debian

Debian 維護者指南

Osamu Aoki, 楊博遠, Fonzie Huang, 且鄭原真

February 5, 2025

Debian 維護者指南
by Osamu Aoki, 楊博遠, Fonzie Huang, 且鄭原真

本指南在撰寫過程中參考了以下幾篇文件：

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

本指南的最新版本應當可以在下列位置找到：

- 在“debmake-doc 套件”中，以及
- 位於“Debian 文件網站”。

Contents

Abstract

本篇《Debian 維護者指南》（2025-02-05）教材文件針對普通 Debian 使用者和未來的開發者，描述了使用 **debmake** 命令構建 Debian 套件的方法。

本指南注重描述現代的打包風格，同時提供了許多簡單的範例。

- POSIX shell 指令碼打包
- Python3 指令碼打包
- C 和 Makefile/Autotools/CMake
- 含有共享程式庫的多個二進位制套件的打包，等等。

本篇《Debian 維護者指南》可看作《Debian 新維護者手冊》的繼承文件。

Chapter 1

前言

If you are a somewhat experienced Debian user [1], you may have encountered the following situations:

- 您想要安裝某一個套件，但是該軟體在 Debian 倉庫中尚不存在。
- 您想要將一個 Debian 套件更新為上游的新版本。
- 您想要新增某些補丁來修復某個 Debian 套件中的缺陷。

If you want to create a Debian package to fulfill these needs and share your work with the community, you are the target audience of this guide as a prospective Debian maintainer. [2] Welcome to the Debian community.

Debian has many social and technical rules and conventions to follow, as it is a large volunteer organization with a rich history. Debian has also developed an extensive array of packaging and archive maintenance tools to build consistent sets of binary packages that address many technical objectives:

- packages have clearly specified package dependencies and patches and build correctly from scratch in a clean build environment (“節 6.6”, “節 6.9”, “節 4.6”)
- packages build across many architectures (“節 9.3”)
- builds are reproducible (“節 10.7”)
- multiarch is supported (“節 10.10”)
- bootstrapping new architectures is possible (“節 10.5”)
- builds use specific compiler flags to harden security (“節 10.6”)
- packages are split optimally into multiple binary packages (“節 10.11”)
- library names and contents are managed to ensure smooth transitions on upgrades (“節 10.18”)
- installations use interactive prompts correctly (if at all) (“節 10.22”)
- continuous integration is used to ensure quality (“節 10.4”)
- …

These factors can be overwhelming for many new prospective Debian maintainers. This guide aims to provide entry points to help them get started. It covers the following:

- 作為未來潛在的維護者，您在參與 Debian 工作之前應該瞭解的東西。
- 製作一個簡單的 Debian 套件大概流程如何。
- 製作 Debian 套件時有哪些規則。

1You need to know a little about Unix programming, but you don’t need to be an expert. You can learn about basic Debian system handling from the “Debian Reference”. It also contains pointers for learning about Unix programming.

2If you’re not interested in sharing the Debian package, you can address your local needs by compiling and installing the fixed upstream source package into **/usr/local/**.

- Tips for making the Debian package with minimal effort.
- Examples of making Debian packages in typical scenarios.

The author recognized the limitations of updating the original “New Maintainers’ Guide”with the **dh-make** package and decided to create an alternative tool with accompanying documentation to address modern requirements such as multi-arch. This resulted in the **debmake** package, initially released as version 4.0 in 2013. The current **debmake** version is 4.5.1. It comes with this updated “Guide for Debian Maintainers”in the **debmake-doc** package (version: 1.21-1). (In 2016, **dh-make** was ported from Perl to Python with updated features.)

Many chores and tips have been integrated into the **debmake** command allowing this guide to be terse. This guide also offers many packaging examples for you to get started.

注意

合適地建立並維護 Debian 套件需要佔用許多時間。Debian 維護者在接受這項挑戰時一定要確保 既能精通技術又能勤勉投入精力。

Some important topics are explained in detail. While some may seem irrelevant to you, please be patient. Certain corner cases are omitted, and some topics are only covered through external references. These are intentional choices to keep this guide simple and maintainable.

Chapter 2

概覽

對 *package-1.0*.**tar.gz**，一個包含了簡單的、符合“GNU 編碼標準”和“FHS（檔案系統層級規範）”的 C 語言源代碼的程式來說，它在 Debian 下打包工作可以按照下列流程，使用 **debmake** 命令進行。

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

如果跳過了對生成的配置檔案的手工調整流程，則最終生成的二進位制套件將缺少有意義的套件描述資訊，但是仍然能為 **dpkg** 命令所使用，在本地部署環境下正常工作。

注意

The **debmake** command only provides decent template files. These template files must be manually adjusted to their perfection to comply with the strict quality requirements of the Debian archive, if the generated package is intended for general consumption.

If you are new to Debian packaging, focus on understanding the overall process rather than worrying about the details.

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 命令設計提供如下特性與功能：

- 現代的打包風格
  - **debian/copyright**: “DEP-5”compliant
  - **debian/control**: **substvar** support, **multiarch** support, multi binary packages, ⋯
  - **debian/rules**: **dh** syntax, compiler hardening options, ⋯
- 靈活性
  - many options (see “節 16.2”, “章 15”, and “章 16”)
- 合理的預設行為
  - 執行過程不中斷，輸出乾淨的結果
  - 生成多架構支援（multiarch）的套件，除非明確指定了 **-m** 選項。
  - generate the non-native Debian package with the Debian source format “**3.0 (quilt)**”, unless the **-n** option is explicitly specified.

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.

- 額外的功能
  - verification of the **debian/copyright** file against the current source (see “節 16.6”)

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**, **-I**, and **-w** options to protect them from shell interference.

提示

非原生套件是標準的 Debian 套件。

提示

本文件中所有套件構建範例的詳細日誌可以由“節 14.14”一段給出的操作來取得。

注

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

Chapter 3

預備知識

Here are the prerequisites you need to understand before getting involved with Debian.

3.1 Debian 社群的工作者

在 Debian 社群中有這幾類常見的角色：

- 上游作者（**upstream author**）：程式的原始作者。
- 上游維護者（**upstream maintainer**）：目前在上游維護程式程式碼的人。
- 套件維護者（**maintainer**）：製作並維護該程式 Debian 套件的人。
- 贊助者（**sponsor**）：幫助維護者上傳套件到 Debian 官方倉庫的人（在通過內容檢查之後）。
- 導師（**mentor**）：幫助新手維護者熟悉和深入打包的人。
- **Debian** 開發者（DD, Debian Developer）：Debian 社群的官方成員。DD 擁有對 Debian 官方倉庫上傳的全部許可權。
- **Debian** 維護者（Debian Maintainer, DM）：擁有對 Debian 官方倉庫部分上傳許可權的人。

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 "`" 節 3.8"`").

3.2 如何做出貢獻

請參考下列文件來了解應當如何為 Debian 作出貢獻：

- "您如何協助 Debian？"（官方）
- "The Debian GNU/Linux FAQ, Chapter 13 - Contributing to the Debian Project"(semi-official)
- "Debian Wiki, HelpDebian"（補充內容）
- "Debian 新成員站點"（官方）
- "Debian Mentors FAQ"（補充內容）

3.3 Debian 的社會驅動力

為做好準備和 Debian 進行互動，請理解 Debian 的社會動力學：

- We are all volunteers.
  - You can't impose tasks on others.
  - You should be self-motivated to do things.
- 友好的合作是我們前行的動力。
  - 您的貢獻不應致使他人增加負擔。
  - 只有當別人欣賞和感激您的貢獻時，它才有真正的價值。
- Debian is not a school where you get automatic attention from teachers.
  - You should be able to learn many things independently.
  - Attention from other volunteers is a scarce resource.
- Debian 一直在不斷進步。
  - Debian 期望您製作出高質量的套件。
  - 您應該隨時調整自己來適應變化。

在這篇指南之後的部分中，我們只關注打包的技術方面。因此，請參考下面的文件來理解 Debian 的社會動力學：

- "Debian: 17 years of Free Software, "do-ocracy", and democracy"(Introductory slides by the ex-DPL)

3.4 技術提醒

Here are some technical reminders to help other maintainers work on your package easily and effectively, maximizing the output of Debian as a whole.

- 讓您的套件容易除錯（debug）。
  - 保持您的套件簡單易懂。
  - 不要對套件過度設計。
- 讓您的套件擁有良好的文件記錄。
  - 使用可讀的程式碼風格。
  - 在程式碼中寫註釋。
  - 格式化程式碼使其風格一致。
  - 維護套件的 git 倉庫 1。

注

對軟體進行除錯（debug）通常會比編寫初始可用的軟體花費更多的時間。

It is unwise to run your base system under the **unstable** suite, even for development purposes.

- Creation and verification of binary **deb** packages should use a minimal **unstable** chroot as described in "節 4.6".

1絕大多數 Debian 維護者使用 **git** 而非其它版本控制系統，如 **hg**、**bzr** 等等。

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

注

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 文件

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

- “Debian Policy Manual”
  - The official “must follow”rules (https://www.debian.org/doc/devel-manuals#policy)
- “Debian Developer’s Reference”
  - The official “best practice”document (https://www.debian.org/doc/devel-manuals#devref)
- “Guide for Debian Maintainers” — this guide
  - 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”.

如果本指南文件的內容與官方的 Debian 文件有所衝突，那麼官方的那些總是對的。請使用 **reportbug** 工具對 **debmake-doc** 套件報告問題。

這裡有一些替代性的教材文件，您可以與本指南一起閱讀進行參考：

- “Debian Packaging Tutorial”
  - https://www.debian.org/doc/devel-manuals#packaging-tutorial
  - https://packages.qa.debian.org/p/packaging-tutorial.html
- “Ubuntu Packaging Guide”(Ubuntu is Debian based.)
  - http://packaging.ubuntu.com/html/
- “Debian New Maintainers’ Guide”(predecessor of this tutorial, deprecated)
  - 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 幫助資源

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

- 套件的資訊可以使用 **aptitude**、**apt-cache** 以及 **dpkg** 命令進行檢視。
- 所有相關套件在 **/usr/share/doc/**套件名目錄下的檔案。
- 所有相關命令在 **man** 命令下輸出的內容。
- 所有相關命令在 **info** 命令下輸出的內容。
- “debian-mentors@lists.debian.org 郵件列表存檔”的內容。
- “debian-devel@lists.debian.org 郵件列表存檔”的內容。

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.

如果您對打包仍然存在疑問，您可以使用以下方式與他人進行溝通：

- 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.
- Teams focusing on a specific set of packages. (Full list at https://wiki.debian.org/Teams)
- 特定語言的郵件列表。
  - “debian-devel-{french,italian,portuguese,spanish}@lists.debian.org”
  - “debian-chinese-gb@lists.debian.org”（該郵件列表用於一般的（簡體）中文討論。）
  - “debian-devel@debian.or.jp”

