debian

Debian 维护者指南

青木修、杨博远、Fonzie Huang 和 xiao sheng wen(肖盛文)

February 5, 2025

Debian 维护者指南
by 青木修、杨博远、Fonzie Huang 和 xiao sheng wen(肖盛文)

本指南在撰写过程中参考了以下几篇文档：

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

本指南的最新版本应当可以在下列位置找到：

- 在 debmake-doc 软件包 中，以及
- 位于 Debian 文档网站。

Contents

Abstract

本篇《Debian 维护者指南》（2025-02-05）教程文档面向普通 Debian 用户和未来的开发者，描述了使用 **debmake** 命令构建 Debian 软件包的方法。

本指南注重描述现代的打包风格，同时提供了许多简单的示例。

- POSIX shell 脚本打包
- Python3 脚本打包
- C 和 Makefile/Autotools/CMake
- 含有共享库的多个二进制软件包的打包，等等。

本篇《Debian 维护者指南》可看作《Debian 新维护者手册》的继承文档。

Chapter 1

前言

如果您在某些方面算得上是有经验的 Debian 用户 1 的话，您可能遇上过这样的情况：

- 您想要安装某一个软件包，但是该软件在 Debian 仓库中尚不存在。
- 您想要将一个 Debian 软件包更新为上游的新版本。
- 您想要添加某些补丁来修复某个 Debian 软件包中的缺陷。

如果您想要创建一个 Debian 软件包来满足这些需求，并将您的工作与社区分享，那么您便是本篇指南的目标读者，即未来的 Debian 维护者。2 欢迎来到 Debian 社区。

Debian 是一个大型的、历史悠久的志愿者组织。因此，它具有许多需要遵守的社交上和技术上的规则和惯例。Debian 也开发出了一长串的打包工具和仓库维护工具，用来构建一套能够解决各种技术目标的二进制软件包：

- 软件包具有清晰明了的依赖关系和补丁，并可以在干净的构建环境下从头开始正确构建（“第 6.6 节”、“第 6.9 节”、“第 4.6 节”）
- 软件包可以跨多个架构得到构建（“第 9.3 节”）
- 构建是可重现的（“第 10.7 节”）
- 具有多架构（multiarch）支持（“第 10.10 节”）
- 可以在新硬件架构下自举（“第 10.5 节”）
- 构建能够使用特定的编译选项增强安全性（“第 10.6 节”）
- 软件按照最佳实践拆分为多个二进制软件包（“第 10.11 节”）
- 程序库的名称和内容得到有效管理，并确保升级时的平滑迁移（“第 10.18 节”）
- 如有必要，能够正确使用交互式命令行进行安装（“第 10.22 节”）
- 利用持续集成保证质量（“第 10.4 节”）
- ……

这些目标对很多新近参与工作的潜在 Debian 维护者来说可能无法立刻全部掌握。本篇指南旨在提供一个着手点，方便读者开展工作。它具体描述了以下内容：

- 作为 Debian 未来的维护者，您在参与 Debian 工作之前应该了解的东西。
- 制作一个简单的 Debian 软件包大概流程如何。
- 制作 Debian 软件包时有哪些规则。
- 省心省力制作 Debian 软件包的小窍门。

1您需要对 Unix 编程有所了解，但不必一定是这方面的专家。在“Debian 参考手册”中，您可以了解到使用 Debian 系统的一些基本操作和关于 Unix 编程的一些指引。

2如果您对分享 Debian 软件包不感兴趣，您可以对上游包含修复内容的源码包进行编译并安装至 **/usr/local** 来满足本地的需求。

- 在某些典型场景下制作 Debian 软件包的示例。

作者在更新原有的使用 **dh-make** 软件包的“新维护者手册”时认识到了该文档的局限性。因此，作者决定创建一个替代工具并编写其对应的文档以解决多架构支持（multi-arch）等现代的需求。其成果便是 **debmake** 软件包（最初版本为 2013 年发布的 4.0 版，当前版本：4.5.1），以及这篇更新的、由 **debmake-doc** 软件包（当前版本：1.22-1）提供的“Debian 维护者指南”。（在 2016 年，**dh-make** 从 Perl 迁移到了 Python 语言并提供了更新的特性。）

许多杂项事务和小提示都被集成进了 **debmake** 命令，以使本指南内容简单易懂。本指南同时提供了许多打包示例来帮助用户上手使用。

小心

合适地创建并维护 Debian 软件包需要占用许多时间。Debian 维护者在接受这项挑战时一定要确保 既能精通技术又能勤勉投入精力。

某些重要的主题会详细进行说明。其中某些可能看起来和您没什么关系。请保持耐心。某些边缘情况会被跳过，一些主题仅以外部引用提及。这些都是有意的行文安排，目标是让这份指南保持简单而可维护。

Chapter 2

概览

对 *package-1.0*.**tar.gz**，一个包含了简单的、符合“GNU 编码标准”和“FHS（文件系统层级规范）”的 C 语言源代码的程序来说，它在 Debian 下打包工作可以按照下列流程，使用 **debmake** 命令进行。

```
$ tar -xvzf package-1.0.tar.gz
$ cd package-1.0
$ debmake
  ... Make manual adjustments of generated configuration files
$ debuild
```

如果跳过了对生成的配置文件的手工调整流程，则最终生成的二进制软件包将缺少有意义的软件包描述信息，但是仍然能为 **dpkg** 命令所使用，在本地部署环境下正常工作。

> 小心
>
> 这里的 **debmake** 命令只提供一些不错的模板文件。如果生成的软件包需要发布出去供公众使用的话，这些模板文件必须手工调整至最佳状态以遵从 Debian 仓库的严格质量标准。

如果您在 Debian 打包方面还是个新手的话，此时不要过多在意细节问题，请先专注于理解整体的流程。

If you are familiar with Debian packaging, you'll notice that **debmake** is similar to the **dh_make** command. This is because **debmake** is designed to replace the functionality historically provided by **dh_make**. 1

debmake 命令设计提供如下特性与功能：

- 现代的打包风格
  - **debian/copyright**：遵循“DEP-5”规范
  - **debian/control**：**substvar** 支持、**multiarch** 支持、生成多个二进制软件包、……
  - **debian/rules**：**dh** 语法、编译器加固选项、……
- 灵活性
  - 许多选项（参见“第 16.2 节”、“第 15 章”和“第 16 章”）
- 合理的默认行为
  - 执行过程不中断，输出干净的结果
  - 生成多架构支持（multiarch）的软件包，除非明确指定了 **-m** 选项。
  - 生成非原生 Debian 软件包，使用“**3.0 (quilt)**”格式，除非明确指定了 **-n** 选项。
- 额外的功能

1Before **dh_make**, the **deb-make** command was popular. The current **debmake** package starts its version from **4.0** to avoid version conflicts with the obsolete **debmake** package, which provided the “**deb-make**”command.

– 根据当前源代码对 **debian/copyright** 文件进行验证（请见“第 16.6 节”）

The **debmake** command delegates most of the heavy lifting to its back-end packages: **debhelper**, **dpkg-dev**, **devscripts**, **sbuild**, **schroot**, etc.

提示

Ensure that you properly quote the arguments of the **-b**, **-f**, **-l**, and **-w** options to protect them from shell interference.

提示

非原生软件包是标准的 Debian 软件包。

提示

本文档中所有软件包构建示例的详细日志可以由“第 14.14 节”一段给出的操作来获取。

注意

The generation of the **debian/copyright** file, and the outputs from the **-c** (see “第 16.3 节”) and **-k** (see “第 16.6 节”) options involve heuristic operations on the copyright and license information. They may produce some erroneous results.

Chapter 3

预备知识

这里给出您在投入 Debian 相关工作之前应当理解掌握的一些必备的预备知识。

3.1 Debian 社区的工作者

在 Debian 社区中有这几类常见的角色：

- 上游作者（**upstream author**）：程序的原始作者。
- 上游维护者（**upstream maintainer**）：目前在上游维护程序代码的人。
- 软件包维护者（**maintainer**）：制作并维护该程序 Debian 软件包的人。
- 赞助者（**sponsor**）：帮助维护者上传软件包到 Debian 官方仓库的人（在通过内容检查之后）。
- 导师（**mentor**）：帮助新手维护者熟悉和深入打包的人。
- **Debian** 开发者（DD, Debian Developer）：Debian 社区的官方成员。DD 拥有向 Debian 官方仓库上传的全部权限。
- **Debian** 维护者（Debian Maintainer, DM）：拥有对 Debian 官方仓库部分上传权限的人。

Please note that you can't become an official **Debian Developer** (DD) overnight, as it requires more than just technical skills. Don't be discouraged by this. If your work is useful to others, you can still upload your package either as a **maintainer** through a **sponsor** or as a **Debian Maintainer**.

Please note that you don't need to create new packages to become an official Debian Developer. Contributing to existing packages can also provide a path to becoming an official Debian Developer. There are many packages waiting for good maintainers (see "`" 第 3.8 节"`").

3.2 如何做出贡献

请参考下列文档来了解应当如何为 Debian 作出贡献：

- "您如何协助 Debian？"（官方）
- "The Debian GNU/Linux 常见问题，第 13 章 - "向 Debian 计划作出贡献""（半官方）
- "Debian Wiki, HelpDebian"（补充内容）
- "Debian 新成员站点"（官方）
- "Debian Mentors FAQ"（补充内容）

3.3 Debian 的社会驱动力

为做好准备和 Debian 进行交互，请理解 Debian 的社会动力学：

- 我们都是志愿者。
  - 任何人都不能把事情强加给他人。
  - 您应该主动地做自己要做的事情。
- 友好的合作是我们前行的动力。
  - 您的贡献不应致使他人增加负担。
  - 只有当别人欣赏和感激您的贡献时，它才有真正的价值。
- Debian 并不是一所学校，在这里没有所谓的老师会自动地注意到您。
  - 您需要有独立学习大量知识和技能的能力。
  - 其他志愿者的关注是比较稀缺的资源。
- Debian 一直在不断进步。
  - Debian 期望您制作出高质量的软件包。
  - 您应该随时调整自己来适应变化。

在这篇指南之后的部分中，我们只关注打包的技术方面。因此，请参考下面的文档来理解 Debian 的社会动力学：

- “Debian：17 年的自由软件、“实干主义”、和民主”（前任 DPL 制作的介绍性幻灯片）

3.4 技术提醒

这里给出一些技术上的建议，参考行事可以让您与其他维护者共同维护软件包时变得更加轻松有效，从而让 Debian 项目的输出成果最大化。

- 让您的软件包容易除错（debug）。
  - 保持您的软件包简单易懂。
  - 不要对软件包过度设计。
- 让您的软件包拥有良好的文档记录。
  - 使用可读的代码风格。
  - 在代码中写注释。
  - 格式化代码使其风格一致。
  - 维护软件包的 git 仓库 [1]。

注意

对软件进行除错（debug）通常会比编写初始可用的软件花费更多的时间。

即使是开发系统，在不稳定（**unstable**）套件下运行基本系统也是不明智的。

- Creation and verification of binary **deb** packages should use a minimal **unstable** chroot as described in “第 4.6 节”.

[1] 绝大多数 Debian 维护者使用 **git** 而非其它版本控制系统，如 **hg**、**bzr** 等等。

- Basic interactive package development activities should use an **unstable** chroot as described in “第 4.7 节”.

注意

Advanced package development activities, such as testing full Desktop systems, network daemons, and system installer packages, should use the **unstable** suite running under “virtualization”.

3.5 Debian 文档

Please make yourself ready to read the pertinent part of the latest Debian documentation to generate perfect Debian packages:

- “Debian 政策手册”
  - 正式的，“必须遵循”的规则 (https://www.debian.org/doc/devel-manuals#policy)
- “Debian 开发者参考”
  - 官方的“最佳实践”文档（https://www.debian.org/doc/devel-manuals#devref）
- “Debian 维护者指南”——即本指南
  - A “tutorial reference”document (https://www.debian.org/doc/devel-manuals#debmake-doc)

All these documents are published on https://www.debian.org using the **unstable** suite versions of corresponding Debian packages. If you wish to have local access to all these documents from your base system, please consider using techniques such as “apt-pinning”and “chroot”.

如果本指南文档的内容与官方的 Debian 文档有所冲突，那么官方的那些总是对的。请使用 **reportbug** 工具向 **debmake-doc** 软件包报告问题。

这里有一些替代性的教程文档，您可以与本指南一起阅读进行参考：

- “《Debian 打包教程》”
  - https://www.debian.org/doc/devel-manuals#packaging-tutorial
  - https://packages.qa.debian.org/p/packaging-tutorial.html
- “Ubuntu 打包指南”（Ubuntu 基于 Debian。）
  - http://packaging.ubuntu.com/html/
- “Debian 新维护者指南”（本文档的前身，已弃用）
  - https://www.debian.org/doc/devel-manuals#maint-guide
  - https://packages.qa.debian.org/m/maint-guide.html

提示

When reading these, you may consider using the **debmake** command in place of the **dh_make** command.

3.6 帮助资源

Before deciding to ask your question in a public forum, please do your part by reading the relevant documentation:

- 软件包的信息可以使用 **aptitude**、**apt-cache** 以及 **dpkg** 命令进行查看。
- 所有相关软件包在 **/usr/share/doc/**软件包名目录下的文件。
- 所有相关命令在 **man** 命令下输出的内容。
- 所有相关命令在 **info** 命令下输出的内容。
- “debian-mentors@lists.debian.org 邮件列表存档”的内容。
- “debian-devel@lists.debian.org 邮件列表存档”的内容。

You can find your desired information effectively by using a well-formed search string such as ”keyword site:lists.debian.org” to limit the search domain of the web search engine.

Creating a small test package is a good way to learn the details of packaging. Inspecting existing well-maintained packages is the best way to learn how other people make packages.

如果您对打包仍然存在疑问，您可以使用以下方式与他人进行沟通：

- debian-mentors@lists.debian.org mailing list. (This mailing list is for the novice.)
- debian-devel@lists.debian.org mailing list. (This mailing list is for the expert.)
- IRC such as #debian-mentors.
- 专注某个特定软件包集合的团队。（完整列表请见 https://wiki.debian.org/Teams）
- 特定语言的邮件列表。
  - “debian-devel-{french,italian,portuguese,spanish}@lists.debian.org”
  - “debian-chinese-gb@lists.debian.org”（该邮件列表用于一般的（简体）中文讨论。）
  - “debian-devel@debian.or.jp”

More experienced Debian developers will gladly help you if you ask properly after making the required efforts.

小心

Debian development is a moving target. Some information found on the web may be outdated, incorrect, or non-applicable. Please use such information carefully.

3.7 仓库状况

请了解 Debian 仓库的当前状况。

- Debian 已经包含了绝大多数种类程序的软件包。
- Debian 仓库内软件包的数量是活跃维护者的数十倍。
- 遗憾的是，某些软件包缺乏维护者的足够关注。

因此，对已经存在于仓库内的软件包做出贡献是十分欢迎的（这也更有可能得到其他维护者的支持和协助上传）。

提示

The **wnpp-alert** command from the **devscripts** package can check for installed packages that are up for adoption or orphaned.

提示

The **how-can-i-help** package can show opportunities for contributing to Debian based on packages installed locally.

3.8 贡献流程

这里使用类 Python 伪代码，给出了向 Debian 贡献名为 **program** 的软件所走的贡献流程：

```
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)
```

其中：

- 对 exist_in_debian() 和 is_team_maintained()，需检查：
  - **aptitude** 命令

  - “Debian 软件包”网页
  - Debian 维基“团队”页面
- 对 is_orphaned()、is_RFA() 和 is_ITPed_by_others()，需检查：
  - **wnpp-alert** 命令的输出。
  - “需要投入精力和未来的软件包（WNPP）”
  - “Debian 缺陷报告记录：在 unstable 版本中 wnpp 伪软件包的缺陷记录”
  - “需要“关爱”的 Debian 软件包”
  - “基于 debtags 浏览 wnpp 缺陷记录”
- 对于 is_good_program()，请检查：
  - 这个程序应当有用。
  - 这个程序不应当向 Debian 系统引入安全和维护上的问题。
  - 这个程序应当有良好的文档，其源代码需要可被理解（即，未经混淆）。
  - 这个程序的作者同意软件被打包，且对 Debian 态度友好。2
- 对 is_it_DFSG()，及 is_its_dependency_DFSG()，请检查：
  - “Debian 自由软件指导方针”（DFSG）。
- 对 is_it_distributable()，请检查：
  - 该软件必须有一个许可证，其中应当允许软件被发行。

You either need to file an **ITP** or adopt a package to start working on it. See the “Debian Developer’s Reference”:

- “5.1. 新软件包”。
- “5.9. 移动、删除、重命名、丢弃、接手和重新引入软件包”。

3.9 新手贡献者和维护者

新手贡献者和维护者可能想知道在开始向 Debian 进行贡献之前需要事先学习哪些知识。根据您个人的侧重点不同，下面有我的一些建议供您参考：

- 打包
  - **POSIX shell** 和 **make** 的基本知识。
  - 一些 **Perl** 和 **Python** 的入门知识。
- 翻译
  - 基于 PO 的翻译系统的工作原理和基本知识。
- 文档
  - 文本标记语言的基础知识（XML、ReST、Wiki 等）。

新手贡献者和维护者可能想知道从哪里开始向 Debian 进行贡献。根据您掌握的技能，下面有我的一些建议供您参考：

- **POSIX shell**、**Perl** 和 **Python** 的技巧：
  - 向 Debian 安装器提交补丁。
  - Send patches to the Debian packaging helper scripts such as **devscripts**, **sbuild**, **schroot**, etc. mentioned in this document.

2这一条不是绝对的要求，但请注意：遇上不友好的上游可能需要大家为此投入大量精力，而一个友好的上游则能协助解决程序的各类问题。

- **C** 和 **C++** 技能：
  - 向具有 **required** 和 **important** 优先级的软件包提交补丁。
- 英语之外的技能：
  - 向 Debian 安装器项目提交补丁。
  - 为具有 **required** 和 **important** 优先级的软件包中的 PO 文件提交补丁。
- 文档技能：
  - 更新“Debian 维基（Wiki）”中的内容。
  - 向已有的“Debian 文档”提交补丁。

这些活动应当能让您在各位 Debian 社区成员之间得到存在感，从而建立您的信誉与名气。
新手维护者应当避免打包具有潜在高度安全隐患的程序：

- **setuid** 或 **setgid** 程序
- 守护进程（**daemon**）程序
- 安装至 **/sbin/** 或 **/usr/sbin/** 目录的程序

在积累足够的打包经验后，您可以再尝试打包这样的程序。

Chapter 4

工具的配置

build-essential 软件包必须在构建环境内预先安装。

The **devscripts** package should be installed in the development environment of the maintainer.

It is a good idea to install and set up all of the popular set of packages mentioned in this chapter. These enable us to share the common baseline working environment, although these are not necessarily absolute requirements.

Please also consider to install the tools mentioned in the “Overview of Debian Maintainer Tools”in the “Debian Developer’s Reference”, as needed.

小心

这里展示的工具配置方式仅作为示例提供，可能与系统上最新的软件包相比有所落后。Debian 的开发具有一个移动的目标。请确保阅读合适的文档并按照需要更新配置内容。

4.1 Email setup

许多 Debian 维护工具识别并使用 shell 环境变量 **$DEBEMAIL** 和 **$DEBFULLNAME** 作为作为您的电子邮件地址和名称。

Let’s set these environment variables by adding the following lines to **~/.bashrc** 1.

添加至 **~/.bashrc** 文件

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

注意

上面的例子使用了本指南作者的个人信息作为示例。本指南展示的配置和操作实例都将使用这里的电子邮件地址和名称设置。在您的系统上，您必须使用您自己的电子邮件地址和姓名。

1这里假设您正在使用 Bash 并以此作为登录默认 shell。如果您设置了其它登录 shell，例如 Z shell，请使用它们对应的配置文件替换 *~/.bashrc* 文件。

4.2 mc 设置

mc 命令提供了管理文件的简单途径。它可以打开二进制 **deb** 文件，并仅需对二进制 **deb** 文件按下回车键便能检查其内容。它调用了 **dpkg-deb** 命令作为其后端。我们可以按照下列方式对其配置，以支持简易 **chdir** 操作。

添加至 **~/.bashrc** 文件

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

4.3 git 设置

如今 **git** 命令已成为管理带历史的源码树的必要工具。

git 命令的用户级全局配置，如您的名字和电子邮件地址，保存在 **~/.gitconfig** 文件中，且可以使用如下方式配置。

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

如果您仍然只习惯 CVS 或者 Subversion 的命令风格，您可以使用如下方式设置几个命令别名。

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

您可以使用如下命令检查全局配置。

```
$ git config --global --list
```

提示

有必要使用某些图形界面 git 工具，例如 **gitk** 或 **gitg** 命令来有效地处理 git 仓库的历史。

4.4 quilt 设置

quilt 命令提供了记录修改的一个基本方式。对 Debian 打包来说，该工具需要进行自定义，从而在 **debian/patches/** 目录内记录修改内容，而非使用默认的 **patches/** 目录。

为了避免改变 **quilt** 命令自身的行为，我们在这里创建一个用于 Debian 打包工作的命令别名：**dquilt**。之后，我们将对应内容写入 **~/.bashrc** 文件。下面给出的第二行为 **dquilt** 命令提供与 **quilt** 命令相同的命令行补全功能。

添加至 **~/.bashrc** 文件

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

然后我们来创建具有如下内容的 **~/.quiltrc-dpkg** 文件。

```
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
```

See **quilt**(1) and "How To Survive With Many Patches or Introduction to Quilt (**quilt.html**)"on how to use the **quilt** command.

要获取使用示例，请查看"第 5.9 节"。

Note that "gbp pq"is able to consume existing **debian/patches**, automate updating and modifying the patches, and export them back into **debian/patches**, all without using quilt nor the need to learn or configure quilt.

4.5 devscripts 设置

debsign 命令由 **devscripts** 软件包提供，它可以使用用户的 GPG 私钥对 Debian 软件包进行签名。

debuild 命令同样由 **devscripts** 软件包提供，它可以构建二进制软件包并使用 **lintian** 命令对其进行检查。**lintian** 命令的详细输出通常都很实用。

您可以将下列内容写入 **~/.devscripts** 文件来进行配置。

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

The **-i** and **-I** options in **DEBUILD_DPKG_BUILDPACKAGE_OPTS** for the **dpkg-source** command help rebuilding of Debian packages without extraneous contents (see "第 8 章").

当前情况下，使用 4096 位的 RSA 密钥是较好的做法。另见"创建一个新 GPG 密钥"。

4.6 sbuild 设置

The **sbuild** package provides a clean room ("chroot") build environment. It offers this efficiently with the help of **schroot** using the bind-mount feature of the modern Linux kernel.

Since it is the same build environment as the Debian's buildd infrastructure, it is always up to date and comes full of useful features.

It can be customized to offer following features:

- The **schroot** package to boost the chroot creation speed.
- **lintian** 软件包能找到所构建软件包中的缺陷。
- The **piuparts** package to find bugs in the package.
- The **autopkgtest** package to find bugs in the package.
- **ccache** 软件包可以加速 **gcc**。（可选）
- **libeatmydata1** 软件包可以加速 **dpkg**。（可选）
- 并行运行 **make** 以提高构建速度。（可选）

Let's set up **sbuild** environment 2:

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

Let's update your group membership to include **sbuild** and verify it:

2Be careful since some older HOWTOs may use different chroot setups.

```
$ newgrp -
$ id
uid=1000(<yourname>) gid=1000(<yourname>) groups=...,132(sbuild)
```

Here, “reboot of system”or “**kill -TERM -1**”can be used instead to update your group membership 3 .

Let’s create the configuration file **~/.sbuildrc** in line with recent Debian practice of “source-only-upload”as:

```
cat >~/.sbuildrc << '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
```

注意

There are some exceptional cases such as NEW uploads, uploads with NEW binary packages, and security uploads where you can’t do source-only-upload but are required to upload with binary packages. The above configuration needs to be adjusted for those exceptional cases.

Following document assumes that **sbuild** is configured this way.

Edit this to your needs. Post-build tests can be turned on and off by assigning 1 or 0 to the corresponding variables,

警告

可选的自定义项可能造成负面影响。如果有疑问，请禁用它们。

3Simply “logout and login under some modern GUI Desktop environment”may not update your group membership.

注意

并行的 **make** 可能在某些已有软件包上运行失败，它同样会使得构建日志难以阅读。

提示

Many **sbuild** related hints are available at “第 9.7 节”and “https://wiki.debian.org/sbuild”.

4.7 Persistent chroot setup

注意

Use of independent copied chroot filesystem prevents contaminating the source chroot used by **sbuild**.

For building new experimental packages or for debugging buggy packages, let’s setup dedicated persistent chroot “**source:unstable-amd64-desktop**”by:

```
$ 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-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
```

Here, **desktop** profile is used instead of **sbuild** profile. Please make sure to adjust **/etc/schroot/desktop/fstab** to make package source accessible from inside of the chroot.

