Debian 新維護人員手冊

Josip Rodin <joy-mg@debian.org>

翻譯：Aron Xu <happyaron.xu@gmail.com>
翻譯：李凌<lilingv@gmail.com>
翻譯：原眞<ycheng@slat.org>
繁簡轉換：陳侃如<koster@debian.org>
繁簡轉換：青木修<osamu@debian.org>

版本1.2.25，2010-12-21 14:06:56 UTC

版權聲明

Copyright © 1998-2002 Josip Rodin
Copyright © 2005-2010 Osamu Aoki
Copyright © 2010 Craig Small
Copyright © 2010 Raphaël Hertzog

本文件可在GNU 通用公共許可證第二版或更高版本的條款規定下使用。

本文件在撰寫過程中參考了以下兩篇文件：

Making a Debian Package (AKA the Debmake Manual), copyright © 1997 Jaldhar Vyas.

The New-Maintainer's Debian Packaging Howto, copyright © 1997 Will Lowe.

Contents

1 從一條正確的路開始 **1**

1.1 開發時需要的軟體 1

1.2 基本術語 4

1.3 正式的Debian Developer 5

1.4 到何處尋求幫助 5

2 第一步 **7**

2.1 選擇你的程式 7

2.2 獲得程式，並且試用它 9

2.3 自由的可移植程式 10

2.4 套件名稱和版本 11

2.5 初始化Debian 套件 11

3 修改原始碼 **15**

3.1 設置quilt 15

3.2 修復上游Bug 15

3.3 把檔案安裝到目的位置 16

3.4 不一樣的函式庫名稱 19

4 debian 目錄中的必須內容 **21**

4.1 control 檔案 21

4.2 copyright 檔案 26

4.3 changelog 檔案 27

4.4 rules 檔案 28

4.4.1 rules 檔案中的target 28

4.4.2 默認的rules 檔案 29
4.4.3 定製rules 檔案 32

5 debian 目錄下的其他檔案 37

5.1 README.Debian 檔案 37
5.2 compat 檔案 38
5.3 conffiles 檔案 38
5.4 *package*.cron.* 檔案 38
5.5 dirs 檔案 39
5.6 *package*.doc-base 檔案 39
5.7 docs 檔案 40
5.8 emacsen-* 檔案 40
5.9 *package*.examples 檔案 40
5.10 *package*.init 和*package*.default 檔案 40
5.11 install 檔案 41
5.12 *package*.info 檔案 41
5.13 {*package*.|source/}lintian-overrides 檔案 42
5.14 manpage.* 檔案 42
5.14.1 manpage.1.ex 檔案 42
5.14.2 manpage.sgml.ex 檔案 43
5.14.3 manpage.xml.ex 檔案 43
5.15 *package*.manpages 檔案 44
5.16 menu 檔案 44
5.17 NEWS 檔案 44
5.18 {post|pre}{inst|rm} 檔案 45
5.19 TODO 檔案 45
5.20 watch 檔案 45
5.21 source/format 檔案 46
5.22 source/local-options 檔案 46
5.23 patches/* 檔案 47

6 構建套件 49

6.1 完整的(重)構建 49
6.2 自動編譯系統 50
6.3 在上傳時包含orig.tar.gz 檔案 51
6.4 debuild 命令 51
6.5 pbuilder 套件 52
6.6 git-buildpackage 和相似命令 54
6.7 快速重構建 54

7 檢套件中的錯誤 57

7.1 檢驗套件安裝過程 57
7.2 檢驗*maintainer scripts* 57
7.3 lintian 套件 58
7.4 debc 命令 59
7.5 debdiff 命令 59
7.6 interdiff 命令 59
7.7 mc 命令 59

8 上傳套件 61

8.1 上傳到Debian 倉庫 61
8.2 跳過的上傳 62

9 更新套件 63

9.1 新的Debian 版本 63
9.2 檢新上游版本 64
9.3 新上游版本 64
9.4 更新打包風格 66
9.5 對更新套件的幾點提示 66

Chapter 1

從一條正確的路開始

本文件試圖普通Debian 用户，和希望對Debian 套件有所瞭解的開發人員講述如何製作Debian 套件。它使用了通俗的語言，並輔以大量可用於實踐的範例。有一句古羅馬諺語說得好：一例勝千言！

本文件已經Debian squeeze 進行了更新。[1]

Debian 能夠成一個頂級的發行版的原因之一就是它的套件系統。儘管相當數量的軟體已經被以Debian 格式打包，有時還是需要安裝一些不是這一格式的軟體。可能你會如何製作自己的套件而迷惑，並可能認這是很困難的任務。是的，如果你是一個Linux 初學者，那麼這的確很難；不過如果你眞的只是一個新手，現在也不會來讀這篇文件了。:-) 你的確需要對Unix 編程有所瞭解，但顯然不需要是這方面的天才。

有一件事是非常明確的：創建並維護一個Debian 套件需要花費很多時間。了能夠不犯錯誤，維護者需要良好的技術基礎並十分勤奮。

本文件會講述每一個細節(在開始時或許給人感覺毫不相關)，並幫助你創建出自己的第一個Debian 套件，從而使你學習到如何打包該軟體的下一個版本或其他軟體的經驗。

如果需要幫助，請閱讀'到何處尋求幫助',5Δ。

本文件的最新版始終可在http://www.debian.org/doc/maint-guide/和maint-guide 套件中找到，其繁體中文版也可在maint-guide-zh-tw 套件中獲得。

1.1 開發時需要的軟體

在開始之前，你需要確認你是否已經正確安裝了開發所需要的附加套件。注意這些套件不包含任何已經被標記essential 或required ——我們假設你已經安裝了它們。

以下這些套件已經隨標準的Debian 安裝而在系統中，所以你可能不需要再動手安裝它們(以及任何附加的依賴套件)。然而，你還是應該用aptitude show *package* 來檢一下。

[1] 本文件默認您使用squeeze 作業系統。如果你需要在lenny 系統上使用本文件記述的方法，則至少必須安裝backports 的dpkg 和debhelper 套件。

在開發系統中要安裝的最重要的套件是build-essential。當你安裝它的時候，將會自動把基本編譯環境裏所需的其他套件一併安裝。

對於某些類型的軟體，以上的就是所需要的全部。然而還有一組套件雖不是對於所有套件都必須，卻可能對你有用或被你的套件所需要：

- file - 這個小程式可以檢測檔案的類型。(參考file(1))
- patch - 這是一個非常有用的工具，它可以把diff 程式生成的差異清單檔案應用到原先的檔案上，從而生成一個補丁版本。(參考patch(1))
- perl - Perl 是當今的類Unix 系統中最常用的解譯型腳本語言之一，常被譽*Unix* 的瑞士軍刀。(參考perl(1))
- python - Python 是Debian 系統中另一個最常用的解譯型腳本語言，它擁有可圈可點的強大功能和十分清晰的語法。(參考python(1))
- autoconf、automake 和autotools-dev - 很多新程式使用configure 腳本和Makefile 檔案來幫助預處理程式。(參考"info autoconf"、"info automake")。autotools-dev 則用於保持指定的自動設定檔案最新，並帶有關於使用那些檔案的最佳方法的文件。
- dh-make 和debhelper - dh-make 是用於創建我們範例套件骨架所必須的，它會使用debhelper 中的一些工具來創建套件。他們不是創建套件所必須的，但對新維護人員而言，我們強烈推薦 使用。它使得整個過程極簡化，並易於在將來維護。(參考dh_make(1)、debhelper(1)、/usr/share/doc/debhelper/README) [2]
- devscripts - 此套件提供了一些非常好非常有用的腳本幫助維護者，但他們並非構建套件所必須。此套件所推薦或建議的套件都值得一看。(參考/usr/share/doc/devscripts/README.gz)
- fakeroot - 這個工具集使你可以在編譯過程中必要的時候用普通用户模擬root 用户環境。(參考fakeroot(1))
- gnupg - 使你可以使用數位簽章 簽署你的套件。這在當你想把它散佈給其他人時特別重要。如果你要把你的工作加入到Debian 發行版中，這是必須的步驟。(參考gpg(1))
- gfortran - GNU Fortran 95 編譯器，如果你的程式是用Fortran 編寫的則必須此套件完成編譯。(參考gfortran(1))
- gpc - GNU Pascal 編譯器。如果你的程式是用Pascal 寫的則需要此套件。值得一提的是fp-compiler，Free Pascal 編譯器(FPC)，也能夠很好地勝任。(參考gpc(1)、ppc386(1))
- xutils-dev - 一些通常用於X11 的程式，用於使用其巨集功能生成Makefile 檔案。(參考imake(1)、xmkmf(1))
- lintian - Debian 套件檢工具，使你可以在編譯套件後知道它是否有常見的錯誤，並對找到的錯誤進行解釋。(參考lintian(1)、/usr/share/doc/lintian/lintian.html/index.html)

[2] 還有幾個類似但更針對某一類軟體的套件，如dh-make-perl、dh-make-php 等。

- pbuilder - 這個套件提供了創建和維護chroot 環境。在此chroot 環境中編譯Debian 套件可以檢編譯依賴是否合適並避免FTBFS (Fails To Build From Source，從原始碼編譯失敗)的Bug。(參考pbuilder(8) 和pdebuild(1))
- patchutils - 此套件提供了一些可以幫助處理補丁的工具，如lsdiff、interdiff 和filterdiff 命令。
- quilt - 此套件幫助你管理一系列的補丁。它們被以邏輯堆疊的方式組織在一起。你可以apply (=push)、un-apply (=pop) 或簡單地更新它們然後再放入堆疊內。(參考quilt(1)、/usr/share/doc/quilt/README.Debian)
- git - 此套件提供了用於快捷處理大型項目的著名版本控制系統- git。它被廣泛用於各種開源項目，最著名的是Linux 核心項目。(參考git(1)、/usr/share/doc/git-doc/index.html)

以下是非常重要 的文件，你應該在讀本文件時同時參考它們：

- debian-policy - Debian Policy Manual (http://www.debian.org/doc/devel-manuals#policy) 包含了對Debian 軟體倉庫、作業系統設計事宜、檔案系統層級標準(FHS，Filesystem Hierarchy Standard)(講述每個檔案和目錄應該放在哪裏)等的描述。對於你而言，最重要的是它描述了套件要進入官方倉庫前必須滿足的條件。(參考/usr/share/doc/debian-policy/policy.html/index.html)
- developers-reference - Debian Developer's Reference (http://www.debian.org/doc/devel-manuals#devref) 描述了打包所需的包含技術細節在內的全部詳細訊息，如倉庫結構、如何重命名/丟棄/接手套件、如何進行NMU(非維護者上傳)、如何管理Bug 以及打包最佳經驗、何時向何處上傳等。(參考/usr/share/doc/developers-reference/)
- Autotools Tutorial (http://www.lrde.epita.fr/~adl/autotools.html) the GNU Build System known as the GNU Autotools (http://en.wikipedia.org/wiki/GNU_build_system) 中最重要的工具——Autoconf、Automake、Libtool 和gettext 提供了很好的文件。
- gnu-standards - 此套件包含了GNU 項目中的兩篇文件：GNU Coding Standards (http://www.gnu.org/prep/standards/html_node/index.html) 和Information for Maintainers of GNU Software (http://www.gnu.org/prep/maintain/html_node/index.html)。儘管Debian 不要求遵守這些規範，它們仍然很有幫助。(參考/usr/share/doc/gnu-standards/standards.html 和/usr/share/doc/gnu-standards/maintain.html)

若本文件所敘述的內容與Debian Policy Manual 或Debian Developer's Reference 有不符，則按照後兩者的要求進行，並向maint-guide 套件提交Bug 報告。

以上給出的簡短描述僅僅是了使你對這些套件有一個基本的印象。在繼續前請詳細閱讀每個程式的文件，至少了解其一般的用途和用法。現在看來這是一項耗時巨大的任務，但在接下來的工作中你將你閱讀了它們而感覺到非常 愉快。