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

注意

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 倉庫狀況

請了解 Debian 倉庫的當前狀況。

- Debian 已經包含了絕大多數種類程式的套件。
- Debian 倉庫內套件的數量是活躍維護者的數十倍。
- 遺憾的是，某些套件缺乏維護者的足夠關注。

因此，對已經存在於倉庫內的套件做出貢獻是十分歡迎的（這也更有可能得到其他維護者的支援和協助上傳）。

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 貢獻流程

這裡使用類 Python 虛擬碼，給出了對 Debian 貢獻名為 **program** 的軟體所走的貢獻流程：

```
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：

- 對 exist_in_debian() 和 is_team_maintained()，需檢查：
  - **aptitude** 命令

  - “Debian 套件”網頁
  - Debian wiki “Teams”page

- 對 is_orphaned()、is_RFA() 和 is_ITPed_by_others()，需檢查：
  - **wnpp-alert** 命令的輸出。
  - “需要投入精力和未來的套件（WNPP）”
  - “Debian 缺陷報告記錄：在 unstable 版本中 wnpp 偽套件的缺陷記錄”
  - “需要“關愛”的 Debian 套件”
  - “基於 debtags 瀏覽 wnpp 缺陷記錄”
- 對於 is_good_program()，請檢查：
  - 這個程式應當有用。
  - 這個程式不應當對 Debian 系統引入安全和維護上的問題。
  - 這個程式應當有良好的文件，其原始碼需要可被理解（即，未經混淆）。
  - 這個程式的作者同意軟體被打包，且對 Debian 態度友好。2
- 對 is_it_DFSG()，及 is_its_dependency_DFSG()，請檢查：
  - “Debian 自由軟體指導方針”（DFSG）。
- 對 is_it_distributable()，請檢查：
  - 該軟體必須有一個許可證，其中應當允許軟體被髮行。

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

- “5.1. 新套件”。
- “5.9. 移動、刪除、重新命名、丟棄、接手和重新引入套件”。

3.9 新手貢獻者和維護者

新手貢獻者和維護者可能想知道在開始對 Debian 進行貢獻之前需要事先學習哪些知識。根據您個人的側重點不同，下面有我的一些建議供您參考：

- 打包
  - **POSIX shell** 和 **make** 的基本知識。
  - 一些 **Perl** 和 **Python** 的入門知識。
- 翻譯
  - 基於 PO 的翻譯系統的工作原理和基本知識。
- 文件
  - Basics of text markups (XML, ReST, Wiki, ⋯).

新手貢獻者和維護者可能想知道從哪裡開始對 Debian 進行貢獻。根據您掌握的技能，下面有我的一些建議供您參考：

- **POSIX shell**、**Perl** 和 **Python** 的技巧：
  - 對 Debian 安裝程式提交補丁。
  - Send patches to the Debian packaging helper scripts such as **devscripts**, **sbuild**, **schroot**, etc. mentioned in this document.

2這一條不是絕對的要求，但請注意：遇上不友好的上游可能需要大家為此投入大量精力，而一個友好的上游則能協助解決程式的各類問題。

- **C** 和 **C++** 技能：
  - 對具有 **required** 和 **important** 優先順序的套件提交補丁。
- 英語之外的技能：
  - 對 Debian 安裝程式專案提交補丁。
  - 為具有 **required** 和 **important** 優先順序的套件中的 PO 檔案提交補丁。
- 文件技能：
  - 更新“Debian 維基（Wiki）”中的內容。
  - 對已有的“Debian 文件”提交補丁。

這些活動應當能讓您在各位 Debian 社群成員之間得到存在感，從而建立您的信譽與名聲。
新手維護者應當避免打包具有潛在高度安全隱患的程式：

- **setuid** 或 **setgid** 程式
- 背景服務程序（**daemon**）程式
- 安裝至 **/sbin/** 或 **/usr/sbin/** 目錄的程式

在積累足夠的打包經驗後，您可以再嘗試打包這樣的程式。

Chapter 4

工具的配置

build-essential 套件必須在構建環境內預先安裝。

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.

注意

這裡展示的工具配置方式僅作為範例提供，可能與系統上最新的套件相比有所落後。Debian 的開發具有一個移動的目標。請確保閱讀合適的文件並按照需要更新配置內容。

4.1 Email setup

許多 Debian 維護工具識別並使用 shell 環境變數 **$DEBEMAIL** 和 **$DEBFULLNAME** 作為作為您的電子郵件地址和名稱。

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

新增至 **~/.bashrc** 檔案

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

注

The above is for the author of this manual. The configuration and operation examples presented in this manual use these email address and name settings. You must use your email address and name for your system.

4.2 mc setup

mc 命令提供了管理檔案的簡單途徑。它可以開啟二進位制 **deb** 檔案，並僅需對二進位制 **deb** 檔案按下回車鍵便能檢查其內容。它呼叫了 **dpkg-deb** 命令作為其後端。我們可以按照下列方式對其配置，以支

1這裡假設您正在使用 Bash 並以此作為登入預設 shell。如果您設定了其它登入 shell，例如 Z shell，請使用它們對應的配置檔案替換 *~/.bashrc* 檔案。

援簡易 **chdir** 操作。
新增至 **~/.bashrc** 檔案

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

4.3 git setup

如今 **git** 命令已成為管理帶歷史的原始碼樹的必要工具。
git 命令的使用者級全域性配置，如您的名字和電子郵件地址，儲存在 **~/.gitconfig** 檔案中，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
```

如果您仍然只習慣 CVS 或者 Subversion 的命令風格，您可以使用如下方式設定幾個命令別名。

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

您可以使用如下命令檢查全域性配置。

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

提示

有必要使用某些圖形介面 git 工具，例如 **gitk** 或 **gitg** 命令來有效地處理 git 倉庫的歷史。

4.4 quilt setup

quilt 命令提供了記錄修改的一個基本方式。對 Debian 打包來說，該工具需要進行設定，從而在 **debian/patches/** 目錄內記錄修改內容，而非使用預設的 **patches/** 目錄。
為了避免改變 **quilt** 命令自身的行為，我們在這裡建立一個用於 Debian 打包工作的命令別名：**dquilt**。之後，我們將對應內容寫入 **~/.bashrc** 檔案。下面給出的第二行為 **dquilt** 命令提供與 **quilt** 命令相同的命令行補全功能。
新增至 **~/.bashrc** 檔案

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

然後我們來建立具有如下內容的 **~/.quiltrc-dpkg** 檔案。

```
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.

要取得使用範例，請檢視“節 5.9”。

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 setup

debsign 命令由 **devscripts** 套件提供，它可以使用使用者的 GPG 私鑰對 Debian 軟體包進行簽名。

debuild 命令同樣由 **devscripts** 套件提供，它可以構建二進位制套件並使用 **lintian** 命令對其進行檢查。**lintian** 命令的詳細輸出通常都很實用。

您可以將下列內容寫入 **~/.devscripts** 檔案來進行配置。

```
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 “章 8”).

當前情況下，使用 4096 位的 RSA 金鑰是較好的做法。另見“建立一個新 GPG 金鑰”。

4.6 sbuild setup

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** 套件能找到所構建套件中的缺陷。
- The **piuparts** package to find bugs in the package.
- The **autopkgtest** package to find bugs in the package.
- **ccache** 套件可以加速 **gcc**。（可選）
- **libeatmydata1** 套件可以加速 **dpkg**。（可選）
- 並行執行 **make** 以提高構建速度。（可選）

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:

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

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

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

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

可選的設定項可能造成負面影響。如果有疑問，請關閉它們。

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

注

並行的 **make** 可能在某些已有套件上執行失敗，它同樣會使得構建日誌難以閱讀。

提示

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

4.7 Persistent chroot setup

注

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-$suffix /srv/chroot/unstable-amd64 ↩
    -desktop
$ sudo tee /etc/schroot/chroot.d/unstable-amd64-desktop << 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/desk-top/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 setup

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

您應當在本地設定 HTTP 快取代理以節約查詢 Debian 軟體倉庫的頻寬。可以考慮以下幾種選項：

- 特化的 HTTP 快取代理，使用 **apt-cacher-ng** 套件。
- 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 倉庫

您可以使用 **reprepro** 套件搭建私有 Debian 倉庫。

4.11 Virtual machines

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 Local network with virtual machines

In order to access virtual machines easily over the local network, setting up multicast DNS service discovery infrastructure by installing **avahi-utils** is a good idea.

For all running virtual machines and the host PC, we can use each host name appended with **.local** for SSH to access each other.

Chapter 5

Simple packaging

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

這裡給出了從簡單的 C 語言原始碼建立簡單的 Debian 套件的例子，並假設上游使用了 **Makefile** 作為構建系統。

我們假設上游原始碼壓縮包（tarball）名稱為 **debhello-0.0.tar.gz**。

這一類原始碼設計可以用這樣的方式安裝成為非系統檔案：

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

在其它更加複雜的構建系統下構建 Debian 套件的例子可以在“章 14”找到。

5.2 大致流程

從上游原始碼壓縮包 **debhello-0.0.tar.gz** 構建單個非原生 Debian 套件的大致流程可以總結如下：

- 維護者取得上游原始碼壓縮包 **debhello-0.0.tar.gz** 並將其內容解壓縮至 **debhello-0.0** 目錄中。
- **debmake** 命令對上游原始碼樹進行 debian 化（debianize），具體來說，是建立一個 **debian** 目錄並僅向該目錄中新增各類模板檔案。
  - 名為 **debhello_0.0.orig.tar.gz** 的符號連結被建立並指向 **debhello-0.0.tar.gz** 檔案。
  - 維護者須自行編輯修改模板檔案。
- **debuild** 命令基於已 debian 化的原始碼樹構建二進位制套件。
  - **debhello-0.0-1.debian.tar.xz** 將被建立，它包含了 **debian** 目錄。

套件構建的大致流程

```
$ 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

如果上游原始碼壓縮包提供了 **.tar.xz** 格式文件，請使用這樣的壓縮包來替代 **.tar.gz** 或 **.tar.bz2** 格式。**xz** 壓縮與 **gzip** 或 **bzip2** 壓縮相比提供了更好的壓縮比。

5.3 什麼是 debmake？

注

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. (“章 15”)

- It creates good template files for the Debian packages.
- 它總是將大多數選項的狀態與引數設定為合理的預設值。
- 它能產生上游原始碼套件，並按需建立所需的符號連結。
- 它不會覆寫 **debian/** 目錄下已存在的配置文件。
- 它支援多架構（**multiarch**）套件。
- It provides short extracted license texts as **debian/copyright** in decent accuracy to help license review.

這些特性使得使用 **debmake** 進行 Debian 打包工作變得簡單而現代化。

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 什麼是 debuild？

這裡給出與 **debuild** 命令類似的一系列命令的一個彙總。

- **debian/rules** 檔案定義了 Debian 二進位制套件該如何構建。
- **dpkg-buildpackge** 是構建 Debian 二進位制套件的正式命令。對於正常的二進位制構建，它大體上會執行以下操作：
  - “**dpkg-source --before-build**”(apply Debian patches, unless they are already applied)
  - “**fakeroot debian/rules clean**”
  - “**dpkg-source --build**”(build the Debian source package)
  - “**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“節 4.5”的說明設定了 **-us** 和 **-us** 選項的話，本步驟將會被跳過。您需要手動執行 **debsign** 命令。
- **debuild** 命令是 **dpkg-buildpackage** 命令的一個封裝指令碼，它可以使用合適的環境變數來構建 Debian 二進位制套件。
- The **sbuild** command is a wrapper script to build the Debian binary package under the proper chroot environment with the proper environment variables.

注

如需瞭解詳細內容，請見 **dpkg-buildpackage**(1)。

5.5 第一步：取得上游原始碼

我們先要取得上游原始碼。

下載 **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
```

這裡的 C 原始碼 **hello.c** 非常的簡單。

hello.c

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

這裡，原始碼中的 **Makefile** 支援“GNU 編碼標準”和“FHS（檔案系統層級規範）”。特別地：

- 構建二進位制程式時會考慮 **$(CPPFLAGS)**、**$(CFLAGS)**、**$(LDFLAGS)**，等等。
- 安裝檔案時採納 **$(DESTDIR)** 作為目標系統鏡像的路徑字首
- 安裝檔案時使用 **$(prefix)** 的值，以便我們將其設定覆蓋為 **/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

對 **$(CFLAGS)** 的 **echo** 命令用於在接下來的例子中驗證所設定的構建引數。

5.6 Step 2: Generate template files with debmake

debmake 命令的輸出十分詳細，如下所示，它可以展示程式的具體操作內容。

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 命令基於命令列選項產生所有這些模板檔案。如果沒有指定具體選項，**debmake** 命令將為您自動選擇合理的預設值：

- 原始碼套件名稱：**debhello**
- 上游版本：**0.0**
- 二進位制套件名稱：**debhello**
- Debian 修訂版本：**1**
- 套件型別：**bin**（ELF 二進位制可執行程式套件）
- The **-x** option: **-x1** (without maintainer script supports for simplicity)

注

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

我們來檢查一下自動產生的模板檔案。
基本 **debmake** 命令執行後的原始碼樹。

```
 $ 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
```

這裡的 **debian/rules** 檔案是應當由套件維護者提供的構建指令碼。此時該檔案是由 **debmake** 命令產生的模板檔案。

debian/rules（模板檔案）：

```
 $ 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
```

這便是使用 **dh** 命令時標準的 **debian/rules** 檔案。（某些內容已被註釋，可供後續修改使用。）

這裡的 **debian/control** 檔案提供了 Debian 套件的主要參數。此時該檔案是由 **debmake** 命令產生的模板檔案。

debian/control（模板檔案）：

```
 $ 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 “章 6”.

注

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.

這裡的 **debian/copyright** 提供了 Debian 套件版權資料的總結。此時該檔案是由 **debmake** 命令產生的模板檔案。

debian/copyright（模板檔案）：

```
 $ 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：編輯模板檔案

作為維護者，要製作一個合適的 Debian 套件當然需要對模板內容進行一些手工的調整。

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（維護者版本）：

```
 $ 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** 檔案中匯出 =**DH_VERBOSE** 環境變數可以強制 **debhelper** 工具輸出細粒度的構建報告。

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

如上匯出 **DEB_CFLAGS_MAINT_APPEND** 可以強制 C 編譯器給出所有型別的警告內容。

如上匯出 **DEB_LDFLAGS_MAINT_APPEND** 可以強制連結器只對真正需要的程式庫進行連結。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**”.

這裡是維護者版本的 **debian/control** 和 **debian/copyright** 檔案。

debian/control（維護者版本）：

```
 $ 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（維護者版本）：

```
 $ 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這裡的做法是為了避免在依賴程式庫情況複雜的情況下過度連結，例如某些 GNOME 程式。這樣做對這裡的簡單例子來說並不是必要的，但應當是無害的。