You can log into this chroot “**source:unstable-amd64-desktop**”by:

```
 $ sudo schroot -c source:unstable-amd64-desktop
```

4.8 gbp 设置

The **git-buildpackage** package offers the **gbp**(1) command. Its user configuration file is **~/.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
```

4.9 HTTP 代理

您应当在本地设置 HTTP 缓存代理以节约访问 Debian 软件仓库的带宽。可以考虑以下几种选项：

- 特化的 HTTP 缓存代理，使用 **apt-cacher-ng** 软件包。
- Generic HTTP caching proxy (**squid** package) configured by **squid-deb-proxy** package

In order to use this HTTP proxy without manual configuration adjustment, it's a good idea to install either **auto-apt-proxy** or **squid-deb-proxy-client** package to everywhere.

4.10 私有 Debian 仓库

您可以使用 **reprepro** 软件包搭建私有 Debian 仓库。

4.11 虚拟机

For testing GUI application, it is a good idea to have virtual machines. Install **virt-manager** and **qemu-kvm** packages.

Use of chroot and virtual machines allows us not to update the whole host PC to the latest **unstable** suite.

4.12 本地网络中的虚拟机

如需经由本地网络轻松访问虚拟机，可以考虑安装 **avahi-utils** 来设置 DNS 多播服务的自动发现基础设施。

对所有运行中的虚拟机以及主机，我们可以使用各自的主机名加上后缀的 **.local** 来使用 SSH 互相访问。

Chapter 5

简单打包

There is an old Latin saying: “**Longum iter est per praecepta, breve et efficax per exempla**”(“It’s a long way by the rules, but short and efficient with examples”).

5.1 Packaging tarball

这里给出了从简单的 C 语言源代码创建简单的 Debian 软件包的例子，并假设上游使用了 **Makefile** 作为构建系统。

我们假设上游源码压缩包（tarball）名称为 **debhello-0.0.tar.gz**。

这一类源代码设计可以用这样的方式安装成为非系统文件：

Basics for the install from the upstream tarball

```
$ tar -xzmf debhello-0.0.tar.gz
$ cd debhello-0.0
$ make
$ make install
```

Debian packaging requires changing this “**make install**”process to install files to the target system image location instead of the normal location under **/usr/local**.

注意

在其它更加复杂的构建系统下构建 Debian 软件包的例子可以在“第 14 章”找到。

5.2 大致流程

从上游源码压缩包 **debhello-0.0.tar.gz** 构建单个非原生 Debian 软件包的大致流程可以总结如下：

- 维护者获取上游源码压缩包 **debhello-0.0.tar.gz** 并将其内容解压缩至 **debhello-0.0** 目录中。
- **debmake** 命令对上游源码树进行 debian 化（debianize），具体来说，是创建一个 **debian** 目录并仅向该目录中添加各类模板文件。
  - 名为 **debhello_0.0.orig.tar.gz** 的符号链接被创建并指向 **debhello-0.0.tar.gz** 文件。
  - 维护者须自行编辑修改模板文件。
- **debuild** 命令基于已 debian 化的源码树构建二进制软件包。
  - **debhello-0.0-1.debian.tar.xz** 将被创建，它包含了 **debian** 目录。

软件包构建的大致流程

```
$ tar -xzmf debhello-0.0.tar.gz
$ cd debhello-0.0
$ debmake
  ... manual customization
$ debuild
  ...
```

提示

The **debuild** command in this and following examples may be substituted by equivalent commands such as the **sbuild** command.

提示

如果上游源码压缩包提供了 **.tar.xz** 格式文件，请使用这样的压缩包来替代 **.tar.gz** 或 **.tar.bz2** 格式。**xz** 压缩与 **gzip** 或 **bzip2** 压缩相比提供了更好的压缩比。

5.3 什么是 debmake？

注意

Actual packaging activities are often performed manually without using **debmake** while referencing only existing similar packages and “Debian Policy Manual”.

The **debmake** command is the helper script for the Debian packaging. (“第 15 章”)

- It creates good template files for the Debian packages.
- 它总是将大多数选项的状态与参数设置为合理的默认值。
- 它能产生上游源码包，并按需创建所需的符号链接。
- 它不会覆写 **debian/** 目录下已存在的配置文件。
- 它支持多架构（**multiarch**）软件包。
- It provides short extracted license texts as **debian/copyright** in decent accuracy to help license review.

这些特性使得使用 **debmake** 进行 Debian 打包工作变得简单而现代化。

In retrospective, I created **debmake** to simplify this documentation. I consider **debmake** to be more-or-less a demonstration session generator for tutorial purpose.

The **debmake** command isn’t the only helper script to make a Debian package. If you are interested alternative packaging helper tools, please see:

- Debian wiki: “AutomaticPackagingTools” — Extensive comparison of packaging helper scripts
- Debian wiki: “CopyrightReviewTools” — Extensive comparison of copyright review helper scripts

5.4 什么是 debuild？

这里给出与 **debuild** 命令类似的一系列命令的一个汇总。

- **debian/rules** 文件定义了 Debian 二进制软件包该如何构建。
- **dpkg-buildpackge** 是构建 Debian 二进制软件包的正式命令。对于正常的二进制构建，它大体上会执行以下操作：
  - “**dpkg-source --before-build**”(apply Debian patches, unless they are already applied)
  - “**fakeroot debian/rules clean**”
  - “**dpkg-source --build**”构建 Debian 源码包）
  - “**fakeroot debian/rules build**”
  - “**fakeroot debian/rules binary**”
  - “**dpkg-genbuildinfo**”(generate a ***.buildinfo** file)
  - “**dpkg-genchanges**”(generate a ***.changes** file)
  - “**fakeroot debian/rules clean**”
  - “**dpkg-source --after-build**”(unapply Debian patches, if they are applied during **--before-build**)
  - “**debsign**”(sign the ***.dsc** and ***.changes** files)
    - 如果您按照“第 4.5 节”的说明设置了 **-us** 和 **-us** 选项的话，本步骤将会被跳过。您需要手动运行 **debsign** 命令。
- **debuild** 命令是 **dpkg-buildpackage** 命令的一个封装脚本，它可以使用合适的环境变量来构建 Debian 二进制软件包。
- The **sbuild** command is a wrapper script to build the Debian binary package under the proper chroot environment with the proper environment variables.

注意

如需了解详细内容，请见 **dpkg-buildpackage**(1)。

5.5 第一步：获取上游源代码

我们先要获取上游源代码。

下载 **debhello-0.0.tar.gz**

```
 $ wget http://www.example.org/download/debhello-0.0.tar.gz
 ...
 $ tar -xzmf debhello-0.0.tar.gz
 $ tree
.
+-- debhello-0.0
|   +-- Makefile
|   +-- README.md
|   +-- src
|       +-- hello.c
+-- debhello-0.0.tar.gz

3 directories, 4 files
```

这里的 C 源代码 **hello.c** 非常的简单。

hello.c

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

这里，源码中的 **Makefile** 支持“GNU 编码标准”和“FHS（文件系统层级规范）”。特别地：

- 构建二进制程序时会考虑 **$(CPPFLAGS)**、**$(CFLAGS)**、**$(LDFLAGS)**，等等。
- 安装文件时采纳 **$(DESTDIR)** 作为目标系统镜像的路径前缀
- 安装文件时使用 **$(prefix)** 的值，以便我们将其设置覆盖为 **/usr**

Makefile

```
 $ 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
```

注意

对 **$(CFLAGS)** 的 **echo** 命令用于在接下来的例子中验证所设置的构建参数。

5.6 Step 2: Generate template files with debmake

debmake 命令的输出十分详细，如下所示，它可以展示程序的具体操作内容。

The output from the debmake command

```
 $ cd /path/to/debhello-0.0
 $ debmake -x1
I: set parameters
I: sanity check of parameters
I: pkg="debhello", ver="0.0", rev="1"
```

```
I: *** start packaging in "debhello-0.0". ***
I: provide debhello_0.0.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-0.0.tar.gz debhello_0.0.orig.tar.gz
I: pwd = "/path/to/debhello-0.0"
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I:  50 %, ext = md
I:  50 %, ext = c
I: check_all_licenses
I: ...
I: check_all_licenses completed for 3 files.
I: bunch_all_licenses
I: format_all_licenses
I: make debian/* template files
I: debmake -x "1" ...
I: creating => debian/control
I: creating => debian/copyright
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra0_changel...
I: creating => debian/changelog
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra0_rules.t...
I: creating => debian/rules
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra0source_f...
I: creating => debian/source/format
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1_README....
I: creating => debian/README.Debian
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1_README....
I: creating => debian/README.source
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1_clean.t...
I: creating => debian/clean
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1_gbp.con...
I: creating => debian/gbp.conf
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1_salsa-c...
I: creating => debian/salsa-ci.yml
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1_watch.t...
I: creating => debian/watch
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1tests_co...
I: creating => debian/tests/control
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1upstream...
I: creating => debian/upstream/metadata
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1patches_...
I: creating => debian/patches/series
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1source.n...
I: creating => debian/source/local-options.ex
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1source.n...
I: creating => debian/source/local-patch-header.ex
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1single_d...
I: creating => debian/dirs
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1single_i...
I: creating => debian/install
I: substituting => /usr/lib/python3/dist-packages/debmake/data/extra1single_l...
I: creating => debian/links
I: $ wrap-and-sort -vast
debian/control
debian/tests/control
debian/copyright
debian/dirs
debian/install
debian/links
--- Modified files ---
debian/control
```

```
debian/dirs
debian/install
debian/links
I: $ wrap-and-sort -vast complete.  Now, debian/* may have a blank line at th...
```

debmake 命令基于命令行选项产生所有这些模板文件。如果没有指定具体选项，**debmake** 命令将为您自动选择合理的默认值：

- 源码包名称：**debhello**
- 上游版本：**0.0**
- 二进制软件包名称：**debhello**
- Debian 修订版本：**1**
- 软件包类型：**bin**（ELF 二进制可执行程序软件包）
- The **-x** option: **-x1** (without maintainer script supports for simplicity)

注意

Here, the **debmake** command is invoked with the **-x1** option to keep this tutorial simple. Use of default **-x3** option is highly recommended.

我们来检查一下自动产生的模板文件。
基本 **debmake** 命令运行后的源码树。

```
 $ cd /path/to
 $ tree
.
+-- debhello-0.0
|   +-- Makefile
|   +-- README.md
|   +-- debian
|   |   +-- README.Debian
|   |   +-- README.source
|   |   +-- changelog
|   |   +-- clean
|   |   +-- control
|   |   +-- copyright
|   |   +-- dirs
|   |   +-- gbp.conf
|   |   +-- install
|   |   +-- links
|   |   +-- patches
|   |   |   +-- series
|   |   +-- rules
|   |   +-- salsa-ci.yml
|   |   +-- source
|   |   |   +-- format
|   |   |   +-- local-options.ex
|   |   |   +-- local-patch-header.ex
|   |   +-- tests
|   |   |   +-- control
|   |   +-- upstream
|   |   |   +-- metadata
|   |   +-- watch
|   +-- src
|       +-- hello.c
+-- debhello-0.0.tar.gz
```

```
+-- debhello_0.0.orig.tar.gz -> debhello-0.0.tar.gz

8 directories, 24 files
```

这里的 **debian/rules** 文件是应当由软件包维护者提供的构建脚本。此时该文件是由 **debmake** 命令产生的模板文件。

debian/rules（模板文件）：

```
 $ cd /path/to/debhello-0.0
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1

%:
        dh $@

#override_dh_auto_install:
#       dh_auto_install -- prefix=/usr

#override_dh_install:
#       dh_install --list-missing -X.pyc -X.pyo
```

这便是使用 **dh** 命令时标准的 **debian/rules** 文件。（某些内容已被注释，可供后续修改使用。）

这里的 **debian/control** 文件提供了 Debian 软件包的主要元信息。此时该文件是由 **debmake** 命令产生的模板文件。

debian/control（模板文件）：

```
 $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.0
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/debhello

Package: debhello
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.
```

警告

If you leave “**Section: unknown**”in the template **debian/control** file unchanged, the **lintian** error may cause the build to fail.

Since this is the ELF binary executable package, the **debmake** command sets “**Architecture: any**”

and “**Multi-Arch: foreign**”. Also, it sets required **substvar** parameters as “**Depends: ${shlibs:Depends}, ${misc:Depends}**”. These are explained in “第 6 章”.

注意

Please note this **debian/control** file uses the RFC-822 style as documented in “5.2 Source package control files — debian/control”of the “Debian Policy Manual” . The use of the empty line and the leading space are significant.

这里的 **debian/copyright** 提供了 Debian 软件包版权数据的总结。此时该文件是由 **debmake** 命令产生的模板文件。

debian/copyright（模板文件）：

```
 $ cat debian/copyright
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: debhello
Upstream-Contact: <preferred name and address to reach the upstream project>
Source: <url://example.com>
#
# Please double check copyright with the licensecheck(1) command.

Files:     Makefile
           README.md
           src/hello.c
Copyright: __NO_COPYRIGHT_NOR_LICENSE__
License:   __NO_COPYRIGHT_NOR_LICENSE__

#-------------------------------------------------------------------------------...
# Files marked as NO_LICENSE_TEXT_FOUND may be covered by the following
# license/copyright files.
```

5.7 第三步：编辑模板文件

作为维护者，要制作一个合适的 Debian 软件包当然需要对模板内容进行一些手工的调整。

In order to install files as a part of the system files, the **$(prefix)** value of **/usr/local** in the **Makefile** should be overridden to be **/usr**. This can be accommodated by the following the **debian/rules** file with the **override_dh_auto_install** target setting “**prefix=/usr**”.

debian/rules（维护者版本）：

```
 $ cd /path/to/debhello-0.0
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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
```

如上在 **debian/rules** 文件中导出 =**DH_VERBOSE** 环境变量可以强制 **debhelper** 工具输出细粒度的构建报告。

Exporting **DEB_BUILD_MAINT_OPTION** as above sets the hardening options as described in the “FEATURE AREAS/ENVIRONMENT”in **dpkg-buildflags**(1). 1

如上导出 **DEB_CFLAGS_MAINT_APPEND** 可以强制 C 编译器给出所有类型的警告内容。

如上导出 **DEB_LDFLAGS_MAINT_APPEND** 可以强制链接器只对真正需要的库进行链接。2

The **dh_auto_install** command for the Makefile based build system essentially runs “**$(MAKE) install DESTDIR=debian/debhello**”. The creation of this **override_dh_auto_install** target changes its behavior to “**$(MAKE) install DESTDIR=debian/debhello prefix=/usr**”.

这里是维护者版本的 **debian/control** 和 **debian/copyright** 文件。

debian/control（维护者版本）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.6.2
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（维护者版本）：

```
 $ vim debian/copyright
 ... hack, hack, hack, ...
 $ 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.
```

1This is a cliché to force a read-only relocation link for the hardening and to prevent the lintian warning “**W: debhello: hardening-no-relro usr/bin/hello**”. This is not really needed for this example but should be harmless. The lintian tool seems to produce a false positive warning for this case which has no linked library.

2这里的做法是为了避免在依赖库情况复杂的情况下过度链接，例如某些 GNOME 程序。这样做对这里的简单例子来说并不是必要的，但应当是无害的。

```
 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.
```

Let’s remove unused template files and edit remaining template files:

- **debian/README.source**
- **debian/source/local-option.ex**
- **debian/source/local-patch-header.ex**
- **debian/patches/series** (No upstream patch)
- **clean**
- **dirs**
- **install**
- **links**

debian/. 下面的模板文件（**0.0** 版）：

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

4 directories, 11 files
```

提示

对于来自 **debhelper** 软件包的各个 **dh**_* 命令来说，它们在读取所使用的配置文件时通常把以 # 开头的行视为注释行。

5.8 Step 4: Building package with debuild

您可以使用 **debuild** 或者等效的命令工具（参见“第 5.4 节”）在这个源码树内构建一个非原生 Debian 软件包。命令的输出通常十分详细，如下所示，它会对构建中执行的操作进行解释。

Building package with debuild

```
 $ cd /path/to/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/debhello-0.0'
# CFLAGS=-g -O2 -Werror=implicit-function-declaration
 ...
Finished running lintian.
```

这里验证了 **CFLAGS** 已经得到了更新，添加了 **-Wall** 和 **-pendatic** 参数；这是我们先前在 **DEB_CFLAGS_MAINT** 变量中所指定的。

根据 **lintian** 的报告，您应该如同后文中的例子那样（请见“第 14 章”）为软件包添加 man 手册页。我们这里暂且跳过这部分内容。

现在我们来看看成果如何。

debhello 0.0 版使用 **debuild** 命令产生的文件：

```
 $ cd /path/to
 $ tree -FL 1
./
+-- debhello-0.0/
+-- debhello-0.0.tar.gz
+-- 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.gz -> debhello-0.0.tar.gz

2 directories, 9 files
```

您可以看见生成的全部文件。

- **debhello_0.0.orig.tar.gz** 是指向上游源码压缩包的符号链接。
- **debhello_0.0-1.debian.tar.xz** 包含了维护者生成的内容。
- **debhello_0.0-1.dsc** 是 Debian 源码包的元数据文件。
- **debhello_0.0-1_amd64.deb** 是 Debian 二进制软件包。
- **debhello-dbgsym_0.0-1_amd64.deb** 是 Debian 的调试符号二进制软件包。另请参见“第 10.21 节”。
- **debhello_0.0-1_amd64.build** 是构建日志文件。
- **debhello_0.0-1_amd64.buildinfo** 是 **dpkg-genbuildinfo**(1) 生成的元数据文件。

- **debhello_0.0-1_amd64.changes** 是 Debian 二进制软件包的元数据文件。

debhello_0.0-1.debian.tar.xz 包含了 Debian 对上游源代码的修改，具体如下所示。
压缩文件 **debhello_0.0-1.debian.tar.xz** 的内容：

```
 $ tar -tzf debhello-0.0.tar.gz
debhello-0.0/
debhello-0.0/src/
debhello-0.0/src/hello.c
debhello-0.0/Makefile
debhello-0.0/README.md
 $ tar --xz -tf debhello_0.0-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
debian/gbp.conf
debian/rules
debian/salsa-ci.yml
debian/source/
debian/source/format
debian/tests/
debian/tests/control
debian/upstream/
debian/upstream/metadata
debian/watch
```

debhello_0.0-1_amd64.deb 包含了将要安装至目标系统中的文件。
The **debhello-debsym_0.0-1_amd64.deb** contains the debug symbol files to be installed to the target system.
所有二进制包的包内容：

```
 $ 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/c4/
-rw-r--r-- root/root ...  ./usr/lib/debug/.build-id/c4/cec6427d45de48efc7f263...
drwxr-xr-x root/root ...  ./usr/share/
drwxr-xr-x root/root ...  ./usr/share/doc/
lrwxrwxrwx root/root ...  ./usr/share/doc/debhello-dbgsym -> debhello
 $ 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
```

生成的依赖列表会给出所有二进制软件包的依赖。
生成的所有二进制软件包的依赖列表（**v=0.0**）：

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

小心

在将软件包上传至 Debian 仓库之前，仍然有很多细节需要进行处理。

注意

如果跳过了对 **debmake** 命令自动生成的配置文件的手工调整步骤，所生成的二进制软件包可能缺少有用的软件包描述信息，某些政策的要求也无法满足。这个不正式的软件包对于 **dpkg** 命令来说可以正常处理，也许这样对您本地的部署来说已经足够好了。

5.9 Step 3 (alternatives): Modification to the upstream source

The above example did not touch the upstream source to make the proper Debian package. An alternative approach as the maintainer is to modify files in the upstream source. For example, **Makefile** may be modified to set the **$(prefix)** value to **/usr**.

注意

The above “第 5.7 节”using the **debian/rules** file is the better approach for packaging for this example. But let’s continue on with this alternative approaches as a leaning experience.

In the following, let’s consider 3 simple variants of this alternative approach to generate **debian/patches/*** files representing modifications to the upstream source in the Debian source format “**3.0 (quilt)**”. These substitute “第 5.7 节”in the above step-by-step example:

- “第 5.10 节”
- “第 5.11 节”
- “第 5.12 节”

Please note the **debian/rules** file used for these examples doesn’t have the **override_dh_auto_install** target as follows:

debian/rules（备选的维护者版本）：

```
 $ cd /path/to/debhello-0.0
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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 by "diff -u"approach

Here, the patch file **000-prefix-usr.patch** is created using the **diff** command.

Patch by diff -u

```
 $ cp -a debhello-0.0 debhello-0.0.orig
 $ vim debhello-0.0/Makefile
 ... hack, hack, hack, ...
 $ diff -Nru debhello-0.0.orig debhello-0.0 >000-prefix-usr.patch
 $ cat 000-prefix-usr.patch
diff -Nru debhello-0.0.orig/Makefile debhello-0.0/Makefile
--- debhello-0.0.orig/Makefile  2024-11-29 07:57:10.299591959 +0000
+++ debhello-0.0/Makefile       2024-11-29 07:57:10.391593434 +0000
@@ -1,4 +1,4 @@
-prefix = /usr/local
+prefix = /usr

 all: src/hello

 $ rm -rf debhello-0.0
 $ mv -f debhello-0.0.orig debhello-0.0
```

Please note that the upstream source tree is restored to the original state after generating a patch file **000-prefix-usr.patch**.

This **000-prefix-usr.patch** is edited to be DEP-3 conforming and moved to the right location as below.

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

```
 $ echo '000-prefix-usr.patch' >debian/patches/series
 $ vim ../000-prefix-usr.patch
 ... hack, hack, hack, ...
 $ mv -f ../000-prefix-usr.patch debian/patches/000-prefix-usr.patch
 $ 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
```

注意

When generating the Debian source package by **dpkg-source** via **dpkg-buildpackage** in the following step of "第 5.8 节", the **dpkg-source** command assumes that no patch was applied to the upstream source, since the **.pc/applied-patches** is missing.

5.11 Patch by dquilt approach

Here, the patch file **000-prefix-usr.patch** is created using the **dquilt** command.

dquilt is a simple wrapper of the **quilt** program. The syntax and function of the **dquilt** command is the same as the **quilt**(1) command, except for the fact that the generated patch is stored in the **debian/-patches/** directory.

Patch by dquilt

```
 $ dquilt new 000-prefix-usr.patch
Patch debian/patches/000-prefix-usr.patch is now on top
 $ dquilt add Makefile
File Makefile added to patch debian/patches/000-prefix-usr.patch
 ... hack, hack, hack, ...
 $ head -1 Makefile
prefix = /usr
 $ dquilt refresh
Refreshed patch debian/patches/000-prefix-usr.patch
 $ dquilt header -e --dep3
 ... edit the DEP-3 patch header with editor
 $ 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
|   +-- gbp.conf
|   +-- install
|   +-- links
|   +-- patches
|   |   +-- 000-prefix-usr.patch
|   |   +-- series
|   +-- rules
|   +-- salsa-ci.yml
|   +-- source
|   |   +-- format
|   |   +-- local-options.ex
|   |   +-- local-patch-header.ex
|   +-- tests
|   |   +-- control
|   +-- upstream
|   |   +-- metadata
|   +-- watch
+-- src
    +-- hello.c

9 directories, 29 files
 $ cat debian/patches/series
000-prefix-usr.patch
 $ 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
```

Here, **Makefile** in the upstream source tree doesn't need to be restored to the original state for the packaging.

> 注意
>
> When generating the Debian source package by **dpkg-source** via **dpkg-buildpackage** in the following step of "第 5.8 节", the **dpkg-source** command assumes that patches were applied to the upstream source, since the **.pc/applied-patches** exists.

The upstream source tree can be restored to the original state for the packaging.

The upstream source tree (restored):

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

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

1 directory, 3 files
```

Here, **Makefile** is restored and the **.pc/applied-patches** is missing.

5.12 Patch by "dpkg-source --auto-commit"approach

Here, the patch file isn't created in this step but the source files are setup to create **debian/patches/*** files in the following step of "第 5.8 节".

我们先来编辑上游源代码。

Modified Makefile

```
 $ vim Makefile
 ... hack, hack, hack, ...
 $ head -n1 Makefile
prefix = /usr
```

Let's edit **debian/source/local-options**:

debian/source/local-options for auto-commit

```
 $ mv debian/source/local-options.ex debian/source/local-options
 $ vim debian/source/local-options
 ... hack, hack, hack, ...
 $ cat debian/source/local-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
```

Let's edit **debian/source/local-patch-header**:
debian/source/local-patch-header for auto-commit

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

Let's remove **debian/patches/*** files and other unused template files.
移除未使用的模板文件

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

4 directories, 13 files
```

There are no **debian/patches/*** files at the end of this step.

注意

When generating the Debian source package by **dpkg-source** via **dpkg-buildpackage** in the following step of "第 5.8 节", the **dpkg-source** command uses options specified in **debian/source/local-options** to auto-commit modification applied to the upstream source as **patches/debian-changes**.

Let's inspect the Debian source package generated after the following "第 5.8 节"step and extracting files from **debhello-0.0.debian.tar.xz**.

Inspect debhello-0.0.debian.tar.xz after debuild

```
 $ tar --xz -xvf debhello_0.0-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
debian/gbp.conf
debian/patches/
debian/patches/debian-changes
debian/patches/series
```

```
debian/rules
debian/salsa-ci.yml
debian/source/
debian/source/format
debian/tests/
debian/tests/control
debian/upstream/
debian/upstream/metadata
debian/watch
```

Let's check generated **debian/patches/*** files.

Inspect debian/patches/* after debuild

