Do not install libtool archive, python .pyc .pyo
#override_dh_install:
#       dh_install --list-missing -X.la -X.pyc -X.pyo
```

Vamos criar este pacote Debian melhor sendo o maintainer.
debian/rules (versão do maintainer, v=1.6):

```
[base_dir] $ cd debhello-1.6
[debhello-1.6] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-1.6] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed
```

```
%:
        dh $@ --with autoreconf

override_dh_auto_configure:
        dh_auto_configure -- \
                --with-math
```

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 5.8”.

14.10 CMake (pacote singular-binário)

Aqui está um exemplo de criar um pacote Debian simples a partir de um programa fonte C simples usando o CMake (**CMakeLists.txt** a alguns ficheiros como o **config.h.in**) como seu sistema de compilação.

O comando **cmake** gera o ficheiro **Makefile** baseado no ficheiro **CMakeLists.txt** e a sua opção **-D**. Também configura o ficheiro como especificado no seu **configure_file(…)** ao substituir strings com **@…@** e alterando a linha **#cmakedefine …**.

Vamos assumir que o tarball do autor seja **debhello-1.7.tar.xz**.

Este tipo de fonte destina-se a ser instalado como um ficheiro não-sistema, por exemplo, como:

```
[base_dir] $ tar --xz -xmf debhello-1.7.tar.xz
[base_dir] $ cd debhello-1.7
[debhello-1.7] $ mkdir obj-x86_64-linux-gnu # for out-of-tree build
[debhello-1.7] $ cd obj-x86_64-linux-gnu
[debhello-1.7] $ cmake ..
[debhello-1.7] $ make
[debhello-1.7] $ make install
```

Vamos obter a fonte e criar o pacote Debian.

Download debhello-1.7.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-1.7.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-1.7.tar.xz
[base_dir] $ tree
.
+-- debhello-1.7
|   +-- CMakeLists.txt
|   +-- LICENSE
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- CMakeLists.txt
|   |   +-- hello.1
|   +-- src
|       +-- CMakeLists.txt
|       +-- config.h.in
|       +-- hello.c
+-- debhello-1.7.tar.xz

5 directories, 11 files
```

Aqui, os conteúdos desta fonte são como se segue.

src/hello.c (v=1.7):

```
[base_dir] $ cat debhello-1.7/src/hello.c
#include "config.h"
#ifdef WITH_MATH
```

```
#  include <math.h>
#endif
#include <stdio.h>
int
main()
{
        printf("Hello, I am " PACKAGE_AUTHOR "!\n");
#ifdef WITH_MATH
        printf("4.0 * atan(1.0) = %10f8\n", 4.0*atan(1.0));
#else
        printf("I can't do MATH!\n");
#endif
        return 0;
}
```

src/config.h.in (v=1.7):

```
[base_dir] $ cat debhello-1.7/src/config.h.in
/* name of the package author */
#define PACKAGE_AUTHOR "@PACKAGE_AUTHOR@"
/* math library support */
#cmakedefine WITH_MATH
```

CMakeLists.txt (v=1.7):

```
[base_dir] $ cat debhello-1.7/CMakeLists.txt
cmake_minimum_required(VERSION 3.31)
project(debhello)
set(PACKAGE_AUTHOR "Osamu Aoki")
add_subdirectory(src)
add_subdirectory(man)
[base_dir] $ cat debhello-1.7/man/CMakeLists.txt
install(
  FILES ${CMAKE_CURRENT_SOURCE_DIR}/hello.1
  DESTINATION share/man/man1
)
[base_dir] $ cat debhello-1.7/src/CMakeLists.txt
# Always define HAVE_CONFIG_H
add_definitions(-DHAVE_CONFIG_H)
# Interactively define WITH_MATH
option(WITH_MATH "Build with math support" OFF)
#variable_watch(WITH_MATH)
# Generate config.h from config.h.in
configure_file(
  "${CMAKE_CURRENT_SOURCE_DIR}/config.h.in"
  "${CMAKE_CURRENT_BINARY_DIR}/config.h"
)
include_directories("${CMAKE_CURRENT_BINARY_DIR}")
add_executable(hello hello.c)
install(TARGETS hello
  RUNTIME DESTINATION bin
)
```

Vamos empacotar isto com o comando **debmake**.

```
[base_dir] $ cd debhello-1.7
[debhello-1.7] $ debmake -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-1.7] $ cd ..
I: Non-native Debian package pkg="debhello", ver="1.7", rev="1" method="dir_d...
I: already in the package-version form: "debhello-1.7"
I: [base_dir] $ ln -sf 'debhello-1.7.tar.xz' 'debhello_1.7.orig.tar.xz'
I: [base_dir] $ cd debhello-1.7
I: parsing option -b ""
I: binary package=debhello Type=bin / Arch=any M-A=foreign
```

```
I: build_type = Cmake
I: ext_type = c                       2 files
I: ext_type = 1                       1 files
I: ext_type = desktop                 1 files
I: creating debian/* files with "-x 1" option
I: [debhello-1.7] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
I: creating debian/rules from extra0_rules
I: creating debian/source/format from extra0source_format
 ...
```

O resultado é semelhante a “Secção 14.8” mas não exactamente o mesmo.
Vamos inspecionar os ficheiros modelo notáveis gerados.
debian/rules (ficheiro modelo, v=1.7):

```
[base_dir] $ cd debhello-1.7
[debhello-1.7] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# See FEATURE AREAS/ENVIRONMENT in dpkg-buildflags(1)
# Apply all hardening options
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
# Package maintainers to append CFLAGS
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
# Package maintainers to append LDFLAGS
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1
#
# With debhelper version 9 or newer, the dh command exports
# all buildflags.  So there is no need to include the
# /usr/share/dpkg/buildflags.mk file here if compat is 9 or newer.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
```

```
        dh $@

# debmake generated override targets
#override_dh_auto_configure:
#       dh_auto_configure -- \
#             -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"
#
# You may need to patch CMakeLists.txt to set the library install path to be:...
#-install(TARGETS <sharedlibname> LIBRARY DESTINATION lib)
#+install(TARGETS <sharedlibname> LIBRARY DESTINATION lib/${CMAKE_LIBRARY_ARC...

# Multiarch package requires library files to be installed to
# /usr/lib/<triplet>/ .  If the build system does not support
# $(DEB_HOST_MULTIARCH), you may need to override some targets such as
# dh_auto_configure or dh_auto_install to use $(DEB_HOST_MULTIARCH) .
```

debian/control (ficheiro modelo, v=1.7):

```
[debhello-1.7] $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 cmake,
Standards-Version: 4.7.3
Homepage: <insert the upstream URL, if relevant>
Rules-Requires-Root: no
#Vcs-Git: https://salsa.debian.org/debian/debhello.git
#Vcs-Browser: https://salsa.debian.org/debian/<project_site>

Package: debhello
Section: unknown
Architecture: any
Multi-Arch: foreign
Depends:
 ${misc:Depends},
 ${shlibs:Depends},
Description: auto-generated package by debmake
 This Debian binary package was auto-generated by the
 debmake(1) command provided by the debmake package.
 .
 ===== This comes from the unmodified template file =====
 .
 Please edit this template file (debian/control) and other package files
 (debian/*) to make them meet all the requirements of the Debian Policy
 before uploading this package to the Debian archive.
 .
 See
  * https://www.debian.org/doc/manuals/developers-reference/best-pkging-pract...
  * https://www.debian.org/doc/manuals/debmake-doc/ch05.en.html#control
 .
 The synopsis description at the top should be about 60 characters and
 written as a phrase.  No extra capital letters or a final period.  No
 articles b''—b'' "a", "an", or "the".
 .
 The package description for general-purpose applications should be
 written for a less technical user.  This means that we should avoid
 jargon.  GNOME or KDE is fine but GTK+ is probably not.
 .
 Use the canonical forms of words:
  * Use X Window System, X11, or X; not X Windows, X-Windows, or X Window.
  * Use GTK+, not GTK or gtk.
  * Use GNOME, not Gnome.
```

```
  * Use PostScript, not Postscript or postscript.
```

Vamos criar este pacote Debian melhor sendo o maintainer.

debian/rules (versão do maintainer, v=1.7):

```
[base_dir] $ cd debhello-1.7
[debhello-1.7] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-1.7] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
        dh $@

override_dh_auto_configure:
        dh_auto_configure -- -DWITH-MATH=1
```

debian/control (versão do maintainer, v=1.7):

```
[debhello-1.7] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-1.7] $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 cmake,
Standards-Version: 4.7.3
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends:
 ${misc:Depends},
 ${shlibs:Depends},
Description: Simple packaging example for debmake
 This Debian binary package is an example package.
 (This is an example only)
```

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 14.8”.