1.2 基本術語

一共有兩種類型的套件。

- **原始碼套件**：原始碼套件包含了程式碼和數據，可以編譯成可執行的程式和格式化的文件。它通常是以`*.orig.tar.gz`、`*.debian.tar.gz` (或`*.diff.gz`) 和`*.dsc` 的組合形式出現的。也可能使用一些其他的格式和壓縮方法。
- **二進位套件**：二進位套件包含了可執行的程式和格式化的文件。它通常是以`*.deb` 檔案的形式提供給普通Debian 系統，以及`*.udeb` 檔案形式提供給Debian 安裝程式。

不要混淆程式原始碼和程式的原始碼套件這樣相似但有區別的概念！

在Debian 社群中有幾個常見的角色。

- **上游作者(upstream author)**：程式的最原始作者。
- **上游維護者(upstream maintainer)**：目前在上游維護程式程式碼的人。
- **維護者(maintainer)**：製作該程式的Debian 套件的人。
- **保證人(sponsor)**：檢內容後幫助維護者上傳套件到Debian 官方倉庫的人。
- **指導者(mentor)**：幫助維護者熟悉和深入打包的人。
- **Debian Developer** (DD)：Debian 社群的官方成員。DD 擁有向Debian 官方倉庫上傳的全部權限。
- **Debian Maintainer** (DM)：擁有對Debian 官方倉庫部分上傳權限的人。

Debian 中使用了幾種不同的版本號。

- **上游原始碼版本**：上游原始碼版本被指代*`version`*。
- **Debian 版本**：Debian 套件的版本被指代*`revision`*。
- **Debian 套件版本**：Debian 套件版本被按以下規則指代。
  - 對於Debian 創建的軟體，其原始碼套件的版本即上文所述的*`version`*。
  - 對於不是Debian 創建的軟體，其原始碼套件的版本*`version-revision`*。

如果你需要更詳細地瞭解術語，請閱讀其他手冊。

1.3 正式的Debian Developer

你不可能在一夜之間成**Debian Developer** (DD)，因一個DD 所需要的遠不只是技術能力，請不要因此感覺到挫敗或沮喪。如果你的軟體對其他人有用，你可以作一個**維護者(maintainer)** 通過一位**保證人(sponsor)** 來上傳軟體，或者申請成一位**Debian Maintainer**。詳見Debian New Maintainers (http://www.debian.org/devel/join/newmaint)。

請注意，要成一位正式的Debian Developer 並不一定要創建新套件。對已有軟體做出貢獻也是成正式的Debian Developer 的路徑。眼下正有很多套件正在等待一個好的維護者對其做出貢獻。(參考'選擇你的程式' ,7Δ)

1.4 到何處尋求幫助

在公共場所提出問題前請先閱讀各種文件(RTFM，Read The Fucking Manual)。包括但不限於/usr/share/doc/dpkg、/usr/share/doc/debian、/usr/share/doc/autotools-dev/README.Debian.gz、/usr/share/doc/*package*/* 檔案和本文件中提及的軟體的man/info 頁面。更多內容參考http://nm.debian.org/

製作小的測試套件是學習打包的好方法，仔細看維護較好的套件則是瞭解他人如何製作套件的最佳辦法。

如果在所有可用的文件和在線資源中都無法找到答案，你可以在Debian Mentors 郵件列表詢問(debian-mentors@lists.debian.org (http://lists.debian.org/debian-mentors/))。在那裏會有經驗更加豐富的Debian 開發人員幫助你，但在發問前請確定要閱讀已有的文件！

關於此郵件列表，參考http://lists.debian.org/debian-mentors/

當你收到一個Bug 報告後(沒錯，眞正的Bug 報告！)，你需要研究Debian Bug Tracking System (http://www.debian.org/Bugs/) (Debian Bug 跟蹤系統，BTS)並閱讀相關的文件以便高效處理這些報告。我推薦閱讀Developer's Reference, 5.8. 'Handling bugs' (http://www.debian.org/doc/manuals/developers-reference/pkgs.html#bug-handling)

如果仍然不能解決所有的問題，則到Debian Developers 郵件列表詢問(debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/))。關於此郵件列表，參考http://lists.debian.org/debian-devel/

即使以上的問題都解決了，也不能高興得太早。什麼？因幾個小時或幾天內就會有人開始使用你的套件，如果你犯了某些嚴重的錯誤，將被無數生氣的Debian 用户的郵件所轟炸.....只是開個玩笑。:-)

放鬆一點並準備好處理Bug 報告，在你的套件完全符合Debian 的各項規範前還需要付出很多努力，處理Bug 也是對你很好的鍛煉(再一次提醒，閱讀那些必須的文件 來了解詳情)。祝你好運！

Chapter 2

第一步

讓我們嘗試創建一個自己的套件(“收養”一個已存在的套件則更好)。

2.1 選擇你的程式

可能你已經選好了要製作的套件。第一件要做的事是使用aptitude 檢它是否已經存在於發行版倉庫中了。

你還可以通過套件搜索頁面(http://www.debian.org/distrib/packages) 和Debian Package Tracking System (http://packages.qa.debian.org/common/index.html) (Debian 套件跟蹤系統，PTS)詢套件訊息。

如果套件已經存在，直接安裝就好了！:-) 如果它是被**拋棄(orphaned)** 的——也就是說它的維護者被設置Debian QA Group (http://qa.debian.org/)，那麼你可以嘗試接手維護它(在Debian Bug report logs: Bugs in package wnpp in unstable (http://bugs.debian.org/wnpp) 檢它的維護者訊息)。你也可以“收養”維護者發出“Request for Adoption”(**RFA**)請求的套件。

有幾種看RFA 套件的方法：

- Work-Needing and Prospective Packages (http://www.debian.org/devel/wnpp/)
- Debian Packages that Need Lovin’ (http://wnpp.debian.net/)
- Browse WNPP bugs based on debtags (http://members.hellug.gr/serzan/wnpp/)

僅作一筆旁註，Debian 已經擁有了絕大多數類型軟體的套件，倉庫中套件的數量也遠遠超過了有上傳權限的貢獻者的數量。因此，已經在倉庫中的套件貢獻力量是非常受其他開發者歡迎的(且更容易獲得sponsorship)[1]。你可以通過非常多的方式來實現這一目的。

[1]當然，也一定總是有值得打包的新軟體。

- 接手被拋棄而仍然被很多人使用的套件。
- 加入打包小組(http://wiki.debian.org/Teams)。
- 某些常用的套件分類Bug。
- 在需要時準備QA 或NMU 上傳(http://www.debian.org/doc/developers-reference/pkgs.html#nmu-qa-upload)。

如果你有能力"收養"那個套件，下載(使用apt-get source *packagename* 或其他類似的工具)並分析它的程式碼。這篇文件沒有包含如何接手套件的詳細訊息，但幸運的是因接手套件時起始的工作已經有人完成，接手的工作應比從頭開始輕鬆得多。儘管如此，請繼續閱讀，下面給出的建議會對你很有幫助。

如果你要製作的套件是全新的，並且希望它出現在Debian 中，請按照以下的步驟進行：

- 首先，你必須知道這個軟體能夠工作，並且需要試用一段時間。
- 你必須在正在被處理的套件列表(http://www.debian.org/devel/wnpp/being_packaged) 上看以確定沒有其他人已經開始了工作。如果沒有，則提交一份ITP (Intent To Package) Bug 報告到wnpp pseudo 套件(可以使用reportbug)。如果已經有人在處理，則在需要的情況下聯繫他或他們。如果不需要你的幫助，就尋找其他你感興趣且沒有人維護的套件。
- 軟體**必須有一個許可證**。
  - 對於main 類的軟體，其許可證**必須滿足Debian Free Software Guidelines (DFSG) 的全部條件** (參考http://www.debian.org/social_contract#guidelines) 以及**不能依賴於main 之外的套件** 以滿足Debian Policy 中對於編譯和運行的要求。正如我們所期望的，main 是容納絕大多數自由軟體的場所。
  - 對於contrib 類的軟體，其許可證必須滿足DFSG 的全部條件，但可以依賴於main 之外的套件以完成編譯或運行。
  - 對於non-free 類的軟體，其許可證可以不滿足DFSG 中的一些條件，但至少需要是**可以散佈的**。

  如果你不清楚你的軟體應該分入哪一類，則把許可證文本發送到debian-legal@lists.debian.org (http://lists.debian.org/debian-legal/) 請他人提出意見。
- 程式**不應該** setuid 到root，甚至可以更好——不setuid 或setgid 到任何用户或組。
- 這個程式不應該是守護進程，也不應該安裝檔案到*/sbin 目錄或者以root 打開任何端口。
- 程式應當最終以二進位可執行程式的形式呈現，函式庫則較難處理。
- 程式應當有很好的文件，最好原始碼也容易理解(比如不混亂)。

- 你應該與程式的作者取得聯繫問一下他是否認程式應當被打包，以及他是否對Debian 友好。能夠詢問作者關於程式的任何問題是非常重要的，不要嘗試打包一個無人維護的軟體。

當然，這些問題都只是了安全，並試圖讓你不至於在比如setuid 守護進程等問題上犯錯誤而激怒了用户... 當你在打包方面有了更多經驗時，就可以處理這些套件了。但即便是富有經驗的開發人員在遇到疑惑時也會發郵件到debian-mentors@lists.debian.org (http://lists.debian.org/debian-mentors/) 郵件列表諮詢，那裏的人會很樂意提供幫助。

關於這些問題的更多幫助，看Debian Developer's Reference (http://www.debian.org/doc/devel-manuals#devref)。

2.2 獲得程式，並且試用它

第一件要做的事就是找到並下載原始的原始碼。我假定你已經從作者的首頁上找到了它的原始碼。Unix 下的自由軟體原始碼通常是以tar+gzip 格式(副檔名.tar.gz)或tar+bzip2 格式(副檔名.tar.bz2)的形式提供的。通常壓縮檔中包含了一個名*programname-version* 的子目錄，裏面包含了全部的原始碼。

如果原始碼的最新版需要通過諸如Git、Subversion 或CVS 這類VCS (Version Control System，版本控制系統)倉庫獲得，那麼你需要使用"git clone""cvs co"或"svn co"等命令獲得程式碼後，再使用tar 程式的--exclude-vcs 選項將其重新打包tar+gzip 格式。

如果你的程式原始碼是以其他形式提供的(比如檔案名以.Z 或.zip 結尾[2])，則使用合適的工具將其解壓縮，再重新打包。

作範例，我將使用一個名gentoo 的程式，它是一個X GTK+ 檔案管理器。[3]

在你的用户主目錄下創建一個子目錄，命名debian 或deb 或任意你覺得合適的其他名字(本例中使用~/gentoo)。把下載好的壓縮檔放在其中並解壓縮(使用tar xzf gentoo-0.9.12.tar.gz 命令)。要確定解壓縮過程中沒有發生錯誤，即便是有一點不恰當也不行，因在別人的系統上解壓縮這些檔案時，如果他們的工具並不忽略這些反常的現象就會出現問題。在你的終端螢幕上，應該看到如下的情形。

```
$ mkdir ~/gentoo ; cd ~/gentoo
$ wget http://www.example.org/gentoo-0.9.12.tar.gz
$ tar xvzf gentoo-0.9.12.tar.gz
$ ls -F
gentoo-0.9.12/
gentoo-0.9.12.tar.gz
```

現在又有了一個新的子目錄，名gentoo-0.9.12。進入該目錄並徹底 讀完其中的文件。通常情況下這些文件被命名README*、INSTALL*、*.lsm 或*.html。你必須找到關於如何正確

[2]當檔案副檔名不足以判斷檔案類型時，可以使用file 命令來判斷。
[3]需要注意的是，這個程式已經被打包好了，並且從寫這篇文件之初到現在它已經發生了很大的變化。

編譯和安裝程式的指導(最可能的是他們會默認你希望把程式安裝到/usr/local/bin 目錄，但事實上你不能那樣做，詳細的內容稍後將在'把檔案安裝到目的位置',16Δ 中說明)。

帶有Makefile 檔案的簡單程式可以很容易地使用make 來編譯。其中的一些還支持make check，可以完成一系列自身測試。編譯好後可以使用make install 來將程式安裝到目標目錄。