```
 $ cat debian/patches/series
debian-changes
 $ 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
```

The Debian source package **debhello-0.0.debian.tar.xz** is confirmed to be generated properly with **debian/patches/*** files for the Debian modification.

Chapter 6

打包工作的基础

Here, a broad overview is presented without using VCS operations for the basic rules of Debian packaging focusing on the non-native Debian package in the "**3.0 (quilt)**"format.

注意

为简明起见，某些细节被有意跳过。请按需查阅对应命令的手册页，例如 **dpkg-source**(1)、**dpkg-buildpackage**(1)、**dpkg**(1)、**dpkg-deb**(1)、**deb**(5)，等等。

Debian 源码包是一组用于构建 Debian 二进制软件包的输入文件，而非单个文件。

Debian 二进制软件包是一个特殊的档案文件，其中包含了一系列可安装的二进制数据及与它们相关的信息。

单个 Debian 源码包可能根据 **debian/control** 文件定义的内容产生多个 Debian 二进制软件包。

The non-native Debian package in the Debian source format "**3.0 (quilt)**"is the most normal Debian source package format.

注意

有许多封装脚本可用。合理使用它们可以帮助您理顺工作流程，但是请确保您能理解它们内部的基本工作原理。

6.1 打包工作流

The Debian packaging workflow to create a Debian binary package involves generating several specifically named files (see "第 6.3 节") as defined in the "Debian Policy Manual". This workflow can be summarized in 10 steps with some over simplification as follows.

1. 下载上游源码压缩包（tarball）并命名为 *package-version*.**tar.gz** 文件。
2. 使上游提供的源码压缩包解压缩后的所有文件存储在 *package-version*/ 目录中。
3. 上游的源码压缩包被复制（或符号链接）至一个特定的文件名 *packagename_version*.**orig.tar.gz**。
   - 分隔 *package* 和 *version* 的符号从 **-**（连字符）更改为 _（下划线）
   - 文件扩展名添加了 **.orig** 部分。
4. Debian 软件包规范文件将被添加至上游源代码中，存放在 *package-version*/**debian/** 目录下。
   - **debian/*** 目录下的必需技术说明文件：

     debian/rules 构建 Debian 软件包所需的可执行脚本（参见"第 6.5 节"）

debian/control 软件包配置文件包含了源码包名称、源码构建依赖、二进制软件包名称、二进制软件包依赖，等等。（参见“第 6.6 节”）

debian/changelog Debian 软件包历史文件，其中第一行定义了上游软件包版本号和 Debian 修订版本号（参见“第 6.7 节”）

debian/copyright 版权和许可证摘要信息（参看“第 6.8 节”）

- 在 **debian/*** 下的可选配置文件（参见“第 6.14 节”）：
- The **debmake** command invoked in the *package-version***/** directory may be used to provide the initial template of these configuration files.
  - 必备的配置文件总会生成，无论是否提供 **-x0** 选项。
  - **debmake** 命令永远不会覆写任何已经存在的配置文件。
- These files must be manually edited to their perfection according to the “Debian Policy Manual” and “Debian Developer’s Reference”.

5. The **dpkg-buildpackage** command (usually from its wrapper **debuild** or **sbuild**) is invoked in the *package-version***/** directory to make the Debian source and binary packages by invoking the **debian/rules** script.
   - The current directory is set as: “**CURDIR=**/*path/to/package-version*/”
   - Create the Debian source package in the Debian source format “**3.0 (quilt)**”using **dpkg-source**(1)
     - *package_version***.orig.tar.gz** (copy or symlink of *package-version***.tar.gz**)
     - *package_version-revision***.debian.tar.xz** (tarball of **debian/** found in *package-version***/**)
     - *package_version-revision***.dsc**
   - Build the source using “**debian/rules build**”into **$(DESTDIR)**
     - “**DESTDIR=debian/***binarypackage***/**”for single binary package [1]
     - “**DESTDIR=debian/tmp/**”for multi binary package
   - 使用 **dpkg-deb**(1)、**dpkg-genbuildinfo**(1) 和 **dpkg-genchanges**(1) 创建 Debian 二进制软件包。
     - *binarypackage_version-revision_arch*.**deb**
     - ……（可能有多个 Debian 二进制包文件。）
     - *package_version-revision_arch*.**changes**
     - *package_version-revision_arch*.**buildinfo**
6. 使用 **lintian** 命令检查 Debian 软件包的质量。（推荐）
   - Follow the rejection guidelines from ftp-master.
     - “软件包被拒绝常见问题解答（REJECT-FAQ）”
     - “新软件包（NEW）检查清单”
     - “Lintian 自动拒绝（autoreject）” (“lintian 标签列表”)
7. 通过手工安装和运行软件包里的程序，来测试生成的 Debian 二进制软件包的可用性。
8. After confirming the goodness, prepare files for the normal source-only upload to the Debian archive.
9. Sign the Debian package file with the **debsign** command using your private GPG key.
   - Use “**debsign** *package_version-revision*_**source.changes**”(normal source-only upload situation)
   - Use “**debsign** *package_version-revision_arch***.changes**”(exceptional binary upload situation such as NEW uploads, and security uploads) files for the binary Debian package upload.
10. Upload the set of the Debian package files with the **dput** command to the Debian archive.
    - Use “**dput** *package_version-revision*_**source.changes**”(source-only upload)

1This is the default up to **debhelper** v13. At **debhelper** v14, it warns the default change. After **debhelper** v15, it will change the default to **DESTDIR=debian/tmp/** .

- Use “**dput** *package_version-revision_arch*.**changes**”(binary upload)

Test building and confirming of the binary package goodness as above is the moral obligation as a diligent Debian developer but there is no physical barrier for people to skip such operations at this moment for the source-only upload.

这里，请将文件名中对应的部分使用下面的方式进行替换：

- 将 *package* 部分替换为 Debian 源码包名称
- 将 *binarypackage* 部分替换为 Debian 二进制软件包名称
- 将 *version* 部分替换为上游版本号
- 将 *revision* 部分替换为 Debian 修订号
- the *arch* part with the package architecture (e.g., **amd64**)

参见“Source-only uploads”。

提示

有很多种通过实践摸索而得到的补丁管理方法和版本控制系统的使用策略与技巧。您没有必要将它们全部用上。

提示

There is very extensive documentation in “Chapter 6. Best Packaging Practices” in the “Debian Developer’s Reference”. Please read it.

6.2 debhelper package

Although a Debian package can be made by writing a **debian/rules** script without using the **debhelper** package, it is impractical to do so. There are too many modern “Debian Policy”required features to be addressed, such as application of the proper file permissions, use of the proper architecture dependent library installation path, insertion of the installation hook scripts, generation of the debug symbol package, generation of package dependency information, generation of the package information files, application of the proper timestamp for reproducible build, etc.

Debhelper package provides a set of useful scripts in order to simplify Debian’s packaging workflow and reduce the burden of package maintainers. When properly used, they will help packagers handle and implement “Debian Policy”required features automatically.

现代化的 Debian 打包工作流可以组织成一个简单的模块化工作流，如下所示：

- 使用 **dh** 命令以自动调用来自 **debhelper** 软件包的许多实用脚本，以及
- 使用 **debian/** 目录下的声明式配置文件配置它们的行为。

您几乎总是应当将 **debhelper** 列为您的软件包的构建依赖之一。本文档在接下来的内容中也假设您正在使用一个版本足够新的 **debhelper** 协助进行打包工作。

注意

For **debhelper** “compat >= 9”, the **dh** command exports compiler flags (**CFLAGS**, **CXXFLAGS**, **FFLAGS**, **CPPFLAGS** and **LDFLAGS**) with values as returned by **dpkg-buildflags** if they are not set previously. (The **dh** command calls **set_buildflags** defined in the **Debian::Debhelper::Dh_Lib** module.)

注意

debhelper(1) changes its behavior with time. Please make sure to read **debhelper-compat-upgrade-checklist**(7) to understand the situation.

6.3 软件包名称和版本

如果所获取上游源代码的形式为 **hello-0.9.12.tar.gz**，您可以将 **hello** 作为上游源代码名称，并将 **0.9.12** 作为上游版本号。

组成 Debian 软件包名称的字符选取存在一定的限制。最明显的限制应当是软件包名称中禁止出现大写字母。这里给出正则表达式形式的规则总结：

- Upstream package name (**-p**): `[-+.a-z0-9]{2,}`
- Binary package name (**-b**): `[-+.a-z0-9]{2,}`
- Upstream version (**-u**): `[0-9][-+.:~a-z0-9A-Z]*`
- Debian revision (**-r**): `[0-9][+.~a-z0-9A-Z]*`

See the exact definition in “Chapter 5 - Control files and their fields”in the “Debian Policy Manual”.

您必须为 Debian 打包工作适当地调整软件包名称和上游版本号。

为了能有效地使用一些流行的工具（如 **aptitude**）管理软件包名称和版本信息，最好能将软件包名称保持在 30 字符以下；版本号和修订号加起来最好能不超过 14 个字符。[2]

为了避免命名冲突，对用户可见的二进制软件包名称不应选择任何常用的单词。

如果上游没有使用像 **2.30.32** 这样正常的版本编号方案，而是使用了诸如 **11Apr29** 这样包含日期、某些代号或者一个版本控制系统散列值等字符串作为版本号的一部分的话，请在上游版本号中将这些部分移除。这些信息可以稍后在 **debian/changelog** 文件中进行记录。如果您需要为软件设计一个版本字符串，可以使用 **YYYYMMDD** 格式，如 **20110429** 的字符串作为上游版本号。这样能保证 **dpkg** 命令在升级时能正确地确定版本的先后关系。如果您想要确保万一上游在未来重新采纳正常版本编号方案，例如 **0.1** 时能够做到顺畅地迁移，可以另行使用 **0~YYMMDD** 的格式，如 **0~110429** 作为上游版本号。

版本字符串可以按如下的方式使用 **dpkg** 命令进行比较。

```
$ dpkg --compare-versions ver1 op ver2
```

版本比较的规则可以归纳如下：

- 字符串按照起始到末尾的顺序进行比较。
- 字符比数字大。
- 数字按照整数顺序进行比较。
- 字符按照 ASCII 编码的顺序进行比较。

对于某些字符，如句点（**.**）、加号（**+**）和波浪号（**~**），有如下的特殊规则。

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

有一个稍需注意的情况，即当上游将 **hello-0.9.12-ReleaseCandidate-99.tar.gz** 这样的版本当作预发布版本，而将 **hello-0.9.12.tar.gz** 作为正式版本时。为了确保 Debian 软件包升级能够顺畅进行，您应当修改版本号命名，如将上游源代码压缩包重命名为 **hello-0.9.12~rc99.tar.gz**。

2对九成以上的软件包来说，软件包名称都不会超过 24 个字符；上游版本号通常不超过 10 个字符，而 Debian 修订版本号也通常不超过 3 个字符。

6.4 原生 Debian 软件包

The non-native Debian package in the Debian source format "**3.0 (quilt)**"is the most normal Debian source package format. The **debian/source/format** file should have "**3.0 (quilt)**"in it as described in **dpkg-source**(1). The above workflow and the following packaging examples always use this format.

而原生 Debian 软件包是较罕见的一种 Debian 软件包格式。它通常只用于打包仅对 Debian 项目有价值、有意义的软件。因此，该格式的使用通常不被提倡。

小心

A native Debian package is often accidentally built when its upstream tarball is not accessible from the **dpkg-buildpackage** command with its correct name *package_version*.**orig.tar.gz** . This is a typical newbie mistake caused by making a symlink name with "**-**"instead of the correct one with "_".

原生 Debian 软件包不对 上游代码和 **Debian** 的修改进行区分，仅包含以下内容：

- *package_version*.**tar.gz** (copy or symlink of *package-version*.**tar.gz** with **debian/*** files.)
- *package_version*.**dsc**

If you need to create a native Debian package, create it in the Debian source format "**3.0 (native)**" using **dpkg-source**(1).

提示

There is no need to create the tarball in advance if the native Debian package format is used. The **debian/source/format** file should have "**3.0 (native)**"in it as described in **dpkg-source**(1) and The **debian/source/format** file should have the version without the Debian revision (**1.0** instead of **1.0-1**). Then, the tarball containing is generated when "**dpkg-source -b**"is invoked in the source tree.

6.5 debian/rules 文件

The **debian/rules** file is the executable script which re-targets the upstream build system to install files in the **$(DESTDIR)** and creates the archive file of the generated files as the **deb** file. The **deb** file is used for the binary distribution and installed to the system using the **dpkg** command.

The Debian policy compliant **debian/rules** file supporting all the required targets can be written as simple as 3:

简单的 **debian/rules**：

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

%:
  dh $@
```

这里的 **dh** 命令的作用是作为一个序列化工具，在合适的时候调用所有所需的"**dh** 目标"命令。[4]

- **dh clean**：清理源码树中的文件。
- **dh build**：在源码树中进行构建
- **dh build-arch**：在源码树中构建架构相关的软件包
- **dh build-indep**：在源代码中构建架构无关的软件包

3**debmake** 命令会产生稍微复杂一些的 **debian/rules** 文件。虽然如此，其核心结构没有什么变化。

[4]这个简化形式在 **debhelper** 软件包第七版或更新的版本中可用。本指南内容假设您在使用 **debhelper** 第 13 版或更新的版本。

- **dh install**：将二进制文件安装至 **$(DESTDIR)**
- **dh install-arch**：为架构相关的软件包将二进制文件安装至 **$(DESTDIR)** 中
- **dh install-indep**：为架构无关的软件包将二进制文件安装进入 **$(DESTDIR)** 中
- **dh binary**：产生 **deb** 文件
- **dh binary-arch**：为架构相关的软件包产生 **deb** 文件
- **dh binary-indep**：为架构无关的软件包产生 **deb** 文件

Here, **$(DESTDIR)** path depends on the build type.

- "**DESTDIR=debian/***binarypackage***/**"for single binary package [5]
- "**DESTDIR=debian/tmp/**"for multi binary package

See "第 9.2 节"and "第 9.3 节"for customization.

提示

Setting "**export DH_VERBOSE = 1**"outputs every command that modifies files on the build system. Also it enables verbose build logs for some build systems.

6.6 debian/control 文件

The **debian/control** file consists of blocks of metadata separated by blank lines. Each block of metadata defines the following, in this order:

- Debian 源码包的元信息数据
- Debian 二进制软件包的元信息

See "Chapter 5 - Control files and their fields"of the "Debian Policy Manual" for the definition of each metadata field.

注意

The **debmake** command sets the **debian/control** file with "**Build-Depends: debhelper-compat (= 13)**"to set the **debhelper** compatibility level.

提示

If an existing package has a **debhelper** compatibility level lower than 13, it's probably time to update its packaging.

[5] This is the default up to **debhelper** v13. At **debhelper** v14, it warns the default change. After **debhelper** v15, it will change the default to **DESTDIR=debian/tmp/** .

6.7 debian/changelog file

The **debian/changelog** file records the Debian package history.

- Edit this file using the **debchange** command (alias **dch**).
- The first line defines the upstream package version and the Debian revision.
- 以特定、正式而简洁的风格记录修改内容。
  - If Debian maintainer modification fixes reported bugs, add "**Closes:** #<*bug_number*>"to close those bugs.
- Even if you're uploading your package yourself, you must document all non-trivial user-visible changes, such as:
  - 安全相关的漏洞修复。
  - 用户界面变动。
- If you're asking a sponsor to upload it, document changes more comprehensively, including all packaging-related ones, to help with package review.
  - The sponsor shouldn't have to guess your reasoning behind package changes.
  - Remember that the sponsor's time is valuable.

After finishing your packaging and verifying its quality, execute the "**dch -r**" command and save the finalized **debian/changelog** file with the suite normally set to **unstable**. [6] If you're packaging for backports, security updates, LTS, etc., use the appropriate distribution names instead.

The **debmake** command creates the initial template file with the upstream package version and the Debian revision. The distribution is set to **UNRELEASED** to prevent accidental uploads to the Debian archive.

提示

The date string used in the **debian/changelog** file can be manually generated by the "**LC_ALL=C date -R**"command.

提示

Use a **debian/changelog** entry with a version string like **1.0.1-1~rc1** when experimenting. Later, consolidate such **changelog** entries into a single entry for the official package.

The **debian/changelog** file is installed in the **/usr/share/doc/***binarypackage* directory as **changelog.Debian.gz** by the **dh_installchangelogs** command.

上游的变更日志则会安装至 **/usr/share/doc/***binarypackage* 目录中，文件名为 **changelog.gz**。

上游的变更日志是由 **dh_installchangelogs** 程序自动进行搜索和处理的；它会使用大小写不敏感的搜索方式寻找上游代码中特定名称的文件，如 **changelog**、**changes**、**changelog.txt**、**changes.txt**、**history**、**history.txt** 或 **changelog.md**。除了根目录，程序还会在 **doc/** 目录和 **docs/** 目录内进行搜索。

6.8 debian/copyright 文件

Debian takes copyright and license matters very seriously. The "Debian Policy Manual" requires a summary of these in the **debian/copyright** file of the package.

[6] If you're using the **vim** editor, make sure to save this with the "**:wq**" command.

- “12.5. Copyright information”
- “2.3. Copyright considerations”
- “License information”

The **debmake** command creates the initial **debian/copyright** template file.

- Double-check copyright information using the **licensecheck**(1) command.
- Format it as a “machine-readable **debian/copyright** file (DEP-5)”.

Unless specifically requested to be pedantic with the **-P** option, the **debmake** command skips reporting auto-generated files with permissive licenses for practicality.

小心

The **debian/copyright** file should be sorted with generic file patterns at the top of the list. See “第 16.6 节”.

注意

如果您发现了这个许可证检查工具存在一些问题，请向 **debmake** 软件包提交缺陷报告并提供包含出现问题的许可证和版权信息在内的相关文本内容。

6.9 debian/patches/* 文件

As demonstrated in “第 5.9 节”, the **debian/patches/** directory holds

- *patch-file-name*.**patch** files providing **-p1** patches and
- the **series** file which defines how these patches are applied.

See how these files are used in:

- “第 13.6 节”to build the Debian source package
- “第 13.7 节”to extract source files from the Debian source package

注意

Header texts of these patches should conform to “DEP-3”.

注意

If you want to use VCS tools such as **git**, **gbp** and **dgit** to create and manage these patches after learning basics here, please refer to later in “第 11 章”.

6.10 debian/source/include-binaries 文件

The “**dpkg-source --commit**”command functions like **dquilt** but has one advantage over the **dquilt** command. While the **dquilt** command can’t handle modified binary files, the “**dpkg-source --commit**” command detects modified binary files and lists them in the **debian/source/include-binaries** file to include them in the Debian tarball as a part of the Debian source package.

6.11 debian/watch 文件

The **uscan**(1) command downloads the latest upstream version using the **debian/watch** file. E.g.:
基本的 **debian/watch** 文件：

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

The **uscan** command may verify the authenticity of the upstream tarball with optional configuration (see “第 6.12 节”).
See **uscan**(1), “第 9.4 节”, “第 8.1 节”, and “第 11.10 节”for more.

6.12 debian/upstream/signing-key.asc file

Some packages are signed by a GPG key and their authenticity can be verified using their public GPG key.
For example, “GNU hello”can be downloaded via HTTP from https://ftp.gnu.org/gnu/hello/ . There are sets of files:

- **hello-*version*.tar.gz**（上游源代码）
- **hello-*version*.tar.gz.sig**（分离的签名）nature)

我们现在来选择最新的版本套装。
下载上游提供的的源码压缩包及其签名。

```
$ wget https://ftp.gnu.org/gnu/hello/hello-2.9.tar.gz
 ...
$ wget https://ftp.gnu.org/gnu/hello/hello-2.9.tar.gz.sig
 ...
$ gpg --verify hello-2.9.tar.gz.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
```

If you know the public GPG key of the upstream maintainer from the mailing list, use it as the **debian/upstream/signing-key.asc** file. Otherwise, use the hkp keyserver and check it via your web of trust.
Download public GPG key for the upstream

```
$ 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
$ gpg --verify hello-2.9.tar.gz.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
```

提示

If your network environment blocks access to the HKP port **11371**, use "**hkp://keyserver.ubuntu.com:80**"instead.

在确认密钥身份 **80EE4A00** 值得信任之后，应当下载其公钥并将其保存在 **debian/upstream/signing-key.asc** 文件中。

Set public GPG key to debian/upstream/signing-key.asc

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

With the above **debian/upstream/signing-key.asc** file and the following **debian/watch** file, the **uscan** command can verify the authenticity of the upstream tarball after its download. E.g.:

Improved debian/watch file with GPG support:

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

6.13 debian/salsa-ci.yml 文件

Install Salsa CI configuration file. See "第 11.3 节".

6.14 其它 debian/* 文件

另外也可以添加一些可选的配置文件并放入 **debian/** 目录。它们大多用于控制由 **debhelper** 软件包提供的 **dh**_* 命令的行为，但也有一些文件会影响 **dpkg-source**、**lintian** 和 **gbp** 这些命令。

提示

Even an upstream source without its build system can be packaged just by using these files. See "第 14.2 节"as an example.

The alphabetical list of notable optional **debian/***binarypackage***.*** configuration files listed below provides very powerful means to set the installation path of files. Please note:

- The "**-x[01234]**"superscript notation that appears in the following list indicates the minimum value for the **debmake -x** option that generates the associated template file. See "第 16.9 节"or **debmake**(1) for details.
- For a single binary package, the "*binarypackage*."part of the filename in the list may be removed.
- For a multi binary package, a configuration file missing the "*binarypackage*"part of the filename is applied to the first binary package listed in the **debian/control**.
- When there are many binary packages, their configurations can be specified independently by prefixing their name to their configuration filenames such as "*package-1***.install**", "*package-2***.install**", etc.
- **debmake** 可能没有自动生成某些模板配置文件。如遇到这种情况，您可以使用文本编辑器手动创建缺失的文件。
- Some configuration template files generated by the **debmake** command with an extra **.ex** suffix need to be activated by removing that suffix.

- 您应当删除 **.ex** 命令生成但对您无用的配置模板文件。
- 请按需复制配置模板文件以匹配其对应的二进制包名称以及您的需求。

binarypackage*.bug-control** [-x3] 将安装至 *binarypackage* 软件包的 **usr/share/bug/binarypackage***/control** 位置。另请参考“第 9.11 节”。

binarypackage*.bug-presubj** [-x3] 将安装至 *binarypackage* 软件包的 **usr/share/bug/binarypackage***/presubj** 位置。另请参考“第 9.11 节”。

binarypackage*.bug-script** [-x3] 将安装至 *binarypackage* 软件包的 **usr/share/bug/binarypackage* or **usr/share/bug/***binarypackage***/script** 位置。另请参考“第 9.11 节”。

***binarypackage*.bash-completion** [-x3] List **bash** completion scripts to be installed.

The **bash-completion** package is required for both build and user environments.

另请参考 **dh_bash-completion**(1)。

clean [-x2] 列出（构建前）未被 **dh_auto_clean** 命令清理，且需要手工清理的文件。

另请参考 **dh_auto_clean**(1) 和 **dh_clean**(1)。

compat [-x4] Set the **debhelper** compatibility level. (deprecated)

Use “**Build-Depends: debhelper-compat (= 13)**”in **debian/control** to specify the compatibility level and remove **debian/compat**.

See “**COMPATIBILITY LEVELS**”in **debhelper**(7).

***binarypackage*.conffiles** [-x3] This optional file is installed into the **DEBIAN** directory within the binary package while supplementing it with all the conffiles auto-detected by **debhelper**.

This file is primarily useful for using ”special” entries such as the remove-on-upgrade feature from **dpkg**(1).

如果您正要打包的程序要求每个用户都对 **/etc** 目录下的配置文件进行修改，可以采取两种常见办法使其不作为 conffile 配置文件出现，避免 **dpkg** 命令处理软件包时给出不必要的处理选项。

- 在 **/etc** 目录下创建一个符号链接，指向 **/var** 目录下的某些文件；实际存在的文件则使用维护者脚本（maintainer script）予以创建。
- 使用维护者脚本（maintainer script）在 **/etc** 目录下创建并维护配置所需的文件。

另请参考 **dh_installdeb**(1)。

***binarypackage*.config** [-x3] 这是 **debconf config** 脚本，用来在配置软件包时向用户询问任何必需的问题。另请参见“第 10.22 节”。

binarypackage*.cron.hourly** [-x3] 安装至 *binarypackage* 包内的 **etc/cron/hourly/binarypackage* 文件。

另请参见 **dh_installcron**(1) 和 **cron**(8)。

binarypackage*.cron.daily** [-x3] 安装至 *binarypackage* 包内的 **etc/cron/daily/binarypackage* 文件。

另请参见 **dh_installcron**(1) 和 **cron**(8)。

binarypackage*.cron.weekly** [-x3] 安装至 *binarypackage* 包内的 **etc/cron/weekly/binarypackage* 文件。

另请参见 **dh_installcron**(1) 和 **cron**(8)。

***binarypackage*.cron.monthly** [-x3] Installed into the *etc/cron/monthly/**binarypackage* file in *binarypackage*.

另请参见 **dh_installcron**(1) 和 **cron**(8)。

binarypackage*.cron.d** [-x3] 安装至 *binarypackage* 包内的 **etc/cron.d/binarypackage* 文件。

参见 **dh_installcron**(1)、**cron**(8) 和 **crontab**(5)。

binarypackage*.default** [-x3] 若该文件存在，它将被安装至 *binarypackage* 包中的 **etc/default/binarypackage* 位置。

参见 **dh_installinit**(1)。

***binarypackage*.dirs** [-x1] 列出 *binarypackage* 包中要创建的目录。

参见 **dh_installdirs**(1)。

通常情况下您并不需要这么做，因为所有的 **dh_install*** 命令都会自动创建所需的目录。请仅在遇到问题时考虑使用这一工具。

***binarypackage*.doc-base** -x1 作为 *binarypackage* 包中的 **doc-base** 控制文件进行安装。

See **dh_installdocs**(1) and “Debian doc-base Manual (**doc-base.html**)”provided by the **doc-base** package.

***binarypackage*.docs** -x1 列出要安装在 *binarypackage* 包中的文档文件。

参见 **dh_installdocs**(1)。

***binarypackage*.emacsen-compat** -x3 安装至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/co** 文件。

参见 **dh_installemacsen**(1)。

***binarypackage*.emacsen-install** -x3 安装至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/inst** 文件。

参见 **dh_installemacsen**(1)。

***binarypackage*.emacsen-remove** -x3 安装至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/re** 文件。

参见 **dh_installemacsen**(1)。

***binarypackage*.emacsen-startup** -x3 安装至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/sta** 文件。

参见 **dh_installemacsen**(1)。

binarypackage*.examples** -x1 列出要安装至 *binarypackage* 包中 **usr/share/doc/binarypackage***/examples/** 位置下的示例文件或目录。

参见 **dh_installexamples**(1)。

gbp.conf -x1 如果该文件存在，它将作为 **gbp** 命令的配置文件发挥作用。

参见 **gbp.conf**(5)、**gbp**(1) 和 **git-buildpackage**(1)。

***binarypackage*.info** -x1 列出要安装至 *binarypackage* 包中的 info 文件。

参见 **dh_installinfo**(1)。

binarypackage*.init** -x4 Installed into **etc/init.d/binarypackage* in *binarypackage*. (deprecated)

参见 **dh_installinit**(1)。

***binarypackage*.install** -x1 列出未被 **dh_auto_install** 命令安装的其它应当安装的文件。

参见 **dh_install**(1) 和 **dh_auto_install**(1)。

***binarypackage*.links** -x1 列出要生成符号链接的源文件和目标文件对。每一对链接均应在单独的一行中列出，源文件和目标文件之间使用空白字符分隔。

参见 **dh_link**(1)。

binarypackage*.lintian-overrides** -x3 安装至软件包构建目录的 **usr/share/lintian/overrides/binarypackage* 位置。该文件用于消除 **lintian** 错误生成的诊断信息。

参见 **dh_lintian**(1)、**lintian**(1) 和“Lintian 用户手册”。

binarypackage*.maintscript** -x2 If this optional file exists, **debhelper** uses this as the template to generate **DEBIAN/binarypackage***.{pre,post}{inst,rm}** files within the binary package while adding “**-- ”$@””**to the **dpkg-maintscript-helper**(1) command.

See **dh_installdeb**(1) and “Chapter 6 - Package maintainer scripts and installation procedure” in the “Debian Policy Manual”.

manpage.* -x3 这些文件是 **debmake** 命令生成的 man 手册页模板文件。请将其重命名为合适的文件名并更新其内容。

Debian Policy requires that each program, utility, and function should have an associated manual page included in the same package. Manual pages are written in **nroff**(1). If you are new to making a manpage, use **manpage.asciidoc** or **manpage.1** as the starting point.

***binarypackage*.manpages** -x1 列出要安装的 man 手册页。

参见 **dh_installman**(1)。

***binarypackage*.menu** （已过时，不再安装） tech-ctte #741573 decided “Debian should use **.desktop** files as appropriate”.

安装至 *binarypackage* 包中的 **usr/share/menu/***binarypackage* Debian 菜单文件。

参见 **menufile**(5) 以了解其格式。另请参见 **dh_installmenu**(1)。

NEWS -x3 安装至 **usr/share/doc/***binarypackage***/NEWS.Debian** 文件。

参见 **dh_installchangelogs**(1)。

patches/* 这是 **-p1** 补丁文件的集合，它们将在使用源代码构建之前应用在上游源码上。

debmake 默认不会生成补丁文件。

参见 **dpkg-source**(1)、“第 4.4 节”和“第 5.9 节”。

patches/series -x1 **patches/*** 补丁文件的应用顺序。

***binarypackage*.preinst** -x2**, *binarypackage*.postinst** -x2**, *binarypackage*.prerm** -x2**, *binarypackage*.postrm**

If these optional files exist, the corresponding files are installed into the **DEBIAN** directory within the binary package after enriched by **debhelper**. Otherwise, these files in the **DEBIAN** directory within the binary package is generated by **debhelper**.

Whenever possible, simpler *binarypackage***.maintscript** should be used instead.

See **dh_installdeb**(1) and “Chapter 6 - Package maintainer scripts and installation procedure” in the “Debian Policy Manual”.

See also **debconf-devel**(7) and “3.9.1 Prompting in maintainer scripts”in the “Debian Policy Manual”.

README.Debian -x1 安装至 **debian/control** 文件列出的第一个二进制软件包中的 **usr/share/doc/***binarypackage***/README.Debian** 位置。

该文件提供了针对该 Debian 软件包的信息。

参见 **dh_installdocs**(1)。

README.source -x1 Installed into the first binary package listed in the **debian/control** file as **usr/share/doc/***binarypackage***/README.source**.

If running “**dpkg-source -x**”on a source package doesn’t produce the source of the package, ready for editing, and allow one to make changes and run **dpkg-buildpackage** to produce a modified package without taking any additional steps, creating this file is recommended.