14.11 Autotools (pacote multi-binário)

Aqui está um exemplo de criar um conjunto de pacotes binário Debian incluindo o pacote executável, o pacote da biblioteca partilhada, o pacote de ficheiros de desenvolvimento, e o pacote de símbolos de depuração a partir de um programa fonte C simples usando Autotools = (Autoconf e Automake os quais usam **Makefile.am** e **configure.ac** como seus ficheiros de entrada) como seu sistema de compilação.

Vamos empacotar isto num modo semelhante a “Secção 14.9”.

Vamos assumir que o tarball do autor seja **debhello-2.0.tar.xz**.

Este tipo de fonte destina-se a ser instalado como um ficheiro não-sistema, por exemplo, como:

```
[base_dir] $ tar --xz -xmf debhello-2.0.tar.xz
[base_dir] $ cd debhello-2.0
```

```
[debhello-2.0] $ autoreconf -ivf # optional
[debhello-2.0] $ ./configure --with-math
[debhello-2.0] $ make
[debhello-2.0] $ make install
```

Vamos obter a fonte e criar o pacote Debian.

Download debhello-2.0.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-2.0.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-2.0.tar.xz
[base_dir] $ tree
.
+-- debhello-2.0
|   +-- LICENSE
|   +-- Makefile.am
|   +-- README.md
|   +-- configure.ac
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- lib
|   |   +-- Makefile.am
|   |   +-- sharedlib.c
|   |   +-- sharedlib.h
|   +-- man
|   |   +-- Makefile.am
|   |   +-- hello.1
|   +-- src
|       +-- Makefile.am
|       +-- hello.c
+-- debhello-2.0.tar.xz

6 directories, 14 files
```

Aqui, os conteúdos desta fonte são como se segue.

src/hello.c (v=2.0):

```
[base_dir] $ cat debhello-2.0/src/hello.c
#include "config.h"
#include <stdio.h>
#include <sharedlib.h>
int
main()
{
        printf("Hello, I am " PACKAGE_AUTHOR "!\n");
        sharedlib();
        return 0;
}
```

lib/sharedlib.h e lib/sharedlib.c (v=1.6):

```
[base_dir] $ cat debhello-2.0/lib/sharedlib.h
int sharedlib();
[base_dir] $ cat debhello-2.0/lib/sharedlib.c
#include <stdio.h>
int
sharedlib()
{
        printf("This is a shared library!\n");
        return 0;
}
```

Makefile.am (v=2.0):

```
[base_dir] $ cat debhello-2.0/Makefile.am
```

```
# recursively process `Makefile.am` in SUBDIRS
SUBDIRS = lib src man
[base_dir] $ cat debhello-2.0/man/Makefile.am
# manpages (distributed in the source package)
dist_man_MANS = hello.1
[base_dir] $ cat debhello-2.0/lib/Makefile.am
# libtool librares to be produced
lib_LTLIBRARIES = libsharedlib.la

# source files used for lib_LTLIBRARIES
libsharedlib_la_SOURCES = sharedlib.c

# C pre-processor flags used for lib_LTLIBRARIES
#libsharedlib_la_CPPFLAGS =

# Headers files to be installed in <prefix>/include
include_HEADERS = sharedlib.h

# Versioning Libtool Libraries with version triplets
libsharedlib_la_LDFLAGS = -version-info 1:0:0
[base_dir] $ cat debhello-2.0/src/Makefile.am
# program executables to be produced
bin_PROGRAMS = hello

# source files used for bin_PROGRAMS
hello_SOURCES = hello.c

# C pre-processor flags used for bin_PROGRAMS
AM_CPPFLAGS = -I$(srcdir) -I$(top_srcdir)/lib

# Extra options for the linker for hello
# hello_LDFLAGS =

# Libraries the `hello` binary to be linked
hello_LDADD = $(top_srcdir)/lib/libsharedlib.la
```

configure.ac (v=2.0):

```
[base_dir] $ cat debhello-2.0/configure.ac
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.
AC_PREREQ([2.69])
AC_INIT([debhello],[2.2],[foo@example.org])
AC_CONFIG_SRCDIR([src/hello.c])
AC_CONFIG_HEADERS([config.h])
echo "Standard customization chores"
AC_CONFIG_AUX_DIR([build-aux])

AM_INIT_AUTOMAKE([foreign])

# Set default to --enable-shared --disable-static
LT_INIT([shared disable-static])

# find the libltdl sources in the libltdl sub-directory
LT_CONFIG_LTDL_DIR([libltdl])

# choose one
LTDL_INIT([recursive])
#LTDL_INIT([subproject])
#LTDL_INIT([nonrecursive])

# Add #define PACKAGE_AUTHOR ... in config.h with a comment
AC_DEFINE(PACKAGE_AUTHOR, ["Osamu Aoki"], [Define PACKAGE_AUTHOR])
# Checks for programs.
AC_PROG_CC
```

```
# only for the recursive case
AC_CONFIG_FILES([Makefile
                 lib/Makefile
                 man/Makefile
                 src/Makefile])
AC_OUTPUT
```

Vamos usar o comando **debmake** para empacotar isto em múltiplos pacotes:

- **debhello**: tipo = **bin**
- **libsharedlib1**: tipo = **lib**
- **libsharedlib-dev**: tipo = **dev**

Aqui, usamos a opção **-b\',libsharedlib1,libsharedlib-dev'** para especificar os pacotes binário adicionais a serem gerados.

```
[base_dir] $ cd debhello-2.0
[debhello-2.0] $ debmake -b',libsharedlib1,libsharedlib-dev' -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-2.0] $ cd ..
I: Non-native Debian package pkg="debhello", ver="2.0", rev="1" method="dir_d...
I: already in the package-version form: "debhello-2.0"
I: [base_dir] $ ln -sf 'debhello-2.0.tar.xz' 'debhello_2.0.orig.tar.xz'
I: [base_dir] $ cd debhello-2.0
I: parsing option -b ",libsharedlib1,libsharedlib-dev"
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: binary package=libsharedlib1 Type=lib / Arch=any M-A=same
I: binary package=libsharedlib-dev Type=dev / Arch=any M-A=same
I: build_type = Autotools with autoreconf
I: ext_type = am                      4 files
I: ext_type = c                       3 files
I: ext_type = 1                       1 files
I: creating debian/* files with "-x 1" option
I: [debhello-2.0] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
 ...
```

O resultado é semelhante a “Secção 14.8” mas com mais ficheiros modelo.

Vamos inspecionar os ficheiros modelo notáveis gerados.

debian/rules (ficheiro modelo, v=2.0):

```
[base_dir] $ cd debhello-2.0
[debhello-2.0] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# See FEATURE AREAS/ENVIRONMENT in dpkg-buildflags(1)
# Apply all hardening options
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
# Package maintainers to append CFLAGS
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
```

```
# Package maintainers to append LDFLAGS
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1
#
# With debhelper version 9 or newer, the dh command exports
# all buildflags.  So there is no need to include the
# /usr/share/dpkg/buildflags.mk file here if compat is 9 or newer.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@ --with autoreconf

# debmake generated override targets
# Set options for ./configure
#CONFIGURE_FLAGS = <options for ./configure>
#overrride_dh_configure:
#       dh_configure -- $(CONFIGURE_FLAGS)
#
# Do not install libtool archive, python .pyc .pyo
#override_dh_install:
#       dh_install --list-missing -X.la -X.pyc -X.pyo
```

Vamos criar este pacote Debian melhor sendo o maintainer.

debian/rules (versão do maintainer, v=2.0):

```
[base_dir] $ cd debhello-2.0
[debhello-2.0] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-2.0] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed

%:
        dh $@ --with autoreconf

override_dh_missing:
        dh_missing -X.la
```

debian/control (versão do maintainer, v=2.0):

```
[debhello-2.0] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-2.0] $ cat debian/control
Source: debhello
```

```
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 dh-autoreconf,
Standards-Version: 4.7.3
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends:
 libsharedlib1 (= ${binary:Version}),
 ${misc:Depends},
 ${shlibs:Depends},
Description: Simple packaging example for debmake
 This package contains the compiled binary executable.
 .
 This Debian binary package is an example package.
 (This is an example only)

Package: libsharedlib1
Section: libs
Architecture: any
Multi-Arch: same
Pre-Depends:
 ${misc:Pre-Depends},
Depends:
 ${misc:Depends},
 ${shlibs:Depends},
Description: Simple packaging example for debmake
 This package contains the shared library.

