debian

Debian 維護者指南

青木修

January 16, 2021

Debian 維護者指南
by 青木修

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》（2021-01-13）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。

前言

如果您在某些方面算得上是有經驗的 Debian 使用者 [1] 的話，您可能遇上過這樣的情況：

- 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。

如果您想要建立一個 Debian 套件來滿足您的需求，並將您的工作與社群分享，您便是本篇指南的目標讀者，即未來的 Debian 維護者。[2] 歡迎來到 Debian 社群。

Debian 是一個大型的、歷史悠久的志願者組織。因此，它具有許多需要遵守的社會上和技術上的規則和慣例。Debian 也開發出了一長串的打包工具和倉庫維護工具，用來構建一套能夠解決各種技術目標的二進位制套件：

- 跨多個架構構建的套件（Section 5.4.4）
- 可重現的構建（Section 5.4.5）
- 在明確指定套件依賴和補丁情況下乾淨地構建（Section 5.5, Section 5.8, Section 7.10）
- 拆分多個二進位制套件的最佳實踐（Section 5.5.1）
- 平滑的程式庫遷移（Section 5.18.2）
- 互動式安裝定製（Section 5.19）
- 多架構（multiarch）支援（Section 5.20）
- 使用特定的編譯選項進行安全增強（Section 5.21）
- 持續整合（Section 5.22）
- 自主生成（boot strapping Section 5.23）
- ……

這些目標也許會讓很多新近參與進 Debian 工作中的潛在 Debian 維護者感到迷茫而不知所措。本篇指南嘗試為這些目標提供一個著手點，方便讀者開展工作。它具體描述了以下內容：

- 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。
- 製作 Debian 套件的小竅門。
- 在某些典型場景下製作 Debian 套件的範例。

[1] 您的確需要對 Unix 程式設計有所瞭解，但顯然沒必要是這方面的天才。在 Debian 參考手冊 中，您可以瞭解到使用 Debian 系統的一些基本方法和關於 Unix 程式設計的一些指引。

[2] 如果您對分享 Debian 套件不感興趣，您當然可以在本地環境中將上游的原始碼套件進行編譯並安裝至 **/usr/local** 來解決問題。

作者在更新原有的使用 **dh-make** 套件的“新維護者手冊”時感受到了文件的侷限性。因此，作者決定建立一個替代工具並編寫其對應的文件以解決某些現代的需求。其成果便是 **debmake**（當前版本：)）套件，以及這篇更新的“Debian 維護者指南”，可從 **debmake-doc**（當前版本：1.16-1）套件取得。

許多雜項事務和小技巧都整合進了 **debmake** 命令，以使本指南內容簡單易懂。本指南同時提供了許多打包範例。

Caution

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。

某些重要的主題會詳細進行說明。其中某些可能看起來和您沒什麼關係。請保持耐心。某些罕見案例會被跳過。某些主題僅使用外部連結提及。這些都是有意的行文安排，目標是讓這份指南保持簡單而可維護。

Chapter 1

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。

> Caution
>
> 這裡的 **debmake** 命令只提供一些不錯的模板檔案。如果生成的套件需要釋出出去供公眾使用的話，這些模板檔案必須手工調整至最佳狀態以遵從 Debian 倉庫的嚴格質量標準。

如果您在 Debian 打包方面還是個新手的話，此時不要過多在意細節問題，請先確立一個大致流程的印象。

如果您曾經接觸過 Debian 打包工作，您會注意到這和 **dh_make** 命令很像。這是因為 **debmake** 命令設計時便旨在替代歷史上由 **dh_make** 命令所提供的功能。[1]

debmake 命令設計提供如下特性與功能：

- 現代的打包風格
  - **debian/copyright**：符合 **DEP-5**
  - **debian/control**：**substvar** 支援、**multiarch** 支援、多個二進位制套件、……
  - **debian/rules**：**dh** 語法、編譯器强化選項、……
- 靈活性
  - 許多選項（Section 5.5.1.1、Chapter 6、Appendix A）
- 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。
  - 生成非原生 Debian 套件，使用“**3.0 (quilt)**”格式，除非明確指定了 **-n** 選項。
- 額外的功能

[1] 歷史上還存在過 **deb-make** 命令，它在 **dh_make** 之前曾經流行過。當前的 **debmake** 套件的版本從 **4.0** 起始，其目的便是避免和廢棄的 **debmake** 套件的版本產生重疊。舊有的對應套件提供了 **deb-make** 命令。

– 根據當前原始碼對 **debian/copyright** 檔案進行驗證（Section 6.5）

debmake 命令將大多數重量級工作分派給了其後端套件：**debhelper**、**dpkg-dev**、**devscripts**、**pbuilder**，等等。

Tip

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。

Tip

非原生套件是標準的 Debian 套件。

Tip

本文件中所有套件構建範例的詳細日誌可以由 Section 8.14 一段給出的操作來取得。

Note

所產生的 **debian/copyright** 檔案，以及 **-c**（Section 6.4）和 **-k**（Section 6.5）選項的輸出都涉及了對版權和授權資訊的啟發式操作。它們具有侷限性，可能會輸出某些錯誤的結果。

Chapter 2

預備知識

這裡給出您在投入 Debian 相關工作之前應當理解掌握的一些必備的預備知識。

2.1 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。

注意，您不可能在一夜之間成為 **Debian** 開發者（DD），因為成為 DD 所需要的遠不只是技術技巧。不過別因此而氣餒，如果您的套件對其他人有用，您可以當這個軟體的套件維護者，然後通過一位贊助者來上傳這份軟體，或者您可以申請成為 **Debian** 維護者。

還有，要成為 Debian 開發者不一定要建立新套件。對已有軟體做出貢獻也是成為 Debian 開發者的理想途徑。眼下正有很多套件等著好的維護者來接手（參見 Section 2.8）。

2.2 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, 第 12 章 - “為 Debian 專案捐贈”（半官方）
- Debian Wiki, HelpDebian（補充內容）
- Debian 新成員站點（官方）
- Debian Mentors FAQ（補充內容）

2.3 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：

- 我們都是志願者。
  - 任何人都不能把事情强加給他人。
  - 您應該主動地做自己想做的事情。
- 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 並不是一所學校，在這裡沒有所謂的老師會自動地注意到您。
  - 您需要有自學大量知識和技能的能力。
  - 其他志願者的關注是非常稀缺的資源。
- 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 年的自由軟體、“實幹主義”、和民主（前任 DPL 製作的介紹投影片）

2.4 技術提醒

這裡給出一些技術上的建議，參考行事可以讓您與其他維護者共同維護套件時變得更加輕鬆有效，從而讓 Debian 專案的輸出成果最大化。

- 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]。

Note

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。

[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。

2.5 Debian 文件

請在閱讀本指南的同時按需閱覽下面這些 Debian 官方文件中的相關部分；這些文件提供的資訊有助於建立質量優良的 Debian 套件：

- 《Debian 政策手冊》
  - “必須遵循”的規則（https://www.debian.org/doc/devel-manuals#policy）
- “Debian 開發者參考”
  - “最佳實踐”文件（https://www.debian.org/doc/devel-manuals#devref）

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 新維護者手冊”（較舊）
  - https://www.debian.org/doc/devel-manuals#maint-guide
  - https://packages.qa.debian.org/m/maint-guide.html
- “Debian 打包教材”
  - https://www.debian.org/doc/devel-manuals#packaging-tutorial
  - https://packages.qa.debian.org/p/packaging-tutorial.html
- “Ubuntu 打包指南”（Ubuntu 基於 Debian。）
  - http://packaging.ubuntu.com/html/

Tip

閱讀這些教材時，您應當考慮使用 **debmake** 命令替代 **dh_make** 命令以獲得更好的模板文件。

2.6 幫助資源

在您決定在某些公共場合問出您的問題之前，請先做好自己能做到的事情，例如，閱讀能找到的文件：

- 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。

要取得您所需要的資訊，一種有效的方法是在網頁搜尋引擎中構建類似“關鍵字 **site:lists.debian.org**”這樣具有限制條件的搜尋字串來限定搜尋的域名。

製作一個小型測試用套件也是瞭解打包細節的一個好辦法。對當前已有的維護良好的套件進行檢查則是瞭解其他人如何製作套件的最好方法。

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 郵件列表。（這個郵件列表為專為新手答疑解惑。）
- debian-devel@lists.debian.org 郵件列表。（這個郵件列表針對熟練使用者和高階開發者。）
- IRC（網際網路中繼聊天） 例如 #debian-mentors。
- 專注某個特定套件集合的團隊。（完整列表請見 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

如果您在做好功課後能在這些場合中合適地提出您的疑問的話，那些更有經驗的 Debian 開發者會很願意幫助您。

Caution

Debian 的開發是一個不斷變動的目標。您在網上找到的某些資訊可能是過時的、不正確的或者不適用的，使用時請留意。

2.7 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）。

Tip

來自 **devscripts** 套件的 **wnpp-alert** 命令可以檢查已安裝軟體中需要接手或已被丟棄的套件。

2.8 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
  - 團隊
- 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。

您或是需要填寫並提交一份 *ITP*，或是需要“收養”一個套件並開始為其工作。請見“Debian 開發者參考（Debian Developer’s Reference）”文件：

- 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。

[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。

2.9 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
  - 文字標記語言的基礎知識（XML、ReST、Wiki 等）。

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。
  - 對 Debian 打包輔助腳本（如本文件中提及的 **devscripts**、**pbuilder** 等項目）提交補丁。
- **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 3

工具的配置

build-essential 套件必須在構建環境內預先安裝。

devscripts 套件應當安裝在維護者的工作環境中。

儘管這個不是絕對的要求，但是在維護者的工作環境內裝上並配置好本章節提到的各個常用的套件會有助於維護者高效投入工作。這些軟體可以構成我們共同確立的一個基準工作環境。

如有需要，請同樣按需安裝在“Debian 開發者參考”文中 Debian 維護者工具概覽 一節提到的各個工具。

Caution

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。

3.1 電子郵件地址

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。

我們可以通過將下面幾行加入 **~/.bashrc**[1] 的方式對這些軟體進行配置。

新增至 **~/.bashrc** 檔案

```
DEBEMAIL="your.email.address@example.org"
DEBFULLNAME="Firstname Lastname"
export DEBEMAIL DEBFULLNAME
```

3.2 mc

mc 命令提供了管理檔案的簡單途徑。它可以開啓二進位制 **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，以支援簡易 **chdir** 操作。

新增至 **~/.bashrc** 檔案

```
# mc related
export HISTCONTROL=ignoreboth
. /usr/lib/mc/mc.sh
```

[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。

3.3 git

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 "Name Surname"
$ git config --global user.email yourname@example.com
```

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
```

Tip

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。

3.4 quilt

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"
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:diff_ctx=35:diff_cctx=33"
    if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi
```

請參考 **quilt**(1) 和 處理大量補丁的方法暨對 Quilt 的介紹 以瞭解如何使用 **quilt** 命令。

要取得使用範例，請檢視 Section 4.8。

3.5 devscripts

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"
```

用於 **dpkg-source** 命令的 **DEBUILD_DPKG_BUILDPACKAGE_OPTS** 中可以額外使用 **-i** 和 **-I** 選項以幫助構建原始碼中具有外來無關內容的套件（參見 Section 5.15）。

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。

3.6 pbuilder

pbuilder 套件提供了淨室（乾淨的）（**chroot**）構建環境。[2]

我們可以搭配使用另外幾個輔助套件對其設定。

- **cowbuilder** 套件能加速 chroot 建立過程。
- **lintian** 套件能找到所構建套件中的缺陷。
- **bash**、**mc** 和 **vim** 套件在構建失敗時用來查詢問題。
- **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）

Warning

可選的設定項可能造成負面影響。如果有疑問，請關閉它們。

我們使用如下給出的內容來建立 **~/.pbuilderrc** 檔案（所有可選功能均已關閉）。

```
AUTO_DEBSIGN="${AUTO_DEBSIGN:-no}"
SOURCE_ONLY_CHANGES="${SOURCE_ONLY_CHANGES:-yes}"
PDEBUILD_PBUILDER=cowbuilder
HOOKDIR="/var/cache/pbuilder/hooks"
MIRRORSITE="http://deb.debian.org/debian/"
#APTCACHE=/var/cache/pbuilder/aptcache
APTCACHE=/var/cache/apt/archives
#BUILDRESULT=/var/cache/pbuilder/result/
BUILDRESULT=../
EXTRAPACKAGES="lintian"
#EXTRAPACKAGES="ccache lintian libeatmydata1"

# enable to use libeatmydata1 for pbuilder
#export LD_PRELOAD=${LD_PRELOAD+$LD_PRELOAD:}libeatmydata.so

# enable ccache for pbuilder
#export PATH="/usr/lib/ccache${PATH+:$PATH}"
#export CCACHE_DIR="/var/cache/pbuilder/ccache"
#BINDMOUNTS="${CCACHE_DIR}"

# parallel make
#DEBBUILDOPTS=-j8
```

[2] **sbuild** 套件提供了另一套 chroot 平臺。

Note

可以考慮建立從 **/root/.pbuilderrc** 到 **/home/<user>/.pbuilderrc** 的符號鏈接以獲得一致的體驗。

Note

由於 缺陷 #606542，您可能需要手動將 **EXTRAPACKAGES** 列出的套件安裝進入 chroot。請見 Section 7.10。

Note

應當在 chroot 環境內外同時安裝上 **libeatmydata1** (>=82-2)，否則即為關閉 **libeatmydata1**。該套件在某些構建系統中可能導致競爭情況（race condition）發生。

Note

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。

我們可以按如下方式建立觸動指令碼。

/var/cache/pbuilder/hooks/A10ccache

```
#!/bin/sh
set -e
# increase the ccache caching size
ccache -M 4G
# output the current statistics
ccache -s
```

/var/cache/pbuilder/hooks/B90lintian

```
#!/bin/sh
set -e
apt-get -y --allow-downgrades install lintian
echo "+++ lintian output +++"
su -c "lintian -i -I --show-overrides /tmp/buildd/*.changes; :" -l pbuilder
echo "+++ end of lintian output +++"
```

/var/cache/pbuilder/hooks/C10shell

```
#!/bin/sh
set -e
apt-get -y --allow-downgrades install vim bash mc
# invoke shell if build fails
cd /tmp/buildd/*/debian/..
/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

Note

所有這些指令碼都需要設定為全域性可執行 :“**-rwxr-xr-x 1 root root**”。

Note

ccache 的快取目錄 **/var/cache/pbuilder/cache** 需要為了 **pbuilder** 命令的使用而設定為全域性可寫 :“**-rwxrwxrwx 1 root root**”。您需要明白這樣會帶來相關的安全隱患。

3.7 git-buildpackage

您也可能需要在 **~/.gbp.conf** 中設定全域性配置資訊

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

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

Tip

這裡的 **gbp** 命令是 **git-buildpackage** 命令的一個別名。

3.8 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：

- 簡單的 HTTP 快取代理，使用 **squid** 軟體包。
- 特化的 HTTP 快取代理，使用 **apt-cacher-ng** 套件。

3.9 私有 Debian 倉庫

您可以使用 **reprepro** 套件搭建私有 Debian 倉庫。

Chapter 4

簡單例子

有一句古羅馬諺語說得好：“**Longum iter est per praecepta, breve et efficax per exempla**”（“一例勝千言！”）。

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：

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

Debian 的打包需要對“**make install**”流程進行改變，從而將檔案安裝至目標系統映象所在位置，而非通常使用的 **/usr/local** 下的位置。

Note

在其它更加複雜的構建系統下構建 Debian 套件的例子可以在 Chapter 8 找到。

4.1 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
  ...
```

Tip

此處和下面例子中的 **debuild** 命令可使用等價的命令進行替換，例如 **pdebuild** 命令。

Tip

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。

4.2 什麼是 **debmake**？

文中的 **debmake** 命令是用於 Debian 打包的一個輔助腳本。

- 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。
- 它能建立良好的模板檔案，例如符合 **DEP-5** 的 **debian/copyright** 檔案。

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

Note

debmake 命令並不是製作一個 Debian 套件的唯一途徑。許多套件是打包者模仿其它已有的打包範例，僅僅使用文字編輯器而編寫完成打包指令碼的。

4.3 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**”（應用 Debian 補丁，除非它們已被應用）
  - “**fakeroot debian/rules clean**”
  - “**dpkg-source --build**”（構建 Debian 原始碼包）
  - “**fakeroot debian/rules build**”
  - “**fakeroot debian/rules binary**”
  - “**dpkg-genbuildinfo**”（產生一個 ***.buildinfo** 檔案）
  - “**dpkg-genchanges**”（產生一個 ***.changes** 檔案）

  - "**fakeroot debian/rules clean**"
  - "**dpkg-source --after-build**"（取消 Debian 補丁，如果它們在 **--before-build** 階段已被應用）
  - "**debsign**"（對 ***.dsc** 和 ***.changes** 檔案進行簽名）
    * 如果您按照 Section 3.5 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。
- **pdebuild** 命令是另一個封裝指令碼，它可以在合適的 chroot 環境下使用合適的環境變數構建 Debian 二進位制套件。
- **git-pbuilder** 命令是又一個用於構建 Debian 二進位制套件的封裝指令碼，它同樣可以確保使用合適的環境變數和 chroot 環境。不同之處在於它提供了一個更容易使用的命令列使用者介面，可以較方便地在不同的構建環境之間進行切換。

Note

如需瞭解詳細內容，請見 **dpkg-buildpackage**(1)。

4.4 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 -xzf debhello-0.0.tar.gz
 $ tree
.
b''├b''b''─b''b''─b'' debhello-0.0
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' Makefile
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-0.0.tar.gz

2 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
```

Note

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。

4.5 第二步：使用 **debmake** 產生模板檔案

Tip

如果 **debmake** 命令呼叫時使用了 **-T** 選項，程式將為模板檔案產生更加詳細的註釋內容。

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。

```
 $ cd debhello-0.0
 $ debmake
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.gz 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: 100 %, ext = c
I: check_all_licenses
I: ..
I: check_all_licenses completed for 2 files.
I: bunch_all_licenses
I: format_all_licenses
I: make debian/* template files
I: single binary package
I: debmake -x "1" ...
I: creating => debian/control
I: creating => debian/copyright
I: substituting => /usr/share/debmake/extra0/changelog
I: creating => debian/changelog
I: substituting => /usr/share/debmake/extra0/rules
I: creating => debian/rules
I: substituting => /usr/share/debmake/extra1/watch
I: creating => debian/watch
I: substituting => /usr/share/debmake/extra1/README.Debian
I: creating => debian/README.Debian
I: substituting => /usr/share/debmake/extra1source/format
I: creating => debian/source/format
I: substituting => /usr/share/debmake/extra1source/local-options
I: creating => debian/source/local-options
I: substituting => /usr/share/debmake/extra1patches/series
I: creating => debian/patches/series
I: run "debmake -x2" to get more template files
I: $ wrap-and-sort
```

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）
- **-x** 選項：**-x1**（是單個二進位制套件的預設值）

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 ..
 $ tree
.
b''├b''b''─b''b''─b'' debhello-0.0
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' Makefile
b''│b''   b''├b''b''─b''b''─b'' debian
b''│b''   b''│b''   b''├b''b''─b''b''─b'' README.Debian
b''│b''   b''│b''   b''├b''b''─b''b''─b'' changelog
b''│b''   b''│b''   b''├b''b''─b''b''─b'' control
b''│b''   b''│b''   b''├b''b''─b''b''─b'' copyright
b''│b''   b''│b''   b''├b''b''─b''b''─b'' patches
b''│b''   b''│b''   b''│b''   b''└b''b''─b''b''─b'' series
b''│b''   b''│b''   b''├b''b''─b''b''─b'' rules
b''│b''   b''│b''   b''├b''b''─b''b''─b'' source
b''│b''   b''│b''   b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''│b''   b''│b''   b''└b''b''─b''b''─b'' local-options
b''│b''   b''│b''   b''└b''b''─b''b''─b'' watch
```

```
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''├b''b''─b''b''─b'' debhello-0.0.tar.gz
b''└b''b''─b''b''─b'' debhello_0.0.orig.tar.gz -> debhello-0.0.tar.gz

5 directories, 14 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）：