See “Debian policy manual section 4.14”.

binarypackage*.service** -x3 如果该文件存在，它将被安装到 *binarypackage* 包下面的 **lib/systemd/system/binarypackage***.service** 位置。

参见 **dh_systemd_enable**(1)、**dh_systemd_start**(1) 和 **dh_installinit**(1)。

source/format -x1 Debian 软件包格式。

- Use “**3.0 (quilt)**”to make this non-native package (recommended)
- Use “**3.0 (native)**”to make this native package

See “SOURCE PACKAGE FORMATS”in **dpkg-source**(1).

source/lintian-overrides -x3 These file is not installed, but are scanned by the **lintian** command to provide overrides for the source package.

另请参考 **dh_lintian**(1) 和 **lintian**(1)。

source/local-options -x1 **dpkg-source** 命令使用此内容作为它的选项，比较重要的选项有：

- **unapply-patches**
- **abort-on-upstream-changes**
- **auto-commit**
- **single-debian-patch**

该文件不会包含在生成的源码包中，仅对维护者在版本控制系统中维护软件包有意义。

See “FILE FORMATS”in **dpkg-source**(1).

source/local-patch-header -x1 自由格式的文本，将被包含在自动生成补丁的顶部。

该文件不会包含在生成的源码包中，仅对维护者在版本控制系统中维护软件包有意义。

See “FILE FORMATS”in **dpkg-source**(1).

source/options -x3 Use **source/local-options** instead to avoid issues with NMUs. See “FILE FORMATS”in **dpkg-source**(1).

source/patch-header -x4 Use **source/local-patch-header** instead to avoid issues with NMUs. See “FILE FORMATS”in **dpkg-source**(1).

***binarypackage*.symbols** [-x1] 这些符号文件如果存在，将交由 **dpkg-gensymbols** 命令进行处理和安装。

参见 **dh_makeshlibs**(1) 和“第 10.16 节”。

***binarypackage*.templates** [-x3] 这是 **debconf** 模板文件，用于在安装过程中向用户询问必需的问题以正确配置软件包。请参阅“第 10.22 节”。

tests/control [-x1] This is the RFC822-style test meta data file defined in DEP-8. See **autopkgtest**(1) and “第 10.4 节”.

TODO [-x3] 安装至 **debian/control** 文件列出的第一个二进制包中的 **usr/share/doc/***binarypackage***/TODO.Debia** 文件。

参见 **dh_installdocs**(1)。

binarypackage*.tmpfile** [-x3] 如果该文件存在，它将被安装至 *binarypackage* 包中的 **usr/lib/tmpfiles.d/binarypackage***.conf** 文件。

参见 **dh_systemd_enable**(1)、**dh_systemd_start**(1) 和 **dh_installinit**(1)。

***binarypackage*.upstart** [-x4] If this exists, it is installed into **etc/init/package.conf** in the package build directory. (deprecated)

参见 **dh_installinit**(1)。

upstream/metadata [-x1] Per-package machine-readable metadata about upstream (DEP-12). See “Upstream MEtadata GAthered with YAml (UMEGAYA)”.

Chapter 7

Quality of packaging

The quality of Debian packaging can be improved by using testing tools.

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

If you follow “第 4 章”, these are automatically executed. You are expected to fix all warnings.

7.1 Reformat debian/* files with wrap-and-sort

It is good idea to reformat **debian/*** files consistently using the **wrap-and-sort**(1) command in **devscripts** package.

Reformat debian/* files

```
$ wrap-and-sort -vast
```

7.2 Validate debian/* files with debputy

The new debputy tool [1] includes subcommands to validate (and fix) most files in **debian/***.

Check correctness of files in debian/*

```
$ debputy lint --spellcheck
```

Format debian/control and debian/tests/control files

```
$ debputy reformat --style black
```

Using the “**debputy reformat**”command obsoletes using “**wrap-and-sort -vast**”.

The debputy tool also includes a language server. You can set up to get real-time feedback while editing **debian/*** files with any modern editor supporting the Language Server Protocol.

1The main purpose of the debputy tool is to offer a new Debian package build paradigm. This new paradigm is beyond the scope of this tutorial.

Chapter 8

Sanitization of the source

There are a few cases that require sanitizing the source to prevent contamination of the generated Debian source package.

- Non-https://www.debian.org/social_contract.html#guidelines[DFSG] compliant content in the upstream source.
  - Debian takes software freedom seriously and adheres to the DFSG.
- Extraneous auto-generated content in the upstream source.
  - Debian packages should rebuild these under the latest system.
- Extraneous VCS content in the upstream source.
  - The **-i** and **-I** options set in “第 4.5 节”for the **dpkg-source**(1) command should avoid these.
    - The **-i** option is intended for non-native Debian packages.
    - The **-I** option is intended for native Debian packages.

There are several methods to avoid including undesirable content.

8.1 Fix with Files-Excluded

This method is suitable for avoiding non-https://www.debian.org/social_contract.html#guidelines[DFSG] compliant content in the upstream source tarball.

- 在 **debian/copyright** 文件中的 **Files-Excluded** 一节中列出需要移除的文件。
- 在 **debian/watch** 文件中列出下载上游源码包（tarball）所使用的 URL。
- 运行 **uscan** 命令以下载新的上游源码包（tarball）。
  - Alternatively, use the “**gbp import-orig --uscan --pristine-tar**”command.
- **mk-origtargz** invoked from **uscan** removes excluded files from the upstream tarball and repack it as a clean tarball.
- 最后得到 tarball 的版本编号会附加一个额外的后缀 **+dfsg**。

See “**COPYRIGHT FILE EXAMPLES**”in **mk-origtargz**(1).

8.2 Fix with “debian/rules clean”

This method is suitable for avoiding auto-generated files by removing them in the ”**debian/rules clean**” target.

注意

The "**debian/rules clean**" target is called before the "**dpkg-source --build**" command by the **dpkg-buildpackage** command. The "**dpkg-source --build**" command ignores removed files.

8.3 Fix with extend-diff-ignore

This is for the non-native Debian package.

The problem of extraneous diffs can be fixed by ignoring changes made to specific parts of the source tree. This is done by adding the "**extend-diff-ignore=**⋯" line in the **debian/source/options** file.

debian/source/options to exclude the config.sub, config.guess and Makefile files:

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

注意

This approach always works, even when you can't remove the file. It saves you from having to make a backup of the unmodified file just to restore it before the next build.

提示

If you use the **debian/source/local-options** file instead, you can hide this setting from the generated source package. This may be useful when local non-standard VCS files interfere with your packaging.

8.4 Fix with tar-ignore

This is for the native Debian package.

You can exclude certain files in the source tree from the generated tarball by adjusting the file glob. Add the "**tar-ignore=**⋯" lines in the **debian/source/options** or **debian/source/local-options** files.

注意

For example, if the source package of a native package needs files with the **.o** extension as part of the test data, the setting in "第 4.5 节"may be too aggressive. You can work around this by dropping the **-I** option for **DEBUILD_DPKG_BUILDPACKAGE_OPTS** in "第 4.5 节"and adding the "**tar-ignore=**⋯" lines in the **debian/source/local-options** file for each package.

8.5 Fix with “git clean -dfx”

The problem of extraneous content in the second build can be avoided by restoring the source tree. This is done by committing the source tree to the Git repository before the first build.

You can restore the source tree before the second package build. For example:

```
$ git reset --hard
$ git clean -dfx
```

This works because the **dpkg-source** command ignores the contents of typical VCS files in the source tree, as specified by the **DEBUILD_DPKG_BUILDPACKAGE_OPTS** setting in “第 4.5 节”.

提示

If the source tree is not managed by a VCS, run ”**git init; git add -A .; git commit**” before the first build.

Chapter 9

More on packaging

Let's explore more fundamentals of Debian packaging.

9.1 软件包自定义

All customization data for the Debian source package resides in the **debian/** directory as presented in "第 5.7 节":

- The Debian package build system can be customized through the **debian/rules** file (see "第 9.2 节").
- 可以使用额外的配置文件（如 **debian/** directory 目录下的 *package*.**install** 和 *package*.**docs** 文件）以配合来自 **debhelper** 软件包的 **dh**_* 命令自定义 Debian 软件包文件的安装路径等信息（请参见 "第 6.14 节"）。

When these are not sufficient to make a good Debian package, **-p1** patches of **debian/patches/*** files are deployed to modify the upstream source. These are applied in the sequence defined in the **debian/patches/series** file before building the package as presented in "第 5.9 节".

You should address the root cause of the Debian packaging problem in the least invasive way possible. This approach will make the generated package more robust for future upgrades.

注意

If the patch addressing the root cause is useful to the upstream project, send it to the upstream maintainer.

9.2 Customized debian/rules

Flexible customization of the 第 6.5 节 is achieved by adding appropriate **override_dh_*** targets and their rules.

When a special operation is required for a certain **dh**_*foo* command invoked by the **dh** command, its automatic execution can be overridden by adding the makefile target **override_dh**_*foo* in the **debian/rules** file.

构建的过程可以使用某些上游提供的接口进行定制化，如使用传递给标准的源代码构建系统的参数。这些构建系统包括但不限于：

- **configure**,
- **Makefile**,
- "**python -m build**", 或者
- **Build.PL**。

In this case, you should add the **override_dh_auto_build** target with "**dh_auto_build --** *arguments*". This ensures that *arguments* are passed to the build system after the default parameters that **dh_auto_build** usually passes.

提示

Avoid executing bare build system commands directly if they are supported by the **dh_auto_build** command.

参见：

- "第 5.7 节"for the basic example.
- "第 10.3 节"to be reminded of the challenge involving the underlying build system.
- "第 10.10 节"for multiarch customization.
- "第 10.6 节"for hardening customization.

9.3 Variables for debian/rules

某些对自定义 **debian/rules** 有用的变量定义可以在 **/usr/share/dpkg/** 目录下的文件中找到。比较重要的包括：

pkg-info.mk Set **DEB_SOURCE**, **DEB_VERSION**, **DEB_VERSION_EPOCH_UPSTREAM**, **DEB_VERSION_UPST** **DEB_VERSION_UPSTREAM**, and **DEB_DISTRIBUTION** variables obtained from **dpkg-parsechangelog**(1). (useful for backport support etc..)

vendor.mk Set **DEB_VENDOR** and **DEB_PARENT_VENDOR** variables; and **dpkg_vendor_derives_from** macro obtained from **dpkg-vendor**(1). (useful for vendor support (Debian, Ubuntu, ⋯).)

architecture.mk Set **DEB_HOST**_* and **DEB_BUILD**_* variables obtained from **dpkg-architecture**(1).

buildflags.mk Set **CFLAGS**, **CPPFLAGS**, **CXXFLAGS**, **OBJCFLAGS**, **OBJCXXFLAGS**, **GCJFLAGS**, **FFLAGS**, **FCFLAGS**, and **LDFLAGS** build flags obtained from **dpkg-buildflags**(1).

For example, you can add an extra option to **CONFIGURE_FLAGS** for **linux-any** target architectures by adding the following to **debian/rules**:

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

参见"第 10.10 节"、**dpkg-architecture**(1) 和 **dpkg-buildflags**(1)。

9.4 新上游版本

When a new upstream release tarball **foo-***newvwesion***.tar.gz** is released, the Debian source package can be updated by invoking commands in the old source tree as:

```
$ uscan
 ... foo-newversion.tar.gz downloaded
$ uupdate -v newversion ../foo-newversion.tar.gz
```

- The **debian/watch** file in the old source tree must be a valid one.
- This make symlink **../foo**_*newvwesion***.orig.tar.gz** pointing to **../foo-***newvwesion***.tar.gz**.

- Files are extracted from **../foo-***newvwesion***.tar.gz** to **../foo-***newversion***/**
- Files are copied from **../foo-***oldversion***/debian/** to **../foo-***newvesion***/debian/** .

After the above, you should refresh **debian/patches/*** files (see "第 9.5 节") and update **debian/changelog** with the **dch**(1) command.

When "**debian uupdate**"is specified at the end of line in the **debian/watch** file, **uscan** automatically executes **uupdate**(1) after downloading the tarball.

9.5 Manage patch queue with dquilt

You can add, drop, and refresh **debian/patches/*** files with **dquilt** to manage patch queue.

- **Add** a new patch **debian/patches/***bugname.patch* recording the upstream source modification on the file *buggy_file* as:

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

- **Drop** (== disable) an existing patch
  - Comment out pertinent line in **debian/patches/series**
  - Erase the patch itself (optional)
- **Refresh debian/patches/*** files to make "**dpkg-source -b**"work as expected after updating a Debian package to the new upstream release.

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

  - If conflicts are encountered with "**dquilt push**"in the above, resolve them and run "**dquilt refresh**"manually for each of them.

9.6 构建命令

Here is a recap of popular low level package build commands. There are many ways to do the same thing.

- **dpkg-buildpackage** = 软件包打包工具的核心
- **debuild** = **dpkg-buildpackage** + **lintian** （在清理后的环境变量下构建）
- **schroot** = core of the Debian chroot environment tool
- **sbuild** = **dpkg-buildpackage** on custom **schroot** (build in the chroot)

9.7 Note on sbuild

The **sbuild**(1) command is a wrapper script of **dpkg-buildpackage** which builds Debian binary packages in a chroot environment managed by the **schroot**(1) command. For example, building for Debian **unstable** suite can be done as:

```
$ sudo sbuild -d unstable
```

In **schroot**(1) terminology, this builds a Debian package in a clean ephemeral chroot “**chroot:unstable-amd64-sbuild**”started as a copy of the clean minimal persistent chroot “**source:unstable-amd64-sbuild**” .

This build environment was set up as described in “第 4.6 节”with “**sbuild-debian-developer-setup** -s unstable”which essentially did the following:

```
$ 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 newgrp -
```

The **schroot**(1) configuration for **unstable-amd64-sbuild** was generated at **/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
```

其中：

- The profile defined in the **/etc/schroot/sbuild/** directory is used to setup the chroot environment.
- **/srv/chroot/unstable-amd64-sbuild** directory holds the chroot filesystem.
- **/etc/sbuild/unstable-amd64-sbuild** is symlinked to **/srv/chroot/unstable-amd64-sbuild** .

You can update this source chroot “**source:unstable-amd64-sbuild**”by:

```
$ sudo sbuild-update -udcar unstable
```

You can log into this source chroot “**source:unstable-amd64-sbuild**”by:

```
$ sudo sbuild-shell unstable
```

提示

If your source chroot filesystem is missing packages such as **libeatmydata1**, **ccache**, and **lintian** for your needs, you may want to install these by logging into it.

9.8 Special build cases

The **orig.tar.gz** file may need to be uploaded for a Debian revision other than **0** or **1** under some exceptional cases (e.g., for a security upload).

When an essential package becomes a non-essential one (e.g., **adduser**), you need to remove it manually from the existing chroot environment for its use by **piuparts**.

9.9 上传 orig.tar.gz

当您第一次向归档上传软件包时，您还需要包含原始的 **orig.tar.gz** 源码。

如果 Debian 修订码是 **1** 或者 **0**，这都是默认的。否则，您必须使用带有 **-sa** 选项的 **dpkg-buildpackage** 命令。

- **dpkg-buildpackage -sa**
- **debuild -sa**
- **sbuild**
- 对于“**gbp buildpackage**”，请编辑 **~/.gbp.conf** 文件。

提示

另一方面，**-sd** 选项将会强制排除原始的 **orig.tar.gz** 源码。

提示

添加至 **~/.bashrc** 文件。

9.10 跳过的上传

如果当跳过上传时，你在 **debian/changelog** 中创建了多个条目，你必须创建一个包含自上次上传以来所有变更的 **debian/changelog** 文件。这可以通过指定 dpkg-buildpackage 选项 **-v** 以及上次上传的版本号，比如 *1.2* 来完成。

- **dpkg-buildpackage -v***1.2*
- **debuild -v***1.2*
- **sbuild --debbuildopts -v***1.2*
- 对于 **gbp buildpackage**，请编辑 **~/.gbp.conf** 文件。

9.11 错误报告

The **reportbug**(1) command used for the bug report of *binarypackage* can be customized by the files in **usr/share/bug/***binarypackage***/**.

dh_bugfiles 命令将安装以下位于 **debian/** 目录中的的模板文件。

- **debian/***binarypackage***.bug-control** → **usr/share/bug/***binarypackage***/control**
  - 该文件包含诸如重定向错误报告至其它软件包的一些指导性内容。
- **debian/***binarypackage***.bug-presubj** → **usr/share/bug/***binarypackage***/presubj**
  - 该文件的内容将由 **reportbug** 命令向用户展示。
- **debian/***binarypackage***.bug-script** → **usr/share/bug/***binarypackage* or **usr/share/bug/***binarypackage***/script**
  - **reportbug** 命令运行此脚本以生成错误报告的模板文件。

See **dh_bugfiles**(1) and “reportbug’s Features for Developers (**README.developers**)”

提示

如果您总是需要提醒提交报告的用户某些注意事项或询问他们某些问题，使用这些文件可以将这个过程自动化。

Chapter 10

高级打包

我们来讨论一下 Debian 打包相关的高级内容。

10.1 Historical perspective

Let me oversimplify historical perspective of Debian packaging practices focused on the non-native packaging.

Debian was started in 1990s when upstream packages were available from public FTP sites such as Sunsite. In those early days, Debian packaging used Debian source format currently known as the Debian source format "**1.0**":

- The Debian source package ships a set of files for the Debian source package.
  - *package_version*.**orig.tar.gz** : symlink to or copy of the upstream released file.
  - *package_version-revision*.**diff.gz** : "**One big patch**"for Debian modifications.
  - *package_version-revision*.**dsc** : package description.
- Several workaround approaches such as **dpatch**, **dbs**, or **cdbs** were deployed to manage multiple topic patches.

The modern Debian source format "**3.0 (quilt)**"was invented around 2008 (see "ProjectsDebSrc3.0"):

- The Debian source package ships a set of files for the Debian source package.
  - *package_version*.**orig.tar.?z** : symlink to or copy of the upstream released file.
  - *package_version-revision*.**debian.tar.?z** : tarball of **debian/** for Debian modifications.
    - The **debian/source/format** file contains "**3.0 (quilt)**".
    - Optional multiple topic patches are stored in the **debian/patches/** directory.
  - *package_version-revision*.**dsc** : package description.
- The standardized approach to manage multiple topic patches using **quilt**(1) is deployed for the Debian source format "**3.0 (quilt)**".

Most Debian packages adopted the Debian source formats "**3.0 (quilt)**"and "**3.0 (native)**".

Now, the **git**(1) is popular with upstream and Debian developers. The **git** and its associated tools are important part of the modern Debian packaging workflow. This modern workflow involving **git** will be mentioned later in "第 11 章".

10.2 当前的趋势

当前的 Debian 打包实践和其趋势也随时间不断发展。请参见：

- "Debian Trends" — Hints for "De facto standard"of Debian practices

  - 构建系统：**dh**
  - Debian 源代码格式 :“**3.0 (quilt)**”
  - VCS: **git**
  - VCS Hosting: salsa
  - Rules-Requires-Root: adopted, fakeroot
  - Copyright format: DEP-5

- “**debhelper-compat-upgrade-checklist**(7) manpage” — Upgrade checklist for **debhelper**
- “DEP - Debian Enhancement Proposals” — Formal proposals to enhance Debian

You can also search entire Debian source code data by yourself, too.

- “Debian Sources” — code search tool
  - “Debian Code Search” — wiki page describing its usage
- “Debian Code Search” — another code search tool

10.3 Note on build system

Auto-generated files of the build system may be found in the released upstream tarball. These should be regenerated when Debian package is build. E.g.:

- “**dh $@ --with autoreconf**”should be used in the **debian/rules** if Autotools (**autoconf** + **automake**) are used.

Some modern build system may be able to download required source codes and binary files from arbitrary remote hosts to satisfy build requirements. Don't use this download feature. The official Debian package is required to be build only with packages listed in **Build-Depends:** of the **debian/control** file.

10.4 持续集成

The **dh_auto_test**(1) command is a **debhelper** command that tries to automatically run the test suite provided by the upstream developer during the Debian package building process.

The **autopkgtest**(1) command can be used after the Debian package building process. It tests generated Debian binary packages in the virtual environment using the **debian/tests/control** RFC822-style metadata file as continuous integration (CI). See:

- Documents in the **/usr/share/doc/autopkgtest/** directory
- “4. autopkgtest: Automatic testing for packages”of the “Ubuntu Packaging Guide”

您可以在 Debian 系统上探索使用不同的持续集成系统。

- The Salsa offers “第 11.3 节”.
- **debci** 软件包：建立在 **autopkgtest** 之上的持续集成平台
- **jenkins** 软件包：通用持续集成平台

10.5 自举

Debian cares about supporting new ports or flavours. The new ports or flavours require bootstrapping operation for the cross-build of the initial minimal native-building system. In order to avoid build-dependency loops during bootstrapping, the build-dependency needs to be reduced using the **DEB_BUILD_PROFILES** environment variable.

See Debian wiki: “BuildProfileSpec”.

提示

If a core package **foo** build depends on a package **bar** with deep build dependency chains but **bar** is only used in the **test** target in **foo**, you can safely mark the **bar** with **<!nocheck>** in the **Build-depends** of **foo** to avoid build loops.

10.6 编译加固

The compiler hardening support spreading for Debian **jessie** (8.0) demands that we pay extra attention to the packaging.

您应当详细阅读下列参考内容。

- Debian wiki: “Hardening”
- Debian wiki: “Hardening Walkthrough”

debmake 命令会向 **debian/rules** 文件中按需添加 **DEB_BUILD_MAINT_OPTIONS**、**DEB_CFLAGS_MAINT_APP** 和 **DEB_LDFLAGS_MAINT_APPEND** 的项目（参见“第 5 章”和 **dpkg-buildflags**(1)）。

10.7 可重现的构建

为了做到软件包可重现的构建，这里给出一些相关的建议。

- 不要嵌入基于系统时间的时间戳。
- Don’t embed the file path of the build environment.
- Use “**dh $@**”in the **debian/rules** to access the latest **debhelper** features.
- Export the build environment as “**LC_ALL=C.UTF-8**”(see “第 12.1 节”).
- 对上游源代码中使用的时间戳，使用 debhelper 提供的环境变量 **$SOURCE_DATE_EPOCH** 的值。
- 阅读“可重现构建”了解更多信息。
  - “可重现构建操作方法”。
  - “可重现构建时间戳处理提议”。

Reproducible builds are important for security and quality assurance. They allow independent verification that no vulnerabilities or backdoors have been introduced during the build process.

由 **dpkg-genbuildinfo**(1) 生成的控制文件 *source-name_source-version_arch*.**buildinfo** 记录了构建环境信息。参见 **deb-buildinfo**(5)

10.8 Substvar

debian/control 也定义了软件包的依赖关系，其中“变量替换机制”（substvar）的功能可以用来将软件包维护者从跟踪（大多数简单的）软件包依赖的重复劳动中解放出来。请参见 **deb-substvars**(5)。

debmake 命令支持下列变量替换指令：

- **${misc:Depends}**，可用于所有二进制软件包
- **${misc:Pre-Depends}**，可用于所有 multiarch 软件包
- **${shlibs:Depends}**，可用于所有含有二进制可执行文件或库的软件包
- **${python:Depends}**，可用于所有 Python 软件包

- **${python3:Depends}**，可用于所有 Python3 软件包
- **${perl:Depends}**，用于所有 Perl 软件包
- **${ruby:Depends}**，用于所有 Ruby 软件包

For the shared library, required libraries found simply by "**objdump -p** */path/to/program* **| grep NEEDED**" are covered by the **shlib** substvar.

For Python and other interpreters, required modules found simply looking for lines with "**import**", "**use**", "**require**", etc., are covered by the corresponding substvars.

对其它没有部署属于自己范畴内的变量替换机制的情况，**misc** 变量替换占位符通常用来覆盖对应的依赖替换需求。

对 POSIX shell 程序来说，并没有简单的办法来验证其依赖关系，substvar 的变量替换也无法自动得出它们的依赖。

对使用动态加载机制，包括"GObject introspection"机制的库和模块来说，现在没有简单的方法可以检查依赖关系，变量替换机制也无法自动推导出所需的依赖。

10.9 库软件包

打包软件库需要您投入更多的工作。下面有一些打包软件库的提醒和建议：

- 库二进制软件包必须根据"第 10.17 节"进行命名。
- Debian 按照 **/usr/lib/<triplet>/lib***foo-0.1***.so.***1.0.0* 这样的路径提供共享链接库（参见"第 10.10 节"）。
- Debian 鼓励在共享库中使用带版本的符号（见"第 10.16 节"）。
- Debian 不提供 ***.la** libtool 库归档文件。
- Debian 不推荐使用、提供 ***.a** 静态库文件。

在打包共享库软件之前，请查阅：

- "Chapter 8 - Shared libraries"of the "Debian Policy Manual"
- "10.2 Libraries"of the "Debian Policy Manual"
- "6.7.2. Libraries"of the "Debian Developer's Reference"

如需研究其历史背景，请参见：

- "逃离依赖地狱"[1]
  - 该文档鼓励在共享库中使用带版本的符号。
- "Debian Library Packaging guide"[2]
  - Please read the discussion thread following its announcement, too.

10.10 多体系结构

Debian **wheezy**（7.0，2013 年 5 月）在 **dpkg** 和 **apt** 中引入了对跨架构二进制软件包安装的多架构支持（特别是 **i386** 架构和 **amd64** 架构，但也支持其它的组合），这部分内容值得我们额外关注。

您应当详细阅读下列参考内容。

- Ubuntu 维基（上游）
  - "多架构规范（MultiarchSpec）"

1该文档是在 **symbols** 文件被引入之前写成的。

2在"第六章 - 开发（-DEV）软件包"中，存在强烈的使用含有 SONAME 版本号的 **-dev** 软件包名而非仅使用 **-dev** 作为名称的偏好，但前 ftp-master 成员（Steve Langasek）对此有不同意见。请注意该文档在 **multiarch** 系统和 **symbols** 引入之前写成，可能有一定程度的过时。

- Debian 维基（Debian 的现状）
  - “Debian 多架构支持”
  - “多架构支持/实现（Multiarch/Implementation）”

多架构支持使用三元组（**<triplet>**）的值，例如 **i386-linux-gnu** 和 **x86_64-linux-gnu**；它们出现在共享链接库的安装路径中，例如 **/usr/lib/<triplet>/**，等等。

- 三元组 **<triplet>** 的值由 **debhelper** 脚本隐式提前设置好，软件包维护者无需担心。
- 不过，在 **debian/rules** 文件中用于 **override_dh_*** 目标的三元组 **<triplet>** 值需要由维护者手动进行显式设置。三元组 **<triplet>** 的值可由 **$(DEB_HOST_MULTIARCH)** 变量在 **debian/rules** 文件中获取到，具体方法如下：