現在嘗試編譯和運行你的程式，確保它工作正常，且在安裝和運行時不會導致其他問題。

你還可以運行make clean (或更好的make distclean)來清理編譯目錄。有時還會帶有make uninstall 用以反安裝已經安裝了的檔案。

2.3 自由的可移植程式

非常多的自由軟體是使用C (http://en.wikipedia.org/wiki/C_(programming_language)) 和C++ (http://en.wikipedia.org/wiki/C++) 語言編寫的。其中的有很多使用Autotools 或CMake 來使其可以在不同平臺上移植。這些工具被用於生成Makefile 和其他必須的源檔案，然後這些程式可以使用正常的make; make install 來編譯和安裝。

Autotools (http://en.wikipedia.org/wiki/GNU_build_system) 是GNU 編譯系統工具，包括Autoconf (http://en.wikipedia.org/wiki/Autoconf)、Automake (http://en.wikipedia.org/wiki/Automake)、Libtool (http://en.wikipedia.org/wiki/GNU_Libtool) 和gettext (http://en.wikipedia.org/wiki/GNU_gettext)。你可以通過configure.ac、Makefile.am 和Makefile.in 等檔案來識別這種類型的原始碼。[4]

使用Autotools 的第一步是上游作者在程式碼中運行autoreconf -i -f 並把生成的檔案同原始碼一起散佈。

```
configure.ac-----+-> autoreconf -+-> configure
Makefile.am -----+        |      +-> Makefile.in
src/Makefile.am -+        |      +-> src/Makefile.in
                          |      +-> config.h.in
                       automake
                       aclocal
                       aclocal.m4
                       autoheader
```

編輯configure.ac 和Makefile.am 等檔案需要一些關於autoconf 和automake 的知識。參考info autoconf 和info automake。

使用Autotools 的流程第二部是用户獲得散佈的原始碼後在原始碼中運行./configure && make 並將其編譯成*binary* (二進位程式)。

[4]參考Autotools Tutorial (http://www.lrde.epita.fr/~adl/autotools.html) 和/usr/share/doc/autotools-dev/README.Debian.gz。

```
Makefile.in -----+                    +-> Makefile -----+-> make -> binary
src/Makefile.in -+-> ./configure -+-> src/Makefile -+
config.h.in -----+                    +-> config.h -----+
                              |
              config.status -+
              config.guess --+
```

你可以通過命令選項來改變Makefile 檔案中的很多設置，比如默認的檔案安裝位置(使用./configure -prefix=/usr)。

儘管不是必須的，還可以使用autoreconf -i -f 更新configure 和其他相關檔案，這樣做可能提高原始碼的兼容性。

CMake (http://en.wikipedia.org/wiki/CMake) 是另一個可選的編譯系統，可以通過CMakeLists.txt 檔案來識別它。

2.4 套件名稱和版本

開始打包時原始碼目錄應當是絕對乾淨(原始)的，或者直接使用剛剛解壓縮得到的原始碼。

了讓套件可以正確地編譯和構建，如果程式原有的名字不是完全小寫的則必須將其改成小寫，並且應當把原始碼以*packagename-version* 的形式命名。

如果程式的名稱超過一個單詞，把它改成一個單詞或縮寫成合適的形式。例如，程式"John's little editor for X"可以命名johnledx 或jle4x，或者隨便你認合適的其他名字。它需要符合一定的限制，比如長度不要超過20 和字符。

還要檢程式的版本號，它將被包含到套件版本內。如果手中的軟體版本號不是X.Y.Z 的形式，而是使用日期等形式。那麼可以繼續使用日期來作版本號，只要新版本程式的版本號能變得比老闆本大即可。最好能與上游使用相同的版本號，如果上游使用了諸如09Oct23 的格式，那麼最好將其轉換YYYYMMDD 的格式，此例中20091023，以便保證dpkg 程式在更新軟體時能正確處理新舊版本。[5]

還有一些程式根本就沒有版本號，這是你應該聯繫上游維護者，看他們是否使用了其他的版本跟蹤方法。

2.5 初始化Debian 套件

首先我們設置兩個環境變量，$DEBEMAIL 和$DEBFULLNAME，使多數Debian 維護工具能夠正確識別你用於維護套件的姓名和電子郵件地址。[6]

[5] Version 字符串可以用dpkg --compare-versions *ver1 op ver2* 命令比較。參考dpkg(1) man 手冊頁。

[6] 以下默認以Bash 作登陸shell。如果你使用其他的shell，例如Z shell，使用它們的設定檔案代替這裏提到的~/.bashrc。

```
$ cat >>~/.bashrc <<EOF
DEBEMAIL=your.email.address@example.org
DEBFULLNAME="Firstname Lastname"
export DEBEMAIL DEBFULLNAME
EOF
```

讓我們按照以下方法使用dh_make 來初始化Debian 套件。

```
$ . ~/.bashrc
$ cd ~/gentoo/gentoo-0.9.12
$ dh_make -f ../gentoo-0.9.12.tar.gz
```

當然，使用你的原始原始碼壓縮檔名替代範例中的檔案名。[7]參考dh_make(1) 以獲得詳細訊息。

此時會出現一些提示訊息。它會詢問你要創建那種類型的套件。Gentoo 是單個的二進位套件——編譯後僅生成一個二進位檔案，也因此只需要一個.deb 檔案——所以我們使用s 鍵選擇第一個選項，檢螢幕上顯示的訊息，無誤後按*ENTER* 確認。[8]

執行dh_make 後，上一級目錄中自動創建了一份上游tarball 的副本，名gentoo_0.9.12.orig.tar.gz，這個檔案和稍後介紹的debian.tar.gz 在一起滿足了Debian 非本地原始碼套件的要求。

```
$ cd ~/gentoo ; ls -F
gentoo-0.9.12/
gentoo-0.9.12.tar.gz
gentoo_0.9.12.orig.tar.gz
```

請注意gentoo_0.9.12.orig.tar.gz 這個檔案名的兩個關鍵特點：

- 套件名稱和版本是以底線_ 分隔的。
- .tar.gz 副檔名前有.orig。

你應該可以注意到debian 目錄下有了許多模板檔案。這些檔案將在‘debian 目錄中的必須內容’,21Δ 和‘debian 目錄下的其他檔案’,37Δ 一一解釋。你還應該明白，打包不是一個自動的過程，你還需要按照‘修改原始碼’,15Δ 中的方法來修改套件。此後，你還要按照‘構建套件’,49Δ 中敘述的合適的方法來編譯Debian 套件、按照‘檢套件中的錯誤’,57Δ 中的方法檢其中錯誤，最終依照‘上傳套件’,61Δ 的介紹將其上傳。我將對所有的這些步驟進行解釋。

再次強調，作一個新維護人員，不推薦創建複雜的套件，例如：

[7]如果上游原始碼提供debian 目錄和內容，運行dh_make 命令是要加上--addmissing 參數。新的3.0 (quilt) 原始碼格式十分強健，在這種情況下不會出現問題。你可能需要你的Debian 套件更新上游提供的debian 目錄內容。

[8]這裏有幾個候選項：s 單個二進位套件(Signle binary)、i 平臺獨立套件(Arch-Independent)、m 多個二進位套件(Multiple binary)、l 函式庫(Library)、k 核心模組(Kernel module)、n 核心補丁(Kernel patch)、b 使用cdbs。本文件重點介紹使用通過dh 命令使用debhelper 打包。本文件側重於單個二進位套件，也涉及平臺獨立和多二進位套件類型的套件。cdbs 套件提供了類似dh 的其他打包腳本框架，它並不在本文件的介紹範圍內。

- 多個二進位套件的套件；
- 函式庫套件；
- 核心模組套件；
- 核心補丁套件；
- 原始碼不是tar.gz 也不是tar.bz2 格式，或
- 原始碼tarball 中包含不可散佈的內容。

它們並非特別難於掌握，但確實需要更多的知識，所以我們不在這裏對所有情況進行描述。

如果你在修改過程中不小心刪除或弄壞了某些模板檔案，你可以使用dh_make 加上--addmissing 參數來將其還原。

更新一個已存在的套件可能比較複雜，因它可能使用舊的技術。請繼續在這裏學習最新的打包技術以便掌握必要的基本功。我們會在稍後的‘更新套件’,63Δ 中回頭講解如何處理這種情況。

Chapter 3

修改原始碼

請注意這裏沒有足夠的篇幅來描述修改上游原始碼的全部 細節，但是這裏介紹了基本的步驟和常見的問題。

3.1 設置quilt

quilt 程式Debian 工作提供了對源碼修改的基本記錄方法。由於我們的需求與默認設定相當的不同，我們把~/.quiltrc 設置成如下的樣子。[1]

```
d=. ; while [ ! -d "$d/debian" -a `readlink -e $d` != / ]; do d="$d/.."; done
if [ -d "$d/debian" ] && [ -z "$QUILT_PATCHES" ]; then
    # Debian packaging case and unset $QUILT_PATCHES
    QUILT_PATCHES=debian/patches
    QUILT_PATCH_OPTS="--unified-reject-files"
    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:di
    if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi
```

參考quilt(1) 和/usr/share/doc/quilt/quilt.html 來了解如何使用quilt。

3.2 修復上游Bug

讓我們假設你在上游的Makefile 檔案中找到了一個錯誤，其中的install: gentoo 應該是install: gentoo-target 才正確。

[1]你可以通過quilt --quiltrc /dev/null ... 來運行quilt 來禁用此設定檔案。

```
install: gentoo
        install ./gentoo $(BIN)
        install icons/* $(ICONS)
        install gentoorc-example $(HOME)/.gentoorc
```

讓我們使用quilt 修復它，並把它命名fix-gentoo-target.patch。[2]

```
$ mkdir debian/patches
$ quilt new fix-gentoo-target.patch
$ quilt add Makefile
```

現在將Makefile 修改如下的樣子。

```
install: gentoo-target
        install ./gentoo $(BIN)
        install icons/* $(ICONS)
        install gentoorc-example $(HOME)/.gentoorc
```

使用quilt 更新補丁以創建debian/patches/fix-gentoo-targe.patch 並添加描述。

```
$ quilt refresh
$ quilt header -e
... 描述補丁
```

3.3 把檔案安裝到目的位置

正常情況下，程式將其本身安裝在/usr/local。因它是保留給系統管理員(或用户)的私有位置，Debian 套件不可以使用該目錄，而是按照檔案系統層級標準(FHS (http://www.debian.org/doc/packaging-manuals/fhs/fhs-2.3.html)，/usr/share/doc/debian-policy/fhs/fhs-2.3.html)使用如/usr/bin 等目錄。

通常在自動編譯程式時使用make(1) 程式，執行make install 就把程式直接按照Makefile 檔案中的install target 安裝到指定的位置。了使Debian 能夠提供二進位套件，編譯系統將檔案安裝到一個臨時目錄中創建的檔案系統樹的鏡像中，而非直接安裝到實際的目標位置。

普通程式安裝過程和Debian 打包安裝過程的區別可以由debhelper 套件中的dh_auto_configure 和dh_auto_install 透明地處理。但程式必須滿足以下條件：

- Makefile 檔案遵循GNU 的規定支持$(DESTDIR) 變量(/usr/share/doc/gnu-standards/standards.html#Makefile-Conventions)。

[2] debian/patches 應當在你按照前面所述的步驟運行dh_make 時生成的。這個例子中我們新創建它，因假設的是在更新一個已存在的套件。

- 原始碼遵循檔案系統層級標準(FHS)。

使用GNU autoconf 的程式自動 遵守了GNU 的規定，這些程式的打包也就幾乎是自動的。通過這項特點和其他發式處理，debhelper 套件可以直接打包約90% 的套件而不需對編譯系統做出大的改變。所以打包也不是它看起來那樣複雜。

如果你需要修改Makefile 檔案，要確保其支持$(DESTDIR) 變量。默認情況下$(DESTDIR) 變量沒有設置並且默認使用程式選擇的安裝位置。打包腳本會將$(DESTDIR) 設置臨時目錄。