```
 $ cat debhello-0.0/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,--as-needed

%:
        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 debhello-0.0/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Firstname Lastname" <email.address@example.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.0
Homepage: <insert the upstream URL, if relevant>

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.
```

Warning

如果您對 **debian/control** 模板檔案中的“**Section: unknown**”部分不做修改的話，後續的 **lintian** 錯誤可能導致構建失敗。

因為這是個 ELF 二進位制可執行檔案套件，**debmake** 命令設定了“**Architecture: any**”和“**Multi-Arch: foreign**”兩項。同時，它將所需的 **substvar** 引數設定為“**Depends: ${shlibs:Depends}, ${misc:Depends}**”。這些內容將在 Chapter 5 部分進行解釋。

Note

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 debhello-0.0/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
           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.

#-------------------------------------------------------------------------...
# License file: LICENSE
 License:
 .
 All files in this archive are licensed under the MIT License as below.
 .
 Copyright 2015 Osamu Aoki <osamu@debian.org>
 .
 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.
```

4.6 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。

為了將安裝檔案變成系統檔案的一部分，**Makefile** 檔案中 **$(prefix)** 預設的 **/usr/local** 的值需要被覆蓋為 **/usr**。要做到這點，可以按照下面給出的方法，在 **debian/rules** 檔案中新增名為 **override_dh_auto_install** 的目標，在其中設置“**prefix=/usr**”。

debian/rules（維護者版本）：

```
 $ vim debhello-0.0/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-0.0/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** 環境變數可以强制 **debhelper** 工具輸出細粒度的構建報告。

如上匯出 **DEB_BUILD_MAINT_OPTION** 變數可以如 **dpkg-buildflags**(1) 手冊頁中“FEATURE AREAS/ENVIRONMENT”部分所說，對强化選項進行設定。[1]

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

如上匯出 **DEB_LDFLAGS_MAINT_APPEND** 可以强制連結器只對真正需要的程式庫進行連結。[2]

對基於 Makefile 的構建系統來說，**dh_auto_install** 命令所做的基本上就是“**$(MAKE) install DESTDIR=debian/debhello**”。這裡建立的 **override_dh_auto_install** 目標將其行為修改為“**$(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 debhello-0.0/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-0.0/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: example package in the debmake-doc package
 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)'.
```

debian/copyright（維護者版本）：

```
 $ vim debhello-0.0/debian/copyright
 ... hack, hack, hack, ...
 $ cat debhello-0.0/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
```

[1] 這裡的做法是為了進行强化而强制啓用只讀重定位連結，以此避免 lintian 的警告“**W: debhello: hardening-no-relro usr/bin/hello**”。其實它在本例中並不是必要的，但加上也沒有什麼壞處。對於沒有外部鏈接程式庫的本例來說，lintian 似乎給出了誤報的警告。

[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。

```
Files:     *
Copyright: 2015 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.
```

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/. 下面的模板檔案（**0.0** 版）：

```
 $ tree debhello-0.0/debian
debhello-0.0/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' patches
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch

2 directories, 9 files
```

Tip

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。

4.7 第四步：使用 **debuild** 構建套件

您可以使用 **debuild** 或者等效的命令工具（參見 Section 4.3）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。

```
 $ cd debhello-0.0
 $ debuild
 dpkg-buildpackage -us -uc -ui -i -i
 ...
 fakeroot debian/rules clean
dh clean
 ...
```

```
 debian/rules build
dh build
   dh_update_autotools_config
   dh_autoreconf
   dh_auto_configure
        install -d /path/to/debhello-0.0/debian/.debhelper/generated/_source/...
   dh_auto_build
        make -j4 "INSTALL=install --strip-program=true"
make[1]: Entering directory '/path/to/debhello-0.0'
# CFLAGS=-g -O2
# -fdebug-prefix-map=/home/osamu/pub/salsa/debmake/debmake-doc/debhello-
 ...
 fakeroot debian/rules binary
dh binary
 ...
Now running lintian -i -I --show-overrides debhello_0.0-1_amd64.changes ...
 ...
W: debhello: binary-without-manpage usr/bin/hello
N:
N:    Each binary in /usr/bin, /usr/sbin, /bin, /sbin or /usr/games should
N:    have a manual page
 ...
```

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 Chapter 8）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 ..
 $ tree -FL 1
.
b''├b''b''─b''b''─b'' debhello-0.0/
b''├b''b''─b''b''─b'' debhello-0.0.tar.gz
b''├b''b''─b''b''─b'' debhello-dbgsym_0.0-1_amd64.deb
b''├b''b''─b''b''─b'' debhello_0.0-1.debian.tar.xz
b''├b''b''─b''b''─b'' debhello_0.0-1.dsc
b''├b''b''─b''b''─b'' debhello_0.0-1_amd64.build
b''├b''b''─b''b''─b'' debhello_0.0-1_amd64.buildinfo
b''├b''b''─b''b''─b'' debhello_0.0-1_amd64.changes
b''├b''b''─b''b''─b'' debhello_0.0-1_amd64.deb
b''└b''b''─b''b''─b'' debhello_0.0.orig.tar.gz -> debhello-0.0.tar.gz

1 directory, 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 Section 5.17.1。
- **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/LICENSE
debhello-0.0/Makefile
debhello-0.0/src/
debhello-0.0/src/hello.c
 $ tar --xz -tf debhello_0.0-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
debian/patches/
debian/patches/series
debian/rules
debian/source/
debian/source/format
debian/watch
```

debhello_0.0-1_amd64.deb 包含了將要安裝至目標系統中的檔案。
debhello-debsym_0.0-1_amd64.deb 包含了將要安裝至目標系統中的除錯符號檔案。
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/66/
-rw-r--r-- root/root ...  ./usr/lib/debug/.build-id/66/73e0826b1e8bd84f511bac...
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 (>= 2.2.5)
```

Caution

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。

Note

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。

4.8 第三步（備選）：修改上游原始碼

上面的例子中，在建立合適的 Debian 套件時沒有修改上游的原始碼。

作為維護者，另一個備選的方案是對上游原始碼做改動，如修改上游的 **Makefile** 以將 $(prefix) 的值設定為 **/usr**。

打包操作基本上和上面的分步範例相同，除了在 Section 4.6 中的兩點：

- 要將維護者對上游原始碼的修改形成對應的補丁檔案存放在 **debian/patches/** 目錄內，並將它們的檔名寫入 **debian/patches/series** 檔案，如 Section 5.8 所述。有數種生成補丁檔案的方式。下面的章節中給出了一些例子：
  - Section 4.8.1
  - Section 4.8.2
  - Section 4.8.3
- 此時維護者對 **debian/rules** 檔案的修改如下所示，它不包含 **override_dh_auto_install** 目標：

  debian/rules（備選的維護者版本）：

```
 $ cd 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 $@
```

這個使用一系列補丁檔案進行 Debian 打包的備選方案對於應對上游未來的改變可能不夠健壯，但是在應對更為複雜的上游原始碼時可以更靈活。（參見 Section 7.13。）

Note

對當前這個特定的打包場景，前文的 Section 4.6 中使用 **debian/rules** 檔案的方式更好一些。但為了演示起見，此時我們先使用本節的方式繼續操作。

Tip

對更復雜的打包場景，可能需要同時應用 Section 4.6 和 Section 4.8 中的方式。

4.8.1 使用 **diff -u** 處理補丁

這裡我們使用 **diff** 命令建立一個 **000-prefix-usr.patch** 檔案作為例子。

```
 $ 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  2020-07-13 00:38:01.407949320 +0900
+++ debhello-0.0/Makefile       2020-07-13 00:38:01.479947950 +0900
@@ -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
```

請注意，上游的原始碼樹應當恢復到原始狀態，補丁檔案此時的名字為 **000-prefix-usr.patch**。

這個 **000-prefix-usr.patch** 檔案隨後應當進行編輯以使其符合 DEP-3，並照如下方式移動至正確的位置。

```
 $ cd debhello-0.0
 $ 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
```

4.8.2 使用 **dquilt** 處理補丁

這裡的例子使用 **dquilt** 命令（一個 **quilt** 程式的簡單封裝）建立 **000-prefix-usr.patch**。**dquilt** 命令的語法和功能與 **quilt**(1) 命令相同，唯一的區別在於補丁儲存在 **debian/patches/** 目錄中。

```
 $ cd debhello-0.0
 $ 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
.
b''├b''b''—b''b''—b'' .pc
b''│b''   b''├b''b''—b''b''—b'' .quilt_patches
b''│b''   b''├b''b''—b''b''—b'' .quilt_series
```

```
b''│b''   b''├b''b''─b''b''─b'' .version
b''│b''   b''├b''b''─b''b''─b'' 000-prefix-usr.patch
b''│b''   b''│b''   b''├b''b''─b''b''─b'' .timestamp
b''│b''   b''│b''   b''└b''b''─b''b''─b'' Makefile
b''│b''   b''└b''b''─b''b''─b'' applied-patches
b''├b''b''─b''b''─b'' LICENSE
b''├b''b''─b''b''─b'' Makefile
b''├b''b''─b''b''─b'' debian
b''│b''   b''├b''b''─b''b''─b'' README.Debian
b''│b''   b''├b''b''─b''b''─b'' changelog
b''│b''   b''├b''b''─b''b''─b'' control
b''│b''   b''├b''b''─b''b''─b'' copyright
b''│b''   b''├b''b''─b''b''─b'' patches
b''│b''   b''│b''   b''├b''b''─b''b''─b'' 000-prefix-usr.patch
b''│b''   b''│b''   b''└b''b''─b''b''─b'' series
b''│b''   b''├b''b''─b''b''─b'' rules
b''│b''   b''├b''b''─b''b''─b'' source
b''│b''   b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''│b''   b''└b''b''─b''b''─b'' local-options
b''│b''   b''└b''b''─b''b''─b'' watch
b''└b''b''─b''b''─b'' src
    b''└b''b''─b''b''─b'' hello.c

6 directories, 19 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
```

這裡，上游原始碼樹中的 **Makefile** 檔案沒有恢復到原始狀態的必要。在 Section 4.7 描述的 Debian 打包過程中呼叫的 **dpkg-source** 命令能夠理解由 **dquilt** 程式在 **.pc/** 目錄中記錄的補丁應用情況。只要所有這些修改都是由 **dquilt** 命令完成的，那麼 Debian 原始碼套件就可以從經過修改的原始碼樹中進行構建。

Note

如果 **.pc/** 目錄不存在，**dpkg-source** 命令就會假定沒有應用任何補丁。這就是更為原始的補丁生成方法，例如 Section 4.8.1 中未生成 **.pc/** 目錄的情況下要求將上游原始碼樹進行恢復的原因。

4.8.3 使用 **dpkg-source --commit** 處理補丁

這裡給出使用“**dpkg-source --commit**”命令生成 **000-prefix-usr.patch** 的例子。

我們先來編輯上游原始碼。

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

我們來進行提交。

```
 $ dpkg-source --commit . 000-prefix-usr.patch
... editor to edit the DEP-3 patch header
...
```

我們來看看效果如何。

```
 $ 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

 $ tree -a
.
b''├b''b''─b''b''─b'' .pc
b''│b''   b''├b''b''─b''b''─b'' .quilt_patches
b''│b''   b''├b''b''─b''b''─b'' .quilt_series
b''│b''   b''├b''b''─b''b''─b'' .version
b''│b''   b''├b''b''─b''b''─b'' 000-prefix-usr.patch
b''│b''   b''│b''   b''├b''b''─b''b''─b'' .timestamp
b''│b''   b''│b''   b''└b''b''─b''b''─b'' Makefile
b''│b''   b''└b''b''─b''b''─b'' applied-patches
b''├b''b''─b''b''─b'' LICENSE
b''├b''b''─b''b''─b'' Makefile
b''├b''b''─b''b''─b'' debian
b''│b''   b''├b''b''─b''b''─b'' README.Debian
b''│b''   b''├b''b''─b''b''─b'' changelog
b''│b''   b''├b''b''─b''b''─b'' control
b''│b''   b''├b''b''─b''b''─b'' copyright
b''│b''   b''├b''b''─b''b''─b'' patches
b''│b''   b''│b''   b''├b''b''─b''b''─b'' 000-prefix-usr.patch
b''│b''   b''│b''   b''└b''b''─b''b''─b'' series
b''│b''   b''├b''b''─b''b''─b'' rules
b''│b''   b''├b''b''─b''b''─b'' source
b''│b''   b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''│b''   b''└b''b''─b''b''─b'' local-options
b''│b''   b''└b''b''─b''b''─b'' watch
b''└b''b''─b''b''─b'' src
    b''└b''b''─b''b''─b'' hello.c

6 directories, 19 files
```

這裡，**dpkg-source** 命令完成了與 Section 4.8.2 一節中使用 **dquilt** 命令完全相同的流程。

Chapter 5

基本内容

這裡展示了 Debian 打包工作中針對非原生套件使用“**3.0 (quilt)**”格式進行打包所遵循基本規則的一個全域性性概覽。

Note

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。

使用“**3.0 (quilt)**”格式的非原生 Debian 套件是最普通的 Debian 原始碼套件格式。

Note

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。

5.1 打包工作流

建立 Debian 二進位制套件的 Debian 打包工作流涉及建立數個特定名稱的檔案（參見 Section 5.2），與《Debian 政策手冊》的定義保持一致。

The oversimplified method for the Debian packaging workflow can be summarized in 10 steps 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/*** 目錄下的必需技術説明檔案：

     debian/rules 構建 Debian 套件所需的可執行指令碼（參見 Section 5.4）

     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 Section 5.5）

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 Section 5.6）

debian/copyright 版權和許可證摘要資訊（參看 Section 5.7）

- 在 **debian/*** 下的可選配置檔案（參見 Section 5.11）：
- 在 *package-version/* 目錄中呼叫 **debmake** 命令將會提供這些配置檔案的一套模板。
  - 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。
- 這些檔案必須手工編輯以達到理想狀態。請使用《Debian 政策手冊》和《Debian 開發者參考》作為編輯依據。

5. **dpkg-buildpackage** 命令（通常由它的封裝命令 **debuild** 或 **pdebuild** 所使用）會在 *package-version/* 目錄中被呼叫，進而以呼叫 **debian/rules** 指令碼的方式製作 Debian 原始碼套件和二進位制套件。
   - 當前工作目錄會被設為：**$(CURDIR)**=*/path/to/package-version/*
   - 使用 **dpkg-source**(1) 以“**3.0 (quilt)**”格式建立 Debian 原始碼套件
     - *package_version*.**orig.tar.gz**（*package-version*.**tar.gz** 的副本或符號連結）
     - *package_version-revision*.**debian.tar.xz**（*package-version*/**debian/*** 的 tar 壓縮包，即 tarball）
     - *package_version-revision*.**dsc**
   - 使用“**debian/rules build**”構建原始碼並安裝到 **$(DESTDIR)** 中
     - **DESTDIR=debian**/*binarypackage/*（單二進位制包）
     - **DESTDIR=debian/tmp/**（多個二進位制包）
   - 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**
     - ……（可能有多個 Debian 二進位制包檔案。）
     - *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）
   - 遵守 ftp-master 的拒絕（rejection）指導方針。
     - 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 normal source-only uploads to the Debian archive.
   - Remove the source tree under */path/to/package-version/* which may contain artifacts from the build process.
   - Regenerate the clean source tree using “**dpkg-source -x** *package_version-revision*.**dsc**”.
   - Remove *package_version-revision*.**debian.tar.xz**.
   - Generate following files using “**dpkg-buildpackage -S -d**” in the clean source tree.
     - *package_version-revision*.**debian.tar.xz**
     - ’package_version-revision’_*source.changes*
     - ’package_version-revision’_*source.buildinfo*
9. Sign the *package_version-revision*.**dsc** and ’package_version-revision’_*source.changes* files with the **debsign** command using your private GPG key.
10. Upload the set of the Debian source package files with the **dput** command to the Debian archive.

Under some exceptional situation such as NEW uploads, uploads to the Debian archive may need to include Debian binary package files. For such situation, skip the step 8, sign *package_version-revision_arch*.**changes** instead of 'package_version-revision'_*source.changes* in the step 9, and upload the set of the Debian source and binary package files in the step 10.

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
- 將 *arch* 部分替換為套件對應架構

See also Source-only uploads.

Tip

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。

Tip

在"Debian 開發者參考"一文的 第 6 章最佳打包實踐 部分有十分詳盡的相關文件。請讀一讀這些內容。

5.1.1 debhelper 套件

儘管 Debian 套件可以僅由編寫 **debian/rules** 指令碼而不使用 **debhelper** 套件來生成，其實這樣做是不切實際的。現代的 Debian"政策"對許多功能特性的實現做了要求，如應用適當的檔案許可權、使用合適的與硬體架構相關的軟體庫安裝路徑、安裝觸動指令碼的插入、除錯符號套件的生成、套件依賴資訊的生成、套件資訊檔案的生成、對時間戳調節以符合可重現構建的要求，等等。

Debhelper 套件提供了一套實用指令碼，用來簡化 Debian 打包工作流並減輕套件維護者的負擔。若能適當運用，它們可以幫助打包者自動地處理並實現 Debian"政策"所要求的功能。

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。

5.2 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。

debmake 的目的是為套件維護者提供開始工作的模板檔案。註釋行以 **#** 開始，其中包含一些教材文字。您在將套件上傳至 Debian 倉庫之前必須刪除或者修改這樣的註釋行。

許可證資訊的提取和賦值過程應用了大量啓發式操作，因此在某些情況下可能不會正常工作。强烈建議您搭配使用其它工具，例如來自 **devscripts** 套件的 **licensecheck** 工具，以配合 **debmake** 的使用。

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：

- 上游套件名稱（**-p**）：[-+.a-z0-9]{2,}
- 二進位制套件名稱（**-b**）：[-+.a-z0-9]{2,}
- 上游版本號（**-u**）：[0-9][-+.:~a-z0-9A-Z]*
- Debian 修訂版本（**-r**）：[0-9][+.~a-z0-9A-Z]*

請在《Debian 政策手冊》的 第 5 章 - Control 檔案及其欄位 一節中檢視其精確定義。

debmake 所假設的打包情景是相對簡單的。因此，所有與直譯器相關的程式都會預設為“**Architecture: all**”的情況。當然，這個假設並非總是成立。

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。[1]

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**。

5.3 原生 Debian 套件

使用“**3.0 (quilt)**”格式的非原生 Debian 套件是最常見最標準的 Debian 原始碼套件格式。根據 **dpkg-source**(1) 的描述，此時的 **debian/source/format** 檔案應當包含“**3.0 (quilt)** 的文字內容。上述的工作流和接下來給出的打包範例都使用了這種格式。

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。

Caution

在上游 tarball 原始碼壓縮包無法使用其正確名稱 *package_version*.**orig.tar.gz** 被 **dpkg-buildpackage** 取得到的時候，會出現意外地構建了原生 Debian 套件的情況。這是新手常見的一個錯誤，通常是因構建中錯誤地在符號連結名稱中使用了“**-**”字元而非正確的“_”字元。[譯註：此處仍然假設打包的場景是已經取得或形成了名為 **package-version.tar.gz** 的上游原始碼 tarball。Debian 的打包工作很大程度上是以上游原始碼 tarball 作為基礎的，這一點須時刻牢記在心。]

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：

[1] 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。

- *package_version*.**tar.gz**（*package-version*.**tar.gz** 檔案的副本或符號連結，包含 **debian/*** 的各個檔案。）
- *package_version*.**dsc**

如果您需要手動建立原生 Debian 套件，可以使用 **dpkg-source**(1) 工具以“**3.0 (native)**”格式進行建立。

Tip

Some people promote packaging even programs that have been written only for Debian in the non-native package format. The required tarball without **debian/*** files needs to be manually generated in advance before the standard workflow in Section 5.1. [a] They claim that the use of non-native package format eases communication with the downstream distributions.