Package: libsharedlib-dev
Section: libdevel
Architecture: any
Multi-Arch: same
Depends:
 libsharedlib1 (= ${binary:Version}),
 ${misc:Depends},
Description: Simple packaging example for debmake
 This package contains the development files.
```

debian/*.install (versão do maintainer, v=2.0):

```
[debhello-2.0] $ vim debian/copyright
 ... hack, hack, hack, ...
[debhello-2.0] $ cat debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: debhello
Upstream-Contact: Osamu Aoki <osamu@debian.org>
Source: https://salsa.debian.org/debian/debmake-doc

Files:     *
Copyright: 2015-2021 Osamu Aoki <osamu@debian.org>
License:   Expat
 Permission is hereby granted, free of charge, to any person obtaining a
 copy of this software and associated documentation files (the "Software"),
 to deal in the Software without restriction, including without limitation
 the rights to use, copy, modify, merge, publish, distribute, sublicense,
 and/or sell copies of the Software, and to permit persons to whom the
 Software is furnished to do so, subject to the following conditions:
 .
```

```
 The above copyright notice and this permission notice shall be included
 in all copies or substantial portions of the Software.
 .
 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Como esta fonte de autor cria o auto-gerado **Makefile** apropriado, não é preciso criar os ficheiros **debian/install** e **debian/manpages**.

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

Ficheiros modelo sob debian/. (v=2.0):

```
[debhello-2.0] $ rm -f debian/clean debian/dirs debian/install debian/links
[debhello-2.0] $ rm -f debian/README.source debian/source/*.ex
[debhello-2.0] $ rm -rf debian/patches
[debhello-2.0] $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- debhello.install
+-- docs
+-- examples
+-- gbp.conf
+-- libsharedlib-dev.install
+-- libsharedlib1.install
+-- libsharedlib1.symbols
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 18 files
```

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 14.8”.

Aqui está a lista de dependências gerada de todos os pacotes binários.

A lista de dependências gerada de todos os pacotes binários (v=2.0):

```
[debhello-2.0] $ dpkg -f debhello-dbgsym_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: debhello (= 2.0-1)
[debhello-2.0] $ dpkg -f debhello_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.0-1), libc6 (>= 2.34)
[debhello-2.0] $ dpkg -f libsharedlib-dev_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.0-1)
[debhello-2.0] $ dpkg -f libsharedlib1-dbgsym_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.0-1)
[debhello-2.0] $ dpkg -f libsharedlib1_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libc6 (>= 2.2.5)
```

14.12 CMake (pacote multi-binário)

Este exemplo demonstra a criação dum conjunto de pacotes binário Debian incluindo o pacote executável, o pacote da biblioteca partilhada, o pacote de ficheiros de desenvolvimento, e o pacote de símbolos de depuração a partir de um programa fonte C simples usando CMake (**CMakeLists.txt** e ficheiros tais como **config.h.in**) como seu sistema de compilação.

Vamos assumir que o tarball do autor seja **debhello-2.1.tar.xz**.

Este tipo de fonte destina-se a ser instalado como um ficheiro não-sistema, por exemplo, como:

```
[base_dir] $ tar --xz -xmf debhello-2.1.tar.xz
[base_dir] $ cd debhello-2.1
[debhello-2.1] $ mkdir obj-x86_64-linux-gnu
[debhello-2.1] $ cd obj-x86_64-linux-gnu
[debhello-2.1] $ cmake ..
[debhello-2.1] $ make
[debhello-2.1] $ make install
```

Vamos obter a fonte e criar o pacote Debian.

Download debhello-2.1.tar.xz

```
[base_dir] $ wget http://www.example.org/download/debhello-2.1.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-2.1.tar.xz
[base_dir] $ tree
.
+-- debhello-2.1
|   +-- CMakeLists.txt
|   +-- LICENSE
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- lib
|   |   +-- CMakeLists.txt
|   |   +-- sharedlib.c
|   |   +-- sharedlib.h
|   +-- man
|   |   +-- CMakeLists.txt
|   |   +-- hello.1
|   +-- src
|       +-- CMakeLists.txt
|       +-- config.h.in
|       +-- hello.c
+-- debhello-2.1.tar.xz

6 directories, 14 files
```

Aqui, os conteúdos desta fonte são como se segue.

src/hello.c (v=2.1):

```
[base_dir] $ cat debhello-2.1/src/hello.c
#include "config.h"
#include <stdio.h>
#include <sharedlib.h>
int
main()
{
        printf("Hello, I am " PACKAGE_AUTHOR "!\n");
        sharedlib();
        return 0;
}
```

src/config.h.in (v=2.1):

```
[base_dir] $ cat debhello-2.1/src/config.h.in
```

```
/* name of the package author */
#define PACKAGE_AUTHOR "@PACKAGE_AUTHOR@"
```

lib/sharedlib.c e lib/sharedlib.h (v=2.1):

```
[base_dir] $ cat debhello-2.1/lib/sharedlib.h
int sharedlib();
[base_dir] $ cat debhello-2.1/lib/sharedlib.c
#include <stdio.h>
int
sharedlib()
{
        printf("This is a shared library!\n");
        return 0;
}
```

CMakeLists.txt (v=2.1):

```
[base_dir] $ cat debhello-2.1/CMakeLists.txt
cmake_minimum_required(VERSION 3.31)
project(debhello)
set(PACKAGE_AUTHOR "Osamu Aoki")
add_subdirectory(lib)
add_subdirectory(src)
add_subdirectory(man)
[base_dir] $ cat debhello-2.1/man/CMakeLists.txt
install(
  FILES ${CMAKE_CURRENT_SOURCE_DIR}/hello.1
  DESTINATION share/man/man1
)
[base_dir] $ cat debhello-2.1/src/CMakeLists.txt
# Always define HAVE_CONFIG_H
add_definitions(-DHAVE_CONFIG_H)
# Generate config.h from config.h.in
configure_file(
  "${CMAKE_CURRENT_SOURCE_DIR}/config.h.in"
  "${CMAKE_CURRENT_BINARY_DIR}/config.h"
  )
include_directories("${CMAKE_CURRENT_BINARY_DIR}")
include_directories("${CMAKE_SOURCE_DIR}/lib")

add_executable(hello hello.c)
target_link_libraries(hello sharedlib)
install(TARGETS hello
  RUNTIME DESTINATION bin
)
```

Vamos empacotar isto com o comando **debmake**.

```
[base_dir] $ cd debhello-2.1
[debhello-2.1] $ debmake -b',libsharedlib1,libsharedlib-dev' -x1
I: debmake (version: 5.1.5)
I: Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>
I: [debhello-2.1] $ cd ..
I: Non-native Debian package pkg="debhello", ver="2.1", rev="1" method="dir_d...
I: already in the package-version form: "debhello-2.1"
I: [base_dir] $ ln -sf 'debhello-2.1.tar.xz' 'debhello_2.1.orig.tar.xz'
I: [base_dir] $ cd debhello-2.1
I: parsing option -b ",libsharedlib1,libsharedlib-dev"
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: binary package=libsharedlib1 Type=lib / Arch=any M-A=same
I: binary package=libsharedlib-dev Type=dev / Arch=any M-A=same
I: build_type = Cmake
I: ext_type = c                       4 files
I: ext_type = 1                       1 files
I: ext_type = desktop                 1 files
```

```
I: creating debian/* files with "-x 1" option
I: [debhello-2.1] $ licensecheck --recursive --copyright --deb-machine  . > d...
I: creating debian/copyright by licensecheck.
I: creating debian/control from control.py
I: creating debian/control by control.py
I: creating debian/changelog from extra0_changelog
 ...
```