對於單個二進位套件dh_auto_install 將臨時目錄設置debian/*package*。[3] 臨時目錄中的全部檔案都將在用户安裝套件時被安裝到用户系統，唯一的區別是dpkg 會把檔案安裝到眞實的根目錄樹中。

請記住，即使你的程式正確安裝到了debian/*package*，仍然要考慮它可以在在實際的根目錄中安裝的情形，即從.deb 套件檔案安裝的情形。所以絕對不允許將諸如/home/me/deb/*package-version*/usr/share/*package* 的內容硬編碼到編譯系統中並裝入套件檔案。

以下是gentoo 的Makefile 檔案中的相關部分[4]：

```
# make install 時放置二進位檔案的位置
BIN      = /usr/local/bin

# make install 時放置圖示檔案的位置
ICONS    = /usr/local/share/gentoo
```

可以看到檔案被放到了/usr/local 下。把他們改變：

```
# make install 時放置二進位檔案的位置
BIN      = $(DESTDIR)/usr/bin

# make install 時放置圖示檔案的位置
ICONS    = $(DESTDIR)/usr/share/gentoo
```

什麼要放在那個目錄，而非其他的呢？因Debian 套件從不在/usr/local 下安裝檔案——那是保留給系統管理員使用的。在Debian 中這些檔案被直接放入/usr。

二進位檔案、圖示和文件等的更詳細位置均在檔案層級標準(FHS，參考/usr/share/doc/debian-policy/fhs/)中作出了詳盡描述。我建議閱讀相關章節以瞭解可能影響你套件的內容。

因此，我們應當把二進位檔案安裝到/usr/bin 而非/usr/local/bin，man 手冊頁則應放在/usr/share/man/man1 而非/usr/local/man/man1 等。注意到gentoo的Makefile

[3]對於多個二進位套件，dh_auto_install 將臨時目錄設置debian/tmp，而dh_install 命令則將檔案按照debian/*package-1*.install 和debian/*package-2*.install 等檔案的描述將debian/tmp 中的檔案分別裝入debian/*package-1* 和debian/*package-2* 等臨時目錄以創建*.deb 套件。

[4]這只是一個Makefile 的例子。如果Makefile 是通過./configure 命令生成的，修復該類的Makefile 的方法是通過dh_auto_configure 執行./configure，並帶上如--prefix=/usr 的默認選項。

裏沒有提及手冊頁，而按照Debian Policy 的要求，每個程式都應當有一個手冊頁，我們將在稍後製作一個並安裝到/usr/share/man/man1

有些程式不使用Makefile 變量定義路徑，這意味你可能需要去編輯C 程式原始碼來使他們使用正確的路徑。但是到哪裏去搜索，哪些才是呢？你可以通過以下的方法找到它們：

```
$ grep -nr -e 'usr/local/lib' --include='*.[c|h]' .
```

grep 會遞歸搜索整個原始碼樹並告訴你所有匹配項的檔案名和行號。

編輯那些檔案，用usr/lib 替換usr/local/lib。

```
$ vim '+argdo %s/usr\/local\/lib/usr\/lib/gce|update' +q \
      $(find . -type f -name '*.[c|h]')
```

小心不要弄亂其他部分的程式碼！:-)

此後你應該找到install target (通常搜索以install: 開頭的行即可)，並把所有的相關目錄以及在Makefile 頂部定義變量均做修改。

在你修復了上游的Bug 後，gentoo 的install target 應是這樣：

```
install: gentoo-target
        install ./gentoo $(BIN)
        install icons/* $(ICONS)
        install gentoorc-example $(HOME)/.gentoorc
```

讓我們把這些修改使用quilt 記錄到debian/patches/install.patch。

```
$ quilt new install.patch
$ quilt add Makefile
```

我們使用編輯器按照以下內容Debian 套件做修改：

```
install: gentoo-target
        install -d $(BIN) $(ICONS) $(DESTDIR)/etc
        install ./gentoo $(BIN)
        install -m644 icons/* $(ICONS)
        install -m644 gentoorc-example $(DESTDIR)/etc/gentoorc
```

你一定會注意到在其他命令前有了一個install -d 命令。原始的Makefile 檔案中沒有它，因通常情況下/usr/local/bin 和用到的其他目錄早已存在於系統，使用make install 時不用對此處理。然而因我們要向空目錄或尚不存在的目錄中安裝，我們必須創建每一個目錄。

我們還可以在末尾添加上其他的內容，比如上游作者有時會省略的附加文件：

```
        install -d $(DESTDIR)/usr/share/doc/gentoo/html
        cp -a docs/* $(DESTDIR)/usr/share/doc/gentoo/html
```

仔細檢後如果沒有問題，使用quilt 更新補丁以創建debian/patches/install.patch 檔案並添加對它的描述。

```
$ quilt refresh
$ quilt header -e
... 描述補丁
```

現在你有了一格系列的補丁。

1 修復上游Bug：debian/patches/fix-gentoo-target.patch

2 Debian 特有的打包修改：debian/patches/install.patch

進行任何不是Debian 特有的修改時，比如debian/patches/fix-gentoo-target.patch，一定要向上遊作者進行反饋，以便上游作者方便在下一版本中以使更多人受益。同時請記住在做不特別針對Debian 或Linux (甚至是Unix！)的修改時要使其可以移植，這會使你的修改更容易被接受。

注意你不一定要把debian/* 都提交到上游。

3.4 不一樣的函式庫名稱

還有另外一個常見的問題：不同平臺之間的函式庫常常是不同的。例如一個Makefile 中可能引了用一個不在Debian 上存在的函式庫。這種情況下我們需要將其修改Debian 中存在的、提供完全相同功能的函式庫。

如果你手中程式的Makefile(或Makefile.in)中有如下的行(且程式無法通過編譯)[5]：

```
LIBS = -lcurses -lsomething -lsomethingelse
```

我們將這個補丁命名debian/patches/ncurse.patch，內容是把curses 改ncurses。

```
$ quilt new ncurse.patch
$ quilt add Makefile
$ sed -i -e "s/-lcurses/-lncurses/g" Makefile
$ quilt refresh
$ quilt header -e
... 描述補丁
```

[5]作者已經意識到這不是一個最好的例子，因我們的libncurses 套件現在提供了一個libcurses.so 的符號鏈接，但作者還沒有想出更好的例子。非常歡迎提出建議:-)

Chapter 4

debian 目錄中的必須內容

運行dh_make 後，程式原始碼目錄下新出現了一個名debian 的子目錄。這個目錄中有很多用於定製套件行 的檔案，需要我們編輯。其中最重要的幾個是control、changelog、copyright 和rules，它們是所有套件必須的。

4.1 control 檔案

這個檔案包含了很多供dpkg、dselect、apt-get、apt-cache、aptitude 等套件管理工具提取訊息以進行管理時所使用的變量。這些變量均在Debian Policy Manual, 5 'Control files and their fields' (http://www.debian.org/doc/debian-policy/ch-controlfields.html) 中定義。

這是dh_make 我們創建的control 檔案：

```
 1 Source: gentoo
 2 Section: unknown
 3 Priority: extra
 4 Maintainer: Josip Rodin <joy-mg@debian.org>
 5 Build-Depends: debhelper (>= 7.0.50~)
 6 Standards-Version: 3.8.4
 7 Homepage: <insert the upstream URL, if relevant>
 8
 9 Package: gentoo
10 Architecture: any
11 Depends: ${shlibs:Depends}, ${misc:Depends}
12 Description: <insert up to 60 chars description>
13  <insert long description, indented with spaces>
```

(注：我它添加了行號。)

第1-6 行是原始碼套件的控制訊息。

第1 行是原始碼套件的名稱。

第2 行是該源碼包要進入發行版中的分類。

你可能已經注意到，Debian 倉庫被分幾個類別：main (自由軟體)、non-free (非自由軟體)以及contrib (依賴於非自由軟體的自由軟體)。在這些大的分類之下還有多個邏輯上的子分類，用以簡短描述套件的用途類別。admin 僅供系統管理員使用的程式，base 基本工具，devel 開發工具，doc 文件，libs 函式庫，mail 電子郵件閱讀器或郵件系統守護程式，net 網絡應用程式或網絡服務守護進程，x11 不屬於其他分類的X11 程式。更多的子分類見Debian Policy Manual, 2.4 'Sections' (http://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections) 和List of sections in 'sid' (http://packages.debian.org/unstable/)

我們將本例設置x11。(main/ 前綴是默認值，可以省略。)

第3 行描述了用户安裝此套件的優先級。具體說明參考Debian Policy Manual, 2.5 'Priorities' (http://www.debian.org/doc/debian-policy/ch-archive.html#s-priorities)

- optional 優先級適用於不與優先級required、important 或standard 的套件衝突的新套件。
- extra 優先級適用於與其他非extra 優先級套件衝突的新套件。

Section 和Priority 常被如aptitude 的前端所使用，以分類套件並計算默認值。一旦你把套件上傳到Debian，這兩項的值可以被倉庫維護人員修改，在被修改時你將收到郵件。

由於這是一個常規優先級的軟體，並不與其他套件衝突，我們將優先級改optional。

第4 行是維護者的姓名和電子郵件地址。請確保此處的值可以直接用於電子郵件頭的To 項。因一旦你將套件上傳至倉庫，Bug 跟蹤系統將使用其向你發送可能的Bug 報告郵件。避免使用逗號、"&"符號或括號。

第5 行中的Build-Depends 項列出了編譯此套件需要的套件。一些情況下你還需要這裏添加一行Build-Depends-Indep (參考Debian Policy Manual, 7.7 'Relationships between source and binary packages - Build-Depends, Build-Depends-Indep, Build-Conflicts, Build-Conflicts-Indep' (http://www.debian.org/doc/debian-policy/ch-relationships.html#s-sourcebinarydeps))。build-essential 依賴的套件，如gcc 和make 等，已經默認安裝而不需再寫到此處。如果你需要其他工具來編譯這個套件，請將它們加到這裏。多個套件應使用半形逗號分隔，繼續閱讀二進位套件依賴關係以增進對這些行的語法的理解。

- 對於所有在debian/rules 檔案中使用dh 命令的套件，必須在Build-Depends 中包含debhelper (>=7.0.50~) 以滿足Debian Policy 中對clean target 的要求。
- 對於生成一些標有Architecture: any 的套件，它們將被autobuilder 編譯。因autobuilder 僅安裝Build-Depends 後便執行debian/rules build 中的內容(參考'自動編譯系統' ,50Δ)，Build-Depends 需要列出所有必須的編譯依賴，而Build-Depends-indep 則很少使用。

- 對於僅有Architecture: all 的套件，Build-Depends-Indep 中應列出Build-Depends 中未列出的所有需要的套件，以便滿足Debian Policy 中對clean target 的要求。

如果你不知道應該使用哪一個，則使用Build-Depends 以保證安全。[1]

要找出編譯你的軟體所需的套件可以使用這個命令：

```
$ dpkg-depcheck -d ./configure
```

要手工地找到*/usr/bin/foo* 的編譯依賴，可以執行：

```
$ objdump -p /usr/bin/foo | grep NEEDED
```

對於列出的每個函式庫，例如libfoo.so.6，運行：

```
$ dpkg -S libfoo.so.6
```

接下來直接將相應的-dev 套件的名稱放到Build-Depends 項內。如果你使用ldd，它也會報告出間接的函式庫依賴關係，這可能造成填寫了某些不必要的依賴。

gentoo 需要xlibs-dev、libgtk1.2-dev 和libglib1.2-dev 才能編譯，所以我們將這些套件加在debhelper 之後。

第6 行是此套件所依據的Debian Policy Manual (http://www.debian.org/doc/devel-manuals#policy) 標準版本號。

在第7 行你可以放置上游項目的首頁地址。

第9 行是二進位套件的名稱。通常情況下與原始碼套件相同，但不是必須的。

第10 行描述了這個套件可以在哪些CPU 構架上編譯。我們將保持它any，因dpkg-gencontrol(1) 將在這個套件可以編譯的平臺上此處填寫合適的訊息。