[a] Use of the “**debmake -t** ⋯” command or “**git deborig -f HEAD**” can help this workflow. See Section 6.2 and **dgit-maint-merge**(7).

Tip

如果使用原生套件格式，沒有必要事先建立 tarball 壓縮包。要建立一個原生 Debian 套件，應當將 **debian/source/format** 檔案的內容設定為“**3.0 (native)**”，適當編寫 **debian/changelog** 文件使得版本號中不包含 Debian 修訂號（例如，**1.0** 而非 **1.0-1**），最後在原始碼樹中調用“**dpkg-source -b .**”命令。這樣做便可以自動生成包含原始碼的 tarball。

5.4 debian/rules

debian/rules 指令碼是用於實際構建 Debian 套件的可執行指令碼。

- **debian/rules** 指令碼重新封裝了上游的構建系統（參見 Section 5.16）以達到將檔案安裝至 **$(DESTDIR)** 並將生成的檔案存入各個 **deb** 格式檔案中的目的。
  - 這裡的 **deb** 檔案用於二進位制的檔案分發，並將被 **dpkg** 命令所使用以將軟體安裝至系統中。
- **dh** 命令通常在 **debian/rules** 指令碼中使用，用作構建系統的一個前端。
- **$(DESTDIR)** 路徑具體值依賴於構建的型別。
  - **$(DESTDIR)=debian/***binarypackage*（單個二進位制套件）
  - **$(DESTDIR)=debian/tmp**（多個二進位制套件）

5.4.1 dh

由 **debhelper** 套件提供的 **dh** 命令與一些相關的套件共同工作，作為典型的上游構建系統的一層封裝，同時它支援所有 Debian 政策（Debian Policy）規定必須在 **debian/rules** 實現的目標（target），以此提供一個統一的查詢介面。

- **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
- **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

Note

對使用了 **debhelper**"compat >=9"的情況，**dh** 命令將在編譯引數未事先設定的情況下根據 **dpkg-buildflags** 命令返回的值設定並匯出各個編譯引數（如 **CFLAGS**、**CXXFLAGS**、**FFLAGS**、**CPPFLAGS** 和 **LDFLAGS**）。（**dh** 命令將呼叫在 **Debian::Debhelper::Dh_Lib** 模組中定義的 **set_buildflags**。）

5.4.2 簡單的 debian/rules

受益於 **dh** 命令對構建目標的抽象化[2]，一個符合 Debian 政策而支援所有必需目標（target）的 **debian/rules** 檔案可以簡單地寫成如下形式[3]：

簡單的 **debian/rules**：

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

%:
        dh $@
```

從本質上來看，這裡的 **dh** 命令的作用是作為一個序列化工具，在合適的時候呼叫所有所需的 **dh_*** 命令。

Tip

設定"**export DH_VERBOSE = 1**"會輸出構建系統中每一條會修改檔案內容的命令。它同時會在某些構建系統中啟用詳細輸出構建日誌的選項。

5.4.3 設定 debian/rules

通過新增合適的 **override_dh_*** 目標（target）並編寫對應的規則，可以實現對 **debian/rules** 指令碼的靈活定製。

如果需要在 **dh** 命令呼叫某些特定的 **dh_*foo*** 命令時採取某些特別的操作，則任何自動執行的操作均可以被 **debian/rules** 中額外新增的 **override_dh_*foo*** 這樣的 Makefile 目標所覆寫。

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**，
- **setup.py**，或
- **Build.PL**。

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

在這種情況下，您應該新增一個 **override_dh_auto_build** 目標並在其中執行“**dh_auto_build --** 設定引數”的命令。這樣可以在 **dh_auto_build** 預設傳遞的引數之後確保將使用者給出的 設定引數繼續傳遞給那些構建系統。

Tip

如果上文提到的構建系統命令已知得到了 **dh_auto_build** 命令的支援的話，請避免直接呼叫這些命令（而讓 **dh_auto_build** 自動處理）。

debmake 命令所建立的初始模版檔案除了應用了上文提到的簡單 **debian/rules** 檔案的優點外，同時為後續可能涉及的套件强化等情景添加了一些額外的定製選項。您需要先了解整個構建系統背後的工作原理（參見 Section 5.16），之後才能收放自如地定製套件來處理某些非常規的工作情況。

- 請參考 Section 4.6 一節以瞭解如何對 **debmake** 命令生成的 **debian/rules** 檔案模版進行定製。
- 請參見 Section 5.20 以瞭解與 multiarch 相關的定製方法。
- 請參見 Section 5.21 以瞭解與套件强化相關的定製方法。

5.4.4 debian/rules 中的變數

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 **DEB_SOURCE**、**DEB_VERSION**、**DEB_VERSION_EPOCH_UPSTREAM**、**DEB_VERSION_UPSTR**
DEB_VERSION_UPSTREAM 和 **DEB_DISTRIBUTION** 變數。它們在向後移植（backport）支援等場景下能起到一定的作用。

vendor.mk **DEB_VENDOR** 和 **DEB_PARENT_VENDOR** 變數，以及 **dpkg_vendor_derives_from** 巨集。它們在系統提供方的支援方面（Debian、Ubuntu 等）有其特定用處。

architecture.mk 設定 **DEB_HOST_*** 和 **DEB_BUILD_*** 變數。除此之外存在一種替代方案，即直接呼叫 **dpkg-architecture** 來取得變數，一次呼叫查詢得到一個變數值。如顯性呼叫 **dpkg-architecture** 以取得必需變數的話，便不再需要在 **debian/rules** 中包含 **architecture.mk** 了（後者會引入全部架構相關的變數）。

buildflags.mk 設定 **CFLAGS**、**CPPFLAGS**、**CXXFLAGS**、**OBJCFLAGS**、**OBJCXXFLAGS**、**GCJFLAGS**、**FFLAGS**、**FCFLAGS** 和 **LDFLAGS** 這些構建標誌（build flags）。

如果您希望在 **debian/rules** 中使用其中的某些變數，您可以將相關的程式碼複製到 **debian/rules** 檔案中，或是重寫一份簡單的替代實現。總而言之請保持 **debian/rules** 檔案儘量簡單。

例如，您按如下的方法在 **debian/rules** 檔案中新增內容，從而為 **linux-any** 目標架構添加額外的 **CONFIGURE_FLAGS**：

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

Tip

歷史上對於 **debhelper** 相容等級小於等於 8 的情況下，在 **debian/rules** 檔案中包含 **buildflags.mk** 檔案是很有用的，它可以合適地設定一些構建標誌，如 **CPPFLAGS**、**CFLAGS**、**LDFLAGS** 等，同時保證對特定選項，如 **DEB_CFLAGS_MAINT_APPEND** 和 **DEB_BUILD_MAINT_OPTIONS** 的合適處理。現在您應當使用的 **debhelper** 相容等級大於等於 9，故如無特殊原因，請不要繼續包含 **buildflags.mk**，請交由 **dh** 命令來處理和設定這些構建標誌。

參見 Section 5.20、**dpkg-architecture**(1) 和 **dpkg-buildflags**(1)。

5.4.5 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。
- 在 **debian/rules** 中使用“**dh $@**”以應用最新的 **debhelper** 特性。
- 在構建環境中匯出環境變數“**LC_ALL=C.UTF-8**”（參見 Section 7.15）。
- 對上游原始碼中使用的時間戳，使用 debhelper 提供的環境變數 **$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。

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)

5.5 debian/control

debian/control 檔案包含了由空行分隔的數塊參數資料。每塊元資料按照如下的順序定義了下面這些內容：

- 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 meta data.

Note

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

5.5.1 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：

- Section 8.11（基於 Autotools）
- Section 8.12（基於 CMake）

5.5.1.1 debmake -b

debmake 命令的 **-b** 選項提供了一個符合直覺又靈活的功能，可以用來建立 **debian/control** 的初始模板檔案，其中可以定義多個 Debian 二進位制套件，每節中含有如下欄位：

- **Package:**
- **Architecture:**
- **Multi-Arch:**
- **Depends:**
- **Pre-Depends:**

debmake 命令也會在每個適當的依賴欄位中設置合適的變數替換佔位符（substvars）。
我們在這裡直接引用 **debmake** 手冊頁中的相關一部分內容。

-b " 二進位制套件名 ***[:type],*…", --binaryspec "** 二進位制套件名 ***[:type],*…"** 設定二進位制套件的指定型別內容，使用一個用逗號分隔的二進位制套件名*:* 型別成對列表；例如，使用完整形式"**foo:bin,foo-doc:doc,libfoo1:lib,libfoo-dev:dev**" 或者使用短形式，"**-doc,libfoo1,libfoo-dev**"。

這裡，二進位制套件是二進位制套件名稱，可選的類型應當從下面的型別值中進行選取：

- **bin**：C/C++ 預編譯 ELF 二進位制程式碼套件（any，foreign）（預設，別名："**"**，即，空字串）
- **data**：資料（字型、影象、……）套件（all，foreign）（別名：**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**)
- **ruby**：Ruby 指令碼套件（all，foreign）（別名：**rb**）
- **nodejs**: Node.js based JavaScript package (all, foreign) (alias: **js**)
- **script**：Shell 指令碼套件（all，foreign）（別名：**sh**）

括號內成對的值，例如（any，foreign），是套件的架構和多架構（**Multi-Arch**）特性的值，它們將設定在 **debian/control** 檔案中。

大多數情況下，**debmake** 命令可以有效地從二進位制套件的名稱猜測出正確的型別。如果型別的值並不明顯，程式將回退到將型別設定為 **bin**。例如，**libfoo** 設定型別為 **lib**，而 **font-bar** 會令程式設定型別為 **data**，……

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。

5.5.1.2 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
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

5.5.1.3 程式庫套件名稱

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。

> Warning
>
> 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。
    - 其相關聯的程式庫套件在程式庫變遷進行時需要從 **lib***foo7* 重新命名為 **lib***foo8* 以避免 **unstable** 倉庫內依賴關係的破壞。
  - 該方法適用於簡單 binNMU 可以快速解決所有受影響套件對該程式庫依賴的情況下。（ABI 有變化，過時的 API 被移除而常用、活躍的 API 未變化。）
  - 該方法有時也能適用於可協調進行手動的程式碼更新，影響範圍限定在有限的一些套件中的情況下。（API 有變化）
- 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。
  - 該方法適用於存在多個依賴該程式庫的套件，且升級時常常涉及手動程式碼更新的場景。（API 有變化）否則，受影響的套件會無法從原始碼重新構建並導致對發行而言致命的 bug 出現。

> Tip
>
> 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 Section 5.18。

5.5.2 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

對共享連結程式庫來說，所需要的依賴程式庫是由執行“**objdump -p** */path/to/program* **| grep NEEDED**”這樣的命令來得到的，由 **shlib** 佔位符進行變數替換。

對於 Python 和其它直譯器來說，所需的模組通常由對包含類似“**import**”、“**use**”、“**require**”等等關鍵字的行進行解析，並會體現在各自對應的變數替換佔位符所在位置上。

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。

5.5.3 binNMU 安全

一個 binNMU（二進位制非維護者上傳） 是為程式庫遷移或其它目的所作的非維護者套件上傳。在一次 binNMU 上傳中，只有“**Architecture: any**”的套件會被重構建，其版本號會在末尾附加一個編號（例如，原來版本為 2.3.4-3，新上傳的包版本會變成 2.3.4-3+b1）。所有“**Architecture: all**”的包將不會重新構建。

來自同一個原始碼套件的各個二進位制包如果在 **debian/control** 檔案中有互相的依賴關係，這些二進位制包通常情況下應當對 binNMU 是安全的（即，進行 binNMU 不會破壞依賴關係）。然而，在“**Architecture: any**”和“**Architecture: all**”的套件同時由同一原始碼套件產出，且互相之間有依賴關係時，需要小心對待所依賴的版本，必要時應做出調整。

- “**Architecture: any**”的套件依賴於“**Architecture: any**”*foo* 套件
  - **Depends:** *foo* **(= ${binary:Version})**
- “**Architecture: any**”的套件依賴於“**Architecture: all**”*bar* 套件
  - **Depends:** *bar* **(= ${source:Version}**
- “**Architecture: all**”的套件依賴於“**Architecture: any**”*baz* 套件
  - **Depends:** *baz* **(>= ${source:Version})**, *baz* **(<< ${source:Version}.0~)**

5.6 debian/changelog

debian/changelog 檔案記錄了 Debian 軟體包的歷史並在其第一行定義了上游套件的版本和 Debian 修訂版本。所有改變的內容應當以明確、正式而簡明的語言風格進行記錄。

- 即便您在自己獨立進行套件上傳，您也必須記錄所有較重要、使用者可見的變更，例如：
  - 安全相關的漏洞修復。
  - 使用者介面變動。
- 如果您需要他人協助您進行上傳，您應當更詳盡地記錄變更內容，包括所有打包相關的變動，從而方便他人對您的套件進行審查。
  - 協助上傳的人員不應該也通常不會猜測您沒有寫出來的想法，所以請認真書寫變更信息。
  - 通常來說，協助您上傳的人的時間比您的時間更寶貴。

debmake 命令會建立初始的模板檔案，其中帶有上游套件版本和 Debian 打包修訂編號。發行版部分被設定為 **UNRELEASED** 以避免半成品不小心被上傳進入 Debian 倉庫。

通常使用 **debchange** 命令（它具有一個別名，即 **dch**）對其進行編輯。

Tip

您也可以手動使用任何文字編輯器修改 **debian/changelog** 檔案，只要您能夠遵循 **debchange** 命令所使用的特定文字排版格式即可。

Tip

debian/changelog 檔案使用的日期字串可以使用“**LC_ALL=C date -R**”命令手動生成。

該檔案將由 **dh_installchangelogs** 命令安裝到 **/usr/share/doc/***binarypackage* 目錄，檔名為 **changelog.Debian.gz**。

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。

當您完成了主要打包工作並驗證了其質量之後，請考慮執行“**dch -r**”命令並將最終完成的 **debian/changelog** 檔案中發行版（distribution）部分進行設定，通常該欄位應當使用 **unstable**。[4] 如果您的打包是一次向後移植（backports）、是安全更新或是對長期支援版的上傳等等其它情況，請使用相應合適的發行版名稱。

5.7 debian/copyright

Debian 以十分嚴肅的態度對待版權和許可證資訊。《Debian 政策手冊》強制規定軟體包的 **debian/copyright** 檔案中需要提供相關資訊的摘要。

您應當按照 機器可解析的 debian/copyright 檔案（DEP-5）對其進行排版。

Caution

這裡的 **debian/copyright** 檔案中描述的許可證資訊匹配資訊應當合適地進行排序，以確保越寬泛的檔案匹配越靠前。請參見 Section 6.5。

debmake 命令會以掃描整個原始碼樹的方式建立初步的、相容 DEP-5 的模板檔案。它會內部呼叫許可證檢查工具來對許可證文字進行分類。[5]

除非明確指定（有些嚴格過頭的）**-P** 選項，**debmake** 命令會為了實用性而跳過對自動生成的檔案的檢查與報告，預設它們採用寬鬆的許可證。

Note

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。

[4] 如果您在使用 **vim** 編輯器，請確保使用“**:wq**”命令對內容進行儲存。

[5] 程式以前會在內部呼叫來自 **devscripts** 軟體包的 **licensecheck** 命令來進行檢查。現在的 **licensecheck** 命令由獨立的 **licensecheck** 套件所提供，相較從前的實現也有了較大的改進。

Note

debmake 命令專注於在建立 **debian/copyright** 模板時聚合相同的授權和許可證資訊並收錄其詳細內容。為了在有限的時間內儘可能完成工作，工具將只會提取檔案中第一塊看起來像授權文字或許可證宣告的部分。因此，生成的許可證資訊可能並不完美。請同時考慮使用其它工具，如 **licensecheck** 輔助進行檢查。

Tip

我們強烈推薦您使用 **licensecheck**(1) 命令再次檢查原始碼許可證的狀態，並在有必要的情況下進行人工程式碼審查。

5.8 debian/patches/*

在構建過程開始之前，**debian/patches/** 目錄內的 **-p1** 等級的補丁將會按照在 **debian/patches/series** 檔案中指定的順序依次應用於上游程式碼樹中。

Note

原生 Debian 套件（參見 Section 5.3）將不使用這些檔案。

要準備這一系列 **-p1** 等級的補丁，有幾種不同的方式可供您選擇。

- **diff** 命令
  - 參見 Section 4.8.1
  - 原始但萬能的方法
    - 補丁的來源多種多樣，它可以來自其它發行版、郵件列表中的帖文或是來自上游 **git** 倉庫的揀選補丁，由“**git format-patches**”生成
  - 不涉及 **.pc/** 目錄的問題
  - 不修改上游原始碼樹
  - 手工更新 **debian/patches/series** 檔案
- **dquilt** 命令
  - 參見 Section 3.4
  - 基本的便利方案
  - 會以合適方式生成 **.pc/** 目錄及其中的資料
  - 會修改上游原始碼樹
- “**dpkg-source --commit**”命令
  - 參見 Section 4.8.3
  - 更新、更優雅一些的方案
  - 會以合適方式生成 **.pc/** 目錄及其中的資料
  - 會修改上游原始碼樹
- 由 **dpkg-buildpackage** 自動生成補丁

  - 參見 Section 5.14
  - 在 **debian/source/local-options** 檔案中添加 **single-debian-patch** 這一行
  - 設定 **debian/source/local-patch-header** 文件
  - 不涉及 **.pc/** 目錄的問題
  - 在 Debian 分支中（常見為 **master** 分支）存儲經過修改的上游原始碼樹

- **gbp-pq** 命令
  - 配合 **git-buildpackage** 工具的基本 **git** 工作流
  - 不涉及 **.pc/** 目錄的問題
  - 在可丟棄分支上儲存經過修改的上游原始碼樹（**patch-queue/master**）
  - 在 Debian 分支中（常見為 **master** 分支）存儲未經修改的原始碼樹

- **gbp-dpm** 命令
  - 配合 **git-dpm** 套件的更細緻的 **git** 工作流
  - 不涉及 **.pc/** 目錄的問題
  - 在補丁分支中（通常命名為 **patched/**隨便啥名字）儲存經過修改的上游原始碼樹
  - 在 Debian 分支中（通常命名為 **master/**隨便啥名字）儲存未經修改的上游原始碼樹

無論這些補丁的來源如何，都建議使用兼容於 DEP-3 的頭部資訊對其進行標記。

Tip

dgit 套件提供了另外一套直接使用 git 集成操作 Debian 套件倉庫的工具。

5.8.1 dpkg-source -x

“**dpkg-source -x**”命令可以對 Debian 原始碼包進行解壓縮。

該命令通常會將 **debian/patches/** 目錄內的補丁應用在原始碼樹中，並將補丁狀態記錄在 **.pc/** 目錄中。

如果您想保持原始碼樹不做修改（例如，為了在 Section 5.13 中繼續使用），請在命令列中使用 **--skip-patches** 選項。

5.8.2 dquilt 和 dpkg-source

在 **dpkg-source** 工具 1.16.1 版本之前，該工具還未提供 **--commit** 選項對應的功能，此時需要 **quilt** 命令（或者對它的封裝，**dquilt** 命令）來管理 **debian/patches/** 目錄中的 **-p1** 等級的補丁。

在使用 **dpkg-source** 命令時，補丁應當能夠乾淨地進行應用。因此在補丁行數出現偏移或者其它情況出現時，您不能直接將舊補丁原封不動地複製到新版上游釋出對應打包版本的目錄中。

與此相對的是 **dquilt** 命令（參見 Section 3.4）對補丁的處理更加寬容。您可以呼叫 **dquilt** 命令對補丁進行正常化。