```
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)
```

参见：

- “第 9.3 节”
- “第 16.2 节”
- “第 10.12 节”
- “**dpkg-architecture**(1) manpage”

10.11 Debian 二进制软件包的拆分

对行为良好的构建系统来说，对 Debian 二进制包的拆分可以由如下方式实现。

- 为所有二进制软件包在 **debian/control** 文件中创建对应的二进制软件包条目。
- 在对应的 **debian/**二进制软件包名**.install** 文件中列出所有文件的路径（相对于 **debian/tmp** 目录）。

请查看本指南中相关的例子：

- “第 14.11 节”（基于 Autotools）
- “第 14.12 节”（基于 CMake）

An intuitive and flexible method to create the initial template **debian/control** file defining the split of the Debian binary packages is accommodated with the **-b** option. See “第 16.2 节”.

10.12 拆包的场景和例子

对于下面这样的上游源代码示例，我们在这里给出使用 **debmake** 处理时一些典型的 multiarch 软件包拆分的场景和做法：

- 一个软件库源码 *libfoo-1.0***.tar.gz**
- 一个软件工具源码 *bar-1.0***.tar.gz**，软件由编译型语言编写
- 一个软件工具源码 *baz-1.0***.tar.gz**，软件由解释型语言编写

二进制软件包	类型	**Architecture:**	**Multi-Arch:**	软件包内容

二进制软件包	类型	**Architecture:**	**Multi-Arch:**	软件包内容
lib*foo1*	**lib***	any	same	共享库，可共同安装
lib*foo***-dev**	**dev***	any	same	共享库头文件及相关开发文件，可共同安装
lib*foo***-tools**	**bin***	any	foreign	运行时支持程序，不可共同安装
lib*foo***-doc**	**doc***	all	foreign	共享库文档
bar	**bin***	any	foreign	编译好的程序文件，不可共同安装
*bar***-doc**	**doc***	all	foreign	程序的配套文档文件
baz	**script**	all	foreign	解释型程序文件

10.13 Multiarch library path

Debian policy requires to comply with the "Filesystem Hierarchy Standard (FHS), version 3.0", with the exceptions noted in "File System Structure".

The most notable exception is the use of **/usr/lib/<triplet>/** instead of **/usr/lib<qual>/** (e.g., **/lib32/** and **/lib64/**) to support a multiarch library.

Table 10.2 多架构库路径选项

经典路径	**i386** 多体系结构路径	**amd64** 多体系结构路径
/lib/	**/lib/i386-linux-gnu/**	**/lib/x86_64-linux-gnu/**
/usr/lib/	**/usr/lib/i386-linux-gnu/**	**/usr/lib/x86_64-linux-gnu/**

对基于 Autotools 且由 **debhelper**（compat>=9）管理的软件包来说，这些路径设置已由 **dh_auto_configure** 命令自动处理。

对于其它使用不支持的构建系统的软件包，您需要按照下面的方式手动调整安装路径。

- If "**./configure**"is used in the **override_dh_auto_configure** target in **debian/rules**, make sure to replace it with "**dh_auto_configure --**"while re-targeting the install path from **/usr/lib/** to **/usr/lib/$(DEB_HOST_MULTIARCH)/**.
- 请在 **debian/***foo***.install** 文件中将所有出现的 **/usr/lib/** 字符串替换为 **/usr/lib/*/**。

所有启用多架构的软件包安装至相同路径的文件必须内容完全相同。您必须小心处理，避免数据字节序或者压缩算法等等问题带来的文件内容差异。

位于默认路径 **/usr/lib/** 和 **/usr/lib/<triplet>/** 的共享库可被自动加载。

对位于其它路径的共享库，必须使用 **pkg-config** 命令设置 GCC 选项 **-l** 以正确进行加载。

10.14 Multiarch header file path

在支持多架构的 Debian 系统上，GCC 默认会同时包含、使用 **/usr/include/** 和 **/usr/include/<triplet>/** 下的头文件。

如果头文件不在这些路径中，必须使用 **pkg-config** 命令设置 GCC 的 **-I** 参数以使得"**#include <***foo.h***>**"正常工作。

Table 10.3 多架构头文件路径选项

经典路径	**i386** 多体系结构路径	**amd64** 多体系结构路径
/usr/include/	**/usr/include/i386-linux-gnu/**	**/usr/include/x86_64-linux-gnu/**
/usr/include/软件包名**/**	**/usr/include/i386-linux-gnu/**软件包名**/**	**/usr/include/x86_64-linux-gnu/**软件包名**/**
	/usr/lib/i386-linux-gnu/软件包名**/**	**/usr/lib/x86_64-linux-gnu/**软件包名**/**

为库文件使用 **/usr/lib/<triplet>/**软件包名**/** 路径可帮助上游维护者对使用 **/usr/lib/<triplet>** 的多架构系统和使用 **/usr/lib<qual>/** 的双架构系统使用相同的安装脚本。3

使用包含 *packagename* 的文件路径也使得在同一系统上同时安装多个架构的开发库成为可能。

10.15 Multiarch *.pc file path

packagename 用来获取系统上已安装库的信息。它在 ***.pc** 文件中存储配置参数，用来设置 GCC 的 **-I** 和 **-l** 选项。

Table 10.4 *.pc 文件路径选项

经典路径	**i386** 多体系结构路径	**amd64** 多体系结构路径
/usr/lib/pkgconfig/	**/usr/lib/pkgconfig/**	**/usr/lib/x86_64-linux-gnu/pkgconfig/**

10.16 库符号

Debian **lenny**（5.0，2009 年 5 月）中引入的 **dpkg** 符号支持可以帮助我们管理同一共享链接库软件包的向后 ABI 兼容性（backward ABI compatibility）。二进制软件包中的 **DEBIAN/symbols** 文件提供了每个符号及其对应的最小版本号。

一个极其简化的软件库打包流程大概如下所示。

- Extract the old **DEBIAN/symbols** file of the immediate previous binary package with the "**dpkg-deb -e**"command.
  - 或者，**mc** 命令也可以用来解压得到 **DEBIAN/symbols** 文件。
- 将其复制为 **debian/***binarypackage***.symbols** 文件。
  - 如果这是第一次打包的话，可以只创建一个空文件。
- 构建二进制软件包。
  - 如果 **dpkg-gensymbols** 命令警告添加了新的符号的话：
    - Extract the updated **DEBIAN/symbols** file with the "**dpkg-deb -e**"command.
    - 将其中的 Debian 修订版本号，例如 **-1**，从文件中去除。
    - 将其复制为 **debian/***binarypackage***.symbols** 文件。
    - 重新构建二进制软件包。
  - 如果 **dpkg-gensymbols** 命令不报和新链接符号有关的警告：
    - 您已完成了共享库的打包工作。

如需了解详细信息，您应当阅读下列第一手参考资料。

- "8.6.3 The symbols system"of the "Debian Policy Manual"
- "**dh_makeshlibs**(1) manapage"
- "**dpkg-gensymbols**(1) manapage"
- "**dpkg-shlibdeps**(1) manapage"
- "**deb-symbols**(5) manapage"

您也应当查看：

- Debian 维基"使用符号文件"

3This path is compliant with the FHS. "Filesystem Hierarchy Standard: /usr/lib : Libraries for programming and packages" states "Applications may use a single subdirectory under **/usr/lib**. If an application uses a subdirectory, all architecture-dependent data exclusively used by the application must be placed within that subdirectory."

- Debian wiki: “Projects/ImprovedDpkgShlibdeps”
- Debian kde team: “Working with symbols files”
- “第 14.11 节”
- “第 14.12 节”

提示

For C++ libraries and other cases where the tracking of symbols is problematic, follow “8.6.4 The shlibs system”of the “Debian Policy Manual”, instead. Please make sure to erase the empty **debian/***binarypackage***.symbols** file generated by the **debmake** command. For this case, the **DEBIAN/shlibs** file is used.

10.17 Library package name

我们考虑 **lib***foo* 这个库的上游 tarball 源码压缩包的名字从 **lib***foo-7.0***.tar.gz** 更新为了 **lib***foo-8.0***.tar.gz**，同时带有一次 SONAME 大版本的跳跃（并因此影响了其它软件包）。

库的二进制软件包将必须从 **lib***foo7* 重命名为 **lib***foo8* 以保持使用 **unstable** 套件的系统上所有依赖该库的软件包在上传了基于 **lib***foo-8.0***.tar.gz** 的新库后仍然能够正常运行。

警告

如果这个二进制库软件包没有得到更名，许多使用 **unstable** 套件的系统上的各个依赖该库的软件包会在新的库包上传后立刻破损，即便立刻请求进行 binNMU 上传也无法避免这个问题。由于种种原因，binNMU 不可能在上传后立刻开始进行，故无法缓解问题。

-dev 软件包必须遵循以下命名规则：

- 使用不带版本号的 **-dev** 软件包名称：**lib***foo***-dev**
  - 该情况通常适用于依赖关系处于叶节点的库软件包。
  - 仓库内只允许存在一个版本的库源码包。
    - The associated library package needs to be renamed from **lib***foo7* to **lib***foo8* to prevent dependency breakage in the **unstable** suite during the library transition.
  - This approach should be used if the simple binNMU resolves the library dependency quickly for all affected packages. (ABI change by dropping the deprecated API while keeping the active API unchanged.)
  - This approach may still be a good idea if manual code updates, etc. can be coordinated and manageable within limited packages. (API change)
- 使用带版本的 **-dev** 软件包名称：**lib***foo7***-dev** 和 **lib***foo8***-dev**
  - 该情况通常适用于各类重要库软件包。
  - 两个版本的库源码包可同时出现在发行版仓库中。
    - 令所有依赖该库的软件包依赖 **lib***foo***-dev**。
    - 令 **lib***foo7***-dev** 和 **lib***foo8***-dev** 两者都提供 **lib***foo***-dev**。
    - 源码包需要从 **lib***foo-?.0***.tar.gz** 相应地重命名为 **lib***foo7-7.0***.tar.gz** 和 **lib***foo8-8.0***.tar.gz**。
    - 需要仔细选择 **lib***foo7* 和 **lib***foo8* 软件包文件安装时的路径，如头文件等等，以保证它们可以同时安装。
  - 可能的话，不要使用这个重量级的、需要大量人为干预的方法。

- This approach should be used if the update of multiple dependent packages require manual code updates, etc. (API change) Otherwise, the affected packages become RC buggy with FTBFS (Fails To Build From Source).

提示

如果包内数据文件编码方案有所变化（如，从 latin1 变为 utf-8），该场景应比照 API 变化做类似的考虑与处理。

参见“第 10.9 节”。

10.18 库变迁

When you package a new library package version which affects other packages, you must file a transition bug report against the **release.debian.org** pseudo package using the **reportbug** command with the ben file and wait for the approval for its upload from the Release Team.

发行团队提供了“变迁跟踪系统”。参见“变迁（Transition）”。

小心

请确保您按照“第 10.17 节”的描述正确地对二进制软件包进行了重命名。

10.19 binNMU 安全

A “binNMU”is a binary-only non-maintainer upload performed for library transitions etc. In a binNMU upload, only the “**Architecture: any**”packages are rebuilt with a suffixed version number (e.g. version 2.3.4-3 will become 2.3.4-3+b1). The “**Architecture: all**”packages are not built.

The dependency defined in the **debian/control** file among binary packages from the same source package should be safe for the binNMU. This needs attention if there are both “**Architecture: any**”and “**Architecture: all**”packages involved in it.

- “**Architecture: any**”package: depends on “**Architecture: any**”*foo* package
  - **Depends:** *foo* **(= ${binary:Version})**
- “**Architecture: any**”package: depends on “**Architecture: all**”*bar* package
  - **Depends:** *bar* **(= ${source:Version})**
- “**Architecture: all**”package: depends on “**Architecture: any**”*baz* package
  - **Depends:** *baz* **(>= ${source:Version})**, *baz* **(<< ${source:Version}.0~)**

10.20 调试信息

Debian 软件包在构建时都会生成调试信息；但打包生成二进制软件包时，这些打包信息会按照“《Debian 政策手册》”中“第十章 - 文件”的要求进行剥离。

参见

- “6.7.9. Best practices for debug packages”of the “Debian Developer’s Reference”.
- “18.2 Debugging Information in Separate Files”of the “Debugging with gdb”

- “**dh_strip**(1) manapage”
- “**strip**(1) manapage”
- “**readelf**(1) manapage”
- “**objcopy**(1) manapage”
- Debian wiki: “DebugPackage”
- Debian wiki: “AutomaticDebugPackages”
- Debian debian-devel 列表发布的邮件 :“自动调试软件包状态”(2015-08-15)

10.21 -dbgsym package

调试信息由 **dh_strip** 命令的默认行为自动打包并进行剥离。所分离得到的调试软件包名具有 **-dbgsym** 的后缀。

- **debian/rules** 文件不应显式包括 **dh_strip** 。
- 编辑 **debian/control** 文件，在 **Build-Depends** 中写入 **debhelper-compat (>=13)**，同时移除 **Build-Depends** 中对 **debhelper** 的依赖。

10.22 debconf

debconf 软件包可以帮助我们在下列两种情况下配置软件包：

- 在 **debian-installer**（Debian 安装器）预安装时进行非交互式配置。
- interactively from the menu interface (**dialog**, **gnome**, **kde**, ⋯)
  - 软件包安装时：由 **dpkg** 命令调用
  - 对已安装软件包：由 **dpkg-reconfigure** 命令调用

软件包安装时的所有用户交互都必须由这里的 **debconf** 系统进行处理，下列配置文件对这个过程进行控制。

- **debian/***binarypackage***.config**
  - 这是 **debconf config** 脚本，用于向用户询问对于配置软件包必需的问题。
- **debian/***binarypackage***.template**
  - 这是 **debconf templates**（模板）文件，用于向用户询问对于配置软件包必需的问题。

These **debconf** files are called by package configuration scripts in the binary Debian package

- **DEBIAN/***binarypackage***.preinst**
- **DEBIAN/***binarypackage***.prerm**
- **DEBIAN/***binarypackage***.postinst**
- **DEBIAN/***binarypackage***.postrm**

See **dh_installdebconf**(1), **debconf**(7), **debconf-devel**(7) and “3.9.1 Prompting in maintainer scripts” in the “Debian Policy Manual”.

Chapter 11

Packaging with git

Up to “第 10 章”, we focused on packaging operations without using Git or any other VCS. These traditional packaging operations were based on the tarball released by the upstream as mentioned in “第 10.1 节”.

Currently, the **git**(1) command is the de-facto platform for the VCS tool and is the essential part of both upstream development and Debian packaging activities. (See Debian wiki “Debian git packaging maintainer branch formats and workflows”for existing VCS workflows.)

注意

Since the non-native Debian source package uses “**diff -u**”as its backend technology for the maintainer modification, it can’t represent modification involving symlink, file permissions, nor binary data (March 2022 discussion on debian-devel@l.d.o). Please avoid making such maintainer modifications even though these can be recorded in the Git repository.

Since VCS workflows are complicated topic and there are many practice styles, I only touch on some key points with minimal information, here.

Salsa is the remote Git repository service with associated tools. It offers the collaboration platform for Debian packaging activities using a custom GitLab application instance. See:

- “第 11.1 节”
- “第 11.2 节”
- “第 11.3 节”

There are 2 styles of branch names for the Git repository used for the packaging. See “第 11.4 节”.
There are 2 main usage styles for the Git repository for the packaging. See:

- “第 11.5 节”
- “第 11.6 节”

There are 2 notable Debian packaging tools for the Git repository for the packaging.

- **gbp**(1) and its subcommands:
  - This is a tool designed to work with “第 11.5 节”.
  - See “第 11.7 节”.
- **dgit**(1) and its subcommands:
  - This is a tool designed to work with both “第 11.6 节”and “第 11.5 节”.
  - This contains a tool to upload Debian packages using the **dgit** server.
  - See “第 11.8 节”.

11.1 Salsa 存储库

强烈推荐将 Debian 打包用的源代码托管在 Salsa 平台上。目前，超过 90% 的 Debian 源代码打包仓库均托管在 Salsa 上。1

The exact VCS repository hosting an existing Debian source code package can be identified by a metadata field `Vcs-*` which can be viewed with the `apt-cache showsrc <package-name>` command.

11.2 Salsa 账户设置

After signing up for an account on Salsa, make sure that the following pages have the same e-mail address and GPG keys you have configured to be used with Debian, as well as your SSH key:

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

11.3 Salsa 持续集成服务

Salsa runs Salsa CI service as an instance of GitLab CI for “第 10.4 节”.

For every “**git push**”instances, tests which mimic tests run on the official Debian package service can be run by setting Salsa CI configuration file “第 6.13 节”as:

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

# Customizations here
```

11.4 分支名称

The Git repository for the Debian packaging should have at least 2 branches:

- **debian-branch** to hold the current Debian packaging head.
  - old style: **master** (or **debian**, **main**, ⋯)
  - DEP-14 style: **debian/latest**
- **upstream-branch** to hold the upstream releases in the imported form.
  - old style: **upstream**
  - DEP-14 style: **upstream/latest**

In this tutorial, old style branch names are used in examples for simplicity.

> 注意
>
> This **upstream-branch** may need to be created using the tarball released by the upstream independent of the upstream Git repository since it tends to contain automatically generated files.

The upstream Git repository content can co-exit in the local Git repository used for the Debian packaging by adding its copy. E.g.:

1对 git.debian.org 和 alioth.debian.org 的使用已被废弃。

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

This allows easy cherry-picking from the upstream Git repository for bug fixes.

11.5 Patch unapplied Git repository

The patch unapplied Git repository can be summarized as:

- This seems to be the traditional practice as of 2024.
- The source tree matches extracted contents by "**dpkg-source -x --skip-patches**"of the Debian source package.
  - The upstream source is recorded in the Git repository without changes.
  - The maintainer modified contents are confined within the **debian/*** directory.
  - Maintainer changes to the upstream source are recorded in **debian/patches/*** files for the Debian source format "**3.0 (quilt)**".
- This repository style is useful for all variants of traditional workflows and **gdb** based workflow:
  - "第 5.7 节"(no patch)
  - "第 5.10 节"
    - **debian/patches/*** files can also be generated using "**git format-patch**", "**git diff**", or "**gitk**" from **git** commits in the through-away maintainer modification branch or from the upstream unreleased commits.
  - "第 5.11 节"including the last "**dquilt pop -a**"step
  - "第 11.9 节"
- Use helper scripts such as **dquilt**(1) and **gbp-pq**(1) to manage data in **debian/patches/*** files.
  - Add **.pc** line to the **~/.gitignore** file if **dquilt** is used.
  - Add **unapply-patches** and **abort-on-upstream-changes** lines in the **debian/source/local-options** file.
- Use "**dpkg-source -b**"to build the Debian source package.
- Use **dput**(1) to upload the Debian source package.
  - Use "**dgit --gbp push-source**"or "**dgit --gbp push**"instead to upload the Debian package via the **dgit** server (see "**dgit-maint-gbp**(7)").

注意

The **debian/source/local-options** and **debian/source/local-patch-header** files are meant to be recorded by the **git** command. These aren't included in the Debian source package.

11.6 Patch applied Git repository

The patch applied Git repository can be summarized as:

- The source tree matches extracted contents by "**dpkg-source -x**"of the Debian source package.
  - The source tree is buildable and the same as what NMU maintainers see.
  - The source is recorded in the Git repository with maintainer changes including the **debian/** directory.
  - Maintainer changes to the upstream source are also recorded in **debian/patches/*** files for the Debian source format "**3.0 (quilt)**".

Use one of workflow styles:

- **dgit-maint-merge**(7) workflow.
  - Use this if you don't intend to record topic patches in the Debian source package.
  - Good enough for packages only with trivial modifications to the upstream.
  - Only choice for packages with intertwined modification histories to the upstream
  - Add **auto-commit** and **single-debian-patch** lines in the **debian/source/local-options** file
  - Use "**git checkout upstream; git pull**"to pull the new upstream commit and use "**git checkout master ; git merge <new-version-tag>**"to merge it to the **master** branch.
  - Use "**dpkg-source -b**"to build the Debian source package.
  - Use "**dgit push-source**"or "**dgit push**"for uploading the Debian package via the **dgit** server.
  - See "第 5.12 节"for example.
- **dgit-maint-debrebase**(7) workflow.
  - Use this if you wish to commit maintainer changes to the patch applied Git repository with the same granularity as patches of "第 11.9 节".
  - Good for packages with multiple sequenced modifications to the upstream.
  - Use "**dgit build-source**"to build the Debian source package.
  - Use "**dgit push-source**"or "**dgit push**"for uploading the Debian package via the **dgit** server.
  - Details of this workflow are beyond the scope of this tutorial document. See "第 11.12 节"for more.

11.7 Note on gbp

The **gbp** command is provided by the **git-buildpackage** package.

- This command is designed to manage contents of "第 11.5 节"efficiently.
- Use "**gbp import-orig**"to import the new upstream tar to the git repository.
  - The "**--pristine-tar**"option for the "**git import-orig**"command enables storing the upstream tarball in the same git repository.
  - The "**--uscan**"option as the last argument of the "**gbp import-orig**"command enables downloading and committing the new upstream tarball into the git repository.
- Use "**gbp import-dsc**"to import the previous Debian source package to the git repository.
- Use "**gbp dch**"to generate the Debian changelog from the git commit messages.
- Use "**gbp buildpackage**"to build the Debian binary package from the git repository.
  - The **sbuild** package can be used as its clean chroot build backend either by configuration or adding "**--git-builder='sbuild -A -s --source-only-changes -v -d unstable'**"

- Use “**gbp pull**”to update the **debian**, **upstream** and **pristine-tar** branches safely from the remote repository.
- Use “**gbp pq**”to manage quilt patches without using **dquilt** command.
- Use “**gbp clone** *REPOSITORY_URL*”to clone and set up tracking branches for **debian**, **upstream** and **pristine-tar**.

Package history management with the **git-buildpackage** package is becoming the standard practice for many Debian maintainers. See more at:

- “使用 git-buildpackage 构建 Debian 软件包”
- “4 tips to maintain a “3.0 (quilt)”Debian source package in a VCS”
- The **systemd** packaging practice documentation on “Building from source”
- The workflow mentioned in **dgit-maint-gbp**(7) which enables to use this **gbp** with **dgit**

11.8 Note on dgit

The **dgit** command is provided by the **dgit** package.

- This command is designed to manage contents of “第 11.6 节”efficiently.
  - This enables to access the Debian package repository as if it is a **git** remote repository.
- This command supports uploading Debian packages using the **dgit** server from both “第 11.5 节” and “第 11.6 节”.

The new **dgit** package offers commands interact with the Debian repository as if it was a git repository. It does not replace **gbp-buildpackage** and both can be used at the same time. Using plain **gbp-buildpackage** is recommended for developers who want to run git push/pull on Salsa and use things such as Salsa CI or Merge Requests on Salsa.

For more details see the extensive guides:

- **dgit-maint-gbp**(7) — for the Debian source format “**3.0 (quilt)**”package with its Debian Git repository which is kept usable also for people using **gbp-buildpackage**(1) using “第 11.5 节”.
- **dgit-maint-merge**(7) — for the Debian source format “**3.0 (quilt)**”package with its changes flowing both ways between the upstream Git repository and the Debian Git repository which are tightly coupled using “第 11.6 节”.
- **dgit-maint-debrebase**(7) — for the Debian source format “**3.0 (quilt)**”package with its changes flowing mostly one way from the upstream Git repository to the Debian Git repository using “第 11.6 节”.
- **dgit-maint-native**(7) — for the Debian source format “**3.0 (native)**”package in the Debian Git repository. (No maintainer changes)

The **dgit**(1) command can push the easy-to-trace change history to the https://browse.dgit.debian.org/- site and can upload Debian package to the Debian repository properly without using **dput**(1).

The concept around **dgit** is beyond this tutorial document. Please start reading relevant information:

- “dgit: use the Debian archive as a git remote (2015)”
- “tag2upload (2023)”

11.9 Patch by “gbp-pq”approach

For “第 11.5 节”, you can generate **debian/patches/*** files using the **gbp-pq**(1) command from **git** commits in the through-away **patch-queue** branch.

Unlike **dquilt** which offers similar functionality as seen “第 5.11 节”and “第 9.5 节”, **gbp-pq** doesn’t use **.pc/*** files to track patch state, but instead **gbp-pq** utilizes temporary branches in git.

11.10 Manage patch queue with gbp-pq

You can add, drop, and refresh **debian/patches/*** files with **gbp-pq** to manage patch queue.

If the package is managed in "第 11.5 节"using the **git-buildpackage** package, you can revise the upstream source to fix bug as the maintainer and release a new Debian revision using **gbp pq**.

- **Add** a new patch recording the upstream source modification on the file *buggy_file* as:

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

- **Drop** (== disable) an existing patch
  - Comment out pertinent line in **debian/patches/series**
  - Erase the patch itself (optional)
- **Refresh debian/patches/*** files to make "**dpkg-source -b**"work as expected after updating a Debian package to the new upstream release.

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

11.11 gbp import-dscs --debsnap

For Debian source packages named "<*source-package*>"recorded in the snapshot.debian.org archive, an initial git repository managed in "第 11.5 节"with all of the Debian version history can be generated as follows.

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

11.12 Note on dgit-maint-debrebase workflow

Here are some hints around **dgit-maint-debrebase**(7). 2

- Use “**dgit setup-new-tree**”to prepare the local **git** working repository.
- The first maintainer modification commit should contain files only in the **debian/** directory excluding files in the **debian/patches** directory.
- **debian/patches/*** files are generated from the maintainer modification commit history using the “**dgit quilt-fixup**”command automatically invoked from “**dgit build**”and “**dgit push**”.
- Use “**git-debrebase new-version <new-version-tag>**”to rebase the maintainer modification commit history with automatically updated **debian/changelog**.
- Use “**git-debrebase conclude**”to make a new pseudomerge (== “**git merge -s ours**”) to record Debian package with clean ff-history.

See **dgit-maint-debrebase**(7), **dgit**(1) and **git-debrebase**(1) for more.

11.13 Quasi-native Debian packaging

The **quasi-native** packaging scheme packages a source without the real upstream tarball using the **non-native** package format.

提示

Some people promote this **quasi-native** packaging scheme even for programs written only for Debian since it helps to ease communication with the downstream distros such as Ubuntu for bug fixes etc.

This **quasi-native** packaging scheme involves 2 preparation steps:

- Organize its source tree almost like **native** Debian package (see “第 6.4 节”) with **debian/*** files with a few exceptions:
  - Make **debian/source/format** to contain “**3.0 (quilt)**”instead of “**3.0 (native)**”.
  - Make **debian/changelog** to contain *version-revision* instead of *version* .
- Generate missing upstream tarball preferably without **debian/*** files.
  - For Debian source format “**3.0 (quilt)**”, removal of files under **debian/** directory is an optional action.

The rest is the same as the **non-native** packaging workflow as written in “第 6.1 节”.

Although this can be done in many ways (“第 16.4 节”), you can use the Git repository and “**git deborig**”as:

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

2I may be incorrect, here.

Chapter 12

提示

Please also read insightful pages linked from “Notes on Debian”by Russ Allbery (long time Debian developer) which have best practices for advanced packaging topics.

12.1 在 UTF-8 环境下构建

构建环境的默认语言环境是 **C**。

某些程序（如 Python3 的 **read** 函数）会根据区域设置改变行为。

添加以下代码到 **debian/rules** 文件可以确保程序使用 **C.UTF-8** 的区域语言设置（locale）进行构建。

```
LC_ALL := C.UTF-8
export LC_ALL
```

12.2 UTF-8 转换

If upstream documents are encoded in old encoding schemes, converting them to UTF-8 is a good idea.