O resultado é semelhante a “Secção 14.8” mas não exactamente o mesmo.
Vamos inspecionar os ficheiros modelo notáveis gerados.
debian/rules (ficheiro modelo, v=2.1):

```
[base_dir] $ cd debhello-2.1
[debhello-2.1] $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
# See debhelper(7) (un-comment to enable)
# This is an autogenerated template for debian/rules.
#
# Output every command that modifies files on the build system.
#export DH_VERBOSE = 1
#
# Copy some variable definitions from pkg-info.mk and vendor.mk
# under /usr/share/dpkg/ to here if they are useful.
#
# See FEATURE AREAS/ENVIRONMENT in dpkg-buildflags(1)
# Apply all hardening options
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
# Package maintainers to append CFLAGS
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
# Package maintainers to append LDFLAGS
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1
#
# With debhelper version 9 or newer, the dh command exports
# all buildflags.  So there is no need to include the
# /usr/share/dpkg/buildflags.mk file here if compat is 9 or newer.
#
# These are rarely used code. (START)
#
# The following include for *.mk magically sets miscellaneous
# variables while honoring existing values of pertinent
# environment variables:
#
# Architecture-related variables such as DEB_TARGET_MULTIARCH:
#include /usr/share/dpkg/architecture.mk
# Vendor-related variables such as DEB_VENDOR:
#include /usr/share/dpkg/vendor.mk
# Package-related variables such as DEB_DISTRIBUTION
#include /usr/share/dpkg/pkg-info.mk
#
# You may alternatively set them susing a simple script such as:
# DEB_VENDOR ?= $(shell dpkg-vendor --query  Vendor)
#
# These are rarely used code. (END)
#

### main packaging script based on post dh7 syntax
%:
        dh $@

# debmake generated override targets
#override_dh_auto_configure:
#       dh_auto_configure -- \
#             -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"
```

```
#
# You may need to patch CMakeLists.txt to set the library install path to be:...
#-install(TARGETS <sharedlibname> LIBRARY DESTINATION lib)
#+install(TARGETS <sharedlibname> LIBRARY DESTINATION lib/${CMAKE_LIBRARY_ARC...

# Multiarch package requires library files to be installed to
# /usr/lib/<triplet>/ .  If the build system does not support
# $(DEB_HOST_MULTIARCH), you may need to override some targets such as
# dh_auto_configure or dh_auto_install to use $(DEB_HOST_MULTIARCH) .
```

Vamos criar este pacote Debian melhor sendo o maintainer.
debian/rules (versão do maintainer, v=2.1):

```
[base_dir] $ cd debhello-2.1
[debhello-2.1] $ vim debian/rules
 ... hack, hack, hack, ...
[debhello-2.1] $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1
export DEB_BUILD_MAINT_OPTIONS = hardening=+all
export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed
DEB_HOST_MULTIARCH ?= $(shell dpkg-architecture -qDEB_HOST_MULTIARCH)

%:
        dh $@

override_dh_auto_configure:
        dh_auto_configure -- \
              -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_HOST_MULTIARCH)"
```

debian/control (versão do maintainer, v=2.1):

```
[debhello-2.1] $ vim debian/control
 ... hack, hack, hack, ...
[debhello-2.1] $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 cmake,
Standards-Version: 4.7.3
Homepage: https://salsa.debian.org/debian/debmake-doc
Rules-Requires-Root: no

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends:
 libsharedlib1 (= ${binary:Version}),
 ${misc:Depends},
 ${shlibs:Depends},
Description: Simple packaging example for debmake
 This package contains the compiled binary executable.
 .
 This Debian binary package is an example package.
 (This is an example only)

Package: libsharedlib1
Section: libs
Architecture: any
Multi-Arch: same
Pre-Depends:
```

```
 ${misc:Pre-Depends},
Depends:
 ${misc:Depends},
 ${shlibs:Depends},
Description: Simple packaging example for debmake
 This package contains the shared library.

Package: libsharedlib-dev
Section: libdevel
Architecture: any
Multi-Arch: same
Depends:
 libsharedlib1 (= ${binary:Version}),
 ${misc:Depends},
Description: Simple packaging example for debmake
 This package contains the development files.
```

debian/*.install (versão do maintainer, v=2.1):

```
[debhello-2.1] $ vim debian/copyright
 ... hack, hack, hack, ...
[debhello-2.1] $ cat debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: debhello
Upstream-Contact: Osamu Aoki <osamu@debian.org>
Source: https://salsa.debian.org/debian/debmake-doc

Files:     *
Copyright: 2015-2021 Osamu Aoki <osamu@debian.org>
License:   Expat
 Permission is hereby granted, free of charge, to any person obtaining a
 copy of this software and associated documentation files (the "Software"),
 to deal in the Software without restriction, including without limitation
 the rights to use, copy, modify, merge, publish, distribute, sublicense,
 and/or sell copies of the Software, and to permit persons to whom the
 Software is furnished to do so, subject to the following conditions:
 .
 The above copyright notice and this permission notice shall be included
 in all copies or substantial portions of the Software.
 .
 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

O ficheiro CMakeLists.txt do autor precisa receber uma patch para lidar com o caminho multiarch corretamente.

debian/patches/* (versão do maintainer, v=2.1):

```
 ... hack, hack, hack, ...
[debhello-2.1] $ cat debian/libsharedlib1.symbols
libsharedlib.so.1 libsharedlib1 #MINVER#
 sharedlib@Base 2.1
```

Como esta fonte de autor cria o auto-gerado **Makefile** apropriado, não é preciso criar os ficheiros **debian/install** e **debian/manpages**.

Existem vários outros ficheiros modelo sob o directório **debian/**. Estes também precisam de ser atualizados.

Ficheiros modelo sob debian/. (v=2.1):

```
[debhello-2.1] $ rm -f debian/clean debian/dirs debian/install debian/links
[debhello-2.1] $ rm -f debian/README.source debian/source/*.ex
```

```
[debhello-2.1] $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- debhello.install
+-- docs
+-- examples
+-- gbp.conf
+-- libsharedlib-dev.install
+-- libsharedlib1.install
+-- libsharedlib1.symbols
+-- manpages
+-- patches/
|   +-- 000-cmake-multiarch.patch
|   +-- series
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

5 directories, 20 files
```

O resto das atividades de empacotamento são praticamente o mesmo que em “Secção 14.8”.

Aqui está a lista de dependências gerada de todos os pacotes binários.

A lista de dependências gerada de todos os pacotes binários (v=2.1):

```
[debhello-2.1] $ dpkg -f debhello-dbgsym_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: debhello (= 2.1-1)
[debhello-2.1] $ dpkg -f debhello_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.1-1), libc6 (>= 2.34)
[debhello-2.1] $ dpkg -f libsharedlib-dev_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.1-1)
[debhello-2.1] $ dpkg -f libsharedlib1-dbgsym_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.1-1)
[debhello-2.1] $ dpkg -f libsharedlib1_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libc6 (>= 2.2.5)
```

14.13 Internacionalização

Aqui está um exemplo de actualizar a fonte C simples do autor **debhello-2.0.tar.xz** apresentado em “Secção 14.11” para internacionalização (i18n) e criar a fonte C do autor actualizada **debhello-2.0.tar.xz**.

Na situação real, o pacote já deveria estar internacionalizado. Assim este exemplo é educativo para você compreender como esta internacionalização é implementada.

Dica



A atividade de rotina do maintainer para o i18n é simplesmente adicionar ficheiros po de tradução reportados para si via Bug Tracking System (BTS) para o directório **po/** e actualizar a lista de linguagens no ficheiro **po/LINGUAS**.

Vamos obter a fonte e criar o pacote Debian.

Download debhello-2.0.tar.xz (i18n)