```
$ while dquilt push; do dquilt refresh ; done
$ dquilt pop -a
```

使用 **dpkg-source** 命令比起使用 **dquilt** 命令來說存在一大優勢：**dquilt** 命令無法自動處理二進位制檔案出現變更的情況，而 **dpkg-source** 命令能夠探測出現內容變動的二進位制檔案，並將其列入 **debian/source/include-binaries** 檔案以便在 Debian 打包用壓縮包中將檔案囊括其中。

5.9 debian/upstream/signing-key.asc

某些套件由 GPG 金鑰進行了簽名。

例如，GNU hello 可使用 HTTP 協議從 https://ftp.gnu.org/gnu/hello/ 下載。它含有以下檔案：

- **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。

```
$ 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
```

如果您從郵件列表獲知上游維護者所使用的 GPG 公鑰資訊，請將它作為 **debian/upstream/signing-key.asc** 檔案進行儲存。否則，請使用 hkp 公鑰伺服器並經由您的信任網進行驗證。

```
$ 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
```

> Tip
>
> 如果您的網路環境阻擋了對 HKP **11371** 埠的連線，請考慮使用 “**hkp://keyserver.ubuntu.com:80**”。

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。

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

之後，應相應地在 **debian/watch** 檔案中做如下的修改。

```
version=4
pgpsigurlmangle=s/$/.sig/  https://ftp.gnu.org/gnu/hello/ hello-(\d[\d.]*)\.tar ↩
    \.(?:gz|bz2|xz)
```

現在 **uscan** 命令會在掃描時自動使用 GPG 籤名驗證上游原始碼套件的真實性。

5.10 debian/watch 和 DFSG

Debian 嚴肅地對待軟體自由，遵循 Debian 自由軟體指導方針（DFSG）。

在使用 **uscan** 命令來更新 Debian 打包所用程式碼時，上游原始碼套件（tarball）中不符合Debian 自由軟體指導方針（DFSG）的部分可以利用該工具簡單地進行移除。

- 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）。
  - 作為替代方案，您也可以使用“**gbp import-orig --uscan --pristine-tar**”命令。
- 最後得到 tarball 的版本編號會附加一個額外的字尾 **+dfsg**。

5.11 其它 debian/* 檔案

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。

> Tip
>
> 
>
> 請檢查 **debhelper**(7) 的內容以瞭解當前可用的 **dh_*** 命令列表。

這些 **debian/***binarypackage***.*** 的檔案提供了設定檔案安裝路徑的強大功能。即使上游原始碼沒有構建系統，這個軟體依然可以利用這裡提到的這些文件來進行打包。請參考 Section 8.2 的範例。

下面列表中出現的“-x[1234]”上標指示了 **debmake -x** 選項生成對應模板檔案所需要的最小值。請參考 **debmake**(1) 以瞭解詳情。

下面按照字母表順序列出一些值得注意的可選配置檔案。

binarypackage*.bug-control** [-x3] 將安裝至 *binarypackage* 套件的 **usr/share/bug/binarypackage***/control** 位置。另請參考 Section 5.24。

binarypackage*.bug-presubj** [-x3] 將安裝至 *binarypackage* 套件的 **usr/share/bug/binarypackage***/presubj** 位置。另請參考 Section 5.24。

binarypackage*.bug-script** [-x3] 將安裝至 *binarypackage* 套件的 **usr/share/bug/binarypackage* or **usr/share/bug/***binarypackag* 位置。另請參考 Section 5.24。

***binarypackage*.bash-completion** 列出需要安裝的 `bash` 補全指令碼。

需要在構建環境和使用者環境內均安裝 `bash-completion` 套件。

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 [-x3] Preciously, this set the **debhelper** compatibility level.

Now, use `Build-Depends: debhelper-compat (= 13)` in **debian/control** to specify the compatibility level.

另請參考 **debhelper**(8) 中“COMPATIBILITY LEVELS”一節。

***binarypackage*.conffile** 如果相容等級大於 3（“compat >= 3”），您沒有建立該檔案的必要，因為所有 **etc/** 目錄下的檔案都是配置檔案（conffiles）。

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** 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 Section 5.19。

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 安裝至 *binarypackage* 包內的 **etc/cron/monthly**/*binarypackage* 檔案。

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** -x3 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** -x2 作為 *binarypackage* 包中的 **doc-base** 控制檔案進行安裝。

參見 **dh_installdocs**(1) 和 **doc-base** 套件提供的 Debian doc-base 手冊。

***binarypackage*.docs** -x2 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/compat**/*binarypack* 檔案。

參見 **dh_installemacsen**(1)。

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

參見 **dh_installemacsen**(1)。

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

參見 **dh_installemacsen**(1)。

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

參見 **dh_installemacsen**(1)。

***binarypackage*.examples** -x2 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 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** -x2 列出要安裝至 *binarypackage* 包中的 info 檔案。

參見 **dh_installinfo**(1)。

***binarypackage*.init** -x3 安裝至 *binarypackage* 包中的 **etc/init.d**/*binarypackage* 檔案。

參見 **dh_installinit**(1)。

***binarypackage*.install** -x2 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)。

license-examples/* -x4 這是 **debmake** 命令生成的版權宣告檔案示例，請用它們作為 **debian/copyright** 檔案的參考。

請在最終工作成果中刪除這些檔案。

***binarypackage*.links** -x2 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。

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 的政策要求套件為其包含的每個程式、工具和函式同時提供一份相關的手冊頁。手冊頁使用 **nroff**(1) 語法寫成。

如果您不熟悉如何編寫使用者手冊頁，請以 **manpage.asciidoc** 或 **manpage.1** 為起點。

***binarypackage*.manpages** -x2 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 決定“Debian 應該在合適的情況下使用 **.desktop** 檔案”。

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 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。

參見 **dpkg-source**(1)、Section 3.4 和 Section 4.8。

debmake 預設不會生成補丁檔案。

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

***binarypackage*.preinst** -x2 **, *binarypackage*.postinst** -x2 **, *binarypackage*.prerm** -x2 **, *binarypackage*.postrm** -x2 這些維護者指令碼將安裝至套件的 **DEBIAN** 目錄下。

在這些指令碼中，**#DEBHELPER#** 記號將由其它 **debhelper** 命令進行處理，將其替換為相應的 shell 指令碼片段。

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***/READM** 位置。

參見 **dh_installdocs**(1)。

該檔案提供了針對該 Debian 套件的資訊。

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***.se** 位置。

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

source/format -x1 Debian 套件格式。

- 使用“**3.0 (quilt)**”以製作這個非原生套件（推薦）
- 使用“**3.0 (native)**”以製作這個原生套件

參見 **dpkg-source**(1) 的“原始碼套件格式”一節。

source/lintian-overrides 或 **source.lintian-overrides** [-x3] 這些檔案不會最終被安裝，但 **lintian** 會對它們進行掃描以提供原始碼套件的 override 資訊。

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。

參見 **dpkg-source**(1) 中的“檔案格式”一節。

source/local-patch-header 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。

+ 參見 **dpkg-source**(1) 的“檔案格式”一節。

***binarypackage*.symbols** [-x2] 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 Section 5.18.1。

***binarypackage*.templates** 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 Section 5.19。

tests/control 這是一個 RFC822 格式的測試元資料檔案，定義在 DEP-8。參見 **autopkgtest**(1) 和 Section 5.22。

TODO 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.Debian** 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** [-x3] 如果該檔案存在，它將被安裝至套件構建目錄的 etc/init/package.conf 位置。（已棄用）

參見 **dh_installinit**(1) 和 Section 8.1。

watch [-x1] 用於下載最新上游版本的 **uscan** 命令的控制檔案。

該控制檔案也可配置以使用其 GPG 簽名自動驗證上游 tarball 的真實性（參見 Section 5.9）。

參見 Section 5.10 和 **uscan**(1)。

這裡給出針對上面列表中資訊的一些額外提醒。

- 對只生成一個二進位制包的情況，列表檔名中的 *binarypackage***.** 這一部分可以不出現。
- 對有多個二進位制包的原始碼套件，一個缺少檔名裡 *binarypackage***.** 部分的配置檔案，會被應用於 **debian/control** 裡列出的第一個二進位制包。
- 在生成多個二進位制包的情況下，各個二進位制包可以分別指定配置檔案；只需在其對應配置檔案的檔名之前加上它們各自對應的包名即可，如 *package-1***.install**、*package-2***.install** 等等。
- **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。
- **debmake** 命令生成的帶額外 **.ex** 字尾名的配置檔案必須在移除這個多餘字尾名後才能發揮作用。
- 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。

5.12 Debian 打包的定製化

我們來重新歸納一下 Debian 打包定製化的相關內容。

所有對 Debian 套件進行定製化的資料都存在於 **debian/** 目錄中。我們在 Section 4.6這裡給出了一個簡單的例子。通常情況下，定製化會涉及以下各個專案：

- Debian 套件構建系統可以經由 **debian/rules** 檔案進行定製（參見 Section 5.4.3）。
- 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 Section 5.11）。

如果以上提到的手段仍然不足以製作滿足要求的 Debian 套件的話，對上游原始碼的修改應當使用 **-p1** 補丁檔案存放在打包原始碼樹 **debian/patches/** 目錄下。這些補丁將按照 **debian/patches/series** 檔案所規定的順序在構建套件之前應用（參見 Section 5.8）。Section 4.8 這裡給出了一些簡單的例子。

您應當以引入最少修改的方式解決打包中出現的根本問題。所生成的套件應當考慮到未來的更新需求並有一定的健壯性。

Note

如果補丁對上游有所幫助的話，也請將解決根本問題的補丁反饋給上游作者和維護者。

5.13 在版本控制系統中進行記錄（標準）

通常情況下，Debian 打包活動使用 Git 作為版本控制系統（VCS）進行記錄；通常會用到下列的分支。

- **master** 分支
  - 記錄用於 Debian 打包的原始碼樹。
  - 原始碼樹的上游部分將照原樣記錄，不做修改。
  - Debian 打包中需要對上游原始碼所作的修改記錄在 **debian/patches/** 目錄中，以 **-p1** 等級的補丁形式存在。
- **upstream** 分支
  - 記錄從上游釋出的 tarball（原始碼壓縮檔案）解壓縮所得到的原始碼樹。

Tip

新增一個 **.gitignore** 檔案並將 **.pc** 檔案列入其中也是一個好主意。

Tip

可以在 **debian/source/local-options** 檔案中新增 **unapply-patches** 和 **abort-on-upstream-changes** 這兩行以保持上游原始碼處於未修改狀態。

Tip

您也可以使用除 **upstream** 分支以外其它名稱的分支追蹤上游版本控制資料以方便揀選補丁。

5.14 在版本控制系統中進行記錄（備選方案）

您也可以選擇不建立 **-p1** 等級的補丁。這時，您可以使用下列分支來記錄 Debian 打包活動。

- **master** 分支
  - 記錄用於 Debian 打包的原始碼樹。
  - 原始碼樹的上游部分在應用了為 Debian 打包所作的修改之後進行記錄。
- **upstream** 分支
  - 記錄從上游釋出的 tarball（原始碼壓縮檔案）解壓縮所得到的原始碼樹。

如下在 **debian/** 目錄下額外新增一些檔案即可達到目的。

```
$ tar -xvzf <package-version>.tar.gz
$ ln -sf <package_version>.orig.tar.gz
$ cd <package-version>/
... hack...hack...
$ echo "single-debian-patch" >> debian/source/local-options
$ cat >debian/source/local-patch-header <<END
This patch contains all the Debian-specific changes mixed
together. To review them separately, please inspect the VCS
history at https://git.debian.org/?=collab-maint/foo.git.
```

如此可讓 Debian 打包過程（**dpkg-buildpackage**、**debuild** 等）所呼叫的 **dpkg-source** 命令自動生成一個 **-p1** 等級的補丁檔案 **debian/patches/debian-changes**。

Tip

這種做法可以應用在任何版本控制工具中。這麼做會把所有修改合併到同一個補丁檔案中而丟失其開發歷史，因此請務必保持版本控制系統的資料公開可見。

Tip

debian/source/local-options 和 **debian/source/local-patch-header** 檔案只用於在版本控制系統中記錄資訊。它們不應包含在 Debian 原始碼套件中。

5.15 構建套件時排除不必要的內容

在某些情況下，直接使用自動生成的 Debian 原始碼套件會引入不必要的一些內容。

- 上游原始碼樹可能由某個版本控制系統進行管理。直接從這個原始碼樹進行構建時，所生成的 Debian 原始碼套件會包含來自版本控制系統的多餘檔案。

- 上游原始碼樹可能包含了一些自動生成的檔案。當從這個原始碼樹重新構建套件時，所生成的 Debian 原始碼套件會包含這些自動生成的不必要的檔案。

通常情況下，Section 3.5 中設定的用於 **dpkg-source** 命令的 **-i** 和 **-I** 選項可以避免這些問題。這裡 **-i** 針對非原生套件而 **-I** 則針對原生套件。請參見 **dpkg-source**(1) 和“**dpkg-source --help**”的輸出。

以下幾種方法均可避免引入不必要的內容。

5.15.1 使用 **debian/rules clean** 進行修復

含有多餘檔案的問題可以使用“**debian/rules clean**”這個 Makefile 目標的呼叫來解決，只需在該目標內刪除檔案即可。它也能處理自動生成的檔案。

Note

執行 **dpkg-buildpackage** 命令時，它會在“**dpkg-source --build**”命令之前被調用“**debian/rules clean**”目標，而“**dpkg-source --build**”命令會忽略被刪除的檔案。

5.15.2 使用版本控制系統修復

含有多餘檔案的問題還可以使用版本控制系統修復；具體來說，可以在首次構建之前將原始碼樹提交到版本控制系統中。

您可以在第二次構建套件之前恢復最初的原始碼樹。例如：

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

這裡工作的原理是 **dpkg-source** 命令會忽略原始碼樹中典型的版本控制系統相關的檔案，相關的設定可以在 Section 3.5 的 **DEBUILD_DPKG_BUILDPACKAGE_OPTS** 設定中找到。

Tip

如果原始碼樹未受版本控制系統管理，您可以在第一次構建之前執行“**git init; git add -A .; git commit**”來初始化。

5.15.3 使用 **extend-diff-ignore** 修復

這種做法僅適合非原生套件。

含有多餘檔案的問題可以使用在 **debian/source/options** 檔案中新增忽略資訊的方式解決，令編譯系統忽略多餘的檔案；具體配置語法為新增 **extend-diff-ignore**=…”一行內容。

如需排除 **config.sub**、**config.guess** 和 **Makefile** 檔案：

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

Note

即使您無法刪除檔案，這種做法總可以正常工作。您無需在每次構建之前手動刪除檔案並手動進行恢復。

Tip

如果您轉而使用 **debian/source/local-options** 檔案，您可以在生成的原始碼套件中隱藏該項設定。這種做法在本地非標準版本控制系統和您的打包工作有衝突時可能有用。

5.15.4 使用 **tar-ignore** 修復

這個方法只適用於原生套件。

您可以使用這種做法在生成的原始碼套件中排除某些檔案；只需在 **debian/source/options** 檔案或者 **debian/source/local-options** 檔案中新增含有萬用字元的“**tar-ignore=⋯**”一行內容即可。

Note

例如，如果您的原生套件的原始碼套件使用了一些具有 **.o** 副檔名的檔案作為測試資料的話，Section 3.5 的預設設定就過於激進了，這些檔案會被當作多餘的檔案預設自動排除。如需解決這個問題，您可以在 Section 3.5 中的 **DEBUILD_DPKG_BUILDPACKAGE_OPTS** 引數中移除 **-I** 選項，同時在每個套件的 **debian/source/local-options** 檔案中新增“**tar-ignore=⋯**”的配置行。

5.16 上游構建系統

上游的構建系統設計為經過數個步驟以從原始碼發行檔案得到並在系統中安裝所生成的二進位制檔案。

Tip

在嘗試製作 Debian 套件之前，您應當熟悉瞭解上游原始碼所使用的構建系統並嘗試構建軟體。

5.16.1 Autotools

使用 Autotools（**autoconf** + **automake**）包括四個步驟。

1. 設定構建系統（“**vim configure.ac Makefile.am**”和“**autoreconf -ivf**”）
2. 配置構建系統（“**./configure**”）
3. 構建原始碼樹（“**make**”）
4. 安裝二進位制檔案（“**make install**”）

第一步通常由上游維護者完成並使用“**make dist**”命令生成上游原始碼壓縮包（tarball）。（所生成的原始碼壓縮包不僅含有原始的版本控制系統中的檔案，也含有其它生成的檔案。）

套件維護者至少要處理第二步到第四步的工作。可以在 **debian/rules** 檔案中使用“**dh $@ --with autotools-dev**”的命令以自動處理這些步驟。

套件維護者也可以想要處理第一步到第四步所有的工作。這時，可以在 **debian/rules** 檔案中使用“**dh $@ --with autoreconf**”命令。這樣會將所有自動生成的檔案更新到最新的版本，通常可以提供對新架構的更好支援。

對於使用 **compat** level（相容等級）**10** 或更高等級的原始碼套件，使用最簡單的“**dh $@**”而不帶“**--with autoreconf**”選項已可自動處理上述第一步到第四步全部內容。

如果您想進一步學習 Autotools，請參考：

- GNU Automake 文件
- GNU Autoconf 文件
- Autotools 教材
- 對 autotools 的介紹（autoconf、automake 和 libtool）
- Autotools 流言終結者

5.16.2 CMake

使用 CMake 通常也包含四個步驟。

1. 設定構建系統（“**vim CMakeLists.txt config.h.in**”）
2. 配置構建系統（“**cmake**”）
3. 構建原始碼樹（“**make**”）
4. 安裝二進位制檔案（“**make install**”）

上游原始碼套件（tarball）不包含自動生成的檔案，通常是在第一步之後直接由 **tar** 命令打包生成。
套件維護者需要處理第二步到第四步的工作。
如果您想進一步學習 CMake，請參考：

- CMake
- CMake 教材

5.16.3 Python distutils

使用 Python distutils 通常包含三個步驟。

1. 設定並配置構建系統（“**vim setup.py**”）
2. 構建原始碼樹（“**python setup.py build**”）
3. 安裝二進位制檔案（“**python setup.py install**”）

上游維護者通常會處理好第一步並使用“**python setup.py sdist**”命令構建好上游原始碼套件並進行發行。
套件維護者需要處理第二步工作。在 **jessie** 釋出後，打包時只需要在 **debian/rules** 檔案中使用最簡單的“**dh $@**”命令。
其它構建系統，如 CMake，其使用方法和 Python 這裡的情況都很類似。
要更多瞭解 Python3 和 **distutils** 請參見：

- Python3
- distutils

5.17 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

- 《Debian 開發者參考》中的 6.7.9. 除錯套件的最佳實踐。
- “Debugging with gdb”中的 18.2 單獨檔案中的除錯資訊
- **dh_strip**(1)
- **strip**(1)
- **readelf**(1)

- **objcopy**(1)
- Debian 維基 除錯套件
- Debian 維基 自動除錯套件
- Debian debian-devel 列表釋出的郵件：自動除錯套件狀態 (2015-08-15)

5.17.1 新的 **-dbgsym** 套件（**Stretch 9.0** 或更新）

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。

在 Stretch 9.0 的釋出之後，如果在 **debian/control** 檔案中從未定義過任何 **-dbg** 軟體包，則不需任何特殊配置。

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

如果在 **debian/control** 檔案中曾經定義過 **-dbg** 套件，在 Stretch 9.0 釋出之後對舊套件更新需要進行額外的處理。

- 在 **debian/control** 檔案中刪除 **-dbg** 套件定義的部分。
- 將 **debian/rules** 檔案中“**dh_strip --dbg-package=***package*”這部分替換為“**dh_strip --dbgsym-migration=***package*”以避免自動產生的除錯套件與（現在過時的）**-dbg** 檔案產生衝突。參見 **dh_strip**(1)。
- Remove **debian/compat**.
- Set the **Build-Depends** to **debhelper-compat (>=13)** while removing **Build-Depends** to **debhelper** in **debian/control**.

5.18 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 Section 5.5.1.3 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 Section 5.20）。
- Debian 鼓勵在共享程式庫中使用帶版本的符號（見 Section 5.18.1）。
- 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”
- 《Debian 開發者參考》中的 6.7.2. 軟體程式庫

如需研究其歷史背景，請參見：

- 逃離依賴地獄 [6]
  - 該文件鼓勵在共享程式庫中使用帶版本的符號。
- Debian 程式庫打包指南 [7]
  - 也請閱讀這裡的公告後面跟隨的討論串。

[6]該文件是在 **symbols** 檔案被引入之前寫成的。

[7]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。

5.18.1 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。

- 從前一個二進位制套件中使用“**dpkg-deb -e**”命令解壓縮得到舊有的 **DEBIAN/symbols** 檔案。
  - 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：
    - 使用“**dpkg-deb -e**”命令解壓得到更新的 **DEBIAN/symbols** 檔案。
    - 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)
- **dpkg-gensymbols**(1)
- **dpkg-shlibdeps**(1)
- **deb-symbols**(5)

您也應當檢視：

- Debian 維基 使用符號檔案
- Debian 維基 專案/改進的 DpkgShlibdeps
- Debian kde 團隊 處理 symbols 檔案
- Section 8.11
- Section 8.12

Tip

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.

5.18.2 程式庫變遷

當您打包新版本的程式庫套件而且此次更新影響到其它的套件時，您必須對 **release.debian.org** 偽套件提交一個變遷 bug 報告並附帶一個ben 檔案；您可以使用 **reportbug** 工具進行提交。提交後，請等待發行團隊的稽核批准方可進行下一步。

發行團隊提供了變遷追蹤系統。參見 變遷（Transition）。

Caution

請確保您按照 Section 5.5.1.3 的描述正確地對二進位制套件進行了重命名。

5.19 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。
- 使用選單介面進行互動式配置（對話方塊（**dialog**）、**gnome**、**kde** 等等）
  - 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。
- 套件配置指令碼
  - **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”.

5.20 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）
  - 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：

- Section 5.4.4
- **dpkg-architecture**(1)
- Section 5.5.1.1
- Section 5.5.1.2

5.20.1 多架構程式庫路徑

Debian 政策要求遵守檔案系統層級標準。其中 /usr/lib：程式和套件的程式庫 宣告“**/usr/lib** 包含目標檔案、程式庫和其它不應由使用者或 shell 指令碼直接呼叫的內部二進位制檔案。”

Debian 在檔案系統層級標準的基礎上新增一項例外，即使用 **/usr/lib/<triplet>/** 而非 **/usr/lib<qual>/**（例如，**/lib32/** 和 **/lib64/**）以對多架構程式庫提供支持。

Table 5.1 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。

- 如果在 **debian/rules** 檔案中設定了 **override_dh_auto_configure** 目標且其中手動呼叫了“**./configure**”命令，請確保將其替換為“**dh_auto_configure --**”，這樣可以將安裝路徑從 **/usr/lib/** 替換為 **/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/*/**。

所有啓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。

Note

./configure 的 **--libexecdir** 選項指定了由程式而非使用者所使用的可執行檔案的預設安裝路徑。其 Autotools 的預設值為 **/usr/libexec/** 但在未啟用多架構特性的 Debian 系統上其預設值是 **/usr/lib/**。如果這些可執行程式屬於被標記為“Multi-arch: foreign”的套件，最好還是使用例如 **/usr/lib/** 或者 **/usr/lib/**套件名這樣的路徑而非使用 **dh_auto_configure** 設定的 **/usr/lib/<triplet>/** 路徑。GNU 程式設計規範：7.2.5 用於安裝目錄的變數 對 **libexecdir** 的描述是“**libexecdir** 的定義對所有套件相同，所以您應當將您的資料安裝在其下的一個子目錄中。大多數套件將資料安裝至 **$(libexecdir)/package-name/** 之中……”（在與 Debian 政策不衝突的前提下，遵守 GNU 的標準總是更好的。）

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。

5.20.2 多架構標頭檔案路徑

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 5.2 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。[8]
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。

5.20.3 多架構支援下的 *.pc 檔案路徑

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 5.3 *.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/**

5.21 編譯强化

自 Debian **jessie**（8.0 開始）的編譯器强化支援要求我們在打包時加以注意。
您應當詳細閱讀下列參考內容。

- Debian 維基 Hardening（强化）
- Debian 維基 Hardening Walkthrough（强化指南）

debmake 命令會對 **debian/rules** 檔案中按需新增 **DEB_BUILD_MAINT_OPTIONS**、**DEB_CFLAGS_MAINT_APPE** 和 **DEB_LDFLAGS_MAINT_APPEND** 的專案（參見 Chapter 4 和 **dpkg-buildflags**(1)）。

5.22 持續整合

DEP-8 定義了 **debian/tests/control** 檔案的格式，它是 RFC822 風格的測試元資料檔案，用於 Debian 套件的持續整合（CI）。
它在完成構建包含 **debian/tests/control** 文件的原始碼套件、得到二進位制包之後發揮作用。在執行 **autopkgtest** 命令時，所生成的二進位制套件會根據這個檔案在虛擬環境中自動進行安裝和測試。
請參考 **/usr/share/doc/autopkgtest/** 目錄下的文件和《Debian 打包指導”中的 3. autopkgtest: 套件的自動測試。
您可以在 Debian 系統上探索使用不同的持續整合系統。

[8]這個路徑和 FHS 相容。檔案系統層級標準：/usr/lib：程式和套件的程式庫 稱“應用程式可以使用 **/usr/lib** 下的一個子目錄。如果一個應用程式使用一個子目錄，所有由此程式所使用的架構相關資料均須放置於該子目錄下。”

- **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

5.23 自主生成 (Bootstrapping)

Debian 關心對新硬體架構的移植工作。新架構的移植工作對自主生成（bootstrapping）操作有所要求，以完成對初始最小本地構建系統的交叉編譯。為了在自主生成時避免構建依賴成環的問題，需要使用 配置型別（profile） 的構建功能特性來縮減所需構建依賴。

Tip

如果一個核心套件 `foo` 構建時依賴於 `bar` 套件，但後者會引入一長串構建依賴鏈而且 `bar` 僅在 `foo` 的 **test** 目標中使用（即僅用於構建後測試），那麼您可以安全地在 `foo` 套件的 **Build-depends** 一欄中將 `bar` 標記為 **<!nocheck>** 以規避構建依賴環。

5.24 錯誤報告

reportbug 命令用於提交 *binarypackage* 套件的錯誤報告；**usr/share/bug/***binarypackage/* 可以對針對該軟體所提交報告的內容進行設定。

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。

參見 **dh_bugfiles**(1) 和 為開發者提供的 reportbug 功能特性

Tip

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 6

debmake 選項

這裡提供 **debmake** 命令的一些重要選項。

6.1 快捷選項（**-a**，**-i**）

debmake 命令提供了兩個快捷選項。

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

前文中 Chapter 4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
```