Use the **iconv** command in the **libc-bin** package to convert the encoding of plain text files.

```
$ iconv -f latin1 -t utf8 foo_in.txt > foo_out.txt
```

使用 **w3m**(1) 将 HTML 文件转换为 UTF-8 纯文本文件。执行此操作时，请确保在 UTF-8 语言环境下执行它。

```
$ 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
```

在 **debian/rules** 文件的 **override_dh_*** 目标中运行这些脚本。

12.3 Hints for Debugging

当您遇到构建问题或者生成的二进制程序核心转储时，您需要自行解决他们。这就是除错（**debug**）。

This is too deep a topic to describe here. So, let me just list few pointers and hints for some typical debug tools.

- Wikipedia: “core dump”
  - “**man core**”
  - Update the “**/etc/security/limits.conf**”file to include the following:

    ```
    * soft core unlimited
    ```

  - “**ulimit -c unlimited**”in **~/.bashrc**
  - “**ulimit -a**”to check

  - Press **Ctrl-** or “**kill -ABRT** ’PID’”to make a core dump file

- **gdb** - The GNU Debugger
  - “**info gdb**”
  - “Debugging with GDB”in **/usr/share/doc/gdb-doc/html/gdb/index.html**
- **strace** - 跟踪系统调用和信号
  - 使用 **/usr/share/doc/strace/examples/** 中的 **strace-graph** 脚本来建立一个好看的树形图
  - “**man strace**”
- **ltrace** - 跟踪库调用
  - “**man ltrace**”
- “**sh -n** *script.sh*”- Syntax check of a Shell script
- “**sh -x** *script.sh*”- Trace a Shell script
- “**python3 -m py_compile** *script.py*”- Syntax check of a Python script
- “**python3 -mtrace --trace** *script.py*”- Trace a Python script
- “**perl -I ../libpath -c** *script.pl*”- Syntax check of a Perl script
- “**perl -d:Trace** *script.pl*”- Trace a Perl script
  - 安装 **libterm-readline-gnu-perl** 软件包或者同类型软件来添加输入行编辑功能与历史记录支持。
- **lsof** - 按进程列出打开的文件
  - “**man lsof**”

提示

script 命令能帮助记录控制台输出。

提示

在 **ssh** 命令中搭配使用 **screen** 和 **tmux** 命令，能够提供安全并且强健的远程连接终端。

提示

libreply-perl（新的）软件包和来自 **libdevel-repl-perl**（旧的）软件包的 **re.pl** 命令为 Perl 提供了一个类似 Python 和 Shell 的 REPL (=READ + EVAL + PRINT + LOOP) 环境。

提示

The **rlwrap** and **rlfe** commands add input line editing capability with history support to any interactive commands. E.g. “**rlwrap dash -i**”.

Chapter 13

工具的使用

这里列出与 Debian 打包相关的值得注意的工具。

注意

The descriptions in this section are intentionally brief. Prospective maintainers are strongly encouraged to search for and read all relevant documentation associated with these commands.

注意

Examples here use the **gz**-compression. The **xz**-compression may be used instead.

13.1 debdiff

您可以使用 **debdiff** 命令来对比两个 Debian 软件包内容的差别。

```
$ debdiff old-package.dsc new-package.dsc
```

您也可以使用 **debdiff** 命令来对比两组二进制 Debian 软件包中的文件列表。

```
$ debdiff old-package.changes new-package.changes
```

当检查源代码包中哪些文件被修改时，这个命令非常有用。它还可以用来检测二进制包中是否有文件在更新过程中发生变动，比如被意外替换或删除。

Debian now enforces the source-only upload when developing packages. So there may be 2 different ***.changes** files:

- *package_version-revision*_**source.changes** for the normal source-only upload
- *package_version-revision_arch***.changes** for the binary upload

13.2 dget

您可以使用 **dget** 命令来下载 Debian 源包的文件集。

```
$ dget https://www.example.org/path/to/package_version-rev.dsc
```

13.3 mk-origtargz

You can make the upstream tarball **../foo-***newversion***.tar.[xg]z** accessible from the Debian source tree as **../foo**_*newversion*.**orig.tar.[xg]z**. This command is useful for renaming and symlinking the upstream tarball to the expected Debian naming convention.

13.4 origtargz

You can fetch the pre-existing orig tarball of a Debian package from various sources, and unpack it with **origtargz** command.

This is basically for **-2**, **-3**, ⋯ revisions.

13.5 git deborig

If the upstream project is hosted in a Git repository without an official tarball release, you can generate its orig tarball from the **git** repository for use by the Debian source package. Execute “git deborig”from the root of the checked-out source tree.

This is basically for **-1** revisions.

13.6 dpkg-source -b

The “**dpkg-source -b**”command packs the upstream source tree into the Debian source package.

It expects a series of patches in the **debian/patches/** directory and their application sequence in **debian/patches/series**.

It is compatible with **dquilt** (see “第 4.4 节”) operations and understands the patch application status from the existence of **.pc/applied-patches**.

The **dpkg-buildpackage** command invokes “**dpkg-source -b**”.

13.7 dpkg-source -x

The “**dpkg-source -x**”command extracts the source tree and applies the patches in the **debian/patches/** directory using the sequence specified in **debian/patches/series** to the upstream source tree. It also adds **.pc/applied-patches**. (See “第 11.6 节”.)

The “**dpkg-source -x --skip-patches**”command extracts source tree only. It doesn’t add **.pc/applied-patches**. (See “第 11.5 节”.)

Both extracted source trees are ready for building Debian binary packages with **dpkg-buildpackage**, **dbuild**, **sbuild**, etc..

13.8 debc

您应该使用 **debc** 命令安装生成的软件包以在本地对齐进行测试。

```
$ debc package_version-rev_arch.changes
```

13.9 piuparts

您应该使用 **piuparts** 命令安装生成的软件包以自动进行测试。

```
$ sudo piuparts package_version-rev_arch.changes
```

注意

这是一个非常慢的过程，因为它需要访问远程 APT 软件包仓库。

13.10 bts

After uploading the package, you will receive bug reports. It is an important duty of a package maintainer to manage these bugs properly, as described in “5.8. Handling bugs”of the “Debian Developer’s Reference”.

bts 命令是一个用以处理“Debian 缺陷追踪系统”上的错误的便捷工具。

```
$ bts severity 123123 wishlist , tags -1 pending
```

Chapter 14

更多示例

There is an old Latin saying: “**fabricando fit faber**”(“practice makes perfect”).
强烈建议使用简单的包来练习和试验 Debian 打包的所有步骤。本章为您的练习提供了许多上游案例。
This should also serve as introductory examples for many programming topics.

- 使用 POSIX shell，Python3 和 C 编程。
- 使用图标图形创建桌面 GUI 程序启动器的方法。
- Conversion of a command from CLI to GUI.
- Conversion of a program to use **gettext** for internationalization and localization: POSIX shell and C sources.
- 构建系统概述：Makefile、Python、Autotools 以及 CMake。

请注意，Debian 对以下事项非常注意：

- 自由软件
- 操作系统的稳定性与安全性
- 通过以下方式以实现通用操作系统：
  - 上游源码的自由选择，
  - CPU 架构的自由选择，以及
  - 用户界面语言的自由选择。

在“第 5 章”中介绍的典型打包示例是本章节的先决条件。
在以下数小节中，有些细节被刻意模糊。请尝试阅读相关文件，并且尝试自行厘清它们。

提示

The best source of a packaging example is the current Debian archive itself. Please use the “Debian Code Search”service to find pertinent examples.

14.1 挑选最好的模板

Here is an example of creating a simple Debian package from a zero-content source in an empty directory.
This is a good way to obtain all the template files without cluttering the upstream source tree you are working on.
让我们假设这个空目录为 **debhello-0.1**。

```
 $ mkdir debhello-0.1
 $ tree
.
+-- debhello-0.1

2 directories, 0 files
```

Let's generate the maximum amount of template files.

Let's also use the "**-p** *debhello* **-t -u** *0.1* **-r** *1*"options to create the missing upstream tarball with default **-x3** and **T** options.

```
 $ cd /path/to/debhello-0.1
 $ debmake -p debhello -t -u 0.1 -r 1
I: set parameters
 ...
```

我们来检查一下自动产生的模板文件。

```
 $ cd /path/to
 $ tree
.
+-- debhello-0.1
|   +-- debian
|       +-- README.Debian
|       +-- README.source
|       +-- changelog
|       +-- clean
|       +-- control
|       +-- copyright
|       +-- debhello.bug-control.ex
|       +-- debhello.bug-presubj.ex
|       +-- debhello.bug-script.ex
|       +-- debhello.conffiles.ex
|       +-- debhello.cron.d.ex
|       +-- debhello.cron.daily.ex
|       +-- debhello.cron.hourly.ex
|       +-- debhello.cron.monthly.ex
|       +-- debhello.cron.weekly.ex
|       +-- debhello.default.ex
|       +-- debhello.emacsen-install.ex
|       +-- debhello.emacsen-remove.ex
|       +-- debhello.emacsen-startup.ex
|       +-- debhello.lintian-overrides.ex
|       +-- debhello.service.ex
|       +-- debhello.tmpfile.ex
|       +-- dirs
|       +-- gbp.conf
|       +-- install
|       +-- links
|       +-- maintscript.ex
|       +-- manpage.1.ex
|       +-- manpage.asciidoc.ex
|       +-- manpage.md.ex
|       +-- manpage.sgml.ex
|       +-- manpage.xml.ex
|       +-- patches
|       |   +-- series
|       +-- postinst.ex
|       +-- postrm.ex
|       +-- preinst.ex
|       +-- prerm.ex
|       +-- rules
|       +-- salsa-ci.yml
|       +-- source
```

```
|       |   +-- format
|       |   +-- lintian-overrides.ex
|       |   +-- local-options.ex
|       |   +-- local-patch-header.ex
|       |   +-- options.ex
|       |   +-- patch-header.ex
|       +-- tests
|       |   +-- control
|       +-- upstream
|       |   +-- metadata
|       +-- watch
+-- debhello-0.1.tar.xz
+-- debhello_0.1.orig.tar.xz -> debhello-0.1.tar.xz

7 directories, 50 files
```

现在，您可以复制 *debhello-0.1/***debian/** 目录下所有生成的模板文件到您的软件包中。

14.2 无 Makefile（shell，命令行界面）

此处是一个从 POSIX shell 命令行界面程序创建简单的 Debian 软件包的示例，我们假设它没有使用任何构建系统。

让我们假设上游的源码包为 **debhello-0.2.tar.gz**。

此类源码不具有自动化方法，所以必须手动安装文件。

例如：

```
 $ tar -xzmf debhello-0.2.tar.gz
 $ cd debhello-0.2
 $ sudo cp scripts/hello /bin/hello
 ...
```

Let's get this source as tar file from a remote site and make it the Debian package.

下载 **debhello-0.2.tar.gz**

```
 $ wget http://www.example.org/download/debhello-0.2.tar.gz
 ...
 $ tar -xzmf debhello-0.2.tar.gz
 $ tree
.
+-- debhello-0.2
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-0.2.tar.gz

5 directories, 6 files
```

这里的 POSIX shell 脚本 **hello** 非常的简单。

hello（**v=0.2**）

```
 $ 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
```

此处的 **hello.desktop** 支持“桌面项（Desktop Entry）规范”。

hello.desktop（**v=0.2**）

```
 $ 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;
```

此处的 **hello.png** 是图标的图像文件。
让我们使用 **debmake** 命令来打包。这里使用 **-b':sh'** 选项来指明生成的二进制包是一个 shell 脚本。

```
 $ cd /path/to/debhello-0.2
 $ debmake -b':sh' -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="0.2", rev="1"
I: *** start packaging in "debhello-0.2". ***
I: provide debhello_0.2.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-0.2.tar.gz debhello_0.2.orig.tar.gz
I: pwd = "/path/to/debhello-0.2"
I: parse binary package settings: :sh
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: analyze the source tree
I: build_type = Unknown
I: scan source for copyright+license text and file extensions
I:  25 %, ext = md
 ...
```

让我们来检查一下自动产生的模板文件。
执行基本的 **debmake** 命令后的源码树。（**v=0.2**）

```
 $ cd /path/to
 $ tree
.
+-- debhello-0.2
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- debian
|   |   +-- README.Debian
|   |   +-- README.source
|   |   +-- changelog
|   |   +-- clean
|   |   +-- control
|   |   +-- copyright
|   |   +-- dirs
|   |   +-- gbp.conf
|   |   +-- install
|   |   +-- links
|   |   +-- patches
|   |   |   +-- series
|   |   +-- rules
|   |   +-- salsa-ci.yml
|   |   +-- source
|   |   |   +-- format
|   |   |   +-- local-options.ex
```

```
|   |   |   +-- local-patch-header.ex
|   |   +-- tests
|   |   |   +-- control
|   |   +-- upstream
|   |   |   +-- metadata
|   |   +-- watch
|   +-- man
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-0.2.tar.gz
+-- debhello_0.2.orig.tar.gz -> debhello-0.2.tar.gz

10 directories, 26 files
```

debian/rules（模板文件，**v=0.2**）：

```
 $ cd /path/to/debhello-0.2
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1

%:
        dh $@
```

这基本上是带有 **dh** 命令的标准 **debian/rules** 文件。因为这是个脚本软件包，所以这个 **debian/rules** 模板文件没有与构建标记（build flag）相关的内容。

debian/control（模板文件，**v=0.2**）：

```
 $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.0
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/debhello

Package: debhello
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.
```

Since this is the shell script package, the **debmake** command sets “**Architecture: all**”and “**Multi-Arch: foreign**”. Also, it sets required **substvar** parameters as “**Depends: ${misc:Depends}**”. These are explained in “第 6 章”.

因为这个上游源码缺少上游的 **Makefile**，所以这个功能需要由维护者提供。这个上游源码仅包含脚本文件和数据文件，没有 C 的源码文件，因此 构建（**build**）的过程可以被跳过，但是需要实现 安装（**install**）的过程。对于这种情况，通过添加 **debian/install** 和 **debian/manpages** 文件可以很好地实现这一功能，且不会使 **debian/rules** 文件变得复杂。

作为维护者，我们要把这个 Debian 软件包做得更好。

debian/rules（维护者版本，**v=0.2**）：

```
 $ cd /path/to/debhello-0.2
 $ vim debian/rules
 ... hack, hack, hack, ...
```

```
 $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
        dh $@
```

debian/control（维护者版本，**v=0.2**）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.6.2
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)
```

警告

If you leave “**Section: unknown**”in the template **debian/control** file unchanged, the **lintian** error may cause a build failure.

debian/install（维护者版本，**v=0.2**）：

```
 $ vim debian/install
 ... hack, hack, hack, ...
 $ cat debian/install
data/hello.desktop usr/share/applications
data/hello.png usr/share/pixmaps
scripts/hello usr/bin
```

debian/manpages（维护者版本，**v=0.2**）：

```
 $ vim debian/manpages
 ... hack, hack, hack, ...
 $ cat debian/manpages
man/hello.1
```

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。
debian/ 目录下的模板文件。（**v=0.2**）：

```
 $ rm -f debian/clean debian/dirs debian/links
 $ rm -f debian/README.source debian/source/*.ex
 $ rm -rf debian/patches
 $ tree -F debian
debian/
+-- README.Debian
```

```
+-- changelog
+-- control
+-- copyright
+-- gbp.conf
+-- install
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 13 files
```

您可以在此源代码树中使用 **debuild** 命令（或其等效命令）创建非原生的 Debian 软件包。如下所示，该命令的输出非常详细，并且解释了它所做的事。

```
 $ cd /path/to/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.
```

现在我们来看看成果如何。

通过 **debuild** 生成的第 **0.2** 版的 **debhello** 文件：

```
 $ cd /path/to
 $ tree -FL 1
./
+-- debhello-0.2/
+-- debhello-0.2.tar.gz
+-- 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.gz -> debhello-0.2.tar.gz

2 directories, 8 files
```

您可以看见生成的全部文件。

- **debhello_0.2.orig.tar.gz** 是指向上游源码包的符号链接。
- **debhello_0.2-1.debian.tar.xz** 包含了维护者生成的内容。
- **debhello_0.2-1.dsc** 是 Debian 源码包的元数据文件。
- The **debhello_0.2-1_all.deb** 是 Debian 二进制软件包。
- **debhello_0.2-1_amd64.build** 是 Debian 二进制软件包。
- **debhello_0.2-1_amd64.buildinfo** 文件是由 **dpkg-genbuildinfo**(1) 自动生成的元文件。
- **debhello_0.2-1_amd64.changes** 是 Debian 二进制软件包的元数据文件。

debhello_0.2-1.debian.tar.xz 包含了 Debian 对上游源代码的修改，具体如下所示。
压缩过的归档文件 **debhello_0.2-1.debian.tar.xz** 中的内容物：

```
 $ tar -tzf debhello-0.2.tar.gz
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
 $ tar --xz -tf debhello_0.2-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
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
```

debhello_0.2-1_amd64.deb 包含了将要安装至系统中的文件，如下所示。
debhello_0.2-1_all.deb 二进制软件包中的内容：

```
 $ 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
```

此处是生成的 **debhello_0.2-1_all.deb** 的依赖项列表。
debhello_0.2-1_all.deb 的依赖项列表：

```
$ dpkg -f debhello_0.2-1_all.deb pre-depends \
          depends recommends conflicts breaks
```

(No extra dependency packages required since this is a POSIX shell program.)

注意

If you wish to replace upstream provided PNG file **data/hello.png** with maintainer provided one **debian/hello.png**, editing **debian/install** isn't enough. When you add **debian/hello.png**, you need to add a line "include-binaries"to **debian/source/options** since PNG is a binary file. See **dpkg-source**(1).

14.3 Makefile（shell，命令行界面）

下面是从 POSIX shell 命令行界面程序创建简单的 Debian 软件包的示例，我们假设它使用 **Makefile** 作为构建系统。
让我们假设上游的源码包为 **debhello-1.0.tar.gz**。
这一类源代码设计可以用这样的方式安装成为非系统文件：

```
$ tar -xzmf debhello-1.0.tar.gz
$ cd debhello-1.0
$ make install
```

Debian packaging requires changing this "**make install**"process to install files to the target system image location instead of the normal location under **/usr/local**.
让我们取得源码并制作 Debian 软件包。
下载 **debhello-1.0.tar.gz**

```
$ wget http://www.example.org/download/debhello-1.0.tar.gz
...
$ tar -xzmf debhello-1.0.tar.gz
$ tree
.
+-- debhello-1.0
|   +-- Makefile
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-1.0.tar.gz

5 directories, 7 files
```

这里的 **Makefile** 正确使用 **$(DESTDIR)** 和 **$(prefix)**。其他的所有文件都和"第 14.2 节"中的一样，并且大多数的打包工作也都一样。
Makefile（**v=1.0**）

```
$ 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
```

让我们使用 **debmake** 命令来打包。这里使用 **-b':sh'** 选项来指明生成的二进制包是一个 shell 脚本。

```
 $ cd /path/to/debhello-1.0
 $ debmake -b':sh' -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.0", rev="1"
I: *** start packaging in "debhello-1.0". ***
I: provide debhello_1.0.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.0.tar.gz debhello_1.0.orig.tar.gz
I: pwd = "/path/to/debhello-1.0"
I: parse binary package settings: :sh
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I:  25 %, ext = md
 ...
```

让我们来检查一下自动产生的模板文件。

debian/rules（模板文件，**v=1.0**）：

```
 $ cd /path/to/debhello-1.0
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1

%:
        dh $@

#override_dh_auto_install:
#       dh_auto_install -- prefix=/usr

#override_dh_install:
#       dh_install --list-missing -X.pyc -X.pyo
```

作为维护者，我们要把这个 Debian 软件包做得更好。
debian/rules（维护者版本，**v=1.0**）：

```
 $ cd /path/to/debhello-1.0
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ cat debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
        dh $@

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

因为上游源码含有正确的上游 **Makefile** 文件，所以没有必要再去创建 **debian/install** 和 **debian/-manpages** 文件。
debian/control 文件和“第 14.2 节”中的完全一致。
在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。
debian/ 目录下的模板文件。（**v=1.0**）：

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

4 directories, 11 files
```

其余的打包操作基本上和“第 14.2 节”中的相同。

14.4 pyproject.toml (Python3, CLI)

Here is an example of creating a simple Debian package from a Python3 CLI program using **pyproject.toml**.
让我们取得源码并制作 Debian 软件包。
下载 **debhello-1.1.tar.gz**

```
 $ wget http://www.example.org/download/debhello-1.1.tar.gz
 ...
 $ tar -xzmf debhello-1.1.tar.gz
 $ 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.gz

6 directories, 10 files
```

Here, the content of this **debhello** source tree as follows.

pyproject.toml (v=1.1) — PEP 517 configuration

```
 $ 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 = {file = "LICENSE.txt"}
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",
  "License :: OSI Approved :: MIT License",
  "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) — for tar-ball.

```
 $ cat debhello-1.1/MANIFEST.in
include data/*
include manpages/*
```

src/debhello/__init__.py (v=1.1)

```
 $ cat debhello-1.1/src/debhello/__init__.py
"""
debhello program (CLI)
"""
```

src/debhello/main.py (v=1.1) — command entry point

```
 $ 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())
```

让我们使用 **debmake** 命令来打包。这里使用 **-b':py3'** 选项来指明生成的二进制包包含 Python3 脚本和模块文件。

```
 $ cd /path/to/debhello-1.1
 $ debmake -b':py3' -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.1", rev="1"
I: *** start packaging in "debhello-1.1". ***
I: provide debhello_1.1.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.1.tar.gz debhello_1.1.orig.tar.gz
I: pwd = "/path/to/debhello-1.1"
I: parse binary package settings: :py3
I: binary package=debhello Type=python3 / Arch=all M-A=foreign
I: analyze the source tree
W: setuptools build system.
I: build_type = Python (pyproject.toml: PEP-518, PEP-621, PEP-660)
I: scan source for copyright+license text and file extensions
 ...
```

让我们来检查一下自动产生的模板文件。

debian/rules（模板文件，**v=1.1**）：

```
 $ cd /path/to/debhello-1.1
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1

%:
        dh $@ --with python3 --buildsystem=pybuild
```

这基本上是带有 **dh** 命令的标准 **debian/rules** 文件。

The use of the "**--with python3**"option invokes **dh_python3** to calculate Python dependencies, add maintainer scripts to byte compiled files, etc. See **dh_python3**(1).

The use of the "**--buildsystem=pybuild**"option invokes various build systems for requested Python versions in order to build modules and extensions. See **pybuild**(1).

debian/control（模板文件，**v=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: 4.7.0
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/debhello

Package: debhello
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.
```

Since this is the Python3 package, the **debmake** command sets "**Architecture: all**"and "**Multi-Arch: foreign**". Also, it sets required **substvar** parameters as "**Depends: ${python3:Depends}, ${misc:Depends}**". These are explained in "第 6 章".

作为维护者，我们要把这个 Debian 软件包做得更好。

debian/rules（维护者版本，**v=1.1**）：

```
 $ cd /path/to/debhello-1.1
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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（维护者版本，**v=1.1**）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ 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: 4.6.2
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)'.
```

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。

This **debhello** command comes with the upstream-provided manpage and desktop file but the upstream **pyproject.toml** doesn't install them. So you need to update **debian/install** and **debian/manpages** as follows:

debian/install (maintainer version, v=1.1):

```
 $ vim debian/copyright
 ... hack, hack, hack, ...
 $ 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 (maintainer version, v=1.1):

```
 $ vim debian/install
 ... hack, hack, hack, ...
 $ cat debian/install
data/hello.desktop usr/share/applications
data/hello.png usr/share/pixmaps
```

其余的打包工作与"第 14.3 节"中的几乎一致。

debian/ 目录下的模板文件。（**v=1.1**）：

```
 $ rm -f debian/clean debian/dirs debian/links
 $ rm -f debian/README.source debian/source/*.ex
 $ rm -rf debian/patches
 $ tree -F debian
debian/
+-- README.Debian
+-- changelog
```

```
+-- control
+-- copyright
+-- gbp.conf
+-- install
+-- manpages
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 13 files
```

此处是生成的 **debhello_1.1-1_all.deb** 包的依赖项列表。
debhello_1.1-1_all.deb 的依赖项列表：

```
 $ dpkg -f debhello_1.1-1_all.deb pre-depends \
           depends recommends conflicts breaks
Depends: python3:any
```

14.5 Makefile（shell，图形界面）

此处是一个从 POSIX shell 图形界面程序构建简单的 Debian 软件包的示例，我们假设程序使用 **Makefile** 作为构建系统。
上游是基于“第 14.3 节”中的源代码，并带有增强的图形界面支持。
让我们假设上游的源码包为 **debhello-1.2.tar.gz**。
让我们取得源码并制作 Debian 软件包。
下载 **debhello-1.2.tar.gz**

```
 $ wget http://www.example.org/download/debhello-1.2.tar.gz
 ...
 $ tar -xzmf debhello-1.2.tar.gz
 $ tree
.
+-- debhello-1.2
|   +-- Makefile
|   +-- README.md
|   +-- data
|   |   +-- hello.desktop
|   |   +-- hello.png
|   +-- man
|   |   +-- hello.1
|   +-- scripts
|       +-- hello
+-- debhello-1.2.tar.gz

5 directories, 7 files
```

此处的 **hello** 已经被重写以便使用 **zenity** 命令来使其成为 GTK+ 图形界面程序。
hello（**v=1.2**）

```
 $ cat debhello-1.2/scripts/hello
#!/bin/sh -e
zenity --info --title "hello" --text "Hello from the shell!"
```

这里，作为图形界面程序，桌面文件被更新为 **Terminal=false**。
hello.desktop（**v=1.2**）

```
 $ 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;
```

其余的所有文件都与“第 14.3 节”中的一致。

Let’s package this with the **debmake** command. Here, the “**-b’:sh’**”option is used to specify that the generated binary package is a shell script.

```
 $ cd /path/to/debhello-1.2
 $ debmake -b':sh' -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.2", rev="1"
I: *** start packaging in "debhello-1.2". ***
I: provide debhello_1.2.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.2.tar.gz debhello_1.2.orig.tar.gz
I: pwd = "/path/to/debhello-1.2"
I: parse binary package settings: :sh
I: binary package=debhello Type=script / Arch=all M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I:  25 %, ext = md
 ...
```

让我们来检查一下自动产生的模板文件。

debian/control（模板文件，**v=1.2**）：

```
 $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.7.0
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/debhello

Package: debhello
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.
```

作为维护者，我们要把这个 Debian 软件包做得更好。

debian/control（维护者版本，**v=1.2**）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
Standards-Version: 4.6.2
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)
```

请注意，这里需要手动添加 **zenity** 依赖。
debian/rules 文件与“第 14.3 节”中的完全一致。
在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。
debian/ 目录下的模板文件。（**v=1.2**）：

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

4 directories, 11 files
```

其余的打包工作与“第 14.3 节”中的几乎一致。
此处是 **debhello_1.2-1_all.deb** 的依赖项列表。
debhello_1.2-1_all.deb 的依赖项列表：

```
 $ dpkg -f debhello_1.2-1_all.deb pre-depends \
            depends recommends conflicts breaks
Depends: zenity
```

14.6 pyproject.toml（Python3, 图形界面）

Here is an example of creating a simple Debian package from a Python3 GUI program using **pyproject.toml**.