如果你的套件是平臺獨立的(例如一個shell 或Perl 腳本，或一些文件)，將這項改變all，然後繼續閱讀'rules 檔案',28Δ 中關於使用binary-indep 指令替代binary-arch 來編譯套件的內容。

第11 行顯示了Debian 套件系統中最強大的特性之一。每個套件都可以和其他套件有各種不同的關係。除Depends 外，還有Recommends、Suggests、Pre-Depends、Breaks、Conflicts、Provides 和Replaces。

套件管理工具通常對這些關係採取相同的操作，如果不是，我將會詳細解釋。(參考dpkg(8)、dselect(8)、apt(8)、aptitude(1) 等。)

以下是各種依賴關係的含義：

[1]這種奇怪的情況是Debian Policy Manual, Footnotes 48 (http://www.debian.org/doc/debian-policy/footnotes.html#f48) 中詳細描述的一種特性。這不是由於在debian/rules 中使用dh 命令所致的，真正的原因是dpkg-buildpackage 如何運行。相同的情形也適用於Ubuntu 的自動編譯系統(https://bugs.launchpad.net/launchpad-buildd/+bug/238141)。

- Depends

  此套件僅當它依賴的套件均已安裝後纔可以安裝。此處寫明你的程式所必須的套件。

- Recommends

  這項中的套件不是嚴格意義上必須安裝纔可以保證程式運行。當用户安裝套件時，所有前端工具都會詢問是否要安裝這些推薦的套件。aptitude 和apt-get 會在安裝你的套件的時候自動安裝推薦的套件(用户可以禁用這個默認行)。dpkg 則會忽略此項。

- Suggests

  此項中的套件可以使你的程式更好地工作，但不是一定需要的。當用户安裝程式時，所有的前端程式將詢問是否安裝建議的套件。aptitude 可以被設定安裝軟體時自動安裝建議的套件。dpkg 和apt-get 將忽略此項。

- Pre-Depends

  此項中的依賴強於Depends 項。套件僅在預依賴的套件已經安裝後纔可以正常安裝並且正確設定 後纔可以正常安裝。在使用此項時應非常慎重，僅當在debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/) 郵件列表討論後才能使用。記住：根本就不要用這項。:-)

- Conflicts

  僅當所有衝突的套件都已經刪除後此套件纔可以安裝。當程式在某些特定套件存在時根本無法運行或存在嚴重問題時使用此項。

- Breaks

  此套件安裝後列出的套件將會受到損壞。通常Breaks 要附帶一個“版本號小於多少”的說明。這樣，套件管理工具將會選擇升級被損壞的特定版本的套件。

- Provides

  某些類型的套件會定義有多個可選的虛擬名稱。你可以在/usr/share/doc/debian-policy/virtual-package-names-list.txt.gz 檔案中找到完整的列表。如果你的程式提供了某個已存在的虛擬套件的功能則使用此項。

- Replaces

  當你的程式要替換其他套件的某些檔案，或是完整地替換另一個套件(與Conflicts 一起使用)。列出的套件中的某些檔案會被你的套件所覆蓋。

所有的這些項都使用相同的語法。它們是一個套件列表，套件名稱間使用半形逗號分隔。也可以寫出有多個可選的套件名稱，這些套件使用| 符號分隔。

這些項內還可以指定某些套件的版本號之間的關係。版本號在括號內，緊隨套件名稱之後，並在以下邏輯符號後寫清具體版本：<<、<=、=、>= 和>>，分別代表嚴格小於、小於或等於、嚴格等於、大於或等於以及嚴格大於。例如：

```
Depends: foo (>= 1.2), libbar1 (= 1.3.4)
Conflicts: baz
```

```
Recommends: libbaz4 (>> 4.0.7)
Suggests: quux
Replaces: quux (<< 5), quux-foo (<= 7.6)
```

你應當知道的最後一個特性是關於${shlibs:Depends}、${perl:Depends}、${misc:Depends}等。這些條目的值會由debhelper 的其他組件在dh_gencontrol(1) 運行時生成。

dh_shlibdeps(1) 會在程式編譯完成並被放入臨時檔案夾後掃描二進位檔案和函式庫檔案來確定動態函式庫的依賴關係，並檢測包含它們的套件名稱。例如libc6 或xlib6g。這些套件將被列在${shlibs:Depends} 的位置。

dh_perl(1) 所生成的套件列表將用於${perl:Depends} 的位置。

一些debhelper 命令可能會使生成的套件需要依賴於某些其他套件，這些套件將被列於${misc:Depends} 的位置。

說過這些以後，我們可以把Depends 項保持現狀不動，並在其下插入一行Suggests: file“，因gentoo 可以使用file 套件提供的某些特性。

第12 行是短描述。絕大多數人的螢幕是80 列寬，所以描述不應超過60 個字符。在這個例子裏我把它寫fully GUI-configurable, two-pane X file manager。

第13 行是長描述開始的地方。這應當是一段更詳細地描述套件的話。每行的第一個格應當留空。描述中不應存在空行，如果必須使用空行，則在行中僅放置一個. (半形句點)。同時，長描述後也不應有超過一行的空白。

接下來我們按照Developer's Reference, 6.2.5. 'Version Control System location' (http://www.debian.org/doc/manuals/developers-reference/best-pkging-practices.html#bpp-vcs) 中記敘的內容在第6 和第7 行之間添加Vcs-* 項。這裏我們假設gentoo 套件處於Debian Alioth Git 服務的git://git.debian.org/git/collab-maint/gentoo.git

到此止，我們做好了control 檔案：

```
 1 Source: gentoo
 2 Section: x11
 3 Priority: optional
 4 Maintainer: Josip Rodin <joy-mg@debian.org>
 5 Build-Depends: debhelper (>= 7.0.5), xlibs-dev, libgtk1.2-dev, libglib1.2-
 6 Standards-Version: 3.8.4
 7 Vcs-Git: git://git.debian.org/git/collab-maint/gentoo.git
 8 Vcs-browser: http://git.debian.org/?p=collab-maint/gentoo.git
 9 Homepage: http://www.obsession.se/gentoo/
10
11 Package: gentoo
12 Architecture: any
13 Depends: ${shlibs:Depends}, ${misc:Depends}
14 Suggests: file
15 Description: fully GUI-configurable, two-pane X file manager
```

```
16  gentoo is a two-pane file manager for the X Window System. gentoo lets th
17  user do (almost) all of the configuration and customizing from within the
18  program itself. If you still prefer to hand-edit configuration files,
19  they're fairly easy to work with since they are written in an XML format.
20  .
21  gentoo features a fairly complex and powerful file identification system,
22  coupled to a object-oriented style system, which together give you a lot
23  of control over how files of different types are displayed and acted upon
24  Additionally, over a hundred pixmap images are available for use in file
25  type descriptions.
26  .
29  gentoo was written from scratch in ANSI C, and it utilises the GTK+ toolk
30  for its interface.
```

(注：我它添加了行號。)

4.2 copyright 檔案

這個檔案包含了上游軟體的資源、版權以及許可證訊息。它的形式沒有在Debian Policy 中詳述，但其內容(Debian Policy Manual, 12.5 'Copyright information' (http://www.debian.org/doc/debian-policy/ch-docs.html#s-copyrightfile))。你還可以看DEP-5: Machine-parseable debian/copyright (http://dep.debian.net/deps/dep5/)。

dh_make 可以給出一個copyright 檔案的模板。在這裏我們使用--copyright gpl2 參數來獲得一個模板寫明gentoo 套件是發佈於GPL-2 許可證下。

你必須填寫上其餘的訊息，如你從何處獲得此軟體，實際的版權聲明和它們的許可證。對於常見的自由軟體許可證，如GNU GPL-1、GNU GPL-2、GNU GPL-3、LGPL-2、LGPL-2.1、LGPL-3、GNU FDL-1.2、GNU FDL-1.3、Apache-2.0 或Artistic 許可證，你可以直接將其指向存在於所有Debian 系統的/usr/share/common-licenses/ 目錄下的相關檔案。其他的許可證則必須包含完整的許可證文本。

簡言之，gentoo 的copyright 檔案如下所示：

```
 1 Format-Specification: http://svn.debian.org/wsvn/dep/web/deps/dep5.mdwn?op
 2 Name: gentoo
 3 Maintainer: Josip Rodin <joy-mg@debian.org>
 4 Source: http://sourceforge.net/projects/gentoo/files/
 5
 6 Copyright: 1998-2010 Emil Brink <emil@obsession.se>
 7 License: GPL-2+
 8
 9 Files: icons/*
10 Copyright: 1998 Johan Hanson <johan@tiq.com>
11 License: GPL-2+
```

```

Files: debian/*
Copyright: 1998-2010 Josip Rodin <joy-mg@debian.org>
License: GPL-2+

License: GPL-2+
 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.
 .
 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 GNU General Public License for more details.
 .
 You should have received a copy of the GNU General Public License along
 with this program; if not, write to the Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
 .
 On Debian systems, the full text of the GNU General Public
 License version 2 can be found in the file
 '/usr/share/common-licenses/GPL-2'.
```

(注：我它添加了行號。)

另外還可以參考ftpmasters 發送到debian-devel-announce 的HOWTO：http://lists.debian.org/debian-devel-announce/2006/03/msg00023.html

4.3 changelog 檔案

這是一個必須的檔案，它的格式在Debian Policy Manual, 4.4 'debian/changelog' (http://www.debian.org/doc/debian-policy/ch-source.html#s-dpkgchangelog) 中有詳細的描述。這種格式被dpkg 和其他程式用以獲得版本號訊息、適應的發行版和緊急程度。

對於你而言，詳細描述你所做出的更改也是很好且很重要的。它將幫助下載你的套件的人瞭解這個套件中是否有他們需要知道的事情。它會被作/usr/share/doc/gentoo/changelog.Debian.gz 保存在二進位套件中。

dh_make 創建了一個默認的檔案，這是它的樣子：

```
gentoo (0.9.12-1) unstable; urgency=low

 * Initial release (Closes: #nnnn)  <nnnn is the bug number of your ITP>
```

```
4
5  -- Josip Rodin <joy-mg@debian.org>  Mon, 22 Mar 2010 00:37:31 +0100
6
```

(注：我它添加了行號。)

第1 行是套件名、版本號、發行版和緊急程度。套件名必須與實際的原始碼套件名相同，發行版可以是unstable(甚至experimental)[2]，緊急程度則不要使用任何高於low 的內容。:-)

第3-5 行是一個很長的條目，記錄了你在這個Debian 版本中做出的修改(不是上游)修改——上游修改由上游作者創建另外一個檔案並維護，它們應被安裝/usr/share/doc/gentoo/changelog.gz)。假設你的ITP (Intent To Package，計劃打包)的Bug 號12345。新行必須插入在上一個以星號* 開頭的行的正下方。你可以使用dch(1) 完成這個工作，也可以使用普通的文本編輯器手工完成。

最後它會成以下的樣子：

```
1  gentoo (0.9.12-1) unstable; urgency=low
2
3   * Initial Release. Closes: #12345
4   * This is my first Debian package.
5   * Adjusted the Makefile to fix $(DESTDIR) problems.
6
7  -- Josip Rodin <joy-mg@debian.org>  Mon, 22 Mar 2010 00:37:31 +0100
8
```

(注：我它添加了行號。)

你可以在關於更新的'更新套件',63Δ 中瞭解更多關於changelog 的內容。

4.4 rules 檔案

現在我們需要看看dpkg-buildpackage(1) 用於實際創建套件的rules 檔案。這個檔案是一個Makefile，但不同於上游原始碼中的那個。和debian 目錄中的其他檔案不同，這個檔案被標記可執行。

4.4.1 rules 檔案中的target

每個rules 檔案都和其他的Makefile 一樣包含多個target 和其中指定如何處理原始碼的規則。Debian Policy Manual, 4.9 'Main building script: debian/rules' (http://www.debian.org/doc/debian-policy/ch-source.html#s-debianrules) 中對此做出了詳細解釋。