Tip

-a 選項也可以使用“**https://www.example.org/DL/package-1.0.tar.gz**”這樣的 URL。

Tip

-a 選項也可以使用“**https://arm.koji.fedoraproject.org/packages/ibus/1.5.7/3.fc21/src/ibus-1.5.7-3.fc21.src.rpm**”這樣的 URL。

6.1.1 Python 模組

您可以使用以原始碼壓縮包（tarball）形式提供的 *pythonmodule-1.0***.tar.gz** Python 模組包直接生成一個獨立二進位制 Debian 套件。這裡的 **-b** 選項可以指定套件型別為 **python**，**-s** 選項可以從上游壓縮包中提取並複製軟體包描述內容至指定位置。

```
$ debmake -s -b':python' -a pythonmodule-1.0.tar.gz -i debuild
```

對其他支援 **-b** 選項的解釋式程式語言，請給 **-b** 選項指定恰當的 *type* 。

對沒有 **-b** 選線支援的其它解釋性語言，您可以指定其為指令碼型別（**script** 型別）並調整 **debian/control** 檔案而將直譯器對應的軟體包新增為套件依賴。

6.2 上游快照（-d，-t）

This packaging scheme is good for the git repository organized as described in **gbp-buildpackage**(7) which uses the master, upstream, and pristine-tar branches.

如果上游套件支援“**make dist**”或者等效的目標，您可以使用 **-d** 選項從上游原始碼樹版本控制系統中得到上游程式碼的快照。

```
$ 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
```

除非您明確使用 **-u** 選項或者在 **debian/changelog** 檔案中提前指定好版本號，預設情況下快照生成的上游版本號將應用協調世界時的日期和時間使用 **0~%y%m%d%H%M** 格式生成，例如 *0~1403012359*。

如果上游版本控制系統位於 套件名/ 目錄而非任意的 上游版本控制系統/ 目錄，引數中的“**-p** 套件名”這部分可以跳過。

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
```

這裡的使用“**debmake -t -i debuild**”命令構建非原生 Debian 二進位制包的流程可以看做擬似原生套件型別進行構建，因為其打包場景和原生 Debian 套件不使用上游原始碼套件直接使用 **debuild** 命令打包很類似。

使用非原生的（**non-native**）套件有助於簡化與下游發行版（如 Ubuntu）之間在缺陷、問題修復上的溝通。

6.3 Upstream snapshot (alternative `git deborig` approach)

This packaging scheme is good for the git repository organized as described in **dgit-maint-merge**(7) which uses the master branch only.

You can create the upstream tarball and Debian package simply as follows.

```
$ cd /path/to/upstream-git
$ git deborig -f HEAD
$ pdebuild
```

This scheme can be applied to the **quasi-native** Debian package scheme when `debian/changelog` contains the non-native version number with revision like `0.16-1`.

For `-1` revision, this use of `git-deborig`(1) as above is how this `debmake-doc` package generates the upstream tarball. For source format `3.0 (quilt)`, files under `debian/` directory in the upstream tarball has no negatives. You may override the lintian warning.

For `-2`, `-3`, ⋯revisions, you need to fetch and use the uploaded upstream tarball instead. For this, `origtargz`(1) may be handy.

6.4 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。

6.5 debmake -k

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
```

“**debmake -k**”命令可以完整解析 **debian/copyright** 並將當前套件中的所有非二進位制文件內含的許可證資訊按照最後一項匹配的方式與 **debian/copyright** 檔案中的資訊進行對比。

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。

Tip

對所有上游釋出新版本的情況，執行“**debmake -k**”可以確保 **debian/copyright** 檔案一直處於最新狀態。

6.6 debmake -j

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 Chapter 4）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。

Note

binarypackage_version-revision_arch.**deb** 命令的 **-j** 選項會呼叫 **dpkg-depcheck**(1) 以在 **strace**(1) 之下執行 **debian/rules**，從而獲得程式庫依賴資訊。然而，這樣操作的執行速度極慢。如果您由其他途徑獲知了套件的程式庫套件依賴資訊，例如外部的 SPEC 文件等等，您可以直接執行“**debmake** …”命令而不帶 **-j** 選項並執行“**debian/rules install**”命令以檢查所生成檔案的安裝路徑。

6.7 debmake -x

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

- 請參見 Section 8.1 以瞭解與揀選使用模板檔案的方式。

Note

debmake 命令不會修改任何已存在的配置文件。

6.8 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。

6.9 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。

Chapter 7

小技巧

這裡有一些關於 Debian 打包的值得注意的小技巧。

7.1 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。

7.2 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
```

7.3 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
```

7.4 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
```

Note

這是一個非常慢的過程，因為它需要查詢遠端 APT 套件倉庫。

7.5 debsign

完成套件的測試後，您可以使用 **debsign** 命令對其進行簽名。

```
$ debsign package_version-rev_arch.changes
```

7.6 dput

使用 **debsign** 命令對包進行簽名後，您可以使用 **dput** 命令上傳 Debian 源和二進位制包的檔案集。

```
$ dput package_version-rev_arch.changes
```

7.7 bts

上傳套件後，您將收到錯誤報告。如《Debian 開發者參考》5.8. 處理缺陷 中所述，正確地管理這些錯誤是套件維護者的一項重要職責。

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
```

7.8 git-buildpackage

git-buildpackage 套件提供了許多命令來使用 git 倉庫自動打包。

- **gbp import-dsc**：在 git 倉庫中匯入先前的 Debian 原始碼套件。
- **gbp import-orig**：在 git 倉庫中匯入新的上遊原始碼。
  - **git import-orig** 命令的 **--pristine-tar** 選項允許將上游原始碼套件儲存在同一個 git 倉庫中。
  - 將 **--uscan** 選項作為 **gbp import-orig** 命令的最後一個引數會允許下載上游原始碼並提交到 git 倉庫中。
- **gbp dch**：從 git 提交資訊中生成 Debian 變更資訊（changelog）。
- **gbp buildpackage**：從 git 倉庫中構建 Debian 二進位制包。
- **gbp pull**：從遠端倉庫中安全更新 **debian**, **upstream** and **pristine-tar** 分支。
- **git-pbuilder**：使用 **pbuilder** 套件從 git 倉庫構建 Debian 二進位制套件。
  - 使用 **cowbuilder** 套件作為後端。
- **gbp pq**、**git-dpm** 或 **quilt**（或者其別名 **dquilt**）命令用於管理相容 quilt 的補丁。
  - **dquilt** 命令是學起來最簡單的，它只要求您使用 **git** 命令手動提交最後的檔案至 **master** 分支。
  - “**gbp pq**” 命令提供了等效的補丁集管理功能，而不需要使用 **dquilt** 並且使用 git 的揀選（cherry-pick）功能，簡化了包含上游 git 倉庫修改的操作流程。
  - “**git dpm**” 命令提供了比 “**gbp pq**” 更强大的功能。

使用 **git-buildpackage** 套件來管理套件歷史，正成為絕大多數 Debian 維護者的實踐標準。
參見：

- 使用 git-buildpackage 構建 Debian 套件
- https://wiki.debian.org/GitPackagingWorkflow
- https://wiki.debian.org/GitPackagingWorkflow/DebConf11BOF
- https://raphaelhertzog.com/2010/11/18/4-tips-to-maintain-a-3-0-quilt-debian-source-package-in-a-vcs/

- **systemd** 打包實踐文件在 從原始碼構建。

Tip

放鬆。您並不需要使用全部的打包工具，您只需要使用您所需要的那個就行。

7.8.1 gbp import-dscs --debsnap

對於記錄在 snapshot.debian.org 歸檔中的名為 <*source-package*> 的 Debian 原始碼套件，可以生成包含所有 Debian 版本歷史的初始 git 儲存庫，如下所示。

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

7.9 上游 git 倉庫

對於使用 **git-buildpackage** 打包的 Debian 套件，遠端儲存庫 **origin** 上的 **upstream** 分支通常用於追蹤已釋出的上游原始碼的內容。

也可以通過將其遠端儲存庫命名為 **upstream** 而不是預設的 **origin** 來追蹤上游 git 倉庫。然後，您可以通過使用 **gitk** 命令和 **gbp-pq** 命令進行挑選，輕鬆地將最近的上游更改新增到 Debian 修訂版中。

Tip

“**gbp import-orig --upstream-vcs-tag**”命令可以通過使用上游 git 倉庫中的指定標籤在 **upstream** 分支上建立一個合併提交的方式來生成乾淨的打包歷史資訊。

Caution

已釋出的上游原始碼的內容可能與上游 git 儲存庫的相應內容並不完全匹配。它可能包含一些自動生成的檔案或遺漏一些檔案。（Autotools、distutils.....）

7.10 chroot

The chroot for a clean package build environment can be created and managed using the tools described in Chapter 3. [1]

以下是可用的套件構建命令的快速總結。有很多方法可以做同樣的事情。

- **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）
- **pbuilder** = Debian chroot 環境工具的核心
- **pdebuild** = **pbuilder** + **dpkg-buildpackage** （在 chroot 環境下構建）
- **cowbuilder** = 提升 **pbuilder** 執行的速度
- **git-pbuilder** = **pdebuild** 的易於使用的命令列語法（由 **gbp buildpackge** 使用）

[1] The **git-pbuilder** style organization is deployed here. See https://wiki.debian.org/git-pbuilder . Be careful since many HOWTOs use different organization.

- **gbp** = 在 git 下管理 Debian 原始碼
- **gbp buildpackge** = **pbuilder** + **dpkg-buildpackage** + **gbp**

可以根據如下方式使用乾淨的 **sid** 版本的 chroot 環境。

- 用於 **sid** 發行版的 chroot 檔案系統建立命令
  - **pbuilder create**
  - **git-pbuilder create**
- **sid** 版本的 chroot 檔案系統的檔案路徑
  - **/var/cache/pbuilder/base.cow**
- **sid** 發行版 chroot 的套件構建命令
  - **pdebuild**
  - **git-pbuilder**
  - **gbp buildpackage**
- 更新 **sid** chroot 的命令
  - **pbuilder --update**
  - **git-pbuilder update**
- 要登入到 **sid** 修改 chroot 檔案系統的命令
  - **git-pbuilder login --save-after-login**

可以根據如下方式使用任意的 *dist* 版本環境。

- 用於 **dist** 版本的 chroot 檔案系統建立命令
  - **pbuilder create --distribution** *dist*
  - **DIST=***dist* **git-pbuilder create**
- **dist** 版本的 chroot 檔案系統的檔案路徑
  - path: **/var/cache/pbuilder/base-***dist***.cow**
- **dist** 版本 chroot 的套件構建命令
  - **pdebuild -- --basepath=/var/cache/pbuilder/base-***dist***.cow**
  - **DIST=***dist* **git-pbuilder**
  - **gbp buildpackage --git-dist=***dist*
- 更新 **dist** chroot 的命令
  - **pbuilder update --basepath=/var/cache/pbuilder/base-***dist***.cow**
  - **DIST=***dist* **git-pbuilder update**
- 登入 **dist** chroot 環境以進行修改的命令
  - **pbuilder --login --basepath=/var/cache/pbuilder/base-***dist***.cow --save-after-login**
  - **DIST=***dist* **git-pbuilder login --save-after-login**

Tip

使用這個“git-pbuilder login --save-after-login”命令，可以非常方便地建立一個包含一些新實驗套件所需的預載入包的設定環境。

Tip

如果您的舊 chroot 檔案系統缺少例如 **libeatmydata1**、**ccache** 和 **lintian** 等套件，您可能需要使用“**git-pbuilder login --save-after-login**”命令來安裝這些套件。

Tip

只需使用“**cp -a base-***dist***.cow base-***customdist***.cow**”命令即可複製 chroot 檔案系統。新的 chroot 可以以“**gbp buildpackage --git-dist=***customdist*”和“**DIST=***customdist* **git-pbuilder** …”查詢。

Tip

當需要為除 **0** 和 **1** 之外的 Debian 修訂版上傳 **orig.tar.gz** 檔案時（例如，對於安全性上傳），將 **-sa** 選項新增到 **dpkg-buildpackage**，**debuild**，**pdebuild** 和 **git-pbuilder** 命令末尾。對於“**gbp buildpackage**”命令，臨時修改 **~/.gbp.conf** 中的 **builder** 設定。

Note

本節中的描述過於簡潔，對大多數潛在的維護者都沒用。這是作者的有意為之。我們強烈建議您搜尋並閱讀與所用命令相關的所有文件。

7.11 新的 **Debian** 版本

Let’s assume that a bug report ***#bug_number*** was filed against your package, and it describes a problem that you can solve by editing the **buggy** file in the upstream source. Here’s what you need to do to create a new Debian revision of the package with the **bugname.patch** file recording the fix.