```
[base_dir] $ wget http://www.example.org/download/debhello-2.0.tar.xz
 ...
[base_dir] $ tar --xz -xmf debhello-2.0.tar.xz
[base_dir] $ tree
.
+-- debhello-2.0
|   +-- LICENSE
|   +-- Makefile.am
|   +-- README.md
|   +-- configure.ac
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- lib
|   |   +-- Makefile.am
|   |   +-- sharedlib.c
|   |   +-- sharedlib.h
|   +-- man
|   |   +-- Makefile.am
|   |   +-- hello.1
|   +-- src
|       +-- Makefile.am
|       +-- hello.c
+-- debhello-2.0.tar.xz

6 directories, 14 files
```

Internacionalizar esta árvore fonte com o comando **gettextize** e remover ficheiros auto-gerados pelo Autotools.

corra gettextize (i18n):

```
[base_dir] $ cd debhello-2.0
 $ gettextize
Creating po/ subdirectory
Creating build-aux/ subdirectory
Copying file ABOUT-NLS
Copying file build-aux/config.rpath
Not copying intl/ directory.
Copying file po/Makefile.in.in
Copying file po/Makevars.template
Copying file po/Rules-quot
Copying file po/boldquot.sed
Copying file po/en@boldquot.header
Copying file po/en@quot.header
Copying file po/insert-header.sin
Copying file po/quot.sed
Copying file po/remove-potcdate.sin
Creating initial po/POTFILES.in
Creating po/ChangeLog
Creating directory m4
Copying file m4/gettext.m4
Copying file m4/iconv.m4
Copying file m4/lib-ld.m4
Copying file m4/lib-link.m4
```

```
Copying file m4/lib-prefix.m4
Copying file m4/nls.m4
Copying file m4/po.m4
Copying file m4/progtest.m4
Creating m4/ChangeLog
Updating Makefile.am (backup is in Makefile.am~)
Updating configure.ac (backup is in configure.ac~)
Creating ChangeLog

Please use AM_GNU_GETTEXT([external]) in order to cause autoconfiguration
to look for an external libintl.

Please create po/Makevars from the template in po/Makevars.template.
You can then remove po/Makevars.template.

Please fill po/POTFILES.in as described in the documentation.

Please run 'aclocal' to regenerate the aclocal.m4 file.
You need aclocal from GNU automake 1.9 (or newer) to do this.
Then run 'autoconf' to regenerate the configure file.

You will also need config.guess and config.sub, which you can get from the CV...
of the 'config' project at http://savannah.gnu.org/. The commands to fetch th...
are
$ wget 'http://savannah.gnu.org/cgi-bin/viewcvs/*checkout*/config/config/conf...
$ wget 'http://savannah.gnu.org/cgi-bin/viewcvs/*checkout*/config/config/conf...

You might also want to copy the convenience header file gettext.h
from the /usr/share/gettext directory into your package.
It is a wrapper around <libintl.h> that implements the configure --disable-nl...
option.

Press Return to acknowledge the previous 6 paragraphs.
[debhello-2.0] $ rm -rf m4 build-aux *~
```

Vamos verificar os ficheiros gerados sob o directório **po/**.
ficheiros em po (i18n):

```
[debhello-2.0] $ ls -l po
total 60
-rw-rw-r-- 1 osamu osamu   494 Jul 22 15:31 ChangeLog
-rw-rw-r-- 1 osamu osamu 17577 Jul 22 15:31 Makefile.in.in
-rw-rw-r-- 1 osamu osamu  3376 Jul 22 15:31 Makevars.template
-rw-rw-r-- 1 osamu osamu    59 Jul 22 15:31 POTFILES.in
-rw-rw-r-- 1 osamu osamu  2203 Jul 22 15:31 Rules-quot
-rw-rw-r-- 1 osamu osamu   217 Jul 22 15:31 boldquot.sed
-rw-rw-r-- 1 osamu osamu  1337 Jul 22 15:31 en@boldquot.header
-rw-rw-r-- 1 osamu osamu  1203 Jul 22 15:31 en@quot.header
-rw-rw-r-- 1 osamu osamu   672 Jul 22 15:31 insert-header.sin
-rw-rw-r-- 1 osamu osamu   153 Jul 22 15:31 quot.sed
-rw-rw-r-- 1 osamu osamu   432 Jul 22 15:31 remove-potcdate.sin
```

Vamos actualizar o **configure.ac** ao adicionar "**AM_GNU_GETTEXT([external])**", etc..
configure.ac (i18n):

```
[debhello-2.0] $ vim configure.ac
 ... hack, hack, hack, ...
[debhello-2.0] $ cat configure.ac
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.
AC_PREREQ([2.69])
AC_INIT([debhello],[2.2],[foo@example.org])
AC_CONFIG_SRCDIR([src/hello.c])
AC_CONFIG_HEADERS([config.h])
echo "Standard customization chores"
```

```
AC_CONFIG_AUX_DIR([build-aux])

AM_INIT_AUTOMAKE([foreign])

# Set default to --enable-shared --disable-static
LT_INIT([shared disable-static])

# find the libltdl sources in the libltdl sub-directory
LT_CONFIG_LTDL_DIR([libltdl])

# choose one
LTDL_INIT([recursive])
#LTDL_INIT([subproject])
#LTDL_INIT([nonrecursive])

# Add #define PACKAGE_AUTHOR ... in config.h with a comment
AC_DEFINE(PACKAGE_AUTHOR, ["Osamu Aoki"], [Define PACKAGE_AUTHOR])
# Checks for programs.
AC_PROG_CC

# desktop file support required
AM_GNU_GETTEXT_VERSION([0.19.3])
AM_GNU_GETTEXT([external])

# only for the recursive case
AC_CONFIG_FILES([Makefile
                 po/Makefile.in
                 lib/Makefile
                 man/Makefile
                 src/Makefile])
AC_OUTPUT
```

Vamos criar o ficheiro **po/Makevars** a partir do modelo **po/Makevars.template**.
po/Makevars (i18n):

```
 ... hack, hack, hack, ...
[debhello-2.0] $ diff -u po/Makevars.template po/Makevars
--- po/Makevars.template        2026-07-22 15:31:09.890068240 +0900
+++ po/Makevars 2026-07-22 15:31:09.986012740 +0900
@@ -18,14 +18,14 @@
 # or entity, or to disclaim their copyright.  The empty string stands for
 # the public domain; in this case the translators are expected to disclaim
 # their copyright.
-COPYRIGHT_HOLDER = Free Software Foundation, Inc.
+COPYRIGHT_HOLDER = Osamu Aoki <osamu@debian.org>

 # This tells whether or not to prepend "GNU " prefix to the package
 # name that gets inserted into the header of the $(DOMAIN).pot file.
 # Possible values are "yes", "no", or empty.  If it is empty, try to
 # detect it automatically by scanning the files in $(top_srcdir) for
 # "GNU packagename" string.
-PACKAGE_GNU =
+PACKAGE_GNU = no

 # This is the email address or URL to which the translators shall report
 # bugs in the untranslated strings:
[debhello-2.0] $ rm po/Makevars.template
```

Vamos actualizar as fontes C para a versão i18n ao envolver as strings com **_(…)**.
src/hello.c (i18n):

```
 ... hack, hack, hack, ...
[debhello-2.0] $ cat src/hello.c
#include "config.h"
#include <stdio.h>
```

```
#include <sharedlib.h>
#include <libintl.h>
#define _(string) gettext (string)
int
main()
{
        printf(_("Hello, I am " PACKAGE_AUTHOR "!\n"));
        sharedlib();
        return 0;
}
```

lib/sharedlib.c (i18n):