```
 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/. 下面的模板檔案（**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

對於來自 **debhelper** 套件的各個 **dh**_* 命令來說，它們在讀取所使用的配置文件時通常把以 # 開頭的行視為註釋行。

5.8 Step 4: Building package with debuild

您可以使用 **debuild** 或者等效的命令工具（參見"節 5.4"）在這個原始碼樹內構建一個非原生 Debian 套件。命令的輸出通常十分詳細，如下所示，它會對構建中執行的操作進行解釋。

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.
```

這裡驗證了 **CFLAGS** 已經得到了更新，新增了 **-Wall** 和 **-pendatic** 引數；這是我們先前在 **DEB_CFLAGS_MAINT**
變數中所指定的。

根據 **lintian** 的報告，您應該如同後文中的例子那樣（請見“章 14”）為套件新增 man 手冊頁。我們這裡暫且跳過這部分內容。

現在我們來看看成果如何。

debhello 0.0 版使用 **debuild** 命令產生的文件：

```
 $ 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
```

您可以看見生成的全部檔案。

- **debhello_0.0.orig.tar.gz** 是指向上游原始碼壓縮包的符號連結。
- **debhello_0.0-1.debian.tar.xz** 包含了維護者生成的內容。
- **debhello_0.0-1.dsc** 是 Debian 原始碼套件的元資料檔案。
- **debhello_0.0-1_amd64.deb** 是 Debian 二進制套件。
- **debhello-dbgsym_0.0-1_amd64.deb** 是 Debian 的除錯符號二進位制套件。另請參見“節 10.21”。
- **debhello_0.0-1_amd64.build** 是構建日誌文件。
- **debhello_0.0-1_amd64.buildinfo** 是 **dpkg-genbuildinfo**(1) 生成的元資料檔案。

- **debhello_0.0-1_amd64.changes** 是 Debian 二進位制套件的元資料檔案。

debhello_0.0-1.debian.tar.xz 包含了 Debian 對上游原始碼的修改，具體如下所示。
壓縮檔案 **debhello_0.0-1.debian.tar.xz** 的內容：

```
 $ 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 包含了將要安裝至目標系統中的檔案。
The **debhello-debsym_0.0-1_amd64.deb** contains the debug symbol files to be installed to the target system.
所有二進位制包的包內容：

```
 $ 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
```

生成的依賴列表會給出所有二進位制套件的依賴。
生成的所有二進位制套件的依賴列表（**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)
```

注意

在將套件上傳至 Debian 倉庫之前，仍然有很多細節需要進行處理。

注

如果跳過了對 **debmake** 命令自動生成的配置檔案的手工調整步驟，所生成的二進位制套件可能缺少有用的套件描述資訊，某些政策的要求也無法滿足。這個不正式的套件對於 **dpkg** 命令來說可以正常處理，也許這樣對您本地的部署來說已經足夠好了。

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

The above “節 5.7”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 “節 5.7”in the above step-by-step example:

- “節 5.10”
- “節 5.11”
- “節 5.12”

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

debian/rules（備選的維護者版本）：

```
 $ 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

When generating the Debian source package by **dpkg-source** via **dpkg-buildpackage** in the following step of "節 5.8", 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

When generating the Debian source package by **dpkg-source** via **dpkg-buildpackage** in the following step of "節 5.8", 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 "節 5.8".
我們先來編輯上游原始碼。
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.
Remove unused template files

```
 $ 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

When generating the Debian source package by **dpkg-source** via **dpkg-buildpackage** in the following step of "節 5.8", 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 "節 5.8"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

Basics for packaging

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

為簡明起見，某些細節被有意跳過。請按需查閱對應命令的手冊頁，例如 **dpkg-source**(1)、**dpkg-buildpackage**(1)、**dpkg**(1)、**dpkg-deb**(1)、**deb**(5)，等等。

Debian 原始碼套件是一組用於構建 Debian 二進位制套件的輸入檔案，而非單個檔案。

Debian 二進位制套件是一個特殊的檔案檔案，其中包含了一系列可安裝的二進位制資料及與它們相關的資訊。

單個 Debian 原始碼套件可能根據 **debian/control** 檔案定義的內容產生多個 Debian 二進位制套件。

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

注

有許多封裝指令碼可用。合理使用它們可以幫助您理順工作流程，但是請確保您能理解它們內部的基本工作原理。

6.1 打包工作流

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

1. 下載上游原始碼壓縮包（tarball）並命名為 *package-version*.**tar.gz** 檔案。
2. 使上游提供的原始碼壓縮包解壓縮後的所有檔案儲存在 *package-version***/** 目錄中。
3. 上游的原始碼壓縮包被複制（或符號連結）至一個特定的檔名 *packagename_version*.**orig.tar.gz**。
   - 分隔 *package* 和 *version* 的符號從 **-**（連字元）更改為 _（下劃線）
   - 副檔名添加了 **.orig** 部分。
4. Debian 套件規範檔案將被新增至上游原始碼中，存放在 *package-version***/debian/** 目錄下。
   - **debian/*** 目錄下的必需技術說明檔案：

     debian/rules 構建 Debian 套件所需的可執行指令碼（參見“節 6.5”）

debian/control 套件配置檔案包含了原始碼套件名稱、原始碼構建依賴、二進位制套件名稱、二進位制套件依賴，等等。(參見"節 6.6")

debian/changelog Debian 套件歷史檔案，其中第一行定義了上游套件版本號和 Debian 修訂版本號（參見"節 6.7"）

debian/copyright 版權和許可證摘要資訊（參看"節 6.8"）

- 在 **debian/*** 下的可選配置檔案（參見"節 6.14"）：
- The **debmake** command invoked in the *package-version***/** directory may be used to provide the initial template of these configuration files.
  - 必備的配置檔案總會生成，無論是否提供 **-x0** 選項。
  - **debmake** 命令永遠不會覆寫任何已經存在的配置檔案。
- 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) 建立 Debian 二進位制套件。
     - *binarypackage_version-revision_arch*.**deb**
     - ⋯ (There may be multiple Debian binary package files.)
     - *package_version-revision_arch*.**changes**
     - *package_version-revision_arch*.**buildinfo**
6. 使用 **lintian** 命令檢查 Debian 套件的質量。(推薦)
   - Follow the rejection guidelines from ftp-master.
     - "套件被拒絕常見問題解答（REJECT-FAQ）"
     - "新套件（NEW）檢查清單"
     - "Lintian 自動拒絕（autoreject）" ("lintian 標籤列表")
7. Test the goodness of the generated Debian binary package manually by installing it and running its programs.
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.

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_*__source.changes__”(source-only upload)
- 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.

這裡，請將檔名中對應的部分使用下面的方式進行替換：

- 將 *package* 部分替換為 Debian 原始碼套件名稱
- 將 *binarypackage* 部分替換為 Debian 二進位制套件名稱
- 將 *version* 部分替換為上游版本號
- 將 *revision* 部分替換為 Debian 修訂號
- the *arch* part with the package architecture (e.g., **amd64**)

See also “Source-only uploads”.

提示

有很多種通過實踐摸索而得到的補丁管理方法和版本控制系統的使用策略與技巧。您沒有必要將它們全部用上。

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.

現代化的 Debian 打包工作流可以組織成一個簡單的模組化工作流，如下所示：

- 使用 **dh** 命令以自動呼叫來自 **debhelper** 套件的許多實用指令碼，以及
- 使用 **debian/** 目錄下的宣告式配置檔案配置它們的行為。

您幾乎總是應當將 **debhelper** 列為您的軟體包的構建依賴之一。本文件在接下來的內容中也假設您正在使用一個版本足夠新的 **debhelper** 協助進行打包工作。

注

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

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

6.3 套件名稱和版本

如果所取得上游原始碼的形式為 **hello-0.9.12.tar.gz**，您可以將 **hello** 作為上游原始碼名稱，並將 **0.9.12** 作為上游版本號。

組成 Debian 套件名稱的字元選取存在一定的限制。最明顯的限制應當是套件名稱中禁止出現大寫字母。這裡給出正則表示式形式的規則總結：

- 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”.

您必須為 Debian 打包工作適當地調整套件名稱和上游版本號。

為了能有效地使用一些流行的工具（如 **aptitude**）管理套件名稱和版本資訊，最好能將套件名稱保持在 30 字元以下；版本號和修訂號加起來最好能不超過 14 個字元。[2]

為了避免命名衝突，對使用者可見的二進位制套件名稱不應選擇任何常用的單詞。

如果上游沒有使用像 **2.30.32** 這樣正常的版本編號方案，而是使用了諸如 **11Apr29** 這樣包含日期、某些代號或者一個版本控制系統雜湊值等字串作為版本號的一部分的話，請在上游版本號中將這些部分移除。這些資訊可以稍後在 **debian/changelog** 檔案中進行記錄。如果您需要為軟體設計一個版本字符串，可以使用 **YYYYMMDD** 格式，如 **20110429** 的字串作為上游版本號。這樣能保證 **dpkg** 命令在升級時能正確地確定版本的先後關係。如果您想要確保萬一上游在未來重新採納正常版本編號方案，例如 **0.1** 時能夠做到順暢地遷移，可以另行使用 **0~YYMMDD** 的格式，如 **0~110429** 作為上游版本號。

版本字串可以按如下的方式使用 **dpkg** 命令進行比較。

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

版本比較的規則可以歸納如下：

- 字串按照起始到末尾的順序進行比較。
- 字元比數字大。
- 數字按照整數順序進行比較。
- 字元按照 ASCII 編碼的順序進行比較。

對於某些字元，如句點（**.**）、加號（**+**）和波浪號（**~**），有如下的特殊規則。

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

有一個稍需注意的情況，即當上游將 **hello-0.9.12-ReleaseCandidate-99.tar.gz** 這樣的版本當作預釋出版本，而將 **hello-0.9.12.tar.gz** 作為正式版本時。為了確保 Debian 套件升級能夠順暢進行，您應當修改版本號命名，如將上游原始碼壓縮包重新命名為 **hello-0.9.12~rc99.tar.gz**。

2對九成以上的套件來說，套件名稱都不會超過 24 個字元；上游版本號通常不超過 10 個字元，而 Debian 修訂版本號也通常不超過 3 個字元。

6.4 原生 Debian 套件

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 套件是較罕見的一種 Debian 套件格式。它通常只用於打包僅對 Debian 專案有價值、有意義的軟體。因此，該格式的使用通常不被提倡。

注意

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 套件不對 上游程式碼和 **Debian** 的修改進行區分，僅包含以下內容：

- *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 file

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:

簡單的 **debian/rules**：

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

%:
  dh $@
```

The **dh** command functions as the sequencer to call all required "**dh** *target*"commands at the right moment. [4]

- **dh clean**：清理原始碼樹中的檔案。
- **dh build**：在原始碼樹中進行構建
- **dh build-arch**：在原始碼樹中構建架構相關的套件

3**debmake** 命令會產生稍微複雜一些的 **debian/rules** 檔案。雖然如此，其核心結構沒有什麼變化。

[4]This simplicity is available since version 7 of the **debhelper** package. This guide assumes the use of **debhelper** version 13 or newer.

- **dh build-indep**：在原始碼中構建架構無關的套件
- **dh install**：將二進位制檔案安裝至 **$(DESTDIR)**
- **dh install-arch**：為架構相關的套件將二進位制檔案安裝至 **$(DESTDIR)** 中
- **dh install-indep**：為架構無關的套件將二進位制檔案安裝進入 **$(DESTDIR)** 中
- **dh binary**：產生 **deb** 檔案
- **dh binary-arch**：為架構相關的套件產生 **deb** 檔案
- **dh binary-indep**：為架構無關的套件產生 **deb** 檔案

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

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

See "節 9.2"and "節 9.3"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 file

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

- Debian 原始碼套件的參數資料
- Debian 二進位制套件的參數

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

注

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.
- Document changes in a specific, formal, and concise style.
  - 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:
  - Security-related bug fixes.
  - User interface changes.
- 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.

上游的變更日誌則會安裝至 **/usr/share/doc/***binarypackage* 目錄中，檔名為 **changelog.gz**。

上游的變更日誌是由 **dh_installchangelogs** 程式自動進行搜尋和處理的；它會使用大小寫不敏感的搜尋方式尋找上游程式碼中特定名稱的檔案，如 **changelog**、**changes**、**changelog.txt**、**changes.txt**、**history**、**history.txt** 或 **changelog.md**。除了根目錄，程式還會在 **doc/** 目錄和 **docs/** 目錄內進行搜尋。

6.8 debian/copyright file

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.

注意

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

注

如果您發現了這個許可證檢查工具存在一些問題，請對 **debmake** 套件提交缺陷報告並提供包含出現問題的許可證和版權資訊在內的相關文字內容。

6.9 debian/patches/* files

As demonstrated in “節 5.9”, 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:

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

注

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

注

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 “章 11”.