使用 **dquilt** 命令準備新的 **Debian** 套件修訂版本

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

```
$ dch -i
```

此外，如果套件是用 **git-buildpackage** 命令以其預設配置管理的 git 倉庫：
使用 **gbp-pq** 命令進行新的 **Debian** 修訂

```
$ git checkout master
$ gbp pq import
$ vim buggy
$ git add buggy
$ git commit
$ git tag pq/<newrev>
$ gbp pq export
$ gbp pq drop
$ git add debian/patches/*
$ dch -i
$ git commit -a -m "Closes: #<bug_number>"
```

請確保簡明扼要地描述修復報告錯誤的更改並通過在 **debian/changelog** 檔案中新增“**Closes: #**<*bug_number*>”來關閉這些錯誤。

Tip

在實驗時使用帶有版本字串的 **debian/changelog** 條目，例如 **1.0.1-1~rc1** 。然後，將這些更改日誌條目整理到官方套件的條目中。

7.12 新上游版本

如果 **foo** 包是以現代“**3.0 (native)**”或“**3.0 (quilt)**”格式正確打包的，則打包新的上游版本時需要將舊的 **debian/** 目錄移動到新的原始碼路徑中。這可以通過在新提取的原始碼路徑中執行“**tar -xvzf** */path/to/foo_oldversion***.debian.tar.gz**”命令來完成。[2] 當然，你還需要做一些修改。

有很多的工具可以用以處理這些情況。在使用這些軟體來更新上游版本後，請在 **debian/changelog** 檔案中簡要描述修復錯誤的上游修改，並新增“**Closes: #***bug_number*”來關閉錯誤。

7.12.1 uupdate + tarball

您可以使用來自 **uupdate** 套件中的 **uupdate** 命令來自動更新到新的上游原始碼。該命令需要舊的 Debian 原始碼套件和新的上游原始碼套件。

```
$ wget https://example.org/foo/foo-newversion.tar.gz
$ cd foo-oldversion
$ uupdate -v newversion ../foo-newversion.tar.gz
...
$ cd ../foo-newversion
$ while dquilt push; do dquilt refresh; done
$ dch
```

7.12.2 uscan

您可以使用來自 **uupdate** 套件中的 **uscan** 命令來自動更新到新的上游原始碼。該命令需要包含 **debian/watch** 檔案的舊的 Debian 原始碼套件。

```
$ cd foo-oldversion
$ uscan
...
$ while dquilt push; do dquilt refresh; done
$ dch
```

[2] 如果 **foo** 包是以舊的 **1.0** 格式打包的，則相對的，只要在新的原始碼路徑中執行“**zcat** */path/to/foo_oldversion***.diff.gz|patch -p1**”命令。

7.12.3 gbp

您可以使用來自 **git-buildpackage** 套件中的“**gbp import-orig --pristine-tar**”命令來自動更新到新的上游原始碼。該命令需要在 git 倉庫中的 Debian 原始碼和新的上游原始碼包。

```
$ ln -sf foo-newversion.tar.gz foo_newversion.orig.tar.gz
$ cd foo-vcs
$ git checkout master
$ gbp pq import
$ git checkout master
$ gbp import-orig --pristine-tar  ../foo_newversion.orig.tar.gz
...
$ gbp pq rebase
$ git checkout master
$ gbp pq export
$ gbp pq drop
$ git add debian/patches
$ dch -v <newversion>
$ git commit -a -m "Refresh patches"
```

Tip

如果上游也使用 git 倉庫，請為 **gbp import-orig** 命令加上 **--upstream-vcs-tag** 選項。

7.12.4 gbp + uscan

您可以使用來自 **git-buildpackage** 套件中的“**gbp import-orig --pristine-tar --uscan**”命令來自動更新到新的上游原始碼。該命令需要在 git 倉庫中的包含 **debian/watch** 檔案的 Debian 原始碼。

```
$ cd foo-vcs
$ git checkout master
$ gbp pq import
$ git checkout master
$ gbp import-orig --pristine-tar --uscan
...
$ gbp pq rebase
$ git checkout master
$ gbp pq export
$ gbp pq drop
$ git add debian/patches
$ dch -v <newversion>
$ git commit -a -m "Refresh patches"
```

Tip

如果上游也使用 git 倉庫，請為 **gbp import-orig** 命令加上 **--upstream-vcs-tag** 選項。

7.13 3.0 原始碼格式

更新套件的風格並不是更新套件所必須的步驟。但是，這麼做可以讓您充分利用現代 **debhelper** 和 **3.0** 原始碼格式的所有能力。

- 如果您因任何原因需要重新建立已刪除的模板檔案，您可以在同一個 Debian 套件源碼樹中再次執行 **debmake**。然後適當地編輯它們。
- 如果套件還未更新到可為 **debian/rules** 文件使用 **dh** 命令，請升級它以便使用該命令（參見 Section 5.4.2）。請根據具體情況更新 **debian/control** 檔案。
- 如果你有一個帶有 **foo.diff.gz** 檔案的 **1.0** 格式的原始碼套件，你可以通過建立帶有“**3.0 (quilt)**”的 **debian/source/format** 檔案來升級至新的“**3.0 (quilt)**”格式。剩下的 **debian/*** 檔案可以直接複製。如果需要的話，可以將“**filterdiff -z -x *debian/* foo.diff.gz > big.diff**”命令生成的 **big.diff** 檔案匯入到你的 quilt 系統中。[3]
- 如果它使用了其他的補丁系統，例如 **dpatch**、**dbs** 或者是帶有 **-p0**、**-p1**、**-p2** 引數的 **cdbs**。請使用 **quilt** 包中的 **deb3** 指令碼來轉換它。
- 如果它使用了帶有“**--with quilt**”選項的 **dh** 命令或者使用了 **dh_quilt_patch** 和 **dh_quilt_unpatch** 命令，請移除這些並且使其使用新的“**3.0 (quilt)**”格式。
- 如果您有一個不帶 **foo.diff.gz** 檔案的 **1.0** 格式的原始碼套件，您可以通過建立包含“**3.0 (native)**”的 **debian/source/format** 檔案，然後將其餘的 **debian/*** 檔案直接複製的方式來更新至新的“**3.0 (native)**”的原始碼格式。

您應該核對一下 DEP——Debian 增强提議 並且採用已接受的提議。
參見 ProjectsDebSrc3.0 以核對 Debian 工具鏈對新 Debian 原始碼格式的支援情況。

7.14 CDBS

Common Debian Build System （**CDBS**）是 **debhelper** 套件的包裝系統。**CDBS** 基於 Makefile 包含機制並且由 **debian/rules** 檔案中設定的 **DEB_*** 變數配置。

在將 **dh** 命令引入第七版的 **debhelper** 套件之前，**CDBS** 是建立簡單乾淨的 **debian/rules** 檔案的唯一方法。

對於很多簡單的套件，現在 **dh** 命令使 **debian/rules** 檔案很簡潔，建議保持構建系統簡潔，而非使用冗長的 **CDBS**。

Note

“**CDBS** 神奇地讓我用更少的命令來完成工作”和“我不懂新的 **dh** 的語法”都不是您繼續使用舊的 **CDBS** 系統的藉口。

對於一些複雜的套件，比如與 GNOME 相關的，當前的維護者有理由利用 **CDBS** 自動化完成他們的統一包裝。如果是這種情況，請不要費心從 **CDBS** 轉換為 **dh** 語法。

Note

如果您正在與維護 團隊 合作，請遵循團隊的既定慣例。

將套件從 **CDBS** 轉換為 **dh** 語法時，請使用以下內容作為參考：

- CDBS 文件
- The Common Debian Build System (CDBS), FOSDEM 2009

[3]您可以使用 **splitdiff** 命令來將 **big.diff** 檔案分割成多個小的增量更新補丁檔案。

7.15 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
```

7.16 UTF-8 轉換

如果上游文件是用舊編碼方案編碼的，那麼將它們轉換為 UTF-8 是個好主意。

請使用 **libc-bin** 包中的 **iconv** 命令來轉換純文字檔案的編碼。

```
$ 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。

7.17 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**
- **pdebuild --debbuildopts -sa**
- **git-pbuilder -sa**
- 對於 **gbp buildpackage**，請編輯 **~/.gbp.conf** 檔案。

Tip

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

Tip

新增至 **~/.bashrc** 檔案。

7.18 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*
- **pdebuild --debbuildopts -v***1.2*
- **git-pbuilder -v***1.2*
- 對於 **gbp buildpackage**，請編輯 **~/.gbp.conf** 檔案。

7.19 高階打包

關於以下內容的可以在 **debhelper**(7) 手冊頁中找到：

- **debhelper** 工具在“**compat** <= 8”選項下不同的行為
- 在數種不同構建條件下構建多種二進位制包
  - 製作上游原始碼的多個副本
  - 在 **override_dh_auto_configure** 目標中呼叫多個“**dh_auto_configure -S** …”指令
  - 在 **override_dh_auto_build** 目標中呼叫多個“**dh_auto_build -S** …”指令
  - 在 **override_dh_auto_install** 目標中呼叫多個“**dh_auto_install -S** …”指令
- building **udeb** packages with “**Package-Type: udeb**” in **debian/control** (see Package-Type)
- 從引導程序中排除某些包（參見 BuildProfileSpec）
  - 在 **debian/control** 中的二進位制包節中新增 **Build-Profiles** 欄位
  - 在 **DEB_BUILD_PROFILES** 環境變數設定成相關配置檔名的條件下構建套件

關於以下內容的提示可以在 **dpkg-source**(1) 手冊頁中找到：

- 多個上游原始碼套件的命名約定
  - 套件名 _ 版本**.orig.tar.gz**
  - 套件名 _ 版本**.orig-**部件名**.tar.gz**
- 將 Debian 更改記錄到上游原始碼套件中
  - **dpkg-source --commit**

7.20 其他發行版

儘管上游的原始碼有著所有構建 Debian 套件所需的資訊，找出使用何種選項的組合仍然不是一件簡單的事。

此外，上游的包可能更專注於功能的增強，而並不那麼重視向後相容性等特性，這是 Debian 打包實踐中的一個重要方面。

利用其他發行版的資訊是解決上述問題的一種選擇。

如果其他發行版是由 Debian 派生的，重新使用它是沒有價值的。

如果其他發行版是基於 RPM 的發行版，請參見 Repackage src.rpm。

通過 **rget** 命令，可以下載並開啓 **src.rpm** 檔案。（請將 **rget** 指令碼新增至 PATH 中）

rget 指令碼

```
#!/bin/sh
FCSRPM=$(basename $1)
mkdir ${FCSRPM}; cd ${FCSRPM}/
wget $1
rpm2cpio ${FCSRPM} | cpio -dium
```

許多上游原始碼套件包含 RPM 系統使用的、以 *packagename*.**spec** 或者 *packagename*.**spec.in** 命名的 SPEC 檔案。這可以被用做 Debian 套件的基點。

7.21 除錯

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.

- 核心轉儲
  - “**man core**”
  - 更新“**/etc/security/limits.conf**”檔案來包含以下程式碼：

    ```
    * soft core unlimited
    ```

  - 在 **~/.bashrc** 中新增“**ulimit -c unlimited**”
  - 使用“**ulimit -a**”來檢查
  - 按下 **Ctrl-** 或者“**kill -ABRT** *PID*”來建立一個核心轉儲檔案
- **gdb** - The GNU Debugger
  - “**info gdb**”
  - 參見 **/usr/share/doc/gdb-doc/html/gdb/index.html** 中的“Debugging with GDB”
- **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*” - Shell 指令碼的語法檢查
- “**sh -x** *script.sh*” - 追蹤 Shell 指令碼
- “**python -m py_compile** *script.py*” - Python 指令碼的語法檢查
- “**python -mtrace --trace** *script.py*” - 追蹤 Python 指令碼
- “**perl -I ../libpath -c** *script.pl*” - Perl 指令碼的語法檢查
- “**perl -d:Trace** *script.pl*” - 追蹤 Perl 指令碼
  - 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** - 按程序列出開啓的檔案
  - “**man lsof**”

Tip

script 命令能幫助記錄控制檯輸出。

Tip

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。

Tip

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。

Tip

rlwrap 和 **rlfe** 命令為所有互動命令提供了輸入行編輯功能。例如“**rlwrap dash -i**”。

Chapter 8

更多範例

有一句古老的拉丁諺語：“**fabricando fit faber**”（“熟能生巧”）。
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。
這也可以作為許多程式設計主題的介紹範例。

- 在 POSIX shell，Python3 和 C 中程式設計。
- 使用圖示圖形建立桌面 GUI 程式啓動器的方法。
- 將 命令列介面 命令轉換為 圖形介面 命令。
- 轉化程式以使用 **gettext** 來為 POSIX shell、Python3 和 C 原始碼的程式進行 國際化和本地化。
- 構建系統概述：Makefile、Python distutils、Autotools 以及 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 Chapter 4 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。

Tip

打包範例的最佳來源就是目前的 Debian 歸檔本身。請使用“Debian 程式碼搜尋”服務來查找相關範例。

8.1 挑選最好的模板

以下是一個從空目錄由零開始構建簡單的 Debian 套件的範例。
這是一個很棒的平臺，可以使您獲得所有的模板檔案，而不會使您正在處理的上游原始碼樹變得一團糟。
讓我們假設這個空目錄為 **debhello-0.1**。

```
 $ mkdir debhello-0.1
 $ tree
.
b''└b''b''—b''b''—b'' debhello-0.1

1 directory, 0 files
```

讓我們通過指定 **-x4** 選項來生成最大數量的模板檔案。
此外，讓我們使用“**-p** *debhello* **-t -u** *0.1* **-r** *1*”選項來製作遺失的上游原始碼套件。

```
 $ debmake -t -p debhello -u 0.1 -r 1 -x4
I: set parameters
 ...
I: debmake -x "4" ...
I: creating => debian/control
I: creating => debian/copyright
I: substituting => /usr/share/debmake/extra0/changelog
 ...
I: creating => debian/license-examples/Expat
I: substituting => /usr/share/debmake/extra4/BSD-3-Clause
I: creating => debian/license-examples/BSD-3-Clause
I: substituting => /usr/share/debmake/extra4/LGPL-3.0+
I: creating => debian/license-examples/LGPL-3.0+
I: $ wrap-and-sort
```

我們來檢查一下自動產生的模板檔案。

```
 $ cd ..
 $ tree
.
b''├b''b''—b''b''—b'' debhello-0.1
b''│b''   b''└b''b''—b''b''—b'' debian
b''│b''       b''├b''b''—b''b''—b'' README.Debian
b''│b''       b''├b''b''—b''b''—b'' changelog
b''│b''       b''├b''b''—b''b''—b'' clean
b''│b''       b''├b''b''—b''b''—b'' compat.ex
b''│b''       b''├b''b''—b''b''—b'' control
b''│b''       b''├b''b''—b''b''—b'' copyright
b''│b''       b''├b''b''—b''b''—b'' debhello.bug-control.ex
b''│b''       b''├b''b''—b''b''—b'' debhello.bug-presubj.ex
b''│b''       b''├b''b''—b''b''—b'' debhello.bug-script.ex
b''│b''       b''├b''b''—b''b''—b'' debhello.conffiles.ex
 ...
b''│b''       b''└b''b''—b''b''—b'' watch
b''├b''b''—b''b''—b'' debhello-0.1.tar.gz
b''└b''b''—b''b''—b'' debhello_0.1.orig.tar.gz -> debhello-0.1.tar.gz

5 directories, 51 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。

Tip

通過使用 **-T** 選項（教材模式）呼叫 **debmake** 命令，可以使生成的模板檔案更加詳細。

8.2 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。

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

讓我們取得原始碼並製作 Debian 套件。
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
.
b''├b''b''─b''b''─b'' debhello-0.2
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' scripts
b''│b''       b''└b''b''─b''b''─b'' hello
b''└b''b''─b''b''─b'' debhello-0.2.tar.gz

4 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 debhello-0.2
$ debmake -b':sh'
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.gz 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
...
```

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 ..
 $ tree
.
b''├b''b''─b''b''─b'' debhello-0.2
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' debian
b''│b''   b''│b''   b''├b''b''─b''b''─b'' README.Debian
b''│b''   b''│b''   b''├b''b''─b''b''─b'' changelog
b''│b''   b''│b''   b''├b''b''─b''b''─b'' control
b''│b''   b''│b''   b''├b''b''─b''b''─b'' copyright
b''│b''   b''│b''   b''├b''b''─b''b''─b'' patches
b''│b''   b''│b''   b''│b''   b''└b''b''─b''b''─b'' series
b''│b''   b''│b''   b''├b''b''─b''b''─b'' rules
b''│b''   b''│b''   b''├b''b''─b''b''─b'' source
b''│b''   b''│b''   b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''│b''   b''│b''   b''└b''b''─b''b''─b'' local-options
b''│b''   b''│b''   b''└b''b''─b''b''─b'' watch
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' scripts
b''│b''       b''└b''b''─b''b''─b'' hello
b''├b''b''─b''b''─b'' debhello-0.2.tar.gz
b''└b''b''─b''b''─b'' debhello_0.2.orig.tar.gz -> debhello-0.2.tar.gz

7 directories, 16 files
```

debian/rules（模板檔案，**v=0.2**）：

```
 $ cat debhello-0.2/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 debhello-0.2/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Firstname Lastname" <email.address@example.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.0
Homepage: <insert the upstream URL, if relevant>

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.
```

因為這是個 shell 指令碼包，所以 **debmake** 命令設定了“**Architecture: all**”和“**Multi-Arch: foreign**”。此外，它還將所需的 **substvar** 引數設定為“**Depends: ${misc:Depends}**”。Chapter 5 對此進行了解釋。

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**）：

```
 $ vim debhello-0.2/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-0.2/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
        dh $@
```

debian/control（維護者版本，**v=0.2**）：

```
 $ vim debhello-0.2/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-0.2/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: ${misc:Depends}
Description: example package in the debmake-doc package
 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)'.
```

Warning

如果您對 **debian/control** 模板檔案中的“**Section: unknown**”部分不作修改的話，後續的 **lintian** 錯誤可能導致構建失敗。

debian/install（維護者版本，**v=0.2**）：

```
 $ vim debhello-0.2/debian/install
 ... hack, hack, hack, ...
 $ cat debhello-0.2/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 debhello-0.2/debian/manpages
 ... hack, hack, hack, ...
 $ cat debhello-0.2/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**）：

```
 $ tree debhello-0.2/debian
debhello-0.2/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' install
b''├b''b''─b''b''─b'' manpages
b''├b''b''─b''b''─b'' patches
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch

2 directories, 11 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 debhello-0.2
 $ debuild
 dpkg-buildpackage -us -uc -ui -i -i
 ...
 fakeroot debian/rules clean
dh clean
 ...
 debian/rules build
dh build
   dh_update_autotools_config
   dh_autoreconf
   create-stamp debian/debhelper-build-stamp
 fakeroot debian/rules binary
dh binary
   dh_testroot
   dh_prep
        rm -f -- debian/debhello.substvars
        rm -fr -- debian/.debhelper/generated/debhello/ debian/debhello/ debi...
 ...
 fakeroot debian/rules binary
dh binary
 ...
```

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 ..
 $ tree -FL 1
.
b''├b''b''─b''b''─b'' debhello-0.2/
b''├b''b''─b''b''─b'' debhello-0.2.tar.gz
b''├b''b''─b''b''─b'' debhello_0.2-1.debian.tar.xz
b''├b''b''─b''b''─b'' debhello_0.2-1.dsc
b''├b''b''─b''b''─b'' debhello_0.2-1_all.deb
b''├b''b''─b''b''─b'' debhello_0.2-1_amd64.build
```

```
b''├b''b''─b''b''─b'' debhello_0.2-1_amd64.buildinfo
b''├b''b''─b''b''─b'' debhello_0.2-1_amd64.changes
b''└b''b''─b''b''─b'' debhello_0.2.orig.tar.gz -> debhello-0.2.tar.gz

1 directory, 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/LICENSE
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
 $ tar --xz -tf debhello_0.2-1.debian.tar.xz
debian/
debian/README.Debian
debian/changelog
debian/control
debian/copyright
debian/install
debian/manpages
debian/patches/
debian/patches/series
debian/rules
debian/source/
debian/source/format
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 br...
```

8.3 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 的打包需要對“**make install**”流程進行改變，從而將檔案安裝至目標系統映象所在位置，而非通常使用的 **/usr/local** 下的位置。

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
.
b''├b''b''─b''b''─b'' debhello-1.0
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' Makefile
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' scripts
b''│b''       b''└b''b''─b''b''─b'' hello
b''└b''b''─b''b''─b'' debhello-1.0.tar.gz

4 directories, 7 files
```