```
 ... hack, hack, hack, ...
[debhello-2.0] $ cat lib/sharedlib.c
#include <stdio.h>
#include <libintl.h>
#define _(string) gettext (string)
int
sharedlib()
{
        printf(_("This is a shared library!\n"));
        return 0;
}
```

O novo **gettext** (v=0.19) consegue lidar coma versão i18n do ficheiro de desktop directamente.
data/hello.desktop.in (i18n):

```
[debhello-2.0] $ fgrep -v '[ja]=' data/hello.desktop > data/hello.desktop.in
[debhello-2.0] $ rm data/hello.desktop
[debhello-2.0] $ cat data/hello.desktop.in
[Desktop Entry]
Name=Hello
Comment=Greetings
Type=Application
Keywords=hello
Exec=hello
Terminal=true
Icon=hello.png
Categories=Utility;
```

Vamos listar os ficheiros de entrada para extrair as strings traduzíveis em **po/POTFILES.in**.
po/POTFILES.in (i18n):

```
 ... hack, hack, hack, ...
[debhello-2.0] $ cat po/POTFILES.in
src/hello.c
lib/sharedlib.c
data/hello.desktop.in
```

Aqui está a raiz actualizada **Makefile.am** com **po** adicionado à variável de ambiente **SUBDIRS**.
Makefile.am (i18n):

```
[debhello-2.0] $ cat Makefile.am
# recursively process `Makefile.am` in SUBDIRS
SUBDIRS = po lib src man

ACLOCAL_AMFLAGS = -I m4

EXTRA_DIST = build-aux/config.rpath m4/ChangeLog
```

Vamos criar um ficheiro modelo de tradução, **debhello.pot**.
po/debhello.pot (i18n):

```
[debhello-2.0] $ xgettext -f po/POTFILES.in -d debhello -o po/debhello.pot -k...
[debhello-2.0] $ cat po/debhello.pot
```

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2026-07-22 15:31+0900\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: src/hello.c:9
#, c-format
msgid "Hello, I am "
msgstr ""

#: lib/sharedlib.c:7
#, c-format
msgid "This is a shared library!\n"
msgstr ""

#: data/hello.desktop.in:2
msgid "Hello"
msgstr ""

#: data/hello.desktop.in:3
msgid "Greetings"
msgstr ""

#: data/hello.desktop.in:5
msgid "hello"
msgstr ""
```

Vamos adicionar uma tradução para Francês.
po/LINGUAS e po/fr.po (i18n):

```
[debhello-2.0] $ echo 'fr' > po/LINGUAS
[debhello-2.0] $ cp po/debhello.pot po/fr.po
[debhello-2.0] $ vim po/fr.po
 ... hack, hack, hack, ...
[debhello-2.0] $ cat po/fr.po
# SOME DESCRIPTIVE TITLE.
# This file is put in the public domain.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: debhello 2.2\n"
"Report-Msgid-Bugs-To: foo@example.org\n"
"POT-Creation-Date: 2015-03-01 20:22+0900\n"
"PO-Revision-Date: 2015-02-21 23:18+0900\n"
"Last-Translator: Osamu Aoki <osamu@debian.org>\n"
"Language-Team: French <LL@li.org>\n"
"Language: ja\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
```

```
#: src/hello.c:34
#, c-format
msgid "Hello, my name is %s!\n"
msgstr "Bonjour, je m'appelle %s!\n"

#: lib/sharedlib.c:29
#, c-format
msgid "This is a shared library!\n"
msgstr "Ceci est une bibliothèque partagée!\n"

#: data/hello.desktop.in:3
msgid "Hello"
msgstr ""

#: data/hello.desktop.in:4
msgid "Greetings"
msgstr "Salutations"

#: data/hello.desktop.in:6
msgid "hello"
msgstr ""

#: data/hello.desktop.in:9
msgid "hello.png"
msgstr ""
```

As atividades de empacotamento são praticamente as mesmas que em “Secção 14.11”.
Você pode encontrar mais exemplos i18n ao seguir “Secção 14.14”.

14.14 Detalhes

Você pode obter informação detalhada acerca dos exemplos apresentados e suas variantes como se segue:

Como obter detalhes

```
[base_dir] $ apt-get source debmake-doc
[base_dir] $ cd debmake-doc*
[debmake-doc-*] $ view examples/README.md
```

Siga as instruções exatas em **examples/README.md**.

```
[debmake-doc-*] $ cd examples
[examples] $ make
```

Agora, cada directório nomeado como **examples/debhello-?.?_build-?** contém o exemplo de empacotamento Debian.

- registo de atividade de linha de comandos de consola emulada: o ficheiro **.log**
- registo de atividade de linha de comandos de consola emulada (curto): o ficheiro **.slog**
- instantâneo da imagem de árvore fonte após o comando **debmake**: o directório **debmake**
- instantâneo da imagem de árvore fonte após empacotamento apropriado: o directório **package**
- instantâneo da imagem de árvore fonte após o comando **debuild**: o directório **test**

Exemplos notáveis incluem:

- Script de shell POSIX com Makefile e suporte a i18n (v=3.0)
- Fonte C com Makefile.in + configure e suporte a i18n (v=3.2)
- Fonte C com Autotools e suporte a i18n (v=3.3)
- Fonte C com CMake e suporte a i18n (v=3.4)

Capítulo 15

manual do debmake(1)

15.1 NOME

debmake - programa para criar um pacote fonte Debian

15.2 RESUMO

debmake [**-h**] [**-n**] [**-p** *pacote*] [**-u** *versão*] [**-r** *revisão*] [**-z** *extensão*] [**-b** "*pacote-binário[:tipo], …*]" [**-D** *valor*] [**-e** *foo@example.org*] [**-f** "*primeiro-nome último-nome*"] [**-i** [**debuild**|**sbuild**|**dgit sbuild**|**gbp buildpackage**|**dpkg-buildpackage**| …]] [**-m**] [**-q**] [**-v**] [**-V**] [**-w** *"addon, …"*] [**-x** [01234]] [**-y**] [**-B**] [*URL*]

15.3 DESCRIÇÃO

debmake ajuda a compilar o pacote Debian a partir da fonte do autor.

Normalmente, isto faz-se como se segue:

- A fonte de autor é obtida como um tarball de um sítio web remoto ou uma árvore de trabalho clonada usando "**git clone**".
  - Para um tarball, é expandido em muitos ficheiros no directório fonte.
  - Para uma árvore de trabalho clonada, é usada como o directório fonte.
- debmake é tipicamente invocado no directório fonte sem nenhuns argumentos.
  - O directório fonte é copiado para o directório ../*pacote-versão*/.
  - Se ../*pacote_versão***.orig.tar.xz** estiver em falta, é gerado.
  - O directório atual é movido para ../pacote-versão/.
  - Os ficheiros modelo são gerados directório ../*pacote-versão***/debian/**
- Os ficheiros no directório ../*pacote-versão***/debian/** deverão ser ajustados manualmente.
- **dpkg-buildpackage** (geralmente a partir do seu invólucro **debuild**, **sbuild**, ...) é invocado no directório ../*pacote-versão*/ para criar pacotes Debian fonte e binário.

E Também, o **debmake** pode ser invocado com um argumento. Este argumento pode ser *URL* para um tarball hospedado num sítio web remoto ou para um código fonte acedido por "**git clone**"; ou *PATH* local para o tarball ou código fonte.

Argumentos para as opções **-b**, **-f**, e **-w** precisam de ser citados para as proteger da shell.

Outras ferramentas também oferecem maneiras de obter o tarball do autor e criar o link simbólico necessário para compilar um pacote Debian dependendo do seu fluxo de trabalho. Por exemplo, **origtargz**, **mk-origtargz**, **git-deborig**, e **pristine-tar**.