6.10 debian/source/include-binaries file

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 file

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

Basic debian/watch file:

```
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 "節 6.12").

See **uscan**(1), "節 9.4", "節 8.1", and "節 11.10"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**（上游原始碼）
- **hello-*version*.tar.gz.sig**（分離的簽名）nature)

我們現在來選擇最新的版本套裝。

Download the upstream tarball and its signature.

```
$ 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.

在確認金鑰身份 **80EE4A00** 值得信任之後，應當下載其公鑰並將其儲存在 **debian/upstream/signing-key.asc** 檔案中。

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 file

Install Salsa CI configuration file. See "節 11.3".

6.14 Other debian/* files

另外也可以新增一些可選的配置檔案並放入 **debian/** 目錄。它們大多用於控制由 **debhelper** 套件提供的 **dh_*** 命令的行為，但也有一些檔案會影響 **dpkg-source**、**lintian** 和 **gbp** 這些命令。

提示

Even an upstream source without its build system can be packaged just by using these files. See "節 14.2"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 "節 16.9"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** 可能沒有自動生成某些模板配置文件。如遇到這種情況，您可以使用文字編輯器手動建立遺失的檔案。
- Some configuration template files generated by the **debmake** command with an extra **.ex** suffix need to be activated by removing that suffix.

- 您應當刪除 **.ex** 命令生成但對您無用的配置模板檔案。
- 請按需複製配置模板檔案以匹配其對應的二進位制包名稱以及您的需求。

binarypackage*.bug-control** -x3 將安裝至 *binarypackage* 套件的 **usr/share/bug/binarypackage***/control** 位置。另請參考“節 9.11”。

binarypackage*.bug-presubj** -x3 將安裝至 *binarypackage* 套件的 **usr/share/bug/binarypackage***/presubj** 位置。另請參考“節 9.11”。

binarypackage*.bug-script** -x3 將安裝至 *binarypackage* 套件的 **usr/share/bug/binarypackage* or **usr/share/bug/***binarypackage***/script** 位置。另請參考“節 9.11”。

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

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

另請參考 **dh_bash-completion**(1)。

clean -x2 列出（構建前）未被 **dh_auto_clean** 命令清理，且需要手工清理的檔案。

另請參考 **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式要求每個使用者都對 **/etc** 目錄下的配置檔案進行修改，可以採取兩種常見辦法使其不作為 conffile 配置檔案出現，避免 **dpkg** 命令處理套件時給出不必要的處理選項。

- 在 **/etc** 目錄下建立一個符號連結，指向 **/var** 目錄下的某些檔案；實際存在的檔案則使用維護者指令碼（maintainer script）予以建立。
- 使用維護者指令碼（maintainer script）在 **/etc** 目錄下建立並維護配置所需的檔案。

另請參考 **dh_installdeb**(1)。

***binarypackage*.config** -x3 這是 **debconf config** 指令碼，用來在配置套件時向用戶詢問任何必需的問題。另請參見“節 10.22”。

binarypackage*.cron.hourly** -x3 安裝至 *binarypackage* 包內的 **etc/cron/hourly/binarypackage* 檔案。

另請參見 **dh_installcron**(1) 和 **cron**(8)。

binarypackage*.cron.daily** -x3 安裝至 *binarypackage* 包內的 **etc/cron/daily/binarypackage* 檔案。

另請參見 **dh_installcron**(1) 和 **cron**(8)。

binarypackage*.cron.weekly** -x3 安裝至 *binarypackage* 包內的 **etc/cron/weekly/binarypackage* 檔案。

另請參見 **dh_installcron**(1) 和 **cron**(8)。

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

另請參見 **dh_installcron**(1) 和 **cron**(8)。

binarypackage*.cron.d** -x3 安裝至 *binarypackage* 包內的 **etc/cron.d/binarypackage* 檔案。

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

binarypackage*.default** -x3 若該檔案存在，它將被安裝至 *binarypackage* 包中的 **etc/default/binarypackage* 位置。

參見 **dh_installinit**(1)。

***binarypackage*.dirs** -x1 列出 *binarypackage* 包中要建立的目錄。

參見 **dh_installdirs**(1)。

通常情況下您並不需要這麼做，因為所有的 **dh_install*** 命令都會自動建立所需的目錄。請僅在遇到問題時考慮使用這一工具。

***binarypackage*.doc-base** -x1 作為 *binarypackage* 包中的 **doc-base** 控制檔案進行安裝。

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

***binarypackage*.docs** -x1 列出要安裝在 *binarypackage* 包中的文件檔案。

參見 **dh_installdocs**(1)。

***binarypackage*.emacsen-compat** -x3 安裝至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/co** 檔案。

參見 **dh_installemacsen**(1)。

***binarypackage*.emacsen-install** -x3 安裝至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/inst** 檔案。

參見 **dh_installemacsen**(1)。

***binarypackage*.emacsen-remove** -x3 安裝至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/re** 檔案。

參見 **dh_installemacsen**(1)。

***binarypackage*.emacsen-startup** -x3 安裝至 *binarypackage* 包中的 **usr/lib/emacsen-common/packages/sta** 檔案。

參見 **dh_installemacsen**(1)。

binarypackage*.examples** -x1 列出要安裝至 *binarypackage* 包中 **usr/share/doc/binarypackage***/examples/** 位置下的範例檔案或目錄。

參見 **dh_installexamples**(1)。

gbp.conf -x1 如果該檔案存在，它將作為 **gbp** 命令的配置檔案發揮作用。

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

***binarypackage*.info** -x1 列出要安裝至 *binarypackage* 包中的 info 檔案。

參見 **dh_installinfo**(1)。

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

參見 **dh_installinit**(1)。

***binarypackage*.install** -x1 列出未被 **dh_auto_install** 命令安裝的其它應當安裝的檔案。

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

***binarypackage*.links** -x1 列出要生成符號連結的原始檔和目標檔案對。每一對連結均應在單獨的一行中列出，源檔案和目標檔案之間使用空白字元分隔。

參見 **dh_link**(1)。

binarypackage*.lintian-overrides** -x3 安裝至套件構建目錄的 **usr/share/lintian/overrides/binarypackage* 位置。該檔案用於消除 **lintian** 錯誤生成的診斷資訊。

參見 **dh_lintian**(1)、**lintian**(1) 和“Lintian 使用者手冊”。

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 這些檔案是 **debmake** 命令生成的 man 手冊頁模板檔案。請將其重新命名為合適的檔名並更新其內容。

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 列出要安裝的 man 手冊頁。

參見 **dh_installman**(1)。

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

安裝至 *binarypackage* 包中的 **usr/share/menu/***binarypackage* Debian 選單文件。

參見 **menufile**(5) 以瞭解其格式。另請參見 **dh_installmenu**(1)。

NEWS -x3 安裝至 **usr/share/doc/***binarypackage***/NEWS.Debian** 檔案。

參見 **dh_installchangelogs**(1)。

patches/* 這是 **-p1** 補丁檔案的集合，它們將在使用源程式碼構建之前應用在上游原始碼上。

debmake 預設不會生成補丁檔案。

參見 **dpkg-source**(1)、“節 4.4”和“節 5.9”。

patches/series -x1 **patches/*** 補丁檔案的應用順序。

***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 安裝至 **debian/control** 檔案列出的第一個二進位制套件中的 **usr/share/doc/***binarypackage***/README.Debian** 位置。

該檔案提供了針對該 Debian 套件的資訊。

參見 **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 如果該檔案存在，它將被安裝到 *binarypackage* 包下面的 **lib/systemd/system/binarypackage***.service** 位置。

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

source/format -x1 Debian 套件格式。

- 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.

另請參考 **dh_lintian**(1) 和 **lintian**(1)。

source/local-options -x1 **dpkg-source** 命令使用此內容作為它的選項，比較重要的選項有：

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

該檔案不會包含在生成的原始碼套件中，僅對維護者在版本控制系統中維護套件有意義。

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

source/local-patch-header -x1 自由格式的文字，將被包含在自動生成補丁的頂部。

該檔案不會包含在生成的原始碼套件中，僅對維護者在版本控制系統中維護套件有意義。

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 這些符號檔案如果存在，將交由 **dpkg-gensymbols** 命令進行處理和安裝。

參見 **dh_makeshlibs**(1) 和“節 10.16”。

***binarypackage*.templates** -x3 這是 **debconf** 模板檔案，用於在安裝過程中向用戶詢問必需的問題以正確配置套件。請參閱“節 10.22”。

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

TODO -x3 安裝至 **debian/control** 檔案列出的第一個二進位制包中的 **usr/share/doc/***binarypackage***/TODO.Deb** 檔案。

參見 **dh_installdocs**(1)。

binarypackage*.tmpfile** -x3 如果該檔案存在，它將被安裝至 *binarypackage* 包中的 **usr/lib/tmpfiles.d/binarypackage***.conf** 檔案。

參見 **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)

參見 **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 “章 4”, 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 “節 4.5”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** 檔案中的 **Files-Excluded** 一節中列出需要移除的檔案。
- 在 **debian/watch** 檔案中列出下載上游原始碼套件（tarball）所使用的 URL。
- 執行 **uscan** 命令以下載新的上游原始碼套件（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.
- 最後得到 tarball 的版本編號會附加一個額外的字尾 **+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

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

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

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 "節 4.5"may be too aggressive. You can work around this by dropping the **-I** option for **DEBUILD_DPKG_BUILDPACKAGE_OPTS** in "節 4.5"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 “節 4.5”.

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 Package customization

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

- The Debian package build system can be customized through the **debian/rules** file (see "節 9.2").
- 可以使用額外的配置檔案（如 **debian/** directory 目錄下的 *package***.install** 和 *package***.docs** 檔案）以配合來自 **debhelper** 套件的 **dh**_* 命令設定 Debian 套件檔案的安裝路徑等資訊（請參見"節 6.14"）。

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 "節 5.9".

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

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 節 6.5 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.

構建的過程可以使用某些上游提供的介面進行定製化，如使用傳遞給標準的原始碼構建系統的引數。這些構建系統包括但不限於：

- **configure**，
- **Makefile**，
- "**python -m build**", or
- **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.