让我们假设上游源码包为 **debhello-1.3.tar.gz**。
让我们取得源码并制作 Debian 软件包。
下载 **debhello-1.3.tar.gz**

```
 $ wget http://www.example.org/download/debhello-1.3.tar.gz
 ...
 $ tar -xzmf debhello-1.3.tar.gz
 $ 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.gz

6 directories, 10 files
```

Here, the content of this **debhello** source tree as follows.
pyproject.toml (v=1.3) — PEP 517 configuration

```
 $ 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 = {file = "LICENSE.txt"}
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",
  "License :: OSI Approved :: MIT License",
  "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) — for tar-ball.

```
 $ cat debhello-1.3/MANIFEST.in
include data/*
include manpages/*
```

src/debhello/__init__.py (v=1.3)

```
 $ cat debhello-1.3/src/debhello/__init__.py
"""
debhello program (GUI)
"""
```

src/debhello/main.py (v=1.3) — command entry point

```
 $ 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()
```

让我们使用 **debmake** 命令来打包。这里使用 **-b':py3'** 选项来指明生成的二进制包包含 Python3 脚本和模块文件。

```
 $ cd /path/to/debhello-1.3
 $ debmake -b':py3' -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.3", rev="1"
```

```
I: *** start packaging in "debhello-1.3". ***
I: provide debhello_1.3.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.3.tar.gz debhello_1.3.orig.tar.gz
I: pwd = "/path/to/debhello-1.3"
I: parse binary package settings: :py3
I: binary package=debhello Type=python3 / Arch=all M-A=foreign
I: analyze the source tree
W: setuptools build system.
I: build_type = Python (pyproject.toml: PEP-518, PEP-621, PEP-660)
I: scan source for copyright+license text and file extensions
 ...
```

The result is practically the same as in “第 14.4 节”.
作为维护者，我们要把这个 Debian 软件包做得更好。
debian/rules（维护者版本，**v=1.3**）：

```
 $ cd /path/to/debhello-1.3
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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（维护者版本，**v=1.3**）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ 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: 4.6.2
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)
```

请注意，此处需要手动添加 **python3-gi** 和 **gir1.2-gtk-3.0** 依赖。
The rest of the packaging activities are practically the same as in <pyproject>>.
此处是 **debhello_1.3-1_all.deb** 的依赖项列表。
debhello_1.3-1_all.deb 的依赖项列表：

```
 $ 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（单个二进制软件包）

这里给出了从简单的 C 语言源代码创建简单的 Debian 软件包的例子，并假设上游使用了 **Makefile** 作为构建系统。

此处的上游源代码是“第 5 章”中的源代码的增强版本。它带有手册页、桌面文件和桌面图标。并且为了更加贴合实际，它还有一个外部库文件 **libm**。

让我们假设上游源码包为 **debhello-1.4.tar.gz**。

这一类源代码设计可以用这样的方式安装成为非系统文件：

```
 $ tar -xzmf debhello-1.4.tar.gz
 $ cd debhello-1.4
 $ make
 $ make install
```

Debian packaging requires changing this “**make install**”process to install files into the target system image location instead of the normal location under **/usr/local**.

让我们取得源码并制作 Debian 软件包。

下载 **debhello-1.4.tar.gz**

```
 $ wget http://www.example.org/download/debhello-1.4.tar.gz
 ...
 $ tar -xzmf debhello-1.4.tar.gz
 $ 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.gz

5 directories, 9 files
```

此处的源码如下所示。

src/hello.c（**v=1.4**）：

```
 $ 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**）：

```
 $ 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**）：

```
 $ cat debhello-1.4/src/config.h
#define PACKAGE_AUTHOR "Osamu Aoki"
```

请注意，此 **Makefile** 含有正确的手册页、桌面文件、桌面图标的 **install** 对象。

让我们使用 **debmake** 命令打包。

```
 $ cd /path/to/debhello-1.4
 $ debmake -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.4", rev="1"
I: *** start packaging in "debhello-1.4". ***
I: provide debhello_1.4.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.4.tar.gz debhello_1.4.orig.tar.gz
I: pwd = "/path/to/debhello-1.4"
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = make
I: scan source for copyright+license text and file extensions
I:  33 %, ext = c
 ...
```

其余的工作与“第 5.6 节”中的几乎一致。

像“第 5.7 节”中所写的一样，让我们这些维护者来把这个 Debian 软件包做的更好。

If the **DEB_BUILD_MAINT_OPTIONS** environment variable is not exported in **debian/rules**, lintian warns “W: debhello: hardening-no-relro usr/bin/hello”for the linking of **libm**.

debian/control 文件与“第 5.7 节”中的完全一致，因为 **libm** 库是 **libc6** 库的一部分，所以它总是可获得的（优先级：必需 / Priority: required）。

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。

debian/ 目录下的模板文件。（**v=1.4**）：

```
 $ rm -f debian/clean debian/dirs debian/links
 $ rm -f debian/README.source debian/source/*.ex
```

```
 $ rm -rf debian/patches
 $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- gbp.conf
+-- install
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 12 files
```

其余的打包步骤与“第 5.8 节”中的基本一致。
此处是生成的二进制包的依赖项列表。
生成的二进制包的依赖项列表（**v=1.4**）：

```
 $ dpkg -f debhello-dbgsym_1.4-1_amd64.deb pre-depends \
           depends recommends conflicts breaks
Depends: debhello (= 1.4-1)
 $ dpkg -f debhello_1.4-1_amd64.deb pre-depends \
           depends recommends conflicts breaks
Depends: libc6 (>= 2.34)
```

14.8 Makefile.in + configure（单个二进制软件包）

这里给出了从简单的 C 语言源代码创建简单的 Debian 软件包的例子，并假设上游使用了 **Makefile.in** 和 **configure** 作为构建系统。

此处的源码示例是“第 14.7 节”中的源代码的增强版本。它也有一个外部链接库 **libm**，并且它的源代码可以使用 **configure** 脚本进行配置，然后生成相应的 **Makefile**、**src/config.h** 文件。

让我们假设上游源码包为 **debhello-1.5.tar.gz**。

此类型的源码旨在作为非系统文件安装，例如：

```
 $ tar -xzmf debhello-1.5.tar.gz
 $ cd debhello-1.5
 $ ./configure --with-math
 $ make
 $ make install
```

让我们取得源码并制作 Debian 软件包。

下载 **debhello-1.5.tar.gz**

```
 $ wget http://www.example.org/download/debhello-1.5.tar.gz
 ...
 $ tar -xzmf debhello-1.5.tar.gz
 $ 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.gz

5 directories, 9 files
```

此处的源码如下所示。

src/hello.c（**v=1.5**）：

```
 $ 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**）：

```
 $ 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**）：

```
 $ 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
```

Please note that the **configure** command replaces strings with @⋯@ in **Makefile.in** to produce **Makefile** and creates **src/config.h**.

让我们使用 **debmake** 命令打包。

```
 $ cd /path/to/debhello-1.5
 $ debmake -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.5", rev="1"
I: *** start packaging in "debhello-1.5". ***
I: provide debhello_1.5.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.5.tar.gz debhello_1.5.orig.tar.gz
I: pwd = "/path/to/debhello-1.5"
```

```
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = configure
I: scan source for copyright+license text and file extensions
I:  17 %, ext = in
 ...
```

结果与“第 5.6 节”中的相似，但是并不完全一致。
让我们来检查一下自动产生的模板文件。
debian/rules（模板文件，**v=1.5**）：

```
 $ cd /path/to/debhello-1.5
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1

%:
        dh $@
```

作为维护者，我们要把这个 Debian 软件包做得更好。
debian/rules（维护者版本，**v=1.5**）：

```
 $ cd /path/to/debhello-1.5
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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"
```

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。
其余的打包步骤与“第 5.8 节”中的基本一致。

14.9 Autotools（单个二进制文件）

Here is an example of creating a simple Debian package from a simple C source program using Autotools = Autoconf and Automake (**Makefile.am** and **configure.ac**) as its build system.

此种源码通常也带有上游自动生成的 **Makefile.in** 和 **configure** 文件。在 **autotools-dev** 软件包的帮助下，我们可以按“第 14.8 节”中所介绍的，使用这些文件进行打包。

The better alternative is to regenerate these files using the latest Autoconf and Automake packages if the upstream provided **Makefile.am** and **configure.ac** are compatible with the latest version. This is advantageous for porting to new CPU architectures, etc. This can be automated by using the “**--with autoreconf**”option for the **dh** command.

让我们假设上游的源码包为 **debhello-1.6.tar.gz**。
此类型的源码旨在作为非系统文件安装，例如：

```
 $ tar -xzmf debhello-1.6.tar.gz
 $ cd debhello-1.6
```

```
 $ autoreconf -ivf # optional
 $ ./configure --with-math
 $ make
 $ make install
```

让我们取得源码并制作 Debian 软件包。
下载 **debhello-1.6.tar.gz**

```
 $ wget http://www.example.org/download/debhello-1.6.tar.gz
 ...
 $ tar -xzmf debhello-1.6.tar.gz
 $ 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.gz

5 directories, 11 files
```

此处的源码如下所示。
src/hello.c（**v=1.6**）：

```
 $ 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**）：

```
 $ cat debhello-1.6/Makefile.am
SUBDIRS = src man
 $ cat debhello-1.6/man/Makefile.am
dist_man_MANS = hello.1
 $ cat debhello-1.6/src/Makefile.am
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```

configure.ac（**v=1.6**）：

```
 $ 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
```

提示

Without “**foreign**”strictness level specified in **AM_INIT_AUTOMAKE()** as above, **automake** defaults to “**gnu**”strictness level requiring several files in the top-level directory. See “3.2 Strictness”in the **automake** document.

让我们使用 **debmake** 命令打包。

```
 $ cd /path/to/debhello-1.6
 $ debmake -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.6", rev="1"
I: *** start packaging in "debhello-1.6". ***
I: provide debhello_1.6.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.6.tar.gz debhello_1.6.orig.tar.gz
I: pwd = "/path/to/debhello-1.6"
```

```
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = Autotools with autoreconf
I: scan source for copyright+license text and file extensions
I:  33 %, ext = am
 ...
```

结果与“第 14.8 节”中的类似，但是并不完全一致。
让我们来检查一下自动产生的模板文件。
debian/rules（模板文件，**v=1.6**）：

```
 $ cd /path/to/debhello-1.6
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1

%:
        dh $@ --with autoreconf

#override_dh_install:
#       dh_install --list-missing -X.la -X.pyc -X.pyo
```

作为维护者，我们要把这个 Debian 软件包做得更好。
debian/rules（维护者版本，**v=1.6**）：

```
 $ cd /path/to/debhello-1.6
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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
```

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。
其余的打包步骤与“第 5.8 节”中的基本一致。

14.10 CMake（单个二进制软件包）

Here is an example of creating a simple Debian package from a simple C source program using CMake (**CMakeLists.txt** and some files such as **config.h.in**) as its build system.

The **cmake** command generates the **Makefile** file based on the **CMakeLists.txt** file and its **-D** option. It also configures the file as specified in its **configure_file(⋯)** by replacing strings with @⋯@ and changing the **#cmakedefine** ⋯ line.

让我们假设上游的源码包为 **debhello-1.7.tar.gz**。
此类型的源码旨在作为非系统文件安装，例如：

```
 $ tar -xzmf debhello-1.7.tar.gz
 $ cd debhello-1.7
 $ mkdir obj-x86_64-linux-gnu # for out-of-tree build
```

```
 $ cd obj-x86_64-linux-gnu
 $ cmake ..
 $ make
 $ make install
```

让我们取得源码并制作 Debian 软件包。
下载 **debhello-1.7.tar.gz**

```
 $ wget http://www.example.org/download/debhello-1.7.tar.gz
 ...
 $ tar -xzmf debhello-1.7.tar.gz
 $ 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.gz

5 directories, 11 files
```

此处的源码如下所示。
src/hello.c（**v=1.7**）：

```
 $ 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**）：

```
 $ 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**）：

```
 $ cat debhello-1.7/CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(debhello)
set(PACKAGE_AUTHOR "Osamu Aoki")
```

```
add_subdirectory(src)
add_subdirectory(man)
 $ cat debhello-1.7/man/CMakeLists.txt
install(
  FILES ${CMAKE_CURRENT_SOURCE_DIR}/hello.1
  DESTINATION share/man/man1
)
 $ 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
)
```

让我们使用 **debmake** 命令打包。

```
 $ cd /path/to/debhello-1.7
 $ debmake -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="1.7", rev="1"
I: *** start packaging in "debhello-1.7". ***
I: provide debhello_1.7.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-1.7.tar.gz debhello_1.7.orig.tar.gz
I: pwd = "/path/to/debhello-1.7"
I: parse binary package settings:
I: binary package=debhello Type=bin / Arch=any M-A=foreign
I: analyze the source tree
I: build_type = Cmake
I: scan source for copyright+license text and file extensions
I:  33 %, ext = text
 ...
```

结果与“第 14.8 节”中的类似，但是并不完全一致。

让我们来检查一下自动产生的模板文件。

debian/rules（模板文件，**v=1.7**）：

```
 $ cd /path/to/debhello-1.7
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1

%:
        dh $@

#override_dh_auto_configure:
#       dh_auto_configure -- \
#             -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"
```

debian/control（模板文件，**v=1.7**）：

```
 $ cat debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Osamu Aoki" <osamu@debian.org>
Build-Depends:
 cmake,
 debhelper-compat (= 13),
Standards-Version: 4.7.0
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/debhello

Package: debhello
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.
```

作为维护者，我们要把这个 Debian 软件包做得更好。
debian/rules（维护者版本，**v=1.7**）：

```
 $ cd /path/to/debhello-1.7
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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（维护者版本，**v=1.7**）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 cmake,
 debhelper-compat (= 13),
Standards-Version: 4.6.2
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)
```

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。
其余的打包工作与“第 14.8 节”中的近乎一致。

14.11 Autotools（多个二进制软件包）

Here is an example of creating a set of Debian binary packages including the executable package, the shared library package, the development file package, and the debug symbol package from a simple C source program using Autotools (Autoconf and Automake, which use **Makefile.am** and **configure.ac** as their input files) as its build system.

Let’s package this in a similar way to “第 14.9 节”.
让我们假设上游源码包为 **debhello-2.0.tar.gz**。
此类型的源码旨在作为非系统文件安装，例如：

```
 $ tar -xzmf debhello-2.0.tar.gz
 $ cd debhello-2.0
 $ autoreconf -ivf # optional
 $ ./configure --with-math
 $ make
 $ make install
```

让我们取得源码并制作 Debian 软件包。
下载 **debhello-2.0.tar.gz**

```
 $ wget http://www.example.org/download/debhello-2.0.tar.gz
 ...
 $ tar -xzmf debhello-2.0.tar.gz
 $ 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.gz

6 directories, 14 files
```

此处的源码如下所示。
src/hello.c（**v=2.0**）：

```
 $ 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 与 lib/sharedlib.c（v=1.6）：

```
 $ cat debhello-2.0/lib/sharedlib.h
int sharedlib();
 $ 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）：

```
 $ cat debhello-2.0/Makefile.am
# recursively process `Makefile.am` in SUBDIRS
SUBDIRS = lib src man
 $ cat debhello-2.0/man/Makefile.am
# manpages (distributed in the source package)
dist_man_MANS = hello.1
 $ 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
 $ 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）：

```
 $ 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
```

Let's use the **debmake** command to package this into multiple packages:

- **debhello**: type = **bin**
- **libsharedlib1**: type = **lib**
- **libsharedlib-dev**: type = **dev**

Here, we use the **-b'libsharedlib1,libsharedlib-dev'** option to specify the additional binary packages to be generated.

```
 $ cd /path/to/debhello-2.0
 $ debmake -b',libsharedlib1,libsharedlib-dev' -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="2.0", rev="1"
I: *** start packaging in "debhello-2.0". ***
I: provide debhello_2.0.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-2.0.tar.gz debhello_2.0.orig.tar.gz
I: pwd = "/path/to/debhello-2.0"
I: parse binary package settings: ,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: analyze the source tree
I: build_type = Autotools with autoreconf
 ...
```

结果与"第 14.8 节"中的相似，但是这个具有更多的模板文件。

让我们来检查一下自动产生的模板文件。

debian/rules（模板文件，**v=2.0**）：

```
 $ cd /path/to/debhello-2.0
 $ cat debian/rules
```

```
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1

%:
        dh $@ --with autoreconf

#override_dh_install:
#       dh_install --list-missing -X.la -X.pyc -X.pyo
```

作为维护者，我们要把这个 Debian 软件包做得更好。
debian/rules（维护者版本，**v=2.0**）：

```
 $ cd /path/to/debhello-2.0
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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（维护者版本，**v=2.0**）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 debhelper-compat (= 13),
 dh-autoreconf,
Standards-Version: 4.6.2
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（维护者版本，**v=2.0**）：

```
 $ vim debian/copyright
 ... hack, hack, hack, ...
 $ 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.
```

因为上游源码已经具有正确的自动生成的 **Makefile** 文件，所以没有必要再去创建 **debian/install** 和 **debian/manpages** 文件。

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。

debian/ 目录下的模板文件。（**v=2.0**）：

```
 $ rm -f debian/clean debian/dirs debian/install debian/links
 $ rm -f debian/README.source debian/source/*.ex
 $ rm -rf debian/patches
 $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- debhello.dirs
+-- debhello.doc-base
```

```
+-- debhello.docs
+-- debhello.examples
+-- debhello.info
+-- debhello.install
+-- debhello.links
+-- debhello.manpages
+-- gbp.conf
+-- libsharedlib-dev.install
+-- libsharedlib1.install
+-- libsharedlib1.symbols
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

4 directories, 22 files
```

其余的打包工作与“第 14.8 节”中的近乎一致。
此处是生成的二进制包的依赖项列表。
生成的二进制包的依赖项列表（**v=2.0**）：

```
 $ dpkg -f debhello-dbgsym_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: debhello (= 2.0-1)
 $ dpkg -f debhello_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.0-1), libc6 (>= 2.34)
 $ dpkg -f libsharedlib-dev_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.0-1)
 $ dpkg -f libsharedlib1-dbgsym_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.0-1)
 $ dpkg -f libsharedlib1_2.0-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libc6 (>= 2.2.5)
```

14.12 CMake（多个二进制软件包）

This example demonstrates creating a set of Debian binary packages including the executable package, the shared library package, the development file package, and the debug symbol package from a simple C source program using CMake (**CMakeLists.txt** and files such as **config.h.in**) as its build system.

让我们假设上游源码包为 **debhello-2.1.tar.gz**。
此类型的源码旨在作为非系统文件安装，例如：

```
 $ tar -xzmf debhello-2.1.tar.gz
 $ cd debhello-2.1
 $ mkdir obj-x86_64-linux-gnu
 $ cd obj-x86_64-linux-gnu
 $ cmake ..
 $ make
 $ make install
```

让我们取得源码并制作 Debian 软件包。
下载 **debhello-2.1.tar.gz**

```
 $ wget http://www.example.org/download/debhello-2.1.tar.gz
 ...
```

```
 $ tar -xzmf debhello-2.1.tar.gz
 $ 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.gz

6 directories, 14 files
```

此处的源码如下所示。

src/hello.c（**v=2.1**）：

```
 $ 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**）：

```
 $ cat debhello-2.1/src/config.h.in
/* name of the package author */
#define PACKAGE_AUTHOR "@PACKAGE_AUTHOR@"
```

lib/sharedlib.c 与 **lib/sharedlib.h**（**v=2.1**）：

```
 $ cat debhello-2.1/lib/sharedlib.h
int sharedlib();
 $ 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**）：

```
 $ cat debhello-2.1/CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project(debhello)
set(PACKAGE_AUTHOR "Osamu Aoki")
add_subdirectory(lib)
```

```
add_subdirectory(src)
add_subdirectory(man)
 $ cat debhello-2.1/man/CMakeLists.txt
install(
  FILES ${CMAKE_CURRENT_SOURCE_DIR}/hello.1
  DESTINATION share/man/man1
)
 $ 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
)
```

让我们使用 **debmake** 命令打包。

```
 $ cd /path/to/debhello-2.1
 $ debmake -b',libsharedlib1,libsharedlib-dev' -x1
I: set parameters
 ...
I: sanity check of parameters
I: pkg="debhello", ver="2.1", rev="1"
I: *** start packaging in "debhello-2.1". ***
I: provide debhello_2.1.orig.tar.?z for non-native Debian package
I: pwd = "/path/to"
I: $ ln -sf debhello-2.1.tar.gz debhello_2.1.orig.tar.gz
I: pwd = "/path/to/debhello-2.1"
I: parse binary package settings: ,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: analyze the source tree
I: build_type = Cmake
 ...
```

结果与“第 14.8 节”中的类似，但是并不完全一致。

让我们来检查一下自动产生的模板文件。

debian/rules（模板文件，**v=2.1**）：

```
 $ cd /path/to/debhello-2.1
 $ cat debian/rules
#!/usr/bin/make -f
# You must remove unused comment lines for the released package.
#export DH_VERBOSE = 1
#export DEB_BUILD_MAINT_OPTIONS = hardening=+all
#export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
#export DEB_LDFLAGS_MAINT_APPEND = -Wl,-O1

%:
        dh $@

#override_dh_auto_configure:
#       dh_auto_configure -- \
#             -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"
```

作为维护者，我们要把这个 Debian 软件包做得更好。
debian/rules（维护者版本，**v=2.1**）：

```
 $ cd /path/to/debhello-2.1
 $ vim debian/rules
 ... hack, hack, hack, ...
 $ 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（维护者版本，**v=2.1**）：

```
 $ vim debian/control
 ... hack, hack, hack, ...
 $ cat debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends:
 cmake,
 debhelper-compat (= 13),
Standards-Version: 4.6.2
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（维护者版本，**v=2.1**）：

```
 $ vim debian/copyright
 ... hack, hack, hack, ...
 $ 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.
```

The upstream CMakeLists.txt file needs to be patched to handle the multiarch path correctly.

debian/patches/*（维护者版本，**v=2.1**）：

```
 ... hack, hack, hack, ...
 $ cat debian/libsharedlib1.symbols
libsharedlib.so.1 libsharedlib1 #MINVER#
 sharedlib@Base 2.1
```

因为上游源码已经具有正确的自动生成的 **Makefile** 文件，所以没有必要再去创建 **debian/install** 和 **debian/manpages** 文件。

在 **debian/** 目录下还有一些其它的模板文件。它们也需要进行更新。

debian/ 目录下的模板文件。（**v=2.1**）：

```
 $ rm -f debian/clean debian/dirs debian/install debian/links
 $ rm -f debian/README.source debian/source/*.ex
 $ tree -F debian
debian/
+-- README.Debian
+-- changelog
+-- control
+-- copyright
+-- debhello.dirs
+-- debhello.doc-base
+-- debhello.docs
+-- debhello.examples
+-- debhello.info
+-- debhello.install
```

```
+-- debhello.links
+-- debhello.manpages
+-- gbp.conf
+-- libsharedlib-dev.install
+-- libsharedlib1.install
+-- libsharedlib1.symbols
+-- patches/
|   +-- 000-cmake-multiarch.patch
|   +-- series
+-- rules*
+-- salsa-ci.yml
+-- source/
|   +-- format
+-- tests/
|   +-- control
+-- upstream/
|   +-- metadata
+-- watch

5 directories, 24 files
```

其余的打包工作与“第 14.8 节”中的近乎一致。
此处是生成的二进制包的依赖项列表。
生成的二进制包的依赖项列表（**v=2.1**）：

```
 $ dpkg -f debhello-dbgsym_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: debhello (= 2.1-1)
 $ dpkg -f debhello_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.1-1), libc6 (>= 2.34)
 $ dpkg -f libsharedlib-dev_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.1-1)
 $ dpkg -f libsharedlib1-dbgsym_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libsharedlib1 (= 2.1-1)
 $ dpkg -f libsharedlib1_2.1-1_amd64.deb pre-depends \
            depends recommends conflicts breaks
Depends: libc6 (>= 2.2.5)
```

14.13 国际化

此处是更新“第 14.11 节”中提供的简单上游 C 语言源代码 **debhello-2.0.tar.gz** 以便进行国际化（i18n）并创建更新后的上游 C 语言源代码 **debhello-2.0.tar.gz** 的示例。
在实际情况中，此软件包应该已被国际化过。所以此示例用作帮助您了解国际化的具体实现方法。

> 提示
>
> 
>
> 负责维护国际化的维护者的日常活动就是将通过缺陷追踪系统（BTS）反馈给您的 po 翻译文件添加至 **po/** 目录，然后更新 **po/LINGUAS** 文件的语言列表。

让我们取得源码并制作 Debian 软件包。
下载 **debhello-2.0.tar.gz**（国际化版）

```
 $ wget http://www.example.org/download/debhello-2.0.tar.gz
 ...
 $ tar -xzmf debhello-2.0.tar.gz
 $ 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.gz

6 directories, 14 files
```

使用 **gettextize** 命令将此源代码树国际化，并删除由 Autotools 自动生成的文件。
运行 **gettextize**（国际化版）：

```
 $ cd /path/to/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.
 $ rm -rf m4 build-aux *~
```

让我们确认一下 **po/** 目录下生成的文件。
po 目录下的文件（国际化版）：

```
 $ ls -l po
total 60
-rw-rw-r-- 1 osamu osamu   494 Nov 29 07:59 ChangeLog
-rw-rw-r-- 1 osamu osamu 17577 Nov 29 07:59 Makefile.in.in
-rw-rw-r-- 1 osamu osamu  3376 Nov 29 07:59 Makevars.template
-rw-rw-r-- 1 osamu osamu    59 Nov 29 07:59 POTFILES.in
-rw-rw-r-- 1 osamu osamu  2203 Nov 29 07:59 Rules-quot
-rw-rw-r-- 1 osamu osamu   217 Nov 29 07:59 boldquot.sed
-rw-rw-r-- 1 osamu osamu  1337 Nov 29 07:59 en@boldquot.header
-rw-rw-r-- 1 osamu osamu  1203 Nov 29 07:59 en@quot.header
-rw-rw-r-- 1 osamu osamu   672 Nov 29 07:59 insert-header.sin
-rw-rw-r-- 1 osamu osamu   153 Nov 29 07:59 quot.sed
-rw-rw-r-- 1 osamu osamu   432 Nov 29 07:59 remove-potcdate.sin
```

Let's update the **configure.ac** by adding "**AM_GNU_GETTEXT([external])**", etc..
configure.ac（国际化版）：

```
 $ vim configure.ac
 ... hack, hack, hack, ...
 $ 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
```

让我们从 **po/Makevars.template** 文件中创建 **po/Makevars** 文件。
po/Makevars（国际化版）：

```
 ... hack, hack, hack, ...
 $ diff -u po/Makevars.template po/Makevars
--- po/Makevars.template        2024-11-29 07:59:15.133577084 +0000
+++ po/Makevars 2024-11-29 07:59:15.209578283 +0000
@@ -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:
 $ rm po/Makevars.template
```

Let's update C sources for the i18n version by wrapping strings with **_(…)**.
src/hello.c （国际化版）：

```
 ... hack, hack, hack, ...
 $ 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（国际化版）：

```
 ... hack, hack, hack, ...
 $ 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;
}
```

新版本的 **gettext**（v = 0.19）可以直接处理桌面文件的国际化版本。
data/hello.desktop.in（国际化版）：

```
 $ fgrep -v '[ja]=' data/hello.desktop > data/hello.desktop.in
 $ rm data/hello.desktop
 $ cat data/hello.desktop.in
[Desktop Entry]
Name=Hello
Comment=Greetings
Type=Application
Keywords=hello
Exec=hello
Terminal=true
Icon=hello.png
Categories=Utility;
```

让我们列出输入文件，以便在 **po/POTFILES.in** 中提取可翻译的字符串。
po/POTFILES.in（国际化版）：

```
 ... hack, hack, hack, ...
 $ cat po/POTFILES.in
src/hello.c
lib/sharedlib.c
data/hello.desktop.in
```

此处是在 **SUBDIRS** 环境变量中添加 **po** 目录后更新过的根 **Makefile.am** 文件。
Makefile.am （国际化版）：

```
 $ 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
```

让我们创建一个翻译模板文件 **debhello.pot**。
po/debhello.pot（国际化版）：

```
 $ xgettext -f po/POTFILES.in -d debhello -o po/debhello.pot -k_
Warning: program compiled against libxml 212 using older 209
 $ 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: 2024-11-29 07:59+0000\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:3
msgid "Hello"
msgstr ""

#: data/hello.desktop.in:4
msgid "Greetings"
msgstr ""

#: data/hello.desktop.in:6
msgid "hello"
msgstr ""
```

让我们添加法语的翻译。

po/LINGUAS 与 **po/fr.po**（国际化版）：

```
 $ echo 'fr' > po/LINGUAS
 $ cp po/debhello.pot po/fr.po
 $ vim po/fr.po
 ... hack, hack, hack, ...
 $ 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 ""
```

打包工作与“第 14.11 节”中的近乎一致。
You can find more i18n examples by following “第 14.14 节”.

14.14 细节

You can obtain detailed information about the examples presented and their variants as follows:
如何取得细节

```
$ apt-get source debmake-doc
$ cd debmake-doc*
$ cd examples
$ view examples/README.md
```

Follow the exact instruction in **examples/README.md**.