15.4 Argumentos posicionais

URL adquirir a árvore fonte a partir do tarball, do repositório git ou da árvore fonte neste *URL* (ou *CAMINHO*) (se em falta, a árvore fonte usa o directório atual)

15.5 Opções

-h, --help mostra esta mensagem de ajuda e termina

-n, **--native** faz um pacote fonte nativo sem **.orig.tar.xz**

-p, --package pacote define o nome do pacote Debian

-u, --upstreamversion *versão* define a versão de pacote do autor ("@" na *versão* é substituído pela marca temporal "**0~aammddHHMM**")

-r, --revision *revisão* define a versão Debian do pacote ("**@""** em *revisão* é substituído pela marca temporal "**0~aammddHHMM**")

-z, --tarz *extensão* define o tipo de compressão do tarball para o tarball de autor em falta, *extensão*=(**tar.xz**|**tar.gz**|**tar.bz2**) (nome alternativo: **z**, **b**, **x**)

-b, --binaryspec *"pacote-binário[:tipo], …""* define as especificações do pacote binário por uma lista separada por vírgulas de pares *pacotebinário:tipo*. Aqui, *pacote-binário* é o nome do pacote binário, e o *tipo* opcional é escolhido a partir dos seguintes valores de *tipo*:

- **bin**: Pacote de código binário ELF compilado C/C++ (any, foreign) (predefinido, nome alternativo: **""**, isto é, **string-nula**)
- **data**: Pacote de dados (fonts, gráficos, …) (all, foreign) (nome alternativo: **da**)
- **dev**: Pacote de desenvolvimento de biblioteca (any, same) (nome alternativo: **de**)
- **doc**: Pacote de documentação (all, foreign) (nome alternativo: **do**)
- **lib**: Pacote biblioteca (any, same) (nome alternativo: **l**)
- **perl**: Pacote script Perl (all, foreign) (nome alternativo: **pl**)
- **python3**: Pacote script Python (versão 3) (all, foreign) (nome alternativo: **py3**, **python**, **py**)
- **ruby**: Pacote script Ruby (all, foreign) (nome alternativo: **rb**)
- **nodejs**: Pacote JavaScript baseado em Node.js (all, foreign) (nome alternativo: **js**)
- **script**: Pacote script shell e outra linguagem interpretada (all, foreign) (nome alternativo: **sh**)

O par de valores dentro de parênteses, tais como (any, foreign), são os valores de estrofe **Arquitectura** e **Multi-Arch** definidos no ficheiro **debian/control**. Em muitos casos, o comando **debmake** consegue adivinhar bem o *tipo* de *pacotebinário*. Se *tipo* não for óbvio, *tipo* é definido para **bin**.

Aqui estão exemplos para cenários típicos de divisão de pacote binário onde o nome do pacote fonte Debian do autor é **foo**:

- Gerando um pacote binário executável **foo**:
  - "**-b'foo:bin'**", ou seu formato curto "**-b'-'**", ou nenhuma opção **-b**
- Gerando um pacote binário executável (python3) **python3-foo**:
  - "**-b'python3-foo:py'**", ou o seu formato curto "**-b'python3-foo'**"
- Gerando um pacote de dados **foo**:
  - "**-b'foo:data'**", ou seu formato curto "**-b'-:data'**"
- Gerando um pacote binário executável **foo** e um de documentação **foo-doc**:
  - "**-b'foo:bin,foo-doc:doc'**", ou seu formato curto "**-b'-:-doc'**"

- Gerando um pacote binário executável **foo**, um pacote biblioteca **libfoo1**, e um pacote de desenvolvimento de biblioteca **libfoo-dev**:
  - "**-b'foo:bin,libfoo1:lib,libfoo-dev:dev'**" ou seu formato curto "**-b'-,libfoo1,libfoo-dev'**"

Se o conteúdo da árvore fonte não corresponder à definição para *tipo*, o comando **debmake** avisa-o.

-e, --email *foo@example.org* define o endereço de e-mail

A predefinição é obtida a partir do valor da variável de ambiente **$DEBEMAIL**.

-D, --debug *valor* define a variável de ambiente **DEBUG** para *valor* para registo de depuração (sub-string de "**spPd**", use "_" para desconfigurar **DEBUG**)

-f, --fullname "*primeiro-nome último-nome*" define o nome completo

A predefinição é obtida a partir do valor da variável de ambiente **$DEBFULLNAME**.

-i, --invoke *[debuild|sbuild|dgit sbuild|gbp buildpackage|dpkg-buildpackage| …]* invoca a ferramenta de compilação do pacote

-m, --monoarch força pacotes a serem não-multi-arquitectura

-q, --quitearly termina cedo antes de criar ficheiros no directório debian

-v, --version mostra informação de versão

-V, --verbose usa --verbose para comandos de shell se disponível

-w, --with *"addon …"* define argumentos de opção adicionais "**dh --with**" em **debian/rules**

Para pacotes baseados em Autotools, se eles instalarem programas Python (versão 3), definir **python3** como *addon* para o argumento de comando **debmake** é necessário pois isto não é óbvio. Mas para pacotes baseados em Python **pyproject.toml**, definir **python3** como *addon* para o argumento de comando **debmake** não é necessário pois isto é óbvio e o comando **debmake** define-o automaticamente para o comando **dh**(1).

-x, --extra *[01234]* gera ficheiros de configuração extra como modelos (predefinição: 2)

Por favor note que **debian/changelog**, **debian/control**, **debian/copyright**, **debian/rules**, e **debian/source/format** são ficheiros de configuração necessários para compilar um pacote binário Debian moderno.

O número determina quais modelos de configuração são gerados.

- **-x0**: todos os 5 ficheiros modelo de configuração requeridos. (opção selecionada se alguns destes ficheiros necessários já existir)
- **-x1**: todos os ficheiros **-x0** + ficheiros modelo de configuração desejáveis com suportes a tipos de pacote binário.
- **-x2**: todos os ficheiros **-x1** + ficheiros modelo de configuração normal com suporte a script de maintainer. (predefinido)
- **-x3**: todos os ficheiros **-x2** + ficheiros modelo de configuração opcionais.
- **-x4**: todos os ficheiros **-x3** + ficheiros modelo de configuração obsoletos.

Alguns ficheiros modelo de configuração são gerados com um sufixo extra **.ex** para facilitar a sua remoção. Para ativar isto, renome os seus nomes de ficheiro para aqueles sem o sufixo **.ex** e edite os seus conteúdos. Os ficheiros de configuração nunca são sobrescritos. Se desejar actualizar alguns dos ficheiros de configuração existentes, or favor renome-os antes de correr o comando **debmake** e junte manualmente os ficheiros de configuração gerados com os antigos que foram renomeados.

-y, --yes use uma vez para "forçar sim" em todas as perguntas, duas vezes para "forçar não"

-B, --backup Mantém os editados pelo utilizador sem o sufixo **.ex** e cria ficheiros modelo com sufixo **.ex**

15.6 EXEMPLOS

Para uma fonte bem comportada, você pode compilar um pacote binário Debian singular instalável e bom-para-uso-local facilmente com um comando. A instalação de teste de tal pacote gerado deste modo oferece uma boa alternativa ao comando tradicional "**make install**" que instala no directório **/usr/local** pois o pacote Debian pode ser removido completamente pelo comando "**dpkg -P** '…'". Aqui estão alguns exemplos de como compilar tais pacotes de teste.

Para uma árvore fonte de programa C típico empacotada com **autoconf/automake**:

- **debmake -i sbuild**

Para uma árvore fonte de módulo Python (versão 3) típica:

- **debmake -b":python3" -i sbuild**

Para um módulo Python (version 3) típico no arquivo *pacote-versão***.tar.xz**:

- **debmake** *pacote-versão***.tar.xz -b":python3" -i sbuild**

Para um módulo Perl típico no arquivo *pacote-versão***.tar.xz**:

- **debmake** *pacote-versão***.tar.xz -b":perl" -i sbuild**

15.7 PACOTES DE AJUDA

O empacotamento pode requerer a instalação de alguns pacotes de ajuda especial adicional.

- Os programas Python (versão 3) podem requerem o pacote **pybuild-plugin-pyproject**.
- O sistema de compilação Autotools (**autoconf** + **automake**) pode requerer o pacote **autotools-dev** ou **dh-autoreconf**.
- Os programas Ruby podem requerer o pacote **gem2deb**.
- Os programas Node.js baseados em JavaScript podem requerer o pacote **pkg-js-tools**.
- Os programas Java podem requerer o pacote **javahelper**.
- Os programas do Gnome podem requerer o pacote **gobject-introspection**.
- etc.