參見：

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

9.3 Variables for debian/rules

某些對設定 **debian/rules** 有用的變數定義可以在 **/usr/share/dpkg/** 目錄下的檔案中找到。比較重要的包括：

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
```

參見"節 10.10"、**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 "節 9.5") 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 Build commands

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

- **dpkg-buildpackage** = 套件打包工具的核心
- **debuild** = **dpkg-buildpackage** + **lintian** （在清理後的環境變數下構建）
- **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 “節 4.6”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 上傳 orig.tar.gz

當您第一次對歸檔上傳套件時，您還需要包含原始的 **orig.tar.gz** 原始碼。

如果 Debian 修訂碼是 **1** 或者 **0**，這都是預設的。否則，您必須使用帶有 **-sa** 選項的 **dpkg-buildpackage** 命令。

- **dpkg-buildpackage -sa**
- **debuild -sa**
- **sbuild**
- For “**gbp buildpackage**”, edit the **~/.gbp.conf** file.

提示

另一方面，**-sd** 選項將會強制排除原始的 **orig.tar.gz** 原始碼。

提示

新增至 **~/.bashrc** 檔案。

9.10 跳過的上傳

如果當跳過上傳時，你在 **debian/changelog** 中建立了多個條目，你必須建立一個包含自上次上傳以來所有變更的 **debian/changelog** 檔案。這可以通過指定 dpkg-buildpackage 選項 **-v** 以及上次上傳的版本號，比如 *1.2* 來完成。

- **dpkg-buildpackage -v***1.2*
- **debuild -v***1.2*
- **sbuild --debbuildopts -v***1.2*
- 對於 **gbp buildpackage**，請編輯 **~/.gbp.conf** 檔案。

9.11 錯誤報告

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

dh_bugfiles 命令將安裝以下位於 **debian/** 目錄中的的模板檔案。

- **debian/***binarypackage***.bug-control** → **usr/share/bug/***binarypackage***/control**
  - 該檔案包含諸如重定向錯誤報告至其它套件的一些指導性內容。
- **debian/***binarypackage***.bug-presubj** → **usr/share/bug/***binarypackage***/presubj**
  - 該檔案的內容將由 **reportbug** 命令對使用者展示。
- **debian/***binarypackage***.bug-script** → **usr/share/bug/***binarypackage* or **usr/share/bug/***binarypackage***/script**
  - **reportbug** 命令執行此指令碼以生成錯誤報告的模板檔案。

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

提示

如果您總是需要提醒提交報告的使用者某些注意事項或詢問他們某些問題，使用這些檔案可以將這個過程自動化。

Chapter 10

高階打包

Let's describe advanced topics on Debian packaging.

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 "章 11".

10.2 Current trends

Current Debian packaging practices and their trends are moving target. See:

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

  - Build systems: **dh**
  - Debian source format: "**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 持續整合

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 系統上探索使用不同的持續整合系統。

- The Salsa offers "節 11.3".
- **debci** 套件：建立在 **autopkgtest** 之上的持續整合平臺
- **jenkins** 套件：通用持續整合平臺

10.5 自主生成 (Bootstrapping)

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 編譯強化

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

您應當詳細閱讀下列參考內容。

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

debmake 命令會對 **debian/rules** 檔案中按需新增 **DEB_BUILD_MAINT_OPTIONS**、**DEB_CFLAGS_MAINT_APP** 和 **DEB_LDFLAGS_MAINT_APPEND** 的專案（參見“章 5”和 **dpkg-buildflags**(1)）。

10.7 可重現的構建

為了做到套件可重現的構建，這裡給出一些相關的建議。

- 不要嵌入基於系統時間的時間戳。
- Don’t embed the file path of the build environment.
- Use “**dh $@**”in the **debian/rules** to access the latest **debhelper** features.
- Export the build environment as “**LC_ALL=C.UTF-8**”(see “節 12.1”).
- 對上游原始碼中使用的時間戳，使用 debhelper 提供的環境變數 **$SOURCE_DATE_EPOCH** 的值。
- 閱讀“可重現構建”瞭解更多資訊。
  - “可重現構建操作方法”。
  - “可重現構建時間戳處理提議”。

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) 生成的控制檔案 *source-name_source-version_arch*.**buildinfo** 記錄了構建環境資訊。參見 **deb-buildinfo**(5)

10.8 Substvar

debian/control 也定義了套件的依賴關系，其中“變數替換機制”（substvar）的功能可以用來將套件維護者從追蹤（大多數簡單的）套件依賴的重複勞動中解放出來。請參見 **deb-substvars**(5)。

debmake 命令支援下列變數替換指令：

- **${misc:Depends}**，可用於所有二進位制套件
- **${misc:Pre-Depends}**，可用於所有 multiarch 套件
- **${shlibs:Depends}**，可用於所有含有二進位制可執行檔案或程式庫的套件
- **${python:Depends}**，可用於所有 Python 套件

- **${python3:Depends}**，可用於所有 Python3 套件
- **${perl:Depends}**，用於所有 Perl 套件
- **${ruby:Depends}**，用於所有 Ruby 套件

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.

對其它沒有部署屬於自己範疇內的變數替換機制的情況，**misc** 變數替換佔位符通常用來覆蓋對應的依賴替換需求。

對 POSIX shell 程式來說，並沒有簡單的辦法來驗證其依賴關係，substvar 的變數替換也無法自動得出它們的依賴。

對使用動態載入機制，包括"GObject introspection"機制的程式庫和模組來說，現在沒有簡單的方法可以檢查依賴關係，變數替換機制也無法自動推匯出所需的依賴。

10.9 程式庫套件

打包軟體程式庫需要您投入更多的工作。下面有一些打包軟體程式庫的提醒和建議：

- 程式庫二進位制套件必須根據"節 10.17"進行命名。
- Debian 按照 **/usr/lib/<triplet>/lib***foo-0.1***.so.***1.0.0* 這樣的路徑提供共享連結程式庫(參見"節 10.10")。
- Debian 鼓勵在共享程式庫中使用帶版本的符號（見"節 10.16")。
- Debian 不提供 ***.la** libtool 程式庫歸檔檔案。
- Debian 不推薦使用、提供 ***.a** 靜態程式庫檔案。

在打包共享程式庫軟體之前，請查閱：

- "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"

如需研究其歷史背景，請參見：

- "逃離依賴地獄"1
  - 該文件鼓勵在共享程式庫中使用帶版本的符號。
- "Debian Library Packaging guide"2
  - Please read the discussion thread following its announcement, too.

10.10 多體系架構

Debian **wheezy**（7.0，2013 年 5 月）在 **dpkg** 和 **apt** 中引入了對跨架構二進位制套件安裝的多架構支援（特別是 **i386** 架構和 **amd64** 架構，但也支援其它的組合），這部分內容值得我們額外關注。

您應當詳細閱讀下列參考內容。

- Ubuntu 維基（上游）
  - "多架構規範（MultiarchSpec）"
- Debian 維基（Debian 的現狀）

1該文件是在 **symbols** 檔案被引入之前寫成的。

2在"第六章 - 開發（-DEV）套件"中，存在強烈的使用含有 SONAME 版本號的 **-dev** 套件名而非僅使用 **-dev** 作為名稱的偏好，但前 ftp-master 成員（Steve Langasek）對此有不同意見。請注意該文件在 **multiarch** 系統和 **symbols** 引入之前寫成，可能有一定程度的過時。

  - “Debian 多架構支援”
  - “多架構支援/實現（Multiarch/Implementation）”

多架構支援使用三元組（**<triplet>**）的值，例如 **i386-linux-gnu** 和 **x86_64-linux-gnu**；它們出現在共享連結程式庫的安裝路徑中，例如 **/usr/lib/<triplet>/**，等等。

- 三元組 **<triplet>** 的值由 **debhelper** 指令碼隱性提前設定好，套件維護者無需擔心。
- 不過，在 **debian/rules** 檔案中用於 **override_dh_*** 目標的三元組 **<triplet>** 值需要由維護者手動進行顯性設定。三元組 **<triplet>** 的值可由 **$(DEB_HOST_MULTIARCH)** 變數在 **debian/rules** 檔案中取得到，具體方法如下：

```
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)
```

參見：

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

10.11 Debian 二進位制套件的拆分

對行為良好的構建系統來說，對 Debian 二進位制包的拆分可以由如下方式實現。

- 為所有二進位制套件在 **debian/control** 檔案中建立對應的二進位制套件條目。
- 在對應的 **debian/**二進位制套件名**.install** 檔案中列出所有檔案的路徑（相對於 **debian/tmp** 目錄）。

請檢視本指南中相關的例子：

- “節 14.11”（基於 Autotools）
- “節 14.12”（基於 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 “節 16.2”.

10.12 拆包的場景和例子

對於下面這樣的上游原始碼範例，我們在這裡給出使用 **debmake** 處理時一些典型的 multiarch 套件拆分的場景和做法：

- 一個程式庫原始碼 *libfoo-1.0*.**tar.gz**
- 一個軟體工具原始碼 *bar-1.0*.**tar.gz**，軟體由編譯型語言編寫
- 一個軟體工具原始碼 *baz-1.0*.**tar.gz**，軟體由解釋型語言編寫

二進位制套件	型別	**Architecture:**	**Multi-Arch:**	套件內容
lib*foo1*	**lib***	any	same	共享程式庫，可共同安裝
lib*foo*-**dev**	**dev***	any	same	共享程式庫標頭檔案及相關開發檔案，可共同安裝

二進位制套件	型別	**Architecture:**	**Multi-Arch:**	套件內容
lib*foo***-tools**	**bin***	any	foreign	執行時支援程式，不可共同安裝
lib*foo***-doc**	**doc***	all	foreign	共享程式庫文件
bar	**bin***	any	foreign	編譯好的程式檔案，不可共同安裝
*bar***-doc**	**doc***	all	foreign	程式的配套文件檔案
baz	**script**	all	foreign	解釋型程式檔案

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 多架構程式庫路徑選項

經典路徑	**i386** 多體系架構路徑	**amd64** 多體系架構路徑
/lib/	**/lib/i386-linux-gnu/**	**/lib/x86_64-linux-gnu/**
/usr/lib/	**/usr/lib/i386-linux-gnu/**	**/usr/lib/x86_64-linux-gnu/**

對基於 Autotools 且由 **debhelper** (compat>=9)管理的套件來說，這些路徑設定已由 **dh_auto_configure** 命令自動處理。

對於其它使用不支援的構建系統的套件，您需要按照下面的方式手動調整安裝路徑。

- 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)/**.
- 請在 **debian/***foo***.install** 檔案中將所有出現的 **/usr/lib/** 字串替換為 **/usr/lib/*/**。

所有啟用多架構的套件安裝至相同路徑的檔案必須內容完全相同。您必須小心處理，避免資料位元組序或者壓縮演算法等等問題帶來的檔案內容差異。

位於預設路徑 **/usr/lib/** 和 **/usr/lib/<triplet>/** 的共享程式庫可被自動載入。

對位於其它路徑的共享程式庫，必須使用 **pkg-config** 命令設定 GCC 選項 **-l** 以正確進行載入。

10.14 Multiarch header file path

在支援多架構的 Debian 系統上，GCC 預設會同時包含、使用 **/usr/include/** 和 **/usr/include/<triplet>/** 下的標頭檔案。

如果標頭檔案不在這些路徑中，必須使用 **pkg-config** 命令設定 GCC 的 **-I** 引數以使得"**#include <***foo.h***>**"正常工作。

Table 10.3 多架構標頭檔案路徑選項

經典路徑	**i386** 多體系架構路徑	**amd64** 多體系架構路徑
/usr/include/	**/usr/include/i386-linux-gnu/**	**/usr/include/x86_64-linux-gnu/**
/usr/include/套件名**/**	**/usr/include/i386-linux-gnu/**套件名**/**	**/usr/include/x86_64-linux-gnu/**套件名**/**
	/usr/lib/i386-linux-gnu/套件名**/**	**/usr/lib/x86_64-linux-gnu/**軟體包名**/**

為程式庫檔案使用 **/usr/lib/<triplet>/**套件名**/** 路徑可幫助上游維護者對使用 **/usr/lib/<triplet>** 的多架構系統和使用 **/usr/lib<qual>/** 的雙架構系統使用相同的安裝指令碼。[3]

使用包含 *packagename* 的檔案路徑也使得在同一系統上同時安裝多個架構的開發程式庫成為可能。

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."

10.15 Multiarch *.pc file path

packagename 用來取得系統上已安裝程式庫的資訊。它在 ***.pc** 檔案中儲存配置引數，用來設定 GCC 的 **-I** 和 **-l** 選項。

Table 10.4 *.pc 檔案路徑選項

經典路徑	**i386** 多體系架構路徑	**amd64** 多體系架構路徑
/usr/lib/pkgconfig/	**/usr/lib/pkgconfig/**	**/usr/lib/x86_64-linux-gnu/pkgconfig/**

10.16 程式庫符號

Debian **lenny**（5.0，2009 年 5 月）中引入的 **dpkg** 符號支援可以幫助我們管理同一共享鏈接程式庫套件的向後 ABI 相容性（backward ABI compatibility）。二進位制套件中的 **DEBIAN/symbols** 檔案提供了每個符號及其對應的最小版本號。

一個極其簡化的軟體程式庫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** 命令也可以用來解壓得到 **DEBIAN/symbols** 檔案。
- 將其複製為 **debian/***binarypackage***.symbols** 檔案。
  - 如果這是第一次打包的話，可以只建立一個空檔案。
- 構建二進位制套件。
  - 如果 **dpkg-gensymbols** 命令警告添加了新的符號的話：
    - Extract the updated **DEBIAN/symbols** file with the "**dpkg-deb -e**"command.
    - 將其中的 Debian 修訂版本號，例如 **-1**，從檔案中去除。
    - 將其複製為 **debian/***binarypackage***.symbols** 檔案。
    - 重新構建二進位制套件。
  - 如果 **dpkg-gensymbols** 命令不報和新連結符號有關的警告：
    - 您已完成了共享程式庫的打包工作。

如需瞭解詳細資訊，您應當閱讀下列第一手參考資料。

- "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"

您也應當檢視：

- Debian wiki: "UsingSymbolsFiles"
- Debian wiki: "Projects/ImprovedDpkgShlibdeps"
- Debian kde team: "Working with symbols files"
- "節 14.11"
- "節 14.12"

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

我們考慮 **lib***foo* 這個程式庫的上游 tarball 原始碼壓縮包的名字從 **lib***foo-7.0***.tar.gz** 更新為了 **lib***foo-8.0***.tar.gz**，同時帶有一次 SONAME 大版本的跳躍（並因此影響了其它套件）。

程式庫的二進位制套件將必須從 **lib***foo7* 重新命名為 **lib***foo8* 以保持使用 **unstable** 套件的系統上所有依賴該程式庫的套件在上傳了基於 **lib***foo-8.0***.tar.gz** 的新程式庫後仍然能夠正常運行。

警告

如果這個二進位制程式庫套件沒有得到更名，許多使用 **unstable** 套件的系統上的各個依賴該程式庫的套件會在新的程式庫包上傳後立刻破損，即便立刻請求進行 binNMU 上傳也無法避免這個問題。由於種種原因，binNMU 不可能在上傳後立刻開始進行，故無法緩解問題。

-dev 套件必須遵循以下命名規則：

- 使用不帶版本號的 **-dev** 套件名稱：**lib***foo***-dev**
  - 該情況通常適用於依賴關係處於葉節點的程式庫套件。
  - 倉庫內只允許存在一個版本的程式庫原始碼套件。
    - 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)
- 使用帶版本的 **-dev** 套件名稱：**lib***foo7***-dev** 和 **lib***foo8***-dev**
  - 該情況通常適用於各類重要程式庫套件。
  - 兩個版本的程式庫原始碼套件可同時出現在發行版倉庫中。
    - 令所有依賴該程式庫的套件依賴 **lib***foo***-dev**。
    - 令 **lib***foo7***-dev** 和 **lib***foo8***-dev** 兩者都提供 **lib***foo***-dev**。
    - 原始碼套件需要從 **lib***foo-?.0***.tar.gz** 相應地重新命名為 **lib***foo7-7.0***.tar.gz** 和 **lib***foo8-8.0***.tar.gz**。
    - 需要仔細選擇 **lib***foo7* 和 **lib***foo8* 套件檔案安裝時的路徑，如標頭檔案等等，以保證它們可以同時安裝。
  - 可能的話，不要使用這個重量級的、需要大量人為干預的方法。
  - 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

如果包內資料檔案編碼方案有所變化（如，從 latin1 變為 utf-8），該場景應比照 API 變化做類似的考慮與處理。

參見“節 10.9”。

10.18 程式庫變遷

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.

發行團隊提供了“變遷追蹤系統”。參見“變遷（Transition）”。

注意

請確保您按照“節 10.17”的描述正確地對二進位制套件進行了重命名。

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 除錯資訊

The Debian package is built with the debugging information but packaged into the binary package after stripping the debugging information as required by “Chapter 10 - Files”of the “Debian Policy Manual”.

參見

- “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 列表釋出的郵件 :“自動除錯套件狀態”(2015-08-15)

10.21 -dbgsym package

除錯資訊由 **dh_strip** 命令的預設行為自動打包並進行移除。所分離得到的除錯套件名具有 **-dbgsym** 的字尾。

- **debian/rules** 檔案不應顯性包括 **dh_strip** 。
- Set the **Build-Depends** to **debhelper-compat (>=13)** while removing **Build-Depends** to **debhelper** in **debian/control**.

10.22 debconf

debconf 套件可以幫助我們在下列兩種情況下配置套件：

- 在 **debian-installer**（Debian 安裝程式）預安裝時進行非互動式配置。
- interactively from the menu interface (**dialog**, **gnome**, **kde**, ⋯)
  - 套件安裝時：由 **dpkg** 命令呼叫
  - 對已安裝套件：由 **dpkg-reconfigure** 命令呼叫

套件安裝時的所有使用者互動都必須由這裡的 **debconf** 系統進行處理，下列配置檔案對這個過程進行控制。

- **debian/***binarypackage***.config**
  - 這是 **debconf config** 指令碼，用於對使用者詢問對於配置套件必需的問題。
- **debian/***binarypackage***.template**
  - 這是 **debconf templates**（模板）檔案，用於對使用者詢問對於配置套件必需的問題。

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 “章 10”, 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 “節 10.1”.

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

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:

- “節 11.1”
- “節 11.2”
- “節 11.3”

There are 2 styles of branch names for the Git repository used for the packaging. See “節 11.4”.

There are 2 main usage styles for the Git repository for the packaging. See:

- “節 11.5”
- “節 11.6”

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

11.1 Salsa repository

It is highly desirable to host Debian source code package on Salsa. Over 90% of all Debian source code packages are hosted on Salsa. 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 account setup

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 CI service

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

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 “節 6.13”as:

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

# Customizations here
```