這裡的 **Makefile** 正確使用 **$(DESTDIR)** 和 **$(prefix)**。其他的所有檔案都和 Section 8.2 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 debhello-1.0
 $ debmake -b':sh'
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.gz 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
...
```

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**）：

```
 $ cat debhello-1.0/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**）：

```
 $ vim debhello-1.0/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-1.0/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 Section 8.2 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**）：

```
 $ tree debhello-1.0/debian
debhello-1.0/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' patches
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch

2 directories, 9 files
```

其餘的打包操作基本上和 Section 8.2 中的相同。

8.4 setup.py（Python3，命令列介面）

此處是一個從 Python3 命令列介面程式建立簡單的 Debian 套件的範例，我們假設程序使用 **setup.py** 作為它的構建系統。

讓我們假設上游的原始碼套件為 **debhello-1.1.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-1.1.tar.gz
 $ cd debhello-1.1
 $ python3 setup.py install
```

Debian 打包要求將最後一行更改為“**python3 setup.py install --install-layout=deb**”以將檔案安裝到目標系統映象所在位置。使用 **dh** 命令進行 Debian 打包時會自動解決此問題。

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
.
b''├b''b''─b''b''─b'' debhello-1.1
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' MANIFEST.in
b''│b''   b''├b''b''─b''b''─b'' PKG-INFO
b''│b''   b''├b''b''─b''b''─b'' hello_py
b''│b''   b''│b''   b''└b''b''─b''b''─b'' __init__.py
b''│b''   b''├b''b''─b''b''─b'' scripts
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello
b''│b''   b''└b''b''─b''b''─b'' setup.py
b''└b''b''─b''b''─b'' debhello-1.1.tar.gz

3 directories, 7 files
```

此處的 **hello** 指令碼和它所關聯的 **hello_py** 模組如下所示。

hello（**v=1.1**）

```
 $ cat debhello-1.1/scripts/hello
#!/usr/bin/python3
import hello_py
```

```
if __name__ == '__main__':
    hello_py.main()
```

hello_py/__init__.py（**v=1.1**）

```
 $ cat debhello-1.1/hello_py/__init__.py
#!/usr/bin/python3
def main():
    print('Hello Python3!')
    input("Press Enter to continue...")
    return

if __name__ == '__main__':
    main()
```

這些是使用帶有 **setup.py** 和 **MANIFEST.in** 檔案的 Python distutils 來打包的。

setup.py（**v=1.1**）

```
 $ cat debhello-1.1/setup.py
#!/usr/bin/python3
# vi:se ts=4 sts=4 et ai:
from distutils.core import setup

setup(name='debhello',
    version='4.0',
    description='Hello Python',
    long_description='Hello Python program.',
    author='Osamu Aoki',
    author_email='osamu@debian.org',
    url='http://people.debian.org/~osamu/',
    packages=['hello_py'],
    package_dir={'hello_py': 'hello_py'},
    scripts=['scripts/hello'],
    classifiers = ['Development Status :: 3 - Alpha',
        'Environment :: Console',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Natural Language :: English',
        'Operating System :: POSIX :: Linux',
        'Programming Language :: Python :: 3',
        'Topic :: Utilities',
    ],
    platforms   = 'POSIX',
    license     = 'MIT License'
)
```

MANIFEST.in（**v=1.1**）

```
 $ cat debhello-1.1/MANIFEST.in
include MANIFEST.in
include LICENSE
```

Tip

許多現代的 Python 套件使用 setuptools 來分發。因為 setuptools 是 disutils 的增強替代品，因此該範例對它們也很有用。

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 debhello-1.1
 $ debmake -b':py3'
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.gz 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
...
```

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**）：

```
 $ cat debhello-1.1/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。

使用“**--with python3**”選項會呼叫 **dh_python3** 來計算 Python 依賴項、將維護者指令碼新增到位元組碼檔案等。參見 **dh_python3**(1)。

使用“**--buildsystem=pybuild**”選項會為要求的 Python 版本呼叫各種構建系統，以便構建模組和擴充套件。參見 **pybuild**(1)。

debian/control（模板檔案，**v=1.1**）：

```
 $ cat debhello-1.1/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Firstname Lastname" <email.address@example.org>
Build-Depends: debhelper-compat (= 13), dh-python, python3-all
Standards-Version: 4.5.0
Homepage: <insert the upstream URL, if relevant>
X-Python3-Version: >= 3.2

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.
```

因為這是 Python3 套件，**debmake** 命令會設定“**Architecture: all**”和“**Multi-Arch: foreign**”。此外，它還將所需的 **substvar** 引數設定為“**Depends: ${python3:Depends}, ${misc:Depends}**”。Chapter 5 對這些做出瞭解釋。

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**）：

```
 $ vim debhello-1.1/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-1.1/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
        dh $@ --with python3 --buildsystem=pybuild
```

debian/control（維護者版本，**v=1.1**）：

```
 $ vim debhello-1.1/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-1.1/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13), dh-python, python3-all
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc
X-Python3-Version: >= 3.2

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: ${misc:Depends}, ${python3:Depends}
Description: example package in the debmake-doc package
 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)'.
```

hello 命令沒有附帶上游提供的手冊頁。讓我們這些維護者給它添上。
debian/manpages 等。（維護者版本，**v=1.1**）：

```
 $ vim debhello-1.1/debian/hello.1
 ... hack, hack, hack, ...
 $ vim debhello-1.1/debian/manpages
 ... hack, hack, hack, ...
 $ cat debhello-1.1/debian/manpages
debian/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。
其餘的打包工作與 Section 8.3 中的幾乎一致。
debian/ 目錄下的模板檔案。（**v=1.1**）：

```
 $ tree debhello-1.1/debian
debhello-1.1/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' hello.1
b''├b''b''─b''b''─b'' manpages
b''├b''b''─b''b''─b'' patches
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch

2 directories, 11 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 br...
Depends: python3:any (>= 3.2~)
```

8.5 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 Section 8.3 中的原始碼，並帶有增强的圖形介面支持。

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
.
b''├b''b''─b''b''─b'' debhello-1.2
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' Makefile
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' scripts
b''│b''       b''└b''b''─b''b''─b'' hello
b''└b''b''─b''b''─b'' debhello-1.2.tar.gz

4 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 Section 8.3 中的一致。

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 debhello-1.2
 $ debmake -b':sh'
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.gz 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
...
```

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 debhello-1.2/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Firstname Lastname" <email.address@example.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.5.0
Homepage: <insert the upstream URL, if relevant>

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 debhello-1.2/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-1.2/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13)
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: zenity, ${misc:Depends}
Description: example package in the debmake-doc package
 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)'.
```

請注意，這裡需要手動新增 **zenity** 依賴。
debian/rules 檔案與 Section 8.3 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**）：

```
 $ tree debhello-1.2/debian
debhello-1.2/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' patches
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
```

```
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch

2 directories, 9 files
```

其餘的打包工作與 Section 8.3 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 br...
Depends: zenity
```

8.6 setup.py（Python3，圖形介面）

此處是一個從 Python3 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 **setup.py** 作為自身的構建系統。
上游是基於 Section 8.4 中的原始碼，並帶有增強的圖形介面、桌面圖示、手冊頁。
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
.
b''├b''b''─b''b''─b'' debhello-1.3
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' MANIFEST.in
b''│b''   b''├b''b''─b''b''─b'' PKG-INFO
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' hello_py
b''│b''   b''│b''   b''└b''b''─b''b''─b'' __init__.py
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''├b''b''─b''b''─b'' scripts
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello
b''│b''   b''└b''b''─b''b''─b'' setup.py
b''└b''b''─b''b''─b'' debhello-1.3.tar.gz

5 directories, 10 files
```

以下是上游原始碼。
hello（**v=1.3**）

```
 $ cat debhello-1.3/scripts/hello
#!/usr/bin/python3
import hello_py

if __name__ == '__main__':
    hello_py.main()
```

hello_py/__init__.py（**v=1.3**）

```
 $ cat debhello-1.3/hello_py/__init__.py
#!/usr/bin/python3
from gi.repository import Gtk

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()
```

setup.py（v=1.3）

```
 $ cat debhello-1.3/setup.py
#!/usr/bin/python3
# vi:se ts=4 sts=4 et ai:
from distutils.core import setup

setup(name='debhello',
    version='4.1',
    description='Hello Python',
    long_description='Hello Python program.',
    author='Osamu Aoki',
    author_email='osamu@debian.org',
    url='http://people.debian.org/~osamu/',
    packages=['hello_py'],
    package_dir={'hello_py': 'hello_py'},
    scripts=['scripts/hello'],
    data_files=[
        ('share/applications', ['data/hello.desktop']),
        ('share/pixmaps', ['data/hello.png']),
        ('share/man/man1', ['man/hello.1']),
    ],
    classifiers = ['Development Status :: 3 - Alpha',
        'Environment :: Console',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Natural Language :: English',
        'Operating System :: POSIX :: Linux',
        'Programming Language :: Python :: 3',
        'Topic :: Utilities',
    ],
    platforms   = 'POSIX',
    license     = 'MIT License'
)
```

MANIFEST.in（v=1.3）

```
 $ cat debhello-1.3/MANIFEST.in
include MANIFEST.in
```

```
include LICENSE
include data/hello.deskto
include data/hello.png
include man/hello.1
```

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 debhello-1.3
 $ debmake -b':py3'
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.gz 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
...
```

其餘的步驟與 Section 8.4 中的基本一致。

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**）：

```
 $ vim debhello-1.3/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-1.3/debian/rules
#!/usr/bin/make -f
export DH_VERBOSE = 1

%:
        dh $@ --with python3 --buildsystem=pybuild
```

debian/control（維護者版本，**v=1.3**）：

```
 $ vim debhello-1.3/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-1.3/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13), dh-python, python3-all
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc
X-Python3-Version: >= 3.2

Package: debhello
Architecture: all
Multi-Arch: foreign
Depends: gir1.2-gtk-3.0, python3-gi, ${misc:Depends}, ${python3:Depends}
Description: example package in the debmake-doc package
 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)'.
```

請注意，此處需要手動新增 **python3-gi** 和 **gir1.2-gtk-3.0** 依賴。

因為上游原始碼已經提供手冊頁，並且其餘的檔案在 **setup.py** 檔案中都有對應條目，就沒有必要再去建立 Section 8.4 中所要求的 **debian/install** 和 **debian/manpages** 檔案。

其餘的打包工作與 Section 8.4 中的幾乎完全一致。

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 br...
Depends: gir1.2-gtk-3.0, python3-gi, python3:any (>= 3.2~)
```

8.7 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 Chapter 4 中的原始碼的增强版本。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 的打包需要對“**make install**”流程進行改變，從而將檔案安裝至系統映象所在位置，而非通常使用的 **/usr/local** 下的位置。

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
.
b''├b''b''─b''b''─b'' debhello-1.4
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' Makefile
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''├b''b''─b''b''─b'' config.h
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-1.4.tar.gz

4 directories, 8 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/src/config.h
#define PACKAGE_AUTHOR "Osamu Aoki"
```

Makefile（**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
```

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 debhello-1.4
 $ debmake
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.gz 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
...
```

其餘的工作與 Section 4.5 中的幾乎一致。

像 Section 4.6 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。

如果環境變數 **DEB_BUILD_MAINT_OPTIONS** 沒有在 **debian/rules** 檔案中進行匯出，lintian 會在連結 **libm** 時丟擲警告：“W: debhello: hardening-no-relro usr/bin/hello”。

debian/control 檔案與 Section 4.6 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**）：

```
 $ tree debhello-1.4/debian
debhello-1.4/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' patches
```

```
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch

2 directories, 9 files
```

其餘的打包步驟與 Section 4.7 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 con...
Depends: debhello (= 1.4-1)
 $ dpkg -f debhello_1.4-1_amd64.deb pre-depends depends recommends conflicts ...
Depends: libc6 (>= 2.3.4)
```

8.8 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 Section 8.7 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
.
b''├b''b''─b''b''─b'' debhello-1.5
b''│b''   b''├b''b''─b''b''─b'' LICENSE
b''│b''   b''├b''b''─b''b''─b'' Makefile.in
b''│b''   b''├b''b''─b''b''─b'' configure
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-1.5.tar.gz

4 directories, 8 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
```

請注意，**configure** 命令替換 **Makefile.in** 檔案中的 **@⋯@** 字串以生成 **Makefile** 與 **src/config.h**。

讓我們使用 **debmake** 命令打包。

```
 $ cd debhello-1.5
 $ debmake
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.gz 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
...
```

結果與 Section 4.5 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**）：

```
 $ cat debhello-1.5/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,--as-needed

%:
        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**）：

```
 $ vim debhello-1.5/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-1.5/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 Section 4.7 中的基本一致。

8.9 Autotools（單個二進位制檔案）

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 Autotools = Autoconf （**Makefile.am** 和 **configure.ac**）作為構建系統。參見 Section 5.16.1。

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 Section 8.8 中所介紹的，使用這些檔案進行打包。

更好的做法是，如果上游提供的 **Makefile.am** 和 **configure.ac** 相容最新版本，我們可以使用最新的 Autoconf 和 Automake 套件重新生成這些（Makefile 和 configure）檔案。這麼做有利於移植到新的 CPU 架構上等優勢。此項工作可以使用帶有“**--with autoreconf**”選項的 **dh** 命令來自動化。

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
.
b''├b''b''─b''b''─b'' debhello-1.6
b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''├b''b''─b''b''─b'' configure.ac
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''├b''b''─b''b''─b'' Makefile.am
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-1.6.tar.gz
```

```
4 directories, 9 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
```

Tip

如果沒有像上述例子中，在 **AM_INIT_AUTOMAKE()** 中指定嚴格級別（strictness level）為“**foreign**”，那麼 **automake** 會預設嚴格級別為“**gnu**”，並需要在頂級目錄中有若干檔案。參見 **automake** 文件的“3.2 Strictness”。

讓我們使用 **debmake** 命令打包。

```
 $ cd debhello-1.6
 $ debmake
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.gz 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
...
```

結果與 Section 8.8 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**）**:**

```
 $ cat debhello-1.6/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,--as-needed

%:
        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**）：

```
 $ vim debhello-1.6/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-1.6/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 Section 4.7 中的基本一致。

8.10 CMake（單個二進位制套件）

此處是一個從簡單的 C 語言原始碼程式生成簡單的 Debian 套件的範例，我們假設上游使用 CMake（**CMakeLists.txt** 和若干形似 **config.h.in** 的檔案）作為構建系統。參見 Section 5.16.2。

cmake 命令根據 **CMakeLists.txt** 檔案和它的 **-D** 選項來生成 **Makefile** 檔案。此外，它還會根據 **configure_file(⋯)** 中指定的條目來替換帶有 **@⋯@** 的字串、更改 **#cmakedefine** ⋯。

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
.
b''├b''b''─b''b''─b'' debhello-1.7
b''│b''   b''├b''b''─b''b''─b'' CMakeLists.txt
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''├b''b''─b''b''─b'' CMakeLists.txt
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''├b''b''─b''b''─b'' CMakeLists.txt
b''│b''       b''├b''b''─b''b''─b'' config.h.in
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-1.7.tar.gz

4 directories, 9 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 debhello-1.7
 $ debmake
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.gz 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
...
```

結果與 Section 8.8 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**）：

```
 $ cat debhello-1.7/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,--as-needed

%:
        dh $@

#override_dh_auto_configure:
#       dh_auto_configure -- \
#             -DCMAKE_LIBRARY_ARCHITECTURE="$(DEB_TARGET_MULTIARCH)"
```

debian/control（模板檔案，**v=1.7**）：

```
 $ cat debhello-1.7/debian/control
Source: debhello
Section: unknown
Priority: optional
Maintainer: "Firstname Lastname" <email.address@example.org>
Build-Depends: cmake, debhelper-compat (= 13)
Standards-Version: 4.5.0
Homepage: <insert the upstream URL, if relevant>

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**）：

```
 $ vim debhello-1.7/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-1.7/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 debhello-1.7/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-1.7/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: cmake, debhelper-compat (= 13)
```

```
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: example package in the debmake-doc package
 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。
其餘的打包工作與 Section 8.8 中的近乎一致。

8.11 Autotools（多個二進位制套件）

此處是從一個簡單的 C 語言原始碼程式建立一個包含可執行套件、共享程式庫包、開發檔案包和除錯符號包的一系列 Debian 二進位制包的範例，我們假設上游使用 Autotools = Autoconf 和 Automake （使用 **Makefile.am** 和 **configure.ac** 作為輸入檔案）作為構建系統。參見 Section 5.16.1。
讓我們用與 Section 8.9 中的相同的方式打包。
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
.
b''├b''b''─b''b''─b'' debhello-2.0
b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''├b''b''─b''b''─b'' configure.ac
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' lib
b''│b''   b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''│b''   b''├b''b''─b''b''─b'' sharedlib.c
b''│b''   b''│b''   b''└b''b''─b''b''─b'' sharedlib.h
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''├b''b''─b''b''─b'' Makefile.am
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-2.0.tar.gz

5 directories, 12 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
```

讓我們用 **debmake** 命令將這些打包到多個包中：

- **debhello**: type = **bin**
- **libsharedlib1**: type = **lib**
- **libsharedlib-dev**: type = **dev**

此處的 **-b',libsharedlib1,libsharedlib-dev'** 選項是用以指明生成的二進位制包。

```
 $ cd debhello-2.0
 $ debmake -b',libsharedlib1,libsharedlib-dev'
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.gz 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 Section 8.8 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**）：

```
 $ cat debhello-2.0/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,--as-needed

%:
        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**）：

```
 $ vim debhello-2.0/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-2.0/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 debhello-2.0/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-2.0/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: debhelper-compat (= 13), dh-autoreconf
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc

Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: libsharedlib1 (= ${binary:Version}),
         ${misc:Depends},
         ${shlibs:Depends}
Description: example executable package
 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)'.
 .
 This package provides the executable program.

Package: libsharedlib1
Section: libs
Architecture: any
Multi-Arch: same
```

```
Pre-Depends: ${misc:Pre-Depends}
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: example shared library package
 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)'.
 .
 This package contains the shared library.

Package: libsharedlib-dev
Section: libdevel
Architecture: any
Multi-Arch: same
Depends: libsharedlib1 (= ${binary:Version}), ${misc:Depends}
Description: example development package
 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)'.
 .
 This package contains the development files.
```

debian/*.install（維護者版本，**v=2.0**）：

```
 $ vim debhello-2.0/debian/debhello.install
 ... hack, hack, hack, ...
 $ cat debhello-2.0/debian/debhello.install
usr/bin/*
usr/share/*
 $ vim debhello-2.0/debian/libsharedlib1.install
 ... hack, hack, hack, ...
 $ cat debhello-2.0/debian/libsharedlib1.install
usr/lib/*/*.so.*
 $ vim debhello-2.0/debian/libsharedlib-dev.install
 ... hack, hack, hack, ...
 $ cat debhello-2.0/debian/libsharedlib-dev.install
###usr/lib/*/pkgconfig/*.pc
usr/include
usr/lib/*/*.so
```

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**）：

```
 $ tree debhello-2.0/debian
debhello-2.0/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' debhello.install
b''├b''b''─b''b''─b'' libsharedlib-dev.install
b''├b''b''─b''b''─b'' libsharedlib1.install
b''├b''b''─b''b''─b'' libsharedlib1.symbols
b''├b''b''─b''b''─b'' patches
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch
```

```
2 directories, 13 files
```

其餘的打包工作與 Section 8.8 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 con...
Depends: debhello (= 2.0-1)
 $ dpkg -f debhello_2.0-1_amd64.deb pre-depends depends recommends conflicts ...
Depends: libsharedlib1 (= 2.0-1), libc6 (>= 2.2.5)
 $ dpkg -f libsharedlib-dev_2.0-1_amd64.deb pre-depends depends recommends co...
Depends: libsharedlib1 (= 2.0-1)
 $ dpkg -f libsharedlib1-dbgsym_2.0-1_amd64.deb pre-depends depends recommend...
Depends: libsharedlib1 (= 2.0-1)
 $ dpkg -f libsharedlib1_2.0-1_amd64.deb pre-depends depends recommends confl...
Depends: libc6 (>= 2.2.5)
```

8.12 CMake（多個二進位制套件）

此處是從一個簡單的 C 語言原始碼程式建立一系列包含可執行套件、共享程式庫包、開發文件包和除錯符號包的 Debian 二進位制包的範例，我們假設上游使用 CMake（**CMakeLists.txt** 和其他形如 **config.h.in** 的檔案）作為構建系統。參見 Section 5.16.2。
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
.
b''├b''b''─b''b''─b'' debhello-2.1
b''│b''   b''├b''b''─b''b''─b'' CMakeLists.txt
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' lib
b''│b''   b''│b''   b''├b''b''─b''b''─b'' CMakeLists.txt
b''│b''   b''│b''   b''├b''b''─b''b''─b'' sharedlib.c
b''│b''   b''│b''   b''└b''b''─b''b''─b'' sharedlib.h
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''├b''b''─b''b''─b'' CMakeLists.txt
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''├b''b''─b''b''─b'' CMakeLists.txt
b''│b''       b''├b''b''─b''b''─b'' config.h.in
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-2.1.tar.gz

5 directories, 12 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 debhello-2.1
 $ debmake -b',libsharedlib1,libsharedlib-dev'
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.gz 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
...
```

結果與 Section 8.8 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**）：

```
 $ cat debhello-2.1/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,--as-needed

%:
        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**）：

```
 $ vim debhello-2.1/debian/rules
 ... hack, hack, hack, ...
 $ cat debhello-2.1/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 debhello-2.1/debian/control
 ... hack, hack, hack, ...
 $ cat debhello-2.1/debian/control
Source: debhello
Section: devel
Priority: optional
Maintainer: Osamu Aoki <osamu@debian.org>
Build-Depends: cmake, debhelper-compat (= 13)
Standards-Version: 4.3.0
Homepage: https://salsa.debian.org/debian/debmake-doc
```

```
Package: debhello
Architecture: any
Multi-Arch: foreign
Depends: libsharedlib1 (= ${binary:Version}),
         ${misc:Depends},
         ${shlibs:Depends}
Description: example executable package
 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)'.
 .
 This package provides the executable program.