以下是對各target 的簡單解釋：

[2]有些人使用無效的發行版名稱，如UNRELEASED 以防止某些套件在共享的VCS 中修改時被意外上傳。

- clean：清理所有編譯的、生成的或編譯樹中無用的檔案。(必須)
- build：在編譯樹中將程式碼編譯程式並生成格式化的文件。(必須)
- install：把檔案安裝到debian 目錄內各個二進位套件構建的檔案樹。如果有定義，binary* target 則會依賴於此target。(可選)
- binary：創建所有二進位套件(是binary-arch 和binary-indep 的合併)。(必須)[3]
- binary-arch：在父目錄中創建平臺依賴(Architecture: any)的二進位套件。(必須)[4]
- binary-indep：在父目錄中創建平臺獨立(Architecture: all)的二進位套件。(必須)[5]
- get-orig-source：從上游站點獲得最新的原始原始碼套件。(可選)

你希望執行的target 規則可以通過命令行參數的方式觸發(例如./debian/rules build 或fakeroot make -f debian/rules binary")。target 名稱後你可以添加其依賴的target 或檔案。此後是以*TAB* 開頭的任意數量的名錄，每行一個。一個新的target 則在行首頂格開始。空行和以# 開頭的行將被視註釋而忽略。

可能你現在感到有些迷惑，在接下來講解dh_make 給出的默認的rules 檔案時會進行詳細的講解。你還應該閱讀info make 來了解更多訊息。

4.4.2 默認的rules 檔案

新版本的dh_make 會生成一個使用dh 命令的非常簡單但非常強大的默認的rules 檔案：

```
 1 #!/usr/bin/make -f
 2 # -*- makefile -*-
 3 # Sample debian/rules that uses debhelper.
 4 # This file was originally written by Joey Hess and Craig Small.
 5 # As a special exception, when this file is copied by dh-make into a
 6 # dh-make output file, you may use that output file without restriction.
 7 # This special exception was added by Craig Small in version 0.37 of dh-ma
 8
 9 # Uncomment this to turn on verbose mode.
10 #export DH_VERBOSE=1
11
12 %:
13         dh $@
```

[3]此target 被dpkg-buildpackage 用於'完整的(重)構建',49Δ 描述的過程中。
[4]此target 被dpkg-buildpackage -B" 用於'自動編譯系統',50Δ 描述的過程中。
[5]此target 被dpkg-buildpackage -A 使用。

(注：我添加了行號。實際的rules 檔案裏開頭的空格是TAB 填充的。)

可能在shell 或Perl 腳本中你已經對第一行的形式很熟悉了，它告訴作業系統這個檔案應使用/usr/bin/make 處理。

第10 行可以去掉註釋並將DH_VERBOSE 變量設置1。此後，dh 命令會輸出哪些dh_* 命令被執行了。你還可以在這裏添加一行export DH_OPTIONS=-v。此後每個dh_* 命令會輸出它所執行的具體命令。這可以幫助你瞭解這個簡單的rules 檔案所執行的內容並幫助調試問題。這個新的dh 是debhelper 工具的關鍵部分，其所有操作都可以讓你看到。

第12 和13 行是整個檔案的全部內容。百分號意味每個target 都只是調用dh + target 名稱的命令。[6] dh 命令是一個包裝腳本並按照參數執行相應的dh_*。[7]

- debian/rules clean 運行了dh clean，實際執行的命令：

```
dh_testdir
dh_auto_clean
dh_clean
```

- debian/rules build 運行了dh build，實際執行的命令：

```
dh_testdir
dh_auto_configure
dh_auto_build
dh_auto_test
```

- fakeroot debian/rules binary 執行了fakeroot dh binary; 實際執行的命令[8]：

```
dh_testroot
dh_prep
dh_installdirs
dh_auto_install
dh_install
dh_installdocs
dh_installchangelogs
dh_installexamples
dh_installman
```

[6]這使用了新的debhelper V7 特性。它的設計理念在Not Your Grandpa's Debhelper (http://joey.kitenet.net/talks/debhelper/debhelper-slides.pdf) 中有詳細解釋。這個幻燈片被debhelper 上游作者於DebConf9 時展示。在lenny 中，dh_make 會創建一個複雜很多的rules 檔案，其中帶有一系列dh_* 腳本命令來完成所需要的各個target。這個新的dh 命令更加簡單靈活。你仍然可以通過override_dh_* target 來對任意target 進行定義。參考'定製rules 檔案',32Δ。它僅是基於debhelper 套件而不像cdbs 那樣使編譯過程模糊不清。

[7]你可以通過dh --no-act *target* 或debian/rules -- '--no-act *target*' 來模擬編譯過程並看實際要執行的dh_* 程式。

[8]此處默認python-support 套件已經安裝在系統中。

```
dh_installcatalogs
dh_installcron
dh_installdebconf
dh_installemacsen
dh_installifupdown
dh_installinfo
dh_pysupport
dh_installinit
dh_installmenu
dh_installmime
dh_installmodules
dh_installlogcheck
dh_installlogrotate
dh_installpam
dh_installppp
dh_installudev
dh_installwm
dh_installxfonts
dh_bugfiles
dh_lintian
dh_gconf
dh_icons
dh_perl
dh_usrlocal
dh_link
dh_compress
dh_fixperms
dh_strip
dh_makeshlibs
dh_shlibdeps
dh_installdeb
dh_gencontrol
dh_md5sums
dh_builddeb
```

- fakeroot debian/rules binary-arch 執行了fakeroot dh binary-arch，其效果等同於fakeroot dh binary 並附加-a 參數於每個命令。
- fakeroot debian/rules binary-indep 執行了fakeroot dh binary-indep，這會運行幾乎和"fakeroot dh binary 一樣的命令，但dh_strip、dh_makeshlibs 和dh_shlibdeps 除外，其他命令則均附加-i 選項。

dh_* 的功能幾乎都可以通過它的名稱來了解。[9]有幾個命令有很值得一提的內容，在此處默認

[9]對於所有的dh_* 腳本實際做的工作以及它們可以使用的選項，請閱讀它們的man 手冊頁和debhelper 文件。

基於此Makefile 使用典型的編譯環境。[10]

- dh_auto_clean 在Makefile 中有distclean target 時執行以下命令。[11]

```
make distclean
```

- dh_auto_configure 在. /configure 存在時通常執行以下命令(省略了部分參數以方便此處閱讀)。

```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var ...
```

- dh_auto_build 通常使用以下命令執行Makefile 中的第一個target。

```
make
```

- dh_auto_test 通常在Makefile 存在且有test target 時執行以下命令。[12]

```
make test
```

- dh_auto_install 通常在Makefile 存在且有install target 時執行以下命令(進行了換行以便閱讀)。

```
make install \
  DESTDIR=/path/to/package_version-revision/debian/package
```

需要fakeroot 命令的都包含了dh_testroot。如果你沒有假裝root，將會報錯並退出。

關於dh_make 生成的rules 檔案，你應該知道的最重要的事是，它僅僅是一個建議。它對多數簡單的套件有效，但對於更復雜的則要大膽對其進行定製以滿足需要。只有一件事不應改變，就是它們的名字，這些名稱由Debian Policy 定義且被所有工具使用。

儘管install target 不是必須的，但也被支持。fakeroot dh install 的操作就像fakeroot dh binary一樣，但停止於dh_fixperms。

4.4.3 定製rules 檔案

有很多方法來定製使用新的dh 命令創建的rules 檔案。

dh $@ 命令可以按以下方式定製。[13]

[10] 這些命令支持其他例如setup.py 的編譯環境，可以通過在套件原始碼目錄執行dh_auto_build --list 列出。

[11] 實際上是依次考慮distclean、realclean 或clean 之一併執行。

[12] 實際上是依次考慮test 或check 並執行。

[13] 如果一個套件安裝了/usr/share/perl5/Debian/Debhelper/Sequence/*custom_name*.pm 檔案，你應當使用dh --with *custom-name* $@ 命令激活其功能。

- 添加dh_pysupport 命令支持。(對於Python 的最佳選擇。)[14]
  - 在Build-Depends 中添加python-support 套件。
  - 照常使用dh $@。(它被默認用)
  - 這會使用python-support 框架處理Python 模塊。
- 添加dh_pycentral 命令支持。
  - 在Build-Depends 中添加python-central 套件。
  - 使用dh --with python-central $@
  - 這樣會同時停用dh_pysupport 命令。
  - 這會使用python-central 框架處理Python 模塊。
- 添加dh_installtex 命令支持。
  - 在Build-Depends 中添加tex-common 套件。
  - 使用dh --with tex $@
  - 這樣會註冊Type 1 字體、斷句樣式或其他使用TeX 的格式。
- 添加dh_quilt_patch 和dh_quilt_unpatch 命令支持。
  - 在Build-Depends 中添加quilt 套件。
  - 使用dh --with quilt $@
  - 這會在你使用1.0 格式的原始碼套件時自動應用或解除debian/patches 目錄中的補丁。
  - 如果你使用新的3.0 (quilt) 原始碼套件格式時不需要這些。
- 添加dh_dkms 命令支持。
  - 在Build-Depends 中添加dkms 套件。
  - 使用dh --with dkms $@
  - 這會使核心模組套件正確使用DKMS。
- 添加dh_autotools-dev_updateconfig 和dh_autotools-dev_restoreconfig 命令支持。
  - 在Build-Depends 中添加autotools-dev 套件。
  - 使用dh --with autotools-dev $@
  - 這會自動更新或還原config.sub 和config.guess 檔案。
- 添加dh_autoreconf 和dh_autoreconf_clean 命令支持。

[14] 在dh_pysupport 和dh_pycentral 命令之間更推薦使用前者，不要使用dh_python 命令。

  - 在Build-Depends 中添加dh-autoreconf 套件。
  - 使用dh --with autoreconf $@
  - 這樣會在編譯時更新GNU 編譯系統檔案並在編譯後對其進行恢復。

- 添加bash 補全特性支持。
  - 在Build-Depends 中添加bash-completion 套件。
  - 使用dh --with bash-completion $@
  - 這會使用debian/*package*.bash-completion 中的設定檔案安裝bash 補全。

對於使用Autotools 的原始碼，使用dh --with autotools-dev --with autoreconf $@ 讓其使用最新的GNU 編譯系統。

很多由新的dh 命令觸發的dh_* 都可以通過修改debian 目錄中的設定檔案來對其行進行定製。參考'debian 目錄下的其他檔案',37Δ 和每個命令的man 手冊頁。

某些由新的dh 命令所觸發的dh_* 命令可能需要額外的參數使其一同執行或者跳過執行。對於這類情況，你可以在rules 檔案中創建一個override_dh_*foo* target 來使其完成你想要的dh_*foo* 命令。它的作用簡單說就是把運行的命令換成我。[15]

請注意dh_auto_* 命令所做的比上述簡化的步驟中介紹的內容更多。除了override_dh_auto_clean 外把上面的簡化命令寫成override_dh_* 中是不明智的，這樣會使得debhelper 的許多智能特性無法體現。

如果你希望把gentoo 套件的設定檔案安裝到/etc/gentoo 而非默認的/etc 目錄，你可以覆蓋dh_auto_configure 默認的使用的--sysconfig=/etc 參數，改向./configure 命令傳遞以下參數：[16]

```
override_dh_auto_configure:
        dh_auto_configure -- --sysconfig=/etc/gentoo
```

在-- 後給出的參數被附加於程式自動添加的默認參數後以對其進行覆蓋。使用dh_auto_configure 命令好於直接使用./configure，因這樣只覆蓋了--sysconfig 參數而保留了其他參數。

如果gentoo 的Makefile 需要指定build 作其編譯用的target[17]，你可以創建一個override_dh_auto_build target 來用它。

```
override_dh_auto_build:
        dh_auto_build -- build
```

[15] 在lenny 下，如果你希望更改某個dh_* 腳本的行，你需要在rules 中找到相應的行然後進行調整。

[16] gentoo 套件使用GNU 編譯系統，又稱Autotools。參考http://en.wikipedia.org/wiki/GNU_build_system