11.4 Branch names

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

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.:

1Use of git.debian.org or alioth.debian.org are deprecated now.

```
$ 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:
  - "節 5.7"(no patch)
  - "節 5.10"
    - **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.
  - "節 5.11"including the last "**dquilt pop -a**"step
  - "節 11.9"
- 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

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 "節 5.12"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 "節 11.9".
  - 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 "節 11.12"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 "節 11.5"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 構建 Debian 套件”
- “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 “節 11.6”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 “節 11.5”and “節 11.6”.

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 “節 11.5”.
- **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 “節 11.6”.
- **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 “節 11.6”.
- **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 “節 11.5”, 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 “節 5.11”and “節 9.5”, **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 "節 11.5"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 "節 11.5"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 “節 6.4”) 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 “節 6.1”.

Although this can be done in many ways (“節 16.4”), 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

小技巧

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 環境下構建

構建環境的預設語言環境是 **C**。

某些程式（如 Python3 的 **read** 函式）會根據區域設定改變行為。

新增以下程式碼到 **debian/rules** 檔案可以確保程式使用 **C.UTF-8** 的區域語言設定（locale）進行構建。

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

12.2 UTF-8 轉換

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) 將 HTML 檔案轉換為 UTF-8 純文字檔案。執行此操作時，請確保在 UTF-8 語言環境下執行它。

```
$ 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** 檔案的 **override_dh_*** 目標中執行這些指令碼。

12.3 Hints for Debugging

當您遇到構建問題或者生成的二進位制程式核心轉儲時，您需要自行解決他們。這就是除錯（**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** - 追蹤系統呼叫和訊號
  - 使用 **/usr/share/doc/strace/examples/** 中的 **strace-graph** 指令碼來建立一個好看的樹形圖
  - “**man strace**”
- **ltrace** - 追蹤程式庫呼叫
  - “**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
  - 安裝 **libterm-readline-gnu-perl** 套件或者同類型軟體來新增輸入行編輯功能與歷史記錄支援。
- **lsof** - 按程序列出開啟的檔案
  - “**man lsof**”

提示

script 命令能幫助記錄控制檯輸出。

提示

在 **ssh** 命令中搭配使用 **screen** 和 **tmux** 命令，能夠提供安全並且強健的遠端連線終端。

提示

libreply-perl（新的）套件和來自 **libdevel-repl-perl**（舊的）套件的 **re.pl** 命令為 Perl 提供了一個類似 Python 和 Shell 的 REPL (=READ + EVAL + PRINT + LOOP) 環境。

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

Tool usages

Here are some notable tools around Debian packaging.

注

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

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

13.1 debdiff

您可以使用 **debdiff** 命令來對比兩個 Debian 套件組成的差別。

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

您也可以使用 **debdiff** 命令來對比兩組二進制 Debian 套件中的檔案列表。

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

當檢查原始碼套件中哪些檔案被修改時，這個命令非常有用。它還可以用來檢測二進位制包中是否有檔案在更新過程中發生變動，比如被意外替換或刪除。

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** 命令來下載 Debian 套件原始碼的檔案集。

```
$ 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 “節 4.4”) 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 “節 11.6”.)

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

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

13.8 debc

您應該使用 **debc** 命令安裝生成的套件以在本地測試它。

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

13.9 piuparts

您應該使用 **piuparts** 命令安裝生成的套件以自動進行測試。

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

注

這是一個非常慢的過程，因為它需要查詢遠端 APT 套件倉庫。

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 命令是一個用以處理“Debian 缺陷追蹤系統”上的錯誤的便捷工具。

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

Chapter 14

更多範例

There is an old Latin saying: “**fabricando fit faber**”(“practice makes perfect”).
強烈建議使用簡單的包來練習和試驗 Debian 打包的所有步驟。本章為您的練習提供了許多上游案例。
This should also serve as introductory examples for many programming topics.

- Programming in the POSIX shell, Python3, and C.
- 使用圖示圖形建立桌面 GUI 程式啟動器的方法。
- Conversion of a command from CLI to GUI.
- Conversion of a program to use **gettext** for internationalization and localization: POSIX shell and C sources.
- Overview of many build systems: Makefile, Python, Autotools, and CMake.

請注意，Debian 對以下事項非常注意：

- 自由軟體
- 作業系統的穩定性與安全性
- 通過以下方式以實現通用作業系統：
  - 上游原始碼的自由選擇，
  - CPU 架構的自由選擇，以及
  - 使用者介面語言的自由選擇。

在“章 5”中介紹的典型打包範例是本章節的先決條件。
在以下數小節中，有些細節被刻意模糊。請嘗試閱讀相關檔案，並且嘗試自行釐清它們。

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 挑選最好的模板

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.
讓我們假設這個空目錄為 **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
 ...
```

我們來檢查一下自動產生的模板檔案。

```
 $ 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
```

現在，您可以複製 *debhello-0.1/***debian/** 目錄下所有生成的模板檔案到您的套件中。

14.2 無 Makefile（shell，命令列介面）

此處是一個從 POSIX shell 命令列介面程式建立簡單的 Debian 套件的範例，我們假設它沒有使用任何構建系統。

讓我們假設上游的原始碼套件為 **debhello-0.2.tar.gz**。
此類原始碼不具有自動化方法，所以必須手動安裝檔案。
For example:

```
$ 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.
下載 **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
```

這裡的 POSIX shell 指令碼 **hello** 非常的簡單。
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
```

此處的 **hello.desktop** 支援"桌面項（Desktop Entry）規範"。
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;
```

此處的 **hello.png** 是圖示的影象檔案。

讓我們使用 **debmake** 命令來打包。這裡使用 **-b':sh'** 選項來指明生成的二進位制包是一個 shell 指令碼。

```
 $ 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
 ...
```

讓我們來檢查一下自動產生的模板檔案。

執行基本的 **debmake** 命令後的原始碼樹。（**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（模板檔案，**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 $@
```

這基本上是帶有 **dh** 命令的標準 **debian/rules** 檔案。因為這是個指令碼套件，所以這個 **debian/rules** 模板檔案沒有與構建標記（build flag）相關的內容。

debian/control（模板檔案，**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 “章 6”.

因為這個上游原始碼缺少上游的 **Makefile**，所以這個功能需要由維護者提供。這個上游原始碼僅包含指令碼檔案和資料檔案，沒有 C 的源碼檔案，因此 構建（**build**）的過程可以被跳過，但是需要實現安裝（**install**）的過程。對於這種情況，通過新增 **debian/install** 和 **debian/manpages** 檔案可以很好地實現這一功能，且不會使 **debian/rules** 檔案變得複雜。

作為維護者，我們要把這個 Debian 套件做得更好。

debian/rules（維護者版本，**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（維護者版本，**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（維護者版本，**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（維護者版本，**v=0.2**）：

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

在 **debian/** 目錄下還有一些其它的模板文件。它們也需要進行更新。

debian/ 目錄下的模板檔案。（**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
```

您可以在此原始碼樹中使用 **debuild** 命令（或其等效命令）建立非原生的 Debian 套件。如下所示，該命令的輸出非常詳細，並且解釋了它所做的事。

```
 $ 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.
```

現在我們來看看成果如何。

通過 **debuild** 生成的第 **0.2** 版的 **debhello** 檔案：

```
 $ 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
```

您可以看見生成的全部檔案。

- **debhello_0.2.orig.tar.gz** 是指向上游原始碼包的符號連結。
- **debhello_0.2-1.debian.tar.xz** 包含了維護者生成的內容。
- **debhello_0.2-1.dsc** 是 Debian 原始碼套件的元資料檔案。
- The **debhello_0.2-1_all.deb** 是 Debian 二進位制套件。
- **debhello_0.2-1_amd64.build** 是 Debian 二進位制套件。
- **debhello_0.2-1_amd64.buildinfo** 檔案是由 **dpkg-genbuildinfo**(1) 自動生成的元檔案。
- **debhello_0.2-1_amd64.changes** 是 Debian 二進位制套件的元資料檔案。

debhello_0.2-1.debian.tar.xz 包含了 Debian 對上游原始碼的修改，具體如下所示。
壓縮過的歸檔檔案 **debhello_0.2-1.debian.tar.xz** 中的內容物：

```
 $ 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 包含了將要安裝至系統中的檔案，如下所示。
debhello_0.2-1_all.deb 二進位制套件中的內容：

```
 $ 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
```

此處是生成的 **debhello_0.2-1_all.deb** 的依賴項列表。
debhello_0.2-1_all.deb 的依賴項列表：

```
$ 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

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，命令列介面）

下面是從 POSIX shell 命令列介面程式建立簡單的 Debian 套件的範例，我們假設它使用 **Makefile** 作為構建系統。
讓我們假設上游的原始碼套件為 **debhello-1.0.tar.gz**。
這一類原始碼設計可以用這樣的方式安裝成為非系統檔案：

```
$ 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**.
讓我們取得原始碼並製作 Debian 套件。
下載 **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
```

這裡的 **Makefile** 正確使用 **$(DESTDIR)** 和 **$(prefix)**。其他的所有檔案都和"節 14.2"中的一樣，並且大多數的打包工作也都一樣。
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
```

讓我們使用 **debmake** 命令來打包。這裡使用 **-b':sh'** 選項來指明生成的二進位制包是一個 shell 指令碼。

```
 $ 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
 ...
```

讓我們來檢查一下自動產生的模板檔案。

debian/rules（模板檔案，**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
```

作為維護者，我們要把這個 Debian 套件做得更好。
debian/rules（維護者版本，**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
```

因為上游原始碼含有正確的上游 **Makefile** 文件，所以沒有必要再去建立 **debian/install** 和 **debian/-manpages** 檔案。
debian/control 檔案和“節 14.2”中的完全一致。
在 **debian/** 目錄下還有一些其它的模板文件。它們也需要進行更新。
debian/ 目錄下的模板檔案。（**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
```

其餘的打包操作基本上和“節 14.2”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**.
讓我們取得原始碼並製作 Debian 套件。
下載 **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())
```

讓我們使用 **debmake** 命令來打包。這裡使用 **-b':py3'** 選項來指明生成的二進位制包包含 Python3 指令碼和模組檔案。

```
 $ 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
 ...
```

讓我們來檢查一下自動產生的模板檔案。

debian/rules（模板檔案，**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
```

這基本上是帶有 **dh** 命令的標準 **debian/rules** 檔案。

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（模板檔案，**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 "章 6".