Package: libsharedlib1
Section: libs
Architecture: any
Multi-Arch: same
Pre-Depends: ${misc:Pre-Depends}
Depends: ${misc:Depends}, ${shlibs:Depends}
Description: example shared library package
 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)'.
 .
 This package contains the shared library.

Package: libsharedlib-dev
Section: libdevel
Architecture: any
Multi-Arch: same
Depends: libsharedlib1 (= ${binary:Version}), ${misc:Depends}
Description: example development package
 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)'.
 .
 This package contains the development files.
```

debian/*.install（維護者版本，**v=2.1**）：

```
 $ vim debhello-2.1/debian/debhello.install
 ... hack, hack, hack, ...
 $ cat debhello-2.1/debian/debhello.install
usr/bin/*
usr/share/*
 $ vim debhello-2.1/debian/libsharedlib1.install
 ... hack, hack, hack, ...
 $ cat debhello-2.1/debian/libsharedlib1.install
usr/lib/*/*.so.*
 $ vim debhello-2.1/debian/libsharedlib-dev.install
 ... hack, hack, hack, ...
 $ cat debhello-2.1/debian/libsharedlib-dev.install
###usr/lib/*/pkgconfig/*.pc
usr/include
usr/lib/*/*.so
```

需要對上游的 CMakeList.txt 進行修補，以便應對多架構的路徑。

debian/patches/*（維護者版本，**v=2.1**）：

```
 ... hack, hack, hack, ...
 $ cat debhello-2.1/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**）：

```
 $ tree debhello-2.1/debian
debhello-2.1/debian
b''├b''b''─b''b''─b'' README.Debian
b''├b''b''─b''b''─b'' changelog
b''├b''b''─b''b''─b'' control
b''├b''b''─b''b''─b'' copyright
b''├b''b''─b''b''─b'' debhello.install
b''├b''b''─b''b''─b'' libsharedlib-dev.install
b''├b''b''─b''b''─b'' libsharedlib1.install
b''├b''b''─b''b''─b'' libsharedlib1.symbols
b''├b''b''─b''b''─b'' patches
b''│b''   b''├b''b''─b''b''─b'' 000-cmake-multiarch.patch
b''│b''   b''└b''b''─b''b''─b'' series
b''├b''b''─b''b''─b'' rules
b''├b''b''─b''b''─b'' source
b''│b''   b''├b''b''─b''b''─b'' format
b''│b''   b''└b''b''─b''b''─b'' local-options
b''└b''b''─b''b''─b'' watch

2 directories, 14 files
```

其餘的打包工作與 Section 8.8 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 con...
Depends: debhello (= 2.1-1)
 $ dpkg -f debhello_2.1-1_amd64.deb pre-depends depends recommends conflicts ...
Depends: libsharedlib1 (= 2.1-1), libc6 (>= 2.2.5)
 $ dpkg -f libsharedlib-dev_2.1-1_amd64.deb pre-depends depends recommends co...
Depends: libsharedlib1 (= 2.1-1)
 $ dpkg -f libsharedlib1-dbgsym_2.1-1_amd64.deb pre-depends depends recommend...
Depends: libsharedlib1 (= 2.1-1)
 $ dpkg -f libsharedlib1_2.1-1_amd64.deb pre-depends depends recommends confl...
Depends: libc6 (>= 2.2.5)
```

8.13 國際化

此處是更新 Section 8.11 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。

Tip

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
.
b''├b''b''─b''b''─b'' debhello-2.0
b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''├b''b''─b''b''─b'' configure.ac
b''│b''   b''├b''b''─b''b''─b'' data
b''│b''   b''│b''   b''├b''b''─b''b''─b'' hello.desktop
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.png
b''│b''   b''├b''b''─b''b''─b'' lib
b''│b''   b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''│b''   b''├b''b''─b''b''─b'' sharedlib.c
b''│b''   b''│b''   b''└b''b''─b''b''─b'' sharedlib.h
b''│b''   b''├b''b''─b''b''─b'' man
b''│b''   b''│b''   b''├b''b''─b''b''─b'' Makefile.am
b''│b''   b''│b''   b''└b''b''─b''b''─b'' hello.1
b''│b''   b''└b''b''─b''b''─b'' src
b''│b''       b''├b''b''─b''b''─b'' Makefile.am
b''│b''       b''└b''b''─b''b''─b'' hello.c
b''└b''b''─b''b''─b'' debhello-2.0.tar.gz

5 directories, 12 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 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
/home/osamu/pub/salsa/debmake/debmake-doc/debhello-2.0-pkg2/step151.cmd: line...
total 60
-rw-rw-r-- 1 osamu osamu   494 Sep 28 23:51 ChangeLog
-rw-rw-r-- 1 osamu osamu 17577 Sep 28 23:51 Makefile.in.in
-rw-rw-r-- 1 osamu osamu  3376 Sep 28 23:51 Makevars.template
-rw-rw-r-- 1 osamu osamu    59 Sep 28 23:51 POTFILES.in
-rw-rw-r-- 1 osamu osamu  2203 Sep 28 23:51 Rules-quot
-rw-rw-r-- 1 osamu osamu   217 Sep 28 23:51 boldquot.sed
-rw-rw-r-- 1 osamu osamu  1337 Sep 28 23:51 en@boldquot.header
-rw-rw-r-- 1 osamu osamu  1203 Sep 28 23:51 en@quot.header
-rw-rw-r-- 1 osamu osamu   672 Sep 28 23:51 insert-header.sin
-rw-rw-r-- 1 osamu osamu   153 Sep 28 23:51 quot.sed
-rw-rw-r-- 1 osamu osamu   432 Sep 28 23:51 remove-potcdate.sin
```

讓我們在 **configure.ac** 檔案中新增“**AM_GNU_GETTEXT([external])**”等條目。
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        2020-07-13 00:39:17.026534688 +0900
+++ po/Makevars 2020-07-13 00:39:17.102533289 +0900
@@ -18,14 +18,14 @@
 # or entity, or to disclaim their copyright.  The empty string stands for
 # the public domain; in this case the translators are expected to disclaim
 # their copyright.
-COPYRIGHT_HOLDER = Free Software Foundation, Inc.
+COPYRIGHT_HOLDER = Osamu Aoki <osamu@debian.org>

 # This tells whether or not to prepend "GNU " prefix to the package
 # name that gets inserted into the header of the $(DOMAIN).pot file.
 # Possible values are "yes", "no", or empty.  If it is empty, try to
 # detect it automatically by scanning the files in $(top_srcdir) for
 # "GNU packagename" string.
-PACKAGE_GNU =
+PACKAGE_GNU = no

 # This is the email address or URL to which the translators shall report
 # bugs in the untranslated strings:
 $ rm po/Makevars.template
```

讓我們通過用 **_(…)** 包裹字串的方式來更新國際化版本的 C 語言原始碼。

src/hello.c（國際化版）：

```
 ... hack, hack, hack, ...
 $ cat src/hello.c
#include "config.h"
#include <stdio.h>
#include <sharedlib.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>
#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_
 $ 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: 2020-07-13 00:39+0900\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
```

```
"Language: \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"

#: src/hello.c:8
#, c-format
msgid "Hello, I am "
msgstr ""

#: lib/sharedlib.c:6
#, 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 ""

#: data/hello.desktop.in:9
msgid "hello.png"
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 Section 8.11 中的近乎一致。
您可以在 Section 8.14 中尋找更多國際化的例子：

- 帶有 Makefile 的 POSIX shell 指令碼（v=3.0），
- 帶有 distutils 的 Python3 指令碼（v=3.1），
- 帶有 Makefile.in + configure 的 C 語言原始碼（v=3.2），
- 帶有 Autotools 的 C 語言原始碼（v=3.3），以及
- 帶有 CMake 的 C 語言原始碼（v=3.4）。

8.14 細節

所示範例的實際細節及其變體可通過以下方式獲得。
如何取得細節

```
$ apt-get source debmake-doc
$ sudo apt-get install devscripts build-essentials
$ cd debmake-doc*
$ sudo apt-get build-dep ./
$ make
```

帶 **-pkg[0-9]** 字尾的每個目錄都包含 Debian 打包範例。

- 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
- 打包後的原始碼樹快照：**packge** 目錄
- 執行 **debuild** 命令後的原始碼樹快照：**test** 目錄

Appendix A

debmake(1) 手冊頁

A.1 名稱

debmake，用來製作 Debian 原始碼套件的程式

A.2 概述

debmake [**-h**] [**-c** | **-k**] [**-n** | **-a** 套件名-版本號**.orig.tar.gz** | **-d** | **-t**] [**-p** *package*] [**-u** *version*] [**-r** 修訂號] [**-z** 擴充套件] [**-b** "*binarypackage, ⋯*]" [**-e** *foo@example.org*] [**-f** " 名稱姓氏"] [**-i** " 構建工具" | **-j**] [**-l** *license_file*] [**-m**] [**-o** *file*] [**-q**] [**-s**] [**-v**] [**-w** "*addon, ⋯*"] [**-x** [01234]] [**-y**] [**-L**] [**-P**] [**-T**]

A.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。
- 在 *package-version/* 目錄下呼叫 **dpkg-buildpackage**（通常使用其高層封裝工具，例如 **debuild** 或者 **pdebuild**）以構建 Debian 套件。

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。

A.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 製作一個原生 Debian 原始碼套件，即不涉及 **.orig.tar.gz**。這樣將製作一個“**3.0 (native)**”格式的包。

如果您正打算打包一個含 **debian/*** 目錄的 Debian 特有的原始碼樹成為一個 Debian 原生套件的話，還請三思。您可以使用“**debmake -d -i debuild**”或者“**debmake -t -i debuild**”命令來建立一個“**3.0 (quilt)**”格式的非原生 Debian 套件。唯一的區別是 **debian/changelog** 檔案必須使用非原生套件對應的命名規範：版本號-修訂號。非原生的套件對下游發行版更友好。

-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。

如果所指定的引數是一個指向上游原始碼壓縮包的 URL（http://、https:// 或 ftp://），程式將會使用 **wget** 或 **curl** 下載這個壓縮包。

-d, --dist 先執行“make dist”命令或其等效命令以生成上游原始碼壓縮包並在打包過程中使用。

“**debmake -d**”命令設計用於在 套件名/ 目錄下使用了上游版本控制系統的場景，且其構建系統支持“**make dist**”或其等效命令。（如 automake/autoconf、Python distutils 等等）

-t, --tar 執行“**tar**”命令以生成上游原始碼壓縮包並在打包過程中使用。

“**debmake -t**”命令設計用於在 套件名/ 目錄下使用了上游版本控制系統的場景。除非您使用了 **-u** 選項或者使用 **debian/changelog** 檔案提供了上游版本號，預設情況下程式將運用協調世界時日期和時間按照 **0~%y%m%d%H%M** 的格式作為快照的上游版本號，例如 *0~1403012359*。所生成的壓縮包將排除上游版本控制系統中的 **debian/** 目錄。（它也會排除常見的版本控制系統目錄：**.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 ” 二進位制套件名 ***[:type],…”, --binaryspec ”*** 二進位制套件名 ***[:type],…”*** 設定二進位制套件的指定型別內容，使用一個用逗號分隔的二進位制套件名*:* 型別成對列表；例如，使用完整形式“**foo:bin,foo-doc:doc,libfoo1:lib,libfoo-dev:dev**”或者使用短形式，“**-doc,libfoo1,libfoo-dev**”。

這裡，二進位制套件是二進位制套件名稱，可選的類型應當從下面的型別值中進行選取：

- **bin**：C/C++ 預編譯 ELF 二進位制程式碼套件（any，foreign）（預設，別名：**””**，即，空字串）
- **data**：資料（字型、影象、……）套件（all，foreign）（別名：**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**)
- **ruby**：Ruby 指令碼套件（all，foreign）（別名：**rb**）
- **nodejs**: Node.js based JavaScript package (all, foreign) (alias: **js**)
- **script**：Shell 指令碼套件（all，foreign）（別名：**sh**）

括號內成對的值，例如（any，foreign），是套件的架構和多架構（**Multi-Arch**）特性的值，它們將設定在 **debian/control** 檔案中。

大多數情況下，**debmake** 命令可以有效地從二進位制套件的名稱猜測出正確的型別。如果型別的值並不明顯，程式將回退到將型別設定為 **bin**。例如，**libfoo** 設定型別為 **lib**，而 **font-bar** 會令程式設定型別為 **data**，……

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**"** 在執行結束時呼叫"構建工具"。構建工具可以是"**dpkg-buildpackage**"、"**debuild**"、"**pdebuild**"、**pdebuild --pbuilder cowbuilder**"等等。

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 " 許可證檔案**,⋯", --license "** 許可證檔案**,⋯"** 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。

預設值是新增 **COPYING** 和 **LICENSE** 檔案，您只需要在許可證檔案部分新增額外的檔名即可，並使用"**,**"分隔各個檔名。

-m, --monoarch 强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 使用上游配置檔案（例如 Python 裡的 setup.py 等）資訊來初始化套件描述內容。

-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 **setup.py** based 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**：最少的配置檔案（這是存在任何已有配置檔案時的預設選項）
- **-x1**：所有 **-x0** 提供的檔案以及用於生成單個二進位制套件可能需要的配置檔案。(這是隻生成單個二進位制套件，且不存在其它已有配置檔案時的預設選項)

- **-x2**：所有 **-x2** 提供的檔案以及用於生成多個二進位制套件可能需要的配置檔案。（這是生成多個二進位制套件，且不存在其它已有配置檔案時的預設選項）
- **-x3**：所有 **-x2** 提供的檔案以及不常使用的配置模板檔案。不常使用的配置模板檔案在生成時會帶上 **.ex** 字尾名以方便對其刪除。如需使用這些配置檔案，請重新命名這些檔案並去除 **.ex** 的字尾。
- **-x4** 選項：全部配置 **-x3** 檔案加版權宣告檔案範例。

-y, --yes 對所有提示“强制選擇是”（不提示選項“詢問 [是/否]”；重複選項兩次則為“强制選擇否”）

-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 在模板檔案中輸出教材註釋行。

A.4 範例

對比較正常的原始碼來說，您可以使用一行命令簡單地構建一個自用的 Debian 二進位制套件。測試安裝這樣生成的套件通常比傳統的“**make install**”命令安裝至 **/usr/local** 目錄更好，因為 Debian 套件可以使用“**dpkg -P …**””命令更乾淨地解除安裝掉。這裡提供構建這類測試套件的一些例子（這些例子應該在大多數情況下足夠使用。如果 **-d** 選項無法工作，請嘗試使用 **-t** 選項。）

對典型的使用 autoconf/automake 的 C 程式原始碼樹：

- **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**

A.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 **dh-python** package.
- Autotools (Autoconf + Automake) 構建系統可能需要 **autotools-dev** 或 **dh-autoreconf** 套件。
- 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。

A.6 注意事項

debmake 的目的是為套件維護者提供開始工作的模板檔案。註釋行以 **#** 開始，其中包含一些教材文字。您在將套件上傳至 Debian 倉庫之前必須刪除或者修改這樣的註釋行。

許可證資訊的提取和賦值過程應用了大量啓發式操作，因此在某些情況下可能不會正常工作。强烈建議您搭配使用其它工具，例如來自 **devscripts** 套件的 **licensecheck** 工具，以配合 **debmake** 的使用。

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：

- 上游套件名稱（**-p**）：[-+.a-z0-9]{2,}
- 二進位制套件名稱（**-b**）：[-+.a-z0-9]{2,}
- 上游版本號（**-u**）：[0-9][-+.:~a-z0-9A-Z]*
- Debian 修訂版本（**-r**）：[0-9][+.~a-z0-9A-Z]*

請在《Debian 政策手冊》的 第 5 章 - Control 檔案及其欄位 一節中檢視其精確定義。

debmake 所假設的打包情景是相對簡單的。因此，所有與直譯器相關的程式都會預設為“**Architecture: all**”的情況。當然，這個假設並非總是成立。

A.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**：列印資訊
- **p**：列出全部全域性引數
- **d**：列出所有二進位制套件解析得到的引數
- **f**：用於掃描授權資訊的輸入檔名
- **y**：授權資訊欄的年份/名稱切分資訊
- **s**：format_state 的行掃描器
- **b**：content_state 掃描迴圈：迴圈開始
- **m**: content_state 掃描迴圈：正則匹配之後
- **e**: content_state 掃描迴圈：迴圈結束
- **c**：列印授權區段文字
- **l**：列印許可證區段文字
- **a**：列印作者/翻譯者區段文字
- **k**：debian/copyright 各節的排序關鍵字
- **n**：debian/copyright 的掃描結果（“**debmake -k**”）

用法如下：

```
$ DEBUG=pdfbmeclak debmake ...
```

檢視原始碼中的 README.developer 檔案以瞭解更多資訊。

A.8 作者

A.9 許可證

Expat 許可證

A.10 參見

debmake-doc 套件提供了“Debian 維護者指南”手冊，以純文字、HTML 和 PDF 三種格式存放在 **/usr/share/doc/debmake-doc/** 目錄下。

另見 **dpkg-source**(1), **deb-control**(5), **debhelper**(7), **dh**(1), **dpkg-buildpackage**(1), **debuild**(1), **quilt**(1), **dpkg-depcheck**(1), **pdebuild**(1), **pbuilder**(8), **cowbuilder**(8), **gbp-buildpackage**(1), **gbp-pq**(1) 和 **git-pbuilder**(1) 的手冊頁。