[17] 沒有參數的dh_auto_build命令將執行Makefile 中的第一個target。

這保證了$(MAKE) 會使用dh_auto_build 傳遞的所有參數並編譯處理build 這個target。

如果gentoo 的Makefile 需要指定packageclean 作其清理用的target，而非Debian 套件通常使用的distclean 或clean，你可以創建一個override_dh_auto_clean target 來用它。

```
override_dh_auto_clean:
        $(MAKE) packageclean
```

如果gentoo 的Makefile 包含了一個test target 但你不想在Debian 套件編譯過程中運行它，可以使用空的override_dh_auto_test target 來跳過它。

```
override_dh_auto_test:
```

如果gentoo 有某個不常見的上游changelog 檔案名FIXES，默認情況下dh_installchangelogs 不會安裝它。dh_installchangelogs 命令需要將FIXES 作它的參數來安裝它。[18]

```
override_dh_installchangelogs:
        dh_installchangelogs FIXES
```

如果你使用新的dh 命令時，還使用'rules 檔案中的target' ,28Δ 中除get-orig-source 的target ，會使得其效果難以預料。請儘量避免使用獨立的或預設的target，如果必須修改默認設置則酌情使用override_dh_*。

[18] debian/changelog 和debian/NEWS 總是會被自動安裝。程式會不分大小寫地搜索以下檔案名來檢測上游changelog：changelog、changes、changelog.txt和changes.txt。

Chapter 5

debian 目錄下的其他檔案

要控制debhelper 在構建套件過程中的行，可以在debian 目錄中放置可選的設定檔案。本章將會對這些檔案和它們的格式進行概述。請閱讀Debian Policy Manual (http://www.debian.org/doc/devel-manuals#policy) 和Debian Developer's Reference (http://www.debian.org/doc/devel-manuals#devref) 來了解更多內容。

dh_make 命令會在debian 目錄中創建一些設定檔案模板，它們的檔案名多帶有.ex 的後綴。其中的一些可能以套件名作前綴，如*package*。現在對它們進行一個大致的瞭解。

還有一些debhelper 的設定檔案模板dh_make 命令沒有直接創建。在這種情況下如果你需要它們，則要使用文本編輯器手工創建。

如果你希望或需要激活它們中的任意一個或多個，請按照下面的方法做。

- 如果模板檔案帶有.ex 或.EX 後綴，則重命名它去掉後綴。
- 按照需要，把帶有套件名的設定檔案的前綴改實際的二進位套件名，例如*package*。
- 修改模板檔案來滿足你的需要。
- 刪除不需要的模板檔案。
- 如果需要，修改control 檔案(參考'control 檔案',21Δ)。
- 如果需要，修改rules 檔案(參考'rules 檔案',28Δ)。

不帶有二進位套件名(例如*package*)前綴的debhelper 設定檔案則默認用於control 檔案中列出的第一個二進位套件，如默認情況下的install 檔案。如果需要其他二進位套件使用設定，則應在設定檔案名前加上二進位套件名，如*package-1*.install、*package-2*.install 等。

5.1 README.Debian 檔案

所有關於你的Debian 版本與原始版本間的額外訊息或差別都應在這裏記錄。

dh_make 創建了一個默認的檔案，以下是它的樣子：

```
gentoo for Debian
-----------------

<possible notes regarding this package - if none, delete this file>

 -- Josip Rodin <joy-mg@debian.org>, Wed, 11 Nov 1998 21:02:14 +0100
```

如果你沒有需要寫在這裏的東西，則刪除這個檔案。參考dh_installdocs(1)

5.2 compat 檔案

compat 檔案定義了debhelper 的兼容級別。目前你應當使用以下命令將其設置debhelper V7。

```
$ echo 7 > debian/compat
```

5.3 conffiles 檔案

系統管理中有件很惱人的事，就是當你付出了很多時間和經歷來自定義一個程式，但是升級後所有的修改都被覆蓋掉了。Debian 通過將設定檔案單獨標記來解決這個問題，當套件升級的時候，你將會被詢問是否要保留你的舊設定檔案。

從debhelper V3 開始，dh_installdeb(1) 將自動 識別所有在/etc 目錄下的檔案設定檔案。所以如果你的程式只有在那裏的設定檔案，則不需再指定它們。

如果你的程式使用設定檔案，但程式會自動對設定進行改寫，則最好也將其標記設定檔案，這樣dpkg 總是會要求用户校驗發生了的變更。

如果你正在打包的程式需要所有用户都自己修改/etc 目錄中的設定，有兩種常見的方法讓dpkg 不將其記錄設定檔案。

- 用*maintainer scripts* 在/etc 目錄中創建指向/var 相關目錄的符號鏈接。
- 用*maintainer scripts* 在/etc 目錄中生成一個檔案。

更多關於*maintainer scripts* 的訊息，參考'{post|pre}{inst|rm} 檔案',45Δ

5.4 *package.cron.** 檔案

如果你的套件需要有計劃運行的操作以保證正常工作，可以使用這個檔案進行設置。既可以設置常規的任務使其在每小時、每天、每星期、每月或其他情況或時間下運行。相應的檔案名是：

- cron.hourly - 安裝/etc/cron.hourly/*package*：每小時運行一次。
- cron.daily - 安裝/etc/cron.daily/*package*：每天運行一次，通常在早晨。
- cron.weekly - 安裝/etc/cron.weekly/*package*：每星期運行一次，通常在星期日早晨。
- cron.monthly - 安裝/etc/cron.monthly/*package*：每月運行一次，通常在該月第一天的早晨。
- cron.d - 安裝/etc/cron.d/*package* 指定的任意時間。

對於前幾個檔案，其格式shell 腳本。唯一不同的是*package*.cron.d，它的格式需要參照crontab(5)

注意這裏的操作不包含日誌旋轉，如果需要日誌旋轉，參考dh_installlogrotate(1)和logrotate(8)

5.5 dirs 檔案

這個檔案指定了我們需要，但正常安裝過程(dh_auto_install 觸發make install DESTDIR=...)裏沒有被安裝的目錄。通常這是由於Makefile 中存在問題。

install 檔案中列出的檔案不需要提前創建目錄，參考'install 檔案',41Δ

最好是先嚐試安裝，如果遇到了相關的問題再使用這個檔案。目錄名中不應有前導的/ 符號。

5.6 *package*.doc-base 檔案

如果你的套件在man 手冊頁和info 訊息文件外還有其他文件，你應該使用doc-base 檔案註冊它們，這樣用户可以使用例如dhelp(1)、dwww(1) 或doccentral(1) 的工具找到它們。

這通常包括HTML、PS 和PDF 檔案，放置在/usr/share/doc/*packagename*/。

以下是gentoo 的doc-base 檔案gentoo.doc-base 的樣子：

```
Document: gentoo
Title: Gentoo Manual
Author: Emil Brink
Abstract: This manual describes what Gentoo is, and how it can be used.
Section: File Management

Format: HTML
Index: /usr/share/doc/gentoo/html/index.html
Files: /usr/share/doc/gentoo/html/*.html
```

關於檔案格式的更多訊息，參考install-docs(8) 和/usr/share/doc/doc-base/doc-base.html/ 中的doc-base 手冊。

關於安裝附加文件的更多訊息，看'把檔案安裝到目的位置',16Δ。

5.7 docs 檔案

這個檔案制定了我們使用dh_installdocs(1) 安裝到臨時目錄的檔案名。

默認情況下它會加入程式碼目錄頂層的所有名BUGS、README*、TODO 等的檔案。

對於gentoo，我還加入了一些其他檔案：

```
BUGS
CONFIG-CHANGES
CREDITS
NEWS
README
README.gtkrc
TODO
```

5.8 emacsen-* 檔案

如果你的套件提供可以在安裝時編譯字節碼的Emacs 檔案，你可以使用這些檔案設置。

它們會被dh_installemacsen(1) 安裝到臨時目錄。

如果你不需要這些，就刪除它們。

5.9 *package*.examples 檔案

dh_installexamples(1) 會將列出的檔案和目錄作範例檔案安裝。

5.10 *package*.init 和*package*.default 檔案

如果你的套件需要在系統動時運行一個守護進程，那麼你顯然沒有按照我最初的建議做事，不是嗎？:-)

package.init 檔案會被安裝/etc/init.d/*package* 腳本。dh_make 創建的init.d.ex 是一個很好的骨架檔案。你可能要對其改名並做很多修改，同時還要提供檔案系統層級標準(FHS，參考/usr/share/doc/debian-policy/fhs/)所要求的檔案頭。它們會被dh_installinit(1) 安裝到臨時目錄。

package.default 檔案會被安裝/etc/default/*package*。該檔案會被init 腳本引用默認設置。多數情況下這個檔案被用於禁止運行一個守護程式、設置默認的標識或超時。如果你的腳本有特定的可設置特性，請把它們安裝到這個檔案，而不是init 腳本。

如果你的上游程式中包含了init 檔案，你可以選擇是否使用它。如果不使用，則創建相應的debian/*package*.init 檔案；如果上游的init 腳本很好且被安裝到正確的位置，你仍然需要設置rc* 符號鏈接。你需要按照以下的方法在rules 檔案中覆蓋dh_installinit：

```
override_dh_installinit:
        dh_installinit --onlyscripts
```

如果你不需要這些，就刪除它們。

5.11 install 檔案

如果你的套件需要標準的make install 沒有安裝的檔案，可以把檔案名和目標路徑寫入install 檔案，它們將被dh_install(1) 安裝。[1] 你需要首先檢要安裝的檔案是否有更有針對性的特定工具會對其進行安裝，例如範例、文件應寫在docs 檔案中安裝。

這個install 檔案每行安裝一份檔案，格式上先是相對於編譯目錄的源目錄，然後是一個空格，接下來是相對於安裝目錄的目標目錄。假設某個二進位檔案沒有被默認安裝，則應在install 檔案添加如下內容：

```
src/foo/mybin usr/bin
```

這意味安裝這個套件時將有一個二進位檔案/usr/bin/mybin

當然install 檔案也可以只包含相對的源路徑而不帶目標位置。這樣的格式通常用於使用*package-1*.install、*package-2*.install 等將大套件分割多個二進位套件的情況。

如果dh_install 命令沒有在當前目錄(或者你可能使用--sourcedir 參數指定的位置)找到檔案，它會回滾至使用debian/tmp 目錄。

5.12 *package*.info 檔案

如果你的套件有info 訊息頁，應該使用dh_installinfo(1) 安裝，要安裝的檔案列於*package*.info 檔案中。

[1] 這取代了已經廢棄的dh_movefiles(1) 命令，它本是用files 檔案所設定的。

5.13 {*package*.|source/}lintian-overrides 檔案

如果lintian 報告了一個Debian Policy 允許的例外的錯誤診斷，你可以使用*package*.lintian-overrides 或source/lintian-overrides 使其不再警告。請閱讀/usr/share/doc/lintian/lintian.html/index.html 並應嚴格避免濫用。

package.lintian-overrides 是對於名*package* 有效的設定，會由dh_lintian 命令安裝到usr/share/lintian/overrides/*package*

source/lintian-overrides 是針對原始碼套件的，不會安裝。

5.14 manpage.* 檔案

你的程式應該有man 手冊頁，如果它們沒有自帶則需要由你來創建。dh_make 命令創建了幾個man 手冊頁的模板。它們需要被妥善地編輯，並且要刪除未使用的模板。

5.14.1 manpage.1.ex 檔案

man 手冊頁通常是使用nroff(1) 的格式編寫的。manpage.1.ex 模板也是使用nroff 格式的。參考man(7) 手冊頁來簡要了解如何編輯這個檔案。

最終的man 手冊頁檔案名應當包含它所描述的程式名，所以我們把它從manpage 改名gentoo；檔案名還應以類似.1 的數字結尾作後綴，.1 代表它是一個用户命令的man 手冊頁。請確認這個章節號是否使用正確。以下是一個man 手冊章節簡表：:

章節	描述	註釋
1	用户命令	可執行命令或腳本。
2	系統調用	核心提供的函數。
3	函數函式庫調用	系統函式庫提供的函數。
4	特殊檔案	常見於 /dev
5	檔案格式	例如 /etc/passwd 的格式。
6	遊戲	遊戲或其他無足輕重的程式。
7	巨集包	例如 man 巨集。
8	系統管理	典型由 root 運行的程式。
9	核心函數	非標準調用和內部控制結構。