15.8 ADVERTÊNCIA

Apesar do **debmake** destinar-se a fornecer ficheiros modelo para o maintainer do pacote trabalhar, as atividades actuais de empacotamento são muitas vezes executadas sem se usar o **debmake** enquanto se referenciam apenas pacotes existentes semelhantes e o "Manual de Política Debian". É requerido que todos os ficheiros modelo gerados pelo **debmake** sejam modificados manualmente.

Existem alguns pontos para o **debmake**:

- **debmake** ajuda a escrever um tutorial de empacotamento conciso "Guia para Maintainers Debian" (pacote **debmake-doc**).
- **debmake** fornece textos de licença curtos extraídos como **debian/copyright** em precisão decente para ajudar na revisão da licença.
- "Guia para Maintainers Debian" também serve como tutorial com exemplos para o uso do debmake.
- **debmake** internamente chama **licensecheck** do pacote **licensecheck** para criar **debian/copyright** se este não existir.
- **debmake** internamente chama **lrc** do pacote **licenserecon** para verificar **debian/copyright** se este já existe.

Existem algumas limitações para quais caracteres podem ser usados como parte do pacote Debian. A limitação mais notável é a proibição de letras maiúsculas no nome do pacote. Aqui está um sumário como um conjunto de expressões regulares:

- Nome de pacote do autor (**-p**): `[-+.a-z0-9]{2,}`
- Nome de pacote binário (**-b**): `[-+.a-z0-9]{2,}`
- Versão do autor (**-u**): `[0-9][-+.:~a-z0-9A-Z]*`
- Revisão Debian (**-r**): `[0-9][+.~a-z0-9A-Z]*`

Veja a definição exacta em "Capítulo 5 - Ficheiros de controle e seus campos" no "Manual de Política Debian".

O **debmake** assume casos de empacotamento relativamente simples. Assim todos programas relacionados com o interpretador são assumidos como sendo "**Architecture: all**". Isto nem sempre é verdade.

15.9 DEBUG

Por favor reporte bugs ao pacote **debmake** usando o comando **reportbug**.

O caractere definido na variável de ambiente **$DEBUG** determina o nível de resultados no relatório.

- **s**: registo de progresso do programa
- **p**: chave para[..] registo de valor
- **P**: todos para[..] registo de valor
- **d**: para["debs"] registo de valor

Use esta funcionalidade como:

```
[base_dir] $ export DEBUG=spd; debmake ...
```

ou

```
[base_dir] $ debmake -D spd ...
```

Veja **README.md** na fonte para mais.

15.10 AUTOR

Copyright © 2014-2026 Osamu Aoki <osamu@debian.org>

15.11 LICENÇA

Licença Expat

15.12 VEJA TAMBÉM

O pacote **debmake-doc** fornece o "Guia para Maintainers Debian" em formato de texto simples, HTML e PDF sob o directório **/usr/share/doc/debmake-doc/**.

Veja também os manuais **licensecheck**(1), **lrc**(1), **dpkg-source**(1), **deb-control**(5), **debhelper**(7), **dh**(1), **dpkg-buildpackage**(1), **debuild**(1), **quilt**(1), **dpkg-depcheck**(1), **sbuild**(1), **gbp-buildpackage**(1), e **gbp-pq**(1).

Capítulo 16

opções do debmake

Aqui estão algumas explicações adicionais para as opções do **debmake**.

16.1 Opção curta (-i)

O comando **debmake** oferece uma opção curta de atalho.

- **-i** : executa script para compilar o pacote binário

O exemplo em cima “Capítulo 5” pode ser feito simplesmente como se segue.

```
[base_dir] $ debmake package-1.0.tar.xz -i debuild
```

Dica



Pode ser usado como um argumento um URL como “**https://www.example.org/DL/package-1.0.tar.xz**” para um tarball, “**https://github.com/username/package.git**” para um repositório git, ou “**/caminho/para/directório-fonte/**” para uma árvore fonte local..

16.2 debmake -b

O comando **debmake** com a opção **-b** fornece um método intuitivo e flexível de criar o ficheiro modelo inicial **debian/control**. Este ficheiro define a divisão dos pacotes binário Debian com as seguintes estrofes:

- **Package:**
- **Architecture:** (ex. **amd64**)
- **Multi-Arch:** (veja “Secção 10.10”)
- **Depends:**
- **Pre-Depends:**

O comando **debmake** também define um conjunto apropriado de substvars (variáveis de substituição) usado em cada estrofe de dependência pertinente.

Vamos citar a parte pertinente do manual do **debmake** aqui.

-b, --binaryspec *”pacote-binário[:tipo], …””* define as especificações do pacote binário por uma lista separada por vírgulas de pares *pacotebinário:tipo*. Aqui, *pacote-binário* é o nome do pacote binário, e o *tipo* opcional é escolhido a partir dos seguintes valores de *tipo*:

- **bin**: Pacote de código binário ELF compilado C/C++ (any, foreign) (predefinido, nome alternativo: **””**, isto é, **string-nula**)
- **data**: Pacote de dados (fonts, gráficos, …) (all, foreign) (nome alternativo: **da**)
- **dev**: Pacote de desenvolvimento de biblioteca (any, same) (nome alternativo: **de**)
- **doc**: Pacote de documentação (all, foreign) (nome alternativo: **do**)
- **lib**: Pacote biblioteca (any, same) (nome alternativo: **l**)
- **perl**: Pacote script Perl (all, foreign) (nome alternativo: **pl**)
- **python3**: Pacote script Python (versão 3) (all, foreign) (nome alternativo: **py3**, **python**, **py**)
- **ruby**: Pacote script Ruby (all, foreign) (nome alternativo: **rb**)
- **nodejs**: Pacote JavaScript baseado em Node.js (all, foreign) (nome alternativo: **js**)
- **script**: Pacote script shell e outra linguagem interpretada (all, foreign) (nome alternativo: **sh**)

O par de valores dentro de parênteses, tais como (any, foreign), são os valores de estrofe **Arquitectura** e **Multi-Arch** definidos no ficheiro **debian/control**. Em muitos casos, o comando **debmake** consegue adivinhar bem o *tipo* de *pacotebinário*. Se *tipo* não for óbvio, *tipo* é definido para **bin**.

Aqui estão exemplos para cenários típicos de divisão de pacote binário onde o nome do pacote fonte Debian do autor é **foo**:

- Gerando um pacote binário executável **foo**:
  - “**-b’foo:bin’**”, ou seu formato curto “**-b’-’**”, ou nenhuma opção **-b**
- Gerando um pacote binário executável (python3) **python3-foo**:
  - “**-b’python3-foo:py’**”, ou o seu formato curto “**-b’python3-foo’**”
- Gerando um pacote de dados **foo**:
  - “**-b’foo:data’**”, ou seu formato curto “**-b’-:data’**”
- Gerando um pacote binário executável **foo** e um de documentação **foo-doc**:
  - “**-b’foo:bin,foo-doc:doc’**”, ou seu formato curto “**-b’-:-doc’**”
- Gerando um pacote binário executável **foo**, um pacote biblioteca **libfoo1**, e um pacote de desenvolvimento de biblioteca **libfoo-dev**:
  - “**-b’foo:bin,libfoo1:lib,libfoo-dev:dev’**” ou seu formato curto “**-b’-,libfoo1,libfoo-dev’**”

Se o conteúdo da árvore fonte não corresponder à definição para *tipo*, o comando **debmake** avisa-o.

16.3 debmake -B

O comando **debmake** invocado com a opção **-B** pode gerar ficheiros modelo com sufixo **.ex**. Isto é útil se você deseja comparar os ficheiros modelo auto-gerados com os existentes.

16.4 debmake -x

A quantidade de ficheiros modelo gerados pelo comando **debmake** depende da opção **-x**[01234].

- Veja “Secção 14.1” para cherry-picking dos ficheiros modelo.

Nota



Nenhum dos ficheiros de configuração existentes são modificados pelo comando **debmake**.