作為維護者，我們要把這個 Debian 套件做得更好。

debian/rules（維護者版本，**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（維護者版本，**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/** 目錄下還有一些其它的模板文件。它們也需要進行更新。

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
```

其餘的打包工作與"節 14.3"中的幾乎一致。

debian/ 目錄下的模板檔案。（**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
```

此處是生成的 **debhello_1.1-1_all.deb** 包的依賴項列表。
debhello_1.1-1_all.deb 的依賴項列表：

```
 $ dpkg -f debhello_1.1-1_all.deb pre-depends \
           depends recommends conflicts breaks
Depends: python3:any
```

14.5 Makefile（shell，圖形介面）

此處是一個從 POSIX shell 圖形介面程式構建簡單的 Debian 套件的範例，我們假設程式使用 **Makefile** 作為構建系統。
上游是基於“節 14.3”中的原始碼，並帶有增強的圖形介面支持。
讓我們假設上游的原始碼套件為 **debhello-1.2.tar.gz**。
讓我們取得原始碼並製作 Debian 套件。
下載 **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
```

此處的 **hello** 已經被重寫以便使用 **zenity** 命令來使其成為 GTK+ 圖形介面程序。
hello（**v=1.2**）

```
 $ cat debhello-1.2/scripts/hello
#!/bin/sh -e
zenity --info --title "hello" --text "Hello from the shell!"
```

這裡，作為圖形介面程式，桌面檔案被更新為 **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;
```

其餘的所有檔案都與“節 14.3”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
 ...
```

讓我們來檢查一下自動產生的模板檔案。

debian/control（模板檔案，**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.
```

作為維護者，我們要把這個 Debian 套件做得更好。

debian/control（維護者版本，**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)
```

請注意，這裡需要手動新增 **zenity** 依賴。
debian/rules 檔案與“節 14.3”中的完全一致。
在 **debian/** 目錄下還有一些其它的模板文件。它們也需要進行更新。
debian/ 目錄下的模板檔案。（**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
```

其餘的打包工作與“節 14.3”中的幾乎一致。
此處是 **debhello_1.2-1_all.deb** 的依賴項列表。
debhello_1.2-1_all.deb 的依賴項列表：

```
 $ dpkg -f debhello_1.2-1_all.deb pre-depends \
            depends recommends conflicts breaks
Depends: zenity
```

14.6 pyproject.toml (Python3, GUI)

Here is an example of creating a simple Debian package from a Python3 GUI program using **pyproject.toml**.

讓我們假設上游原始碼套件為 **debhello-1.3.tar.gz**。
讓我們取得原始碼並製作 Debian 套件。
下載 **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()
```

讓我們使用 **debmake** 命令來打包。這裡使用 **-b':py3'** 選項來指明生成的二進位制包包含 Python3 指令碼和模組檔案。

```
 $ 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 “節 14.4”.
作為維護者，我們要把這個 Debian 套件做得更好。
debian/rules（維護者版本，**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（維護者版本，**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)
```

請注意，此處需要手動新增 **python3-gi** 和 **gir1.2-gtk-3.0** 依賴。
The rest of the packaging activities are practically the same as in <pyproject>>.
此處是 **debhello_1.3-1_all.deb** 的依賴項列表。
debhello_1.3-1_all.deb 的依賴項列表：

```
 $ 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（單個二進位制套件）

這裡給出了從簡單的 C 語言原始碼建立簡單的 Debian 套件的例子，並假設上游使用了 **Makefile** 作為構建系統。

此處的上游原始碼是“章 5”中的原始碼的增強版本。它帶有手冊頁、桌面檔案和桌面圖示。並且為了更加貼合實際，它還有一個外部程式庫檔案 **libm**。

讓我們假設上游原始碼套件為 **debhello-1.4.tar.gz**。

這一類原始碼設計可以用這樣的方式安裝成為非系統檔案：

```
 $ 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**.

讓我們取得原始碼並製作 Debian 套件。

下載 **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
```

此處的原始碼如下所示。

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"
```

請注意，此 **Makefile** 含有正確的手冊頁、桌面檔案、桌面圖示的 **install** 物件。
讓我們使用 **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
 ...
```

其餘的工作與“節 5.6”中的幾乎一致。
像“節 5.7”中所寫的一樣，讓我們這些維護者來把這個 Debian 套件做的更好。
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 檔案與“節 5.7”中的完全一致，因為 **libm** 程式庫是 **libc6** 程式庫的一部分，所以它總是可獲得的（優先順序：必需 / Priority: required）。
在 **debian/** 目錄下還有一些其它的模板文件。它們也需要進行更新。
debian/ 目錄下的模板檔案。（**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
```

其餘的打包步驟與“節 5.8”中的基本一致。
此處是生成的二進位制包的依賴項列表。
生成的二進位制包的依賴項列表（**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（單個二進位制套件）

這裡給出了從簡單的 C 語言原始碼建立簡單的 Debian 套件的例子，並假設上游使用了 **Makefile.in** 和 **configure** 作為構建系統。

此處的原始碼範例是“節 14.7”中的原始碼的增強版本。它也有一個外部連結程式庫 **libm**，並且它的原始碼可以使用 **configure** 指令碼進行配置，然後生成相應的 **Makefile**、**src/config.h** 檔案。

讓我們假設上游原始碼套件為 **debhello-1.5.tar.gz**。
此型別的原始碼旨在作為非系統檔案安裝，例如：

```
 $ tar -xzmf debhello-1.5.tar.gz
 $ cd debhello-1.5
 $ ./configure --with-math
 $ make
 $ make install
```

讓我們取得原始碼並製作 Debian 套件。
下載 **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
```

此處的原始碼如下所示。

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**.

讓我們使用 **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
 ...
```

結果與“節 5.6”中的相似，但是並不完全一致。
讓我們來檢查一下自動產生的模板檔案。
debian/rules（模板檔案，**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 $@
```

作為維護者，我們要把這個 Debian 套件做得更好。
debian/rules（維護者版本，**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/** 目錄下還有一些其它的模板文件。它們也需要進行更新。
其餘的打包步驟與“節 5.8”中的基本一致。

14.9 Autotools（單個二進位制檔案）

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.

此種原始碼通常也帶有上游自動生成的 **Makefile.in** 和 **configure** 檔案。在 **autotools-dev** 套件的幫助下，我們可以按“節 14.8”中所介紹的，使用這些檔案進行打包。

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.

讓我們假設上游的原始碼套件為 **debhello-1.6.tar.gz**。
此型別的原始碼旨在作為非系統檔案安裝，例如：

```
 $ tar -xzmf debhello-1.6.tar.gz
 $ cd debhello-1.6
```

```
 $ autoreconf -ivf # optional
 $ ./configure --with-math
 $ make
 $ make install
```

讓我們取得原始碼並製作 Debian 套件。
下載 **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
```

此處的原始碼如下所示。
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.

讓我們使用 **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
 ...
```

結果與“節 14.8”中的類似，但是並不完全一致。
讓我們來檢查一下自動產生的模板檔案。
debian/rules（模板檔案，**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
```

作為維護者，我們要把這個 Debian 套件做得更好。
debian/rules（維護者版本，**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/** 目錄下還有一些其它的模板文件。它們也需要進行更新。
其餘的打包步驟與“節 5.8”中的基本一致。

14.10 CMake（單個二進位制套件）

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.

讓我們假設上游的原始碼套件為 **debhello-1.7.tar.gz**。
此型別的原始碼旨在作為非系統檔案安裝，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
```

讓我們取得原始碼並製作 Debian 套件。
下載 **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
```

此處的原始碼如下所示。
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
)
```

讓我們使用 **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
 ...
```

結果與“節 14.8”中的類似，但是並不完全一致。

讓我們來檢查一下自動產生的模板檔案。

debian/rules（模板檔案，**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（模板檔案，**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.
```

作為維護者，我們要把這個 Debian 套件做得更好。

debian/rules（維護者版本，**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（維護者版本，**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/** 目錄下還有一些其它的模板文件。它們也需要進行更新。
其餘的打包工作與“節 14.8”中的近乎一致。

14.11 Autotools（多個二進位制套件）

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 “節 14.9”.
讓我們假設上游原始碼套件為 **debhello-2.0.tar.gz**。
此型別的原始碼旨在作為非系統檔案安裝，例如：

```
 $ tar -xzmf debhello-2.0.tar.gz
 $ cd debhello-2.0
 $ autoreconf -ivf # optional
 $ ./configure --with-math
 $ make
 $ make install
```

讓我們取得原始碼並製作 Debian 套件。
下載 **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
```

此處的原始碼如下所示。
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 與 **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
 ...
```

結果與"節 14.8"中的相似，但是這個具有更多的模板檔案。
讓我們來檢查一下自動產生的模板檔案。
debian/rules（模板檔案，**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
```

作為維護者，我們要把這個 Debian 套件做得更好。
debian/rules（維護者版本，**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（維護者版本，**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（維護者版本，**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.
```

因為上游原始碼已經具有正確的自動生成的 **Makefile** 檔案，所以沒有必要再去建立 **debian/install** 和 **debian/manpages** 文件。

在 **debian/** 目錄下還有一些其它的模板文件。它們也需要進行更新。

debian/ 目錄下的模板檔案。（**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
```

其餘的打包工作與“節 14.8”中的近乎一致。
此處是生成的二進位制包的依賴項列表。
生成的二進位制包的依賴項列表（**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（多個二進位制套件）

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.

讓我們假設上游原始碼套件為 **debhello-2.1.tar.gz**。
此型別的原始碼旨在作為非系統檔案安裝，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
```

讓我們取得原始碼並製作 Debian 套件。
下載 **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
```

此處的原始碼如下所示。

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 與 **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
)
```

讓我們使用 **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
 ...
```

結果與“節 14.8”中的類似，但是並不完全一致。

讓我們來檢查一下自動產生的模板檔案。

debian/rules（模板檔案，**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)"
```

作為維護者，我們要把這個 Debian 套件做得更好。
debian/rules（維護者版本，**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（維護者版本，**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（維護者版本，**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/*（維護者版本，**v=2.1**）：

```
 ... hack, hack, hack, ...
 $ cat debian/libsharedlib1.symbols
libsharedlib.so.1 libsharedlib1 #MINVER#
 sharedlib@Base 2.1
```

因為上游原始碼已經具有正確的自動生成的 **Makefile** 檔案，所以沒有必要再去建立 **debian/install** 和 **debian/manpages** 文件。

在 **debian/** 目錄下還有一些其它的模板文件。它們也需要進行更新。

debian/ 目錄下的模板檔案。（**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
```

其餘的打包工作與“節 14.8”中的近乎一致。
此處是生成的二進位制包的依賴項列表。
生成的二進位制包的依賴項列表（**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 國際化

此處是更新“節 14.11”中提供的簡單上游 C 語言原始碼 **debhello-2.0.tar.gz** 以便進行國際化（i18n）並建立更新後的上游 C 語言原始碼 **debhello-2.0.tar.gz** 的範例。
在實際情況中，此套件應該已被國際化過。所以此範例用作幫助您瞭解國際化的具體實現方法。

提示

負責維護國際化的維護者的日常活動就是將通過缺陷追蹤系統（BTS）反饋給您的 po 翻譯檔案新增至 **po/** 目錄，然後更新 **po/LINGUAS** 檔案的語言列表。

讓我們取得原始碼並製作 Debian 套件。
下載 **debhello-2.0.tar.gz**（國際化版）

```
 $ 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** 命令將此原始碼樹國際化，並刪除由 Autotools 自動生成的檔案。
執行 **gettextize**（國際化版）：

```
 $ 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 *~
```

讓我們確認一下 **po/** 目錄下生成的檔案。
po 目錄下的檔案（國際化版）：

```
 $ 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（國際化版）：

```
 $ 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
```

讓我們從 **po/Makevars.template** 檔案中創建 **po/Makevars** 檔案。
po/Makevars（國際化版）：

```
 ... 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 （國際化版）：

```
 ... 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（國際化版）：

```
 ... 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）可以直接處理桌面檔案的國際化版本。
data/hello.desktop.in（國際化版）：

```
 $ 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;
```

讓我們列出輸入檔案，以便在 **po/POTFILES.in** 中提取可翻譯的字串。
po/POTFILES.in（國際化版）：

```
 ... hack, hack, hack, ...
 $ cat po/POTFILES.in
src/hello.c
lib/sharedlib.c
data/hello.desktop.in
```

此處是在 **SUBDIRS** 環境變數中新增 **po** 目錄後更新過的根 **Makefile.am** 檔案。
Makefile.am （國際化版）：

```
 $ 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
```

讓我們建立一個翻譯模板檔案 **debhello.pot**。
po/debhello.pot（國際化版）：

```
 $ 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 ""
```

讓我們新增法語的翻譯。

po/LINGUAS 與 **po/fr.po**（國際化版）：

```
 $ 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 ""
```

打包工作與“節 14.11”中的近乎一致。
You can find more i18n examples by following “節 14.14”.