```
$ cd examples
$ make
```

Now, each directory named as **examples/debhello-?.?_build-?** contains the Debian packaging example.

- 模拟控制台命令行活动日志：**.log** 文件
- 模拟控制台命令行活动日志（缩略版）：**.slog** 文件
- 执行 **debmake** 命令后的源码树快照：**debmake** 目录
- snapshot source tree image after proper packaging: the **package** directory
- 执行 **debuild** 命令后的源码树快照：**test** 目录

Notable examples include:

- POSIX shell script with Makefile and i18n support (v=3.0)
- C source with Makefile.in + configure and i18n support (v=3.2)
- C source with Autotools and i18n support (v=3.3)
- 带有 CMake 和国际化支持的的 C 语言源代码（v=3.4）

Chapter 15

debmake(1) 手册页

15.1 名称

debmake，用来制作 Debian 源码包的程序

15.2 概述

debmake [**-h**] [**-c** | **-k**] [**-n** | **-a** *package-version*.**orig.tar.gz** | **-d** | **-t**] [**-p** *package*] [**-u** *version*] [**-r** *revision*] [**-z** *extension*] [**-b** "*binarypackage[:type], ⋯*]" [**-e** *foo@example.org*] [**-f** "*firstname lastname*"] [**-i** "*buildtool*" | **-j**] [**-l** *license_file*] [**-m**] [**-o** *file*] [**-q**] [**-s**] [**-v**] [**-w** "*addon, ⋯*"] [**-x** [*01234*]] [**-y**] [**-L**] [**-P**] [**-T**]

15.3 描述

debmake 协助从上游源代码构建一个 Debian 软件包，通常做法如下：

- 下载上游源码压缩包（tarball）并命名为 *package-version*.**tar.gz** 文件。
- 对其进行解压缩并将所有文件放置于 *package-version/* 目录之下。
- 在 *package-version/* 目录中调用 debmake，并按需带上参数。
- 手工调整 *package-version*/**debian/** 目录下的文件。
- **dpkg-buildpackage** (usually from its wrapper **debuild** or **sbuild**) is invoked in the *package-version/* directory to make Debian packages.

请确保将 **-b**、**-f**、**-l** 和 **-w** 选项的参数使用引号合适地保护起来，以避免 shell 环境的干扰。

15.3.1 可选参数：

-h, --help 显示本帮助信息并退出。

-c, --copyright 为授权 + 许可证文本而扫描源码，然后退出。

- **-c**：简单输出风格
- **-cc**：正常输出风格（类似 **debian/copyright** 文件）
- **-ccc**：调试输出风格

-k, --kludge 对 **debian/copyright** 文件和源代码进行比较并退出。

debian/copyright 必须将通用的文件匹配模式放在前部并将个别文件的例外放在后部。

- **-k**：基本输出风格
- **-kk**：冗长输出风格

-n, --native make a native Debian source package without **.orig.tar.gz**. This makes a Debian source format "**3.0 (native)**"package.

If you are thinking of packaging a Debian-specific source tree with **debian/** in it into a native Debian package, please think otherwise. You can use the "**debmake -d -i debuild**"or "**debmake -t -i debuild**"commands to make a Debian non-native package using the Debian source format "**3.0 (quilt)**"The only difference is that the **debian/changelog** file must use the non-native version scheme: *version-revision*. The non-native package is more friendly to downstream distributions.

-a *package-version*.tar.gz, --archive *package-version*.tar.gz 直接使用上游源码压缩包。（**-p**, **-u**, **-z**：被覆盖）

上游源码压缩包可以命名为 *package_version*.**orig.tar.gz** 或者 **tar.gz**。在某些情况下，也可使用 **tar.bz2** 或 **tar.xz**。

如果所指定的源码压缩包文件名中包含大写字母，Debian 打包时生成的名称会将其转化为小写字母。

If the specified argument is the URL (`http://`, `https://`, or `ftp://`) to the upstream tarball, the upstream tarball is downloaded from the URL using **wget** or **curl**.

-d, --dist run the "**make dist**"command equivalents first to generate the upstream tarball and use it.

The "**debmake -d**"command is designed to run in the *package/* directory hosting the upstream VCS with the build system supporting the "**make dist**"command equivalents. (`automake`/`autoconf`, ⋯)

-t, --tar run the "**tar**"command to generate the upstream tarball and use it.

The "**debmake -t**"command is designed to run in the *package/* directory hosting the upstream VCS. Unless you provide the upstream version with the **-u** option or with the **debian/changelog** file, a snapshot upstream version is generated in the **0\~%y%m%d%H%M** format, e.g., *0~1403012359*, from the UTC date and time. The generated tarball excludes the **debian/** directory found in the upstream VCS. (It also excludes typical VCS directories: **.git/**, **.hg/**, **.svn/**, **.CVS/**.)

-p 软件包名**, --package** 软件包名 设置 Debian 软件包名称。

-u 上游版本号**, --upstreamversion** 版本号 设置上游软件包版本。

-r 修订号**, --revision** 修订号 设置 Debian 软件包修订号。

-z 扩展名**, --targz** 扩展名 设置源码压缩包类型，扩展名 =(**tar.gz**|**tar.bz2**|**tar.xz**)。（别名：**z**, **b**, **x**）

-b "*binarypackage[:type],*⋯", --binaryspec "*binarypackage[:type],*⋯" set the binary package specs by a comma separated list of *binarypackage:type* pairs. Here, *binarypackage* is the binary package name, and the optional *type* is chosen from the following *type* values:

- **bin**: C/C++ compiled ELF binary code package (any, foreign) (default, alias: **""**, i.e., **null-string**)
- **data**: Data (fonts, graphics, ⋯) package (all, foreign) (alias: **da**)
- **dev**：库开发软件包（any，same）（别名：**de**）
- **doc**：文档软件包（all，foreign）（别名：**do**）
- **lib**：库软件包（any，same）（别名：**l**）
- **perl**：Perl 脚本软件包（all，foreign）（别名：**pl**）
- **python3**: Python (version 3) script package (all, foreign) (alias: **py3**, **python**, **py**)
- **ruby**：Ruby 脚本软件包（all，foreign）（别名：**rb**）
- **nodejs**：基于 Node.js 的 JavaScript 软件包（all，foreign）（别名：**js**）
- **script**: Shell and other interpreted language script package (all, foreign) (alias: **sh**)

The pair values in the parentheses, such as (any, foreign), are the **Architecture** and **Multi-Arch** stanza values set in the **debian/control** file. In many cases, the **debmake** command makes good guesses for *type* from *binarypackage*. If *type* is not obvious, *type* is set to **bin**.

Here are examples for typical binary package split scenarios where the upstream Debian source package name is **foo**:

- Generating an executable binary package **foo**:
  - “**-b**’**foo:bin**’”, or its short form `**-b’-**`”, or no **-b** option
- Generating an executable (python3) binary package **python3-foo**:
  - “**-b**’**python3-foo:py**’”, or its short form “**-b**’**python3-foo**’”
- Generating a data package **foo**:
  - “**-b**’**foo:data**’”, or its short form “**-b’-:data**’”
- Generating a executable binary package **foo** and a documentation one **foo-doc**:
  - “**-b**’**foo:bin,foo-doc:doc**’”, or its short form “**-b’-:-doc**’”
- Generating a executable binary package **foo**, a library package **libfoo1**, and a library development package **libfoo-dev**:
  - “**-b**’**foo:bin,libfoo1:lib,libfoo-dev:dev**’”or its short form “**-b’-,libfoo1,libfoo-dev**’”

如果源码树的内容和类型的设置不一致，**debmake** 命令会发出警告。

-e ***foo@example.org*****, --email** ***foo@example.org*** 设置电子邮件地址。

默认值为环境变量 **$DEBEMAIL** 的值。

-f ” 名称姓氏**”, --fullname ”** 名称姓氏**”** 设置全名。

默认值为环境变量 **$DEBFULLNAME** 的值。

-i ” 构建工具**”, --invoke ”** 构建工具**”** invoke ”*buildtool*” at the end of execution. *buildtool* may be “**dpkg-buildpackage**”, “**debuild**”, “**sbuild**”, etc.

默认情况是不执行任何程序。

设置该选项也会自动设置 **--local** 选项。

-j, --judge 运行 **dpkg-depcheck** 以检查构建依赖和文件路径。检查日志将存储在父目录下。

- 软件包名**.build-dep.log**：**dpkg-depcheck** 的日志文件。
- 软件包名**.install.log**：记录 **debian/tmp** 目录下所安装文件的日志。

-l ”*license_file,*⋯”, --license ”*license_file,*⋯” 在存放许可证扫描结果的 **debian/copyright** 文件末尾添加格式化后的许可证文本。

The default is to add **COPYING** and **LICENSE**, and *license_file* needs to list only the additional file names all separated by “**,**”.

-m, --monoarch 强制软件包不使用多架构特性。

-o 文件**, --option** 文件 从指定 *file* 读取可选参数。（这个选项不适合日常使用。）

文件 *file* 的内容，将在 **para.py** 的末尾作为 Python 代码的源代码。例如，软件包描述信息可以使用下述文件来定义。

```
para['desc'] = 'program short description'
para['desc_long'] = '''\
 program long description which you wish to include.
 .
 Empty line is space + .
 You keep going on ...
'''
```

-q, --quitearly 在创建 **debian/** 目录下的文件之前即提前退出程序。

-s, --spec use upstream spec (**pyproject.py** for Python, etc.) for the package description.

-v, --version 显示版本信息。

-w "*addon,*⋯", --with "*addon,*⋯" 在 **debian/rules** 文件中向 **dh**(1) 命令的参数中添加额外的 **dh**(1) 参数以指定所使用的附加组件（*addon*）。

The *addon* values are listed all separated by "**,**", e.g., "**-w "python3,autoreconf"**".

For Autotools based packages, **autoreconf** as *addon* to run "**autoreconf -i -v -f**"for every package building is default behavior of the **dh**(1) command.

For Autotools based packages, if they install Python (version 3) programs, setting **python3** as *addon* to the **debmake** command argument is needed since this is non-obvious. But for **pyproject.toml** based Python packages, setting **python3** as *addon* to the **debmake** command argument is not needed since this is obvious and the **debmake** command automatically set it to the **dh**(1) command.

-x *n*, --extra *n* 以模板文件的形式创建配置文件（请注意 **debian/changelog**、**debian/control**、**debian/copyright** 和 **debian/rules** 文件是构建 Debian 二进制软件包所需的最小文件集合。）

n 的数字大小决定了生成哪些配置模板文件。

- **-x0**: all required configuration template files. (selected option if any of these files already exist)
- **-x1**: all **-x0** files + desirable configuration template files with binary package type supports.
- **-x2**: all **-x1** files + normal configuration template files with maintainer script supports.
- **-x3**: all **-x2** files + optional configuration template files. (default option)
- **-x4**: all **-x3** files + deprecated configuration template files.

Some configuration template files are generated with the extra **.ex** suffix to ease their removal. To activate these, rename their file names to the ones without the **.ex** suffix and edit their contents. Existing configuration files are never overwritten. If you wish to update some of the existing configuration files, please rename them before running the **debmake** command and manually merge the generated configuration files with the old renamed ones.

-y, --yes "force yes"for all prompts. (without option: "ask [Y/n]"; doubled option: "force no")

-L, --local 为本地软件包生成配置文件以绕过 **lintian**(1) 的检查。

-P, --pedantic 对自动生成的文件进行严格（甚至古板到迂腐程度）的检查。

-T, --tutorial output tutorial comment lines in template files. default when **-x3** or **-x4** is set.

15.4 示例

For a well behaving source, you can build a good-for-local-use installable single Debian binary package easily with one command. Test install of such a package generated in this way offers a good alternative to the traditional "**make install**"command installing into the **/usr/local** directory since the Debian package can be removed cleanly by the "**dpkg -P** '⋯'"command. Here are some examples of how to build such test packages. (These should work in most cases. If the **-d** option does not work, try the **-t** option instead.)

For a typical C program source tree packaged with **autoconf/automake**:

- **debmake -d -i debuild**

对于典型的 Python（版本 3）模块源码树：

- **debmake -s -d -b":python3" -i debuild**

对于 *package-version***.tar.gz** 存档里的一个典型 Python*（版本 3）模块：

- **debmake -s -a** *package-version***.tar.gz -b":python3" -i debuild**

对于典型的以 *package-version***.tar.gz** 归档提供的 Perl 模块：

- **debmake -a** *package-version***.tar.gz -b":perl" -i debuild**

15.5 帮助软件包

打包工作也许需要额外安装一些专用的帮助软件包。

- Python (version 3) programs may require the **pybuild-plugin-pyproject** package.
- The Autotools (**autoconf** + **automake**) build system may require **autotools-dev** or **dh-autoreconf** package.
- Ruby 程序可能需要 **gem2deb** 软件包。
- 基于 JavaScript 的 Node.js 程序可能需要 **pkg-js-tools** 软件包。
- Java 程序可能需要 **javahelper** 软件包。
- Gnome 程序可能需要 **gobject-introspection** 软件包。
- 等等。

15.6 注意事项

Although **debmake** is meant to provide template files for the package maintainer to work on, actual packaging activities are often performed without using **debmake** while referencing only existing similar packages and "Debian Policy Manual". All template files generated by **debmake** are required to be modified manually.

There are 2 positive points for **debmake**:

- **debmake** helps to write terse packaging tutorial "Guide for Debian Maintainers"(**debmake-doc** package).
- **debmake** provides short extracted license texts as **debian/copyright** in decent accuracy to help license review.

Please double check copyright with the **licensecheck**(1) command.

组成 Debian 软件包名称的字符选取存在一定的限制。最明显的限制应当是软件包名称中禁止出现大写字母。这里给出正则表达式形式的规则总结：

- Upstream package name (**-p**): `[-+.a-z0-9]{2,}`
- Binary package name (**-b**): `[-+.a-z0-9]{2,}`
- Upstream version (**-u**): `[0-9][-+.:~a-z0-9A-Z]*`
- Debian revision (**-r**): `[0-9][+.~a-z0-9A-Z]*`

See the exact definition in "Chapter 5 - Control files and their fields"in the "Debian Policy Manual".

debmake assumes relatively simple packaging cases. So all programs related to the interpreter are assumed to be "**Architecture: all**". This is not always true.

15.7 除错

请使用 **reportbug** 命令报告 **debmake** 软件包的问题与错误。

环境变量 **$DEBUG** 中设置的字符用来确定日志输出等级。

- **i**: `main.py` logging
- **p**: `para.py` logging
- **s**: `checkdep5.py` `check_format_style()` logging
- **y**: `checkdep5.py` `split_years_name()` logging
- **b**: `checkdep5.py` `parse_lines()` 1 logging — content_state scan loop: begin-loop

- **m**: `checkdep5.py parse_lines()` 2 logging — content_state scan loop: after regex match
- **e**: `checkdep5.py parse_lines()` 3 logging — content_state scan loop: end-loop
- **a**: `checkdep5.py parse_lines()` 4 logging — print author/translator section text
- **f**: `checkdep5.py check_all_license()` 1 logging — input filename for the copyright scan
- **l**: `checkdep5.py check_all_license()` 2 logging — print license section text
- **c**: `checkdep5.py check_all_license()` 3 logging — print copyright section text
- **k**: `checkdep5.py check_all_license()` 4 logging — sort key for debian/copyright stanza
- **r**: `sed.py` logging
- **w**: `cat.py` logging
- **n**: `kludge.py` logging (“**debmake -k**”)

Use this feature as:

```
$ DEBUG=ipsybmeaflckrwn debmake ...
```

See **README.developer** in the source for more.

15.8 作者

15.9 许可证

Expat 许可证

15.10 参见

The **debmake-doc** package provides the “Guide for Debian Maintainers”in plain text, HTML and PDF formats under the **/usr/share/doc/debmake-doc/** directory.

See also **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), and **gbp-pq**(1) manpages.

Chapter 16

debmake options

Here are some additional explanations for **debmake** options.

16.1 Shortcut options (-a, -i)

debmake 命令提供了两个快捷选项。

- **-a**：打开上游源码压缩包
- **-i**：执行构建二进制包的脚本

前文中“第 5 章”的例子可以使用下面的命令直接达到目的。

```
$ debmake -a package-1.0.tar.gz -i debuild
```

提示

A URL such as “**https://www.example.org/DL/package-1.0.tar.gz**”may be used for the **-a** option.

提示

A URL such as “**https://arm.koji.fedoraproject.org/packages/ibus/1.5.7/-3.fc21/src/ibus-1.5.7-3.fc21.src.rpm**”may be used for the **-a** option, too.

16.2 debmake -b

The **debmake** command with the **-b** option provides an intuitive and flexible method to create the initial template **debian/control** file. This file defines the split of the Debian binary packages with the following stanzas:

- **Package:**
- **Architecture:** (e.g. **amd64**)
- **Multi-Arch:** (see “第 10.10 节”)
- **Depends:**

- **Pre-Depends:**

The **debmake** command also sets an appropriate set of substvars (substitution variables) used in each pertinent dependency stanza.

我们在这里直接引用 **debmake** 手册页中的相关一部分内容。

-b "*binarypackage[:type],*⋯", --binaryspec "*binarypackage[:type],*⋯" set the binary package specs by a comma separated list of *binarypackage:type* pairs. Here, *binarypackage* is the binary package name, and the optional *type* is chosen from the following *type* values:

- **bin**: C/C++ compiled ELF binary code package (any, foreign) (default, alias: **""**, i.e., **null-string**)
- **data**: Data (fonts, graphics, ⋯) package (all, foreign) (alias: **da**)
- **dev**：库开发软件包（any，same）（别名：**de**）
- **doc**：文档软件包（all，foreign）（别名：**do**）
- **lib**：库软件包（any，same）（别名：**l**）
- **perl**：Perl 脚本软件包（all，foreign）（别名：**pl**）
- **python3**: Python (version 3) script package (all, foreign) (alias: **py3**, **python**, **py**)
- **ruby**：Ruby 脚本软件包（all，foreign）（别名：**rb**）
- **nodejs**：基于 Node.js 的 JavaScript 软件包（all，foreign）（别名：**js**）
- **script**: Shell and other interpreted language script package (all, foreign) (alias: **sh**)

The pair values in the parentheses, such as (any, foreign), are the **Architecture** and **Multi-Arch** stanza values set in the **debian/control** file. In many cases, the **debmake** command makes good guesses for *type* from *binarypackage*. If *type* is not obvious, *type* is set to **bin**.

Here are examples for typical binary package split scenarios where the upstream Debian source package name is **foo**:

- Generating an executable binary package **foo**:
  - "**-b**'**foo:bin'**", or its short form "**-b'-'**", or no **-b** option
- Generating an executable (python3) binary package **python3-foo**:
  - "**-b**'**python3-foo:py'**", or its short form "**-b**'**python3-foo'**"
- Generating a data package **foo**:
  - "**-b**'**foo:data'**", or its short form "**-b'-:data'**"
- Generating a executable binary package **foo** and a documentation one **foo-doc**:
  - "**-b**'**foo:bin,foo-doc:doc'**", or its short form "**-b'-:-doc'**"
- Generating a executable binary package **foo**, a library package **libfoo1**, and a library development package **libfoo-dev**:
  - "**-b**'**foo:bin,libfoo1:lib,libfoo-dev:dev'**"or its short form "**-b'-,libfoo1,libfoo-dev'**"

如果源码树的内容和类型的设置不一致，**debmake** 命令会发出警告。

16.3 debmake -cc

debmake 命令在带上 **-cc** 选项时可以向标准输出打印整个源码树的版权和许可证概要信息。

```
$ tar -xvzf package-1.0.tar.gz
$ cd package-1.0
$ debmake -cc | less
```

如果转而使用 **-c** 选项，程序将提供较短的报告。

16.4 Snapshot upstream tarball (-d, -t)

This test building scheme is suitable for git repositories organized as described in **gbp-buildpackage**(7), which uses the master, upstream, and pristine-tar branches.

The upstream snapshot from the upstream source tree in the upstream VCS can be made with the **-d** option if the upstream supports the "**make dist**"equivalence.

```
$ cd /path/to/upstream-vcs
$ debmake -d -i debuild
```

除此之外，也可使用 **-t** 选项以使用 **tar** 命令生成上游源码包。

```
$ cd /path/to/upstream-vcs
$ debmake -p package -t -i debuild
```

Unless you provide the upstream version with the **-u** option or with the **debian/changelog** file, a snapshot upstream version is generated in the **0~%y%m%d%H%M** format, e.g., **0~1403012359**, from the UTC date and time.

If the upstream VCS is hosted in the *package***/** directory instead of the *upstream-vcs***/** directory, the "**-p** *package*"can be skipped.

如果版本控制系统中的上游源码树包含了 **debian/*** 文件，**debmake** 命令在带有 **-d** 选项或者 **-t** 选项并结合 **-i** 选项可以自动化进行使用这些 **debian/*** 文件从版本控制系统快照中构建非原生软件包的流程。

```
$ cp -r /path/to/package-0~1403012359/debian/. /path/to/upstream-vcs/debian
$ dch
  ... update debian/changelog
$ git add -A .; git commit -m "vcs with debian/*"
$ debmake -t -p package -i debuild
```

This **non-native** Debian binary package building scheme without the real upstream tarball is considered a **quasi-native** Debian package. See "第 11.13 节"for more details.

16.5 debmake -j

这是测试性功能。

生成多个二进制软件包通常比只生成一个二进制软件包需要投入更多的工作量。对源码包进行测试构建是其中的必要一环。

例如，我们考虑将相同的 *package-1.0***.tar.gz**（参见"第 5 章"）打包并生成多个二进制软件包。

- 调用 **debmake** 命令并使用 **-j** 选项以测试构建并报告结果。

  ```
  $ debmake -j -a package-1.0.tar.gz
  ```

- 请检查 *package***.build-dep.log** 文件最后的几行以确定 **Build-Depends** 所需填写的构建依赖。（您不需要在 **Build-Depends** 中列出 **debhelper**、**perl** 或 **fakeroot** 所使用的软件包。在只生成单个软件包的情况下也是如此。）
- 请检查 *package***.install.log** 的文件内容以确定各个文件的安装路径，从而决定如何将它们拆分成多个软件包。
- 调用 **debmake** 命令以开始准备打包信息。

  ```
  $ rm -rf package-1.0
  $ tar -xvzf package-1.0.tar.gz
  $ cd package-1.0
  $ debmake -b"package1:type1, ..."
  ```

- 请使用以上信息更新 **debian/control** 和 **debian/***binarypackage***.install** 文件。
- 按需更新其它 **debian/*** 文件。
- 使用 **debuild** 或等效的其它工具构建 Debian 软件包。

```
$ debuild
```

- 所有由 **debian/***binarypackage***.install** 文件指定的二进制软件包条目均会生成 *binarypackage_version-revision_arch***.deb** 的安装包。

注意

The **-j** option for the **debmake** command invokes **dpkg-depcheck**(1) to run **debian/rules** under **strace**(1) to obtain library dependencies. Unfortunately, this is very slow. If you know the library package dependencies from other sources such as the SPEC file in the source, you may just run the "**debmake** ⋯" command without the **-j** option and run the "**debian/rules install**"command to check the install paths of the generated files.

16.6 debmake -k

这是测试性功能。

在使用上游新发行版本更新软件包时，**debmake** 可以使用已有的 **debian/copyright** 文件和整个更新的源码树文件进行对比验证版权和许可证信息。

```
$ cd package-vcs
$ gbp import-orig --uscan --pristine-tar
... update source with the new upstream release
$ debmake -k | less
```

The "**debmake -k**"command parses the **debian/copyright** file from the top to the bottom and compares the license of all the non-binary files in the current package with the license described in the last matching file pattern entry of the **debian/copyright** file.

在您编辑自动生成的 **debian/copyright** 文件时，请确保将最通用的文件匹配模式放在文件前部，最精确的匹配模式放在后部。

提示

For all new upstream releases, run the "**debmake -k**"command to ensure that the **debian/copyright** file is current.

16.7 debmake -P

调用 **debmake** 命令并带上 **-P** 选项将会严厉地检查所有自动生成文件的版权和许可证文本信息；即使它们都使用宽松的许可证也是如此。

此选项不止会影响正常执行过程中所生成的 **debian/copyright** 文件的内容，也会影响带参数 **-k**、**-c**、**-cc** 和 **-ccc** 选项的输出内容。

16.8 debmake -T

调用 **debmake** 命令并带上 **-T** 选项会额外输出详细的教程注释行。这些行在模板文件中用 **###** 进行标注。

16.9 debmake -x

debmake 生成的模板文件数量由 **-x**[01234] 选项进行控制。

- 请参见“第 14.1 节”以了解与拣选使用模板文件的方式。

注意

debmake 命令不会修改任何已存在的配置文件。