14.14 細節

You can obtain detailed information about the examples presented and their variants as follows:
如何取得細節

```
$ 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.

- 模擬控制檯命令列活動日誌：**.log** 檔案
- 模擬控制檯命令列活動日誌（縮略版）：**.slog** 檔案
- 執行 **debmake** 命令後的原始碼樹快照：**debmake** 目錄
- snapshot source tree image after proper packaging: the **package** directory
- 執行 **debuild** 命令後的原始碼樹快照：**test** 目錄

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)
- C source with CMake and i18n support (v=3.4)

Chapter 15

debmake(1) 手冊頁

15.1 名稱

debmake，用來製作 Debian 原始碼套件的程式

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 協助從上游原始碼構建一個 Debian 套件，通常做法如下：

- 下載上游原始碼壓縮包（tarball）並命名為 *package-version*.**tar.gz** 檔案。
- 對其進行解壓縮並將所有檔案放置於 *package-version/* 目錄之下。
- 在 *package-version/* 目錄中呼叫 debmake，並按需帶上引數。
- 手工調整 *package-version*/**debian/** 目錄下的檔案。
- **dpkg-buildpackage** (usually from its wrapper **debuild** or **sbuild**) is invoked in the *package-version/* directory to make Debian packages.

請確保將 **-b**、**-f**、**-l** 和 **-w** 選項的引數使用引號合適地保護起來，以避免 shell 環境的干擾。

15.3.1 可選引數：

-h, --help 顯示本幫助資訊並退出。

-c, --copyright 為授權 + 許可證文字而掃描原始碼，然後退出。

- **-c**：簡單輸出風格
- **-cc**：正常輸出風格（類似 **debian/copyright** 檔案）
- **-ccc**：除錯輸出風格

-k, --kludge 對 **debian/copyright** 檔案和原始碼進行比較並退出。

debian/copyright 必須將通用的檔案匹配模式放在前部並將個別檔案的例外放在後部。

- **-k**：基本輸出風格
- **-kk**：冗長輸出風格

-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 直接使用上游原始碼壓縮包。（**-p**, **-u**, **-z**：被覆蓋）

上游原始碼壓縮包可以命名為 *package_version*.**orig.tar.gz** 或者 **tar.gz**。在某些情況下，也可使用 **tar.bz2** 或 **tar.xz**。

如果所指定的原始碼壓縮包檔名中包含大寫字母，Debian 打包時生成的名稱會將其轉化為小寫字母。

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 套件名**, --package** 套件名 設定 Debian 套件名稱。

-u 上游版本號**, --upstreamversion** 版本號 設定上游套件版本。

-r 修訂號**, --revision** 修訂號 設定 Debian 套件修訂號。

-z 副檔名**, --targz** 副檔名 設定原始碼壓縮包型別，副檔名 =(**tar.gz**|**tar.bz2**|**tar.xz**)。（別名：**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**：程式庫開發套件（any，same）（別名：**de**）
- **doc**：文件套件（all，foreign）（別名：**do**）
- **lib**：程式庫套件（any，same）（別名：**l**）
- **perl**：Perl 指令碼套件（all，foreign）（別名：**pl**）
- **python3**: Python (version 3) script package (all, foreign) (alias: **py3**, **python**, **py**)
- **ruby**：Ruby 指令碼套件（all，foreign）（別名：**rb**）
- **nodejs**: Node.js based JavaScript package (all, foreign) (alias: **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**’”

如果原始碼樹的內容和型別的設定不一致，**debmake** 命令會發出警告。

-e ***foo@example.org*****, --email** ***foo@example.org*** 設定電子郵件地址。

預設值為環境變數 **$DEBEMAIL** 的值。

-f ” 名稱姓氏**”, --fullname** ” 名稱姓氏**”** 設定全名。

預設值為環境變數 **$DEBFULLNAME** 的值。

-i ” 構建工具**”, --invoke** ” 構建工具**”** invoke ”*buildtool*” at the end of execution. *buildtool* may be “**dpkg-buildpackage**”, “**debuild**”, “**sbuild**”, etc.

預設情況是不執行任何程式。

設定該選項也會自動設定 **--local** 選項。

-j, --judge 執行 **dpkg-depcheck** 以檢查構建依賴和檔案路徑。檢查日誌將儲存在父目錄下。

- 套件名**.build-dep.log**：**dpkg-depcheck** 的日誌檔案。
- 套件名**.install.log**：記錄 **debian/tmp** 目錄下所安裝檔案的日誌。

-l ”***license_file,*****⋯”, --license** ”***license_file,*****⋯”** 在存放許可證掃描結果的 **debian/copyright** 檔案末尾新增格式化後的許可證文字。

The default is to add **COPYING** and **LICENSE**, and *license_file* needs to list only the additional file names all separated by “**,**”.

-m, --monoarch 強制套件不使用多架構特性。

-o 檔案**, --option** 檔案 從指定 *file* 讀取可選引數。（這個選項不適合日常使用。）

The content of *file* is sourced as the Python code at the end of **para.py**. For example, the package description can be specified by the following file.

```
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 在建立 **debian/** 目錄下的檔案之前即提前退出程式。

-s, --spec use upstream spec (**pyproject.py** for Python, etc.) for the package description.

-v, --version 顯示版本資訊。

-w "*addon,*⋯", --with "*addon,*⋯" 在 **debian/rules** 檔案中在 **dh**(1) 命令的引數中新增額外的 **dh**(1) 引數以指定所使用的附加元件（*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* 以模板檔案的形式建立配置檔案（請注意 **debian/changelog**、**debian/control**、**debian/copyright** 和 **debian/rules** 檔案是構建 Debian 二進位制套件所需的最小文件集合。）

n 的數字大小決定了生成哪些配置模板檔案。

- **-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 為本地套件生成配置檔案以繞過 **lintian**(1) 的檢查。

-P, --pedantic 對自動生成的檔案進行嚴格（甚至古板到迂腐程度）的檢查。

-T, --tutorial output tutorial comment lines in template files. default when **-x3** or **-x4** is set.

15.4 範例

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**

For a typical Python (version 3) module source tree:

- **debmake -s -d -b":python3" -i debuild**

For a typical Python (version 3) module in the *package-version***.tar.gz** archive:

- **debmake -s -a** *package-version***.tar.gz -b":python3" -i debuild**

對於典型的以 *package-version***.tar.gz** 歸檔提供的 Perl 模組：

- **debmake -a** *package-version***.tar.gz -b":perl" -i debuild**

15.5 幫助套件

打包工作也許需要額外安裝一些專用的幫助套件。

- 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 程式可能需要 **gem2deb** 套件。
- Node.js based JavaScript programs may require the **pkg-js-tools** package.
- Java 程式可能需要 **javahelper** 套件。
- Gnome 程式可能需要 **gobject-introspection** 套件。
- 等等。

15.6 注意事項

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.

組成 Debian 套件名稱的字元選取存在一定的限制。最明顯的限制應當是套件名稱中禁止出現大寫字母。這裡給出正則表示式形式的規則總結：

- 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 除錯

請使用 **reportbug** 命令報告 **debmake** 套件的問題與錯誤。

環境變數 **$DEBUG** 中設定的字元用來確定日誌輸出等級。

- **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 許可證

Expat 許可證

15.10 參見

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 命令提供了兩個快捷選項。

- **-a**：開啟上游原始碼壓縮包
- **-i**：執行構建二進位制包的指令碼

前文中“章 5”的例子可以使用下面的命令直接達到目的。

```
$ 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 “節 10.10”)
- **Depends:**

- **Pre-Depends:**

The **debmake** command also sets an appropriate set of substvars (substitution variables) used in each pertinent dependency stanza.

我們在這裡直接引用 **debmake** 手冊頁中的相關一部分內容。

-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**：程式庫開發套件（any，same）（別名：**de**）
- **doc**：文件套件（all，foreign）（別名：**do**）
- **lib**：程式庫套件（any，same）（別名：**l**）
- **perl**：Perl 指令碼套件（all，foreign）（別名：**pl**）
- **python3**: Python (version 3) script package (all, foreign) (alias: **py3**, **python**, **py**)
- **ruby**：Ruby 指令碼套件（all，foreign）（別名：**rb**）
- **nodejs**: Node.js based JavaScript package (all, foreign) (alias: **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'**"

如果原始碼樹的內容和型別的設定不一致，**debmake** 命令會發出警告。

16.3 debmake -cc

debmake 命令在帶上 **-cc** 選項時可以對標準輸出列印整個原始碼樹的版權和許可證概要資訊。

```
$ tar -xvzf package-1.0.tar.gz
$ cd package-1.0
$ debmake -cc | less
```

如果轉而使用 **-c** 選項，程式將提供較短的報告。

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** 選項以使用 **tar** 命令生成上游原始碼套件。

```
$ 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.

如果版本控制系統中的上游原始碼樹包含了 **debian/*** 檔案，**debmake** 命令在帶有 **-d** 選項或者 **-t** 選項並結合 **-i** 選項可以自動化進行使用這些 **debian/*** 檔案從版本控制系統快照中構建非原生套件的流程。

```
$ 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 "節 11.13"for more details.

16.5 debmake -j

This is an experimental feature.

生成多個二進位制套件通常比只生成一個二進位制套件需要投入更多的工作量。對原始碼包進行測試構建是其中的必要一環。

例如，我們考慮將相同的 *package-1.0***.tar.gz**（參見"章 5"）打包並生成多個二進位制套件。

- 呼叫 **debmake** 命令並使用 **-j** 選項以測試構建並報告結果。

  ```
  $ debmake -j -a package-1.0.tar.gz
  ```

- 請檢查 *package***.build-dep.log** 檔案最後的幾行以確定 **Build-Depends** 所需填寫的構建依賴。（您不需要在 **Build-Depends** 中列出 **debhelper**、**perl** 或 **fakeroot** 所使用的套件。在只生成單個套件的情況下也是如此。）
- 請檢查 *package***.install.log** 的檔案內容以確定各個檔案的安裝路徑，從而決定如何將它們拆分成多個套件。
- 呼叫 **debmake** 命令以開始準備打包資訊。

  ```
  $ rm -rf package-1.0
  $ tar -xvzf package-1.0.tar.gz
  $ cd package-1.0
  $ debmake -b"package1:type1, ..."
  ```

- 請使用以上資訊更新 **debian/control** 和 **debian/***binarypackage***.install** 檔案。
- 按需更新其它 **debian/*** 檔案。
- 使用 **debuild** 或等效的其它工具構建 Debian 套件。

```
$ debuild
```

- 所有由 **debian/***binarypackage***.install** 檔案指定的二進位制套件條目均會生成 *binarypackage_version-revision_arch***.deb** 的安裝檔。

注

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

This is an experimental feature.

在使用上游新發行版本更新套件時，**debmake** 可以使用已有的 **debian/copyright** 檔案和整個更新的原始碼樹檔案進行對比驗證版權和許可證資訊。

```
$ 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.

在您編輯自動生成的 **debian/copyright** 檔案時，請確保將最通用的檔案匹配模式放在檔案前部，最精確的匹配模式放在後部。

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

呼叫 **debmake** 命令並帶上 **-P** 選項將會嚴厲地檢查所有自動生成檔案的版權和許可證文字資訊；即使它們都使用寬鬆的許可證也是如此。

此選項不止會影響正常執行過程中所生成的 **debian/copyright** 檔案的內容，也會影響帶引數 **-k**、**-c**、**-cc** 和 **-ccc** 選項的輸出內容。

16.8 debmake -T

呼叫 **debmake** 命令並帶上 **-T** 選項會額外輸出詳細的教材註釋行。這些行在模板檔案中用 **###** 進行標註。

16.9 debmake -x

debmake 生成的模板檔案數量由 **-x**[01234] 選項進行控制。

- 請參見“節 14.1”以瞭解與揀選使用模板檔案的方式。

注

debmake 命令不會修改任何已存在的配置文件。