所以gentoo 的man 手冊頁應叫gentoo.1。如果原始程式碼中沒有gentoo.1 man 手冊頁，你需要重命名manpage.1.ex 模板gentoo.1 並按照範例和上游文件編輯它。

你也可以使用help2man 命令來通過程式的--help 和--version 參數的輸出創建man 手冊頁。[2]

[2]如果命令沒有info 訊息頁但在/usr/share/*package* 目錄中有其他文件，你應該手工編輯help2man 命令所創建的man 手冊頁。

5.14.2 manpage.sgml.ex 檔案

如果你希望使用SGML 而非nroff 格式編寫man 手冊頁，可以使用manpage.sgml.ex 模板。如果你要這樣，需要進行以下步驟：

- 重命名檔案類似gentoo.sgml 的名稱。
- 安裝docbook-to-man 套件。
- 在control 檔案的Build-Depends 中添加docbook-to-man 套件。
- 在rules 檔案裏添加override_dh_auto_build target：

```
override_dh_auto_build:
        docbook-to-man debian/gentoo.sgml > debian/gentoo.1
        dh_auto_build
```

5.14.3 manpage.xml.ex 檔案

如果你希望使用XML 而非SGML，可以使用manpage.xml.ex 模板。如果你要這樣，需要進行以下步驟：

- 重命名源檔案類似gentoo.1.xml 的名稱。
- 安裝docbook-xsl 套件和一個XSLT 處理器，例如xsltproc (推薦)。
- 在control 檔案的Build-Depends 中添加docbook-xsl、docbook-xml 和xsltproc 套件。
- 在rules 檔案裏添加override_dh_auto_build target：

```
override_dh_auto_build:
        xsltproc --nonet \
         --param make.year.ranges 1 \
         --param make.single.year.ranges 1 \
         --param man.charmap.use.subset 0 \
         -o debian/ \
 http://docbook.sourceforge.net/release/xsl/current/manpages/docbook.xs
        debian/gentoo.1.xml
        dh_auto_build
```

5.15 *package*.manpages 檔案

如果你的套件有man 手冊頁，你應該將它們列在*package*.manpages 檔案中以便dh_installman(1) 安裝。

要將doc/gentoo.1 安裝gentoo 的man 手冊頁，創建一個gentoo.manpages，內容如下：

```
docs/gentoo.1
```

5.16 menu 檔案

X Windows 系統用户通常有窗口管理器，並且帶有可定製的菜單用於動程式。如果他們安裝了Debian 的menu 套件，整個系統中所有已安裝軟體的一系列菜單將會被自動創建。

以下是dh_make 創建的默認的menu.ex 檔案：

```
?package(gentoo):needs="X11|text|vc|wm" \
        section="Applications/see-menu-manual"\
        title="gentoo" command="/usr/bin/gentoo"
```

冒號後的第一個欄位是needs，它指定了程式需要何種界面。修改此處列出的選項之一，例如text 或X11。

接下來是section，包含菜單和子菜單項。當前可用的章節列表[3]位於：/usr/share/doc/debian-policy/menu-policy.html/ch2.html#s2.1

title 欄位是程式的名稱，它可以用大寫字母開頭，只需保證它不要很長。

最後的command 欄位是運行此程式時使用的命令。

我們把檔案名修改menu，並修改其內容：

```
?package(gentoo): needs="X11" \
        section="Applications/Tools" \
        title="Gentoo" command="gentoo"
```

你還可以添加諸如longtitle、icon、hints 等欄位。參考dh_installmenu(1)、menufile(5)、update-menus(1) 和/usr/share/doc/debian-policy/menu-policy.html/ 瞭解更多訊息。

5.17 NEWS 檔案

dh_installchangelogs(1) 命令會安裝這個檔案。

[3] squeeze 的列表和以前的有了較大的調整。

5.18 {post|pre}{inst|rm} 檔案

postinst、preinst、postrm 和prerm 檔案[4] 被稱*maintainer scripts*。它們是放置於套件控制訊息區域，並由dpkg 在套件安裝、升級或反安裝時執行的腳本。

作一個新維護人員，你應當避免手工編輯*maintainer scripts*，因它們常存在各種問題。更多訊息請閱讀Debian Policy Manual, 6 'Package maintainer scripts and installation procedure' (http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html) 並看dh_make 給出的範例。

如果你不聽我的勸告，自己一個套件創建了*maintainer scripts*，你必須保證不僅測試**install** 和**upgrade**，還應測試**remove** 和**purge**。

升級到新版本應當是靜默的(已有用户應當只在發現舊的Bug 被修復或有新特性時注意到升級的變化)。

當更新必須以非靜默模式進行時(例如多個主目錄中的設定檔案都要改完全不同的結構時)，你應該考慮將套件設置到安全的默認設定(例如禁用服務)並按照Debian Policy 提供相應的文件(README.Debian 和NEWS.Debian)。不要在升級時使用*maintainer scripts* 觸發debconf 來打擾用户。

ucf 套件提供了類似*conffile* 的對不應標記*conffiles* 的檔案的保留機制。這可以將與*maintainer scripts* 相關的問題減少到最低。

這些*maintainer scripts* 是Debian 對於軟體系統的增強特性，它們解釋了**人們什麼選擇Debian**。你必須非常小心，保證人們不因此產生煩惱。

5.19 TODO 檔案

dh_installdocs(1) 命令會安裝這個檔案。

5.20 watch 檔案

watch 檔案的格式被詳述於uscan(1) man 手冊頁中。watch 檔案設定了uscan 程式(devscripts 中)以便監視你獲得原始原始碼的網站。它還被用於Debian External Health Status (DEHS) (http://wiki.debian.org/DEHS)。

以下是我寫入的內容：

```
# watch control file for uscan
version=3
http://sf.net/gentoo/gentoo-(.+)\.tar\.gz debian uupdate
```

[4]儘管我使用了Bash 短表達式在此處指出這些檔案是{post|pre}{inst|rm}，但仍然建議你在這些*maintainer scripts* 中儘可能地使用純粹的POSIX (非Bash)腳本來提供最佳的兼容性。

通常，在按照這個watch 檔案執行時，URL http://sf.net/gentoo" 會被下載，程式在其中搜搜<a href=...>。/ 後的基本名稱會按照Perl 正則表達式匹配(參考perlre(1)) gentoo-(.+)\.tar\.gz。在所有匹配的檔案裏，將會下載帶有最大版本號的，此後由uupdate 程式創建更新的原始碼樹。

儘管上述內容對於所有站點都適用，但SourceForge 下載服務(http://sf.net)仍是一個例外。當watch 中包含匹配Perl 正則表達式^http://sf\.net/ 的URL 時，uscan 程式會將其替換http://qa.debian.org/watch/sf.php/ 然後應用此規則。http://qa.debian.org/ 的URL 重定向服務器被設計用於提供一個穩定的重定向服務以滿足watch 檔案中的http://sf.net/*project*/*tar-name*-(.+)\.tar\.gz 形式，這樣解決了關於經常性URL 變更導致的問題。

5.21 source/format 檔案

在debian/source/format 中只包含一行，寫明瞭此原始碼套件的格式(看dpkg-source(1) 獲得完整列表)。在squeeze 後，它應該是以下二者之一：

- 3.0 (native) - Debian native 軟體
- 3.0 (quilt) - 其他所有軟體

全新的3.0 (quilt) 原始碼格式將所有修改使用quilt 補丁系列記錄到debian/patches。這些修改會在解壓原始碼套件時自動應用。[5] Debian 修改保存於debian.tar.gz 壓縮檔，其中包含了整個debian 目錄。這個新格式支持直接添加例如PNG 圖示等的二進位檔案。[6]

dpkg-source 解壓3.0 (quilt) 格式的源碼包時會自動應用所有列於debian/patches/series 的補丁。你可以使用--skip-patches 選項避免在解壓後自動應用補丁。

5.22 source/local-options 檔案

如果你希望使用版本控制系統(VCS)時，你可以創建一個分支(例如叫做upstream) 來跟蹤上游程式碼(對於Git 而言典型的是master 分支)，和另一個分支來跟蹤你的Debian 套件。對於後者，通常會將未應用補丁的上游程式碼和你的debian/* 檔案放在一起以便容易合併上游的新程式碼。

在編譯套件之後，原始碼通常會被保持在應用補丁後的狀態。你需要手工執行quilt pop -a 來解除這些補丁，然後再提交到master 分支。你可以向debian/source/local-options 檔案裏添加一行unapply-patches 來自動實現此目的。這個檔案不會被加入到生成的原始碼套件，它隻影響本地的編譯構建行。這個檔案裏還可以包含abort-on-upstream-changes (參考dpkg-source(1))。

[5] 參考DebSrc3.0 (http://wiki.debian.org/Projects/DebSrc3.0) 瞭解轉換到新的3.0 (quilt) 和3.0 (native) 原始碼格式時的總結考慮。

[6] 實際上新格式還支持多個上游tarball 和更多的壓縮方法，但這已超出本文件講述的範圍。

5.23 patches/* 檔案

舊的1.0 原始碼套件格式使用單一的大diff.gz 檔案保存debian 檔案夾和補丁。這樣的套件比較難於在事後檢和分析。這不是很好。

新的3.0 (quilt) 原始碼套件格式使用quilt 管理debian/patches/* 檔案。這些補丁和其他debian 目錄下的檔案被歸檔debian.tar.gz 檔案。因dpkg-source 不調用quilt 就可以處理3.0 (quilt) 源碼包格式的補丁，故不需要Build-Depends 於quilt 套件。[7]

quilt 命令在quilt(1) 中有詳細描述。它將對原始碼的修改維護於debian/patches 中一系列-p1 級別的補丁檔案中，debian 目錄外的檔案沒有任何修改。這些補丁的順序記錄於debian/patches/series 檔案中。你可以輕鬆地apply (=push)、un-apply (=pop) 和refresh 補丁。[8]

在'修改原始碼',15Δ 中，我們在debian/patches 創建了三個補丁。

因 Debian 補丁位於debian/patches，請確定按照'設置quilt',15Δ 中的方法正確設定quilt。

如果在之後有人提供了一個補丁*foo*.patch，對於3.0 (quilt) 原始碼套件格式可以很容易修改：

```
$ dpkg-source -x gentoo_0.9.12.dsc
$ cd gentoo-0.9.12
$ quilt import ../foo.patch
$ quilt push
$ quilt refresh
$ quilt header -e
... describe patch
```

存儲於新的3.0 (quilt) 原始碼套件格式的補丁必須有清晰的邊界。你應該通過quilt pop -a; while quilt push; do quilt refresh; done 來驗證這點。

[7]有多種不同的補丁維護方式被用於Debian 套件中，而quilt 系統則是其中的首選。其他系統還包括dpatch、dbs、cdbs 等。這些中的很多也將補丁保存於debian/patches/* 檔案中。

[8]如果你需要一個sponsor 上傳你的套件，這種情況下，清晰的套件分解和對你做出的更改的描述對於套件的審都非常重要。

Chapter 6

構建套件

現在我們已經構建套件做好了準備。

6.1 完整的(重)構建

保證完整的套件(重)構建能順利進行，你必須保證系統中已經安裝：

- build-essential 套件；
- 列於Build-Depends 欄位的套件(參考'control 檔案' ,21Δ)；
- 列於Build-Depends-indep 欄位的套件(參考'control 檔案' ,21Δ)。

然後在原始碼目錄中執行以下命令：

```
$ dpkg-buildpackage
```

這樣會自動完成所有從原始碼套件構建二進位套件的工作，包括：

- 清理原始碼樹(debian/rules clean)
- 構建原始碼套件(dpkg-source -b)
- 構建程式(debian/rules build)
- 構建二進位套件(fakeroot debian/rules binary)
- 使用gpg 簽署.dsc 檔案
- 使用dpkg-genchanges 和gpg 創建並簽署上傳用的.changes 檔案

這個過程中你需要做的就是輸入兩次你的GPG 私鑰密碼。

之後看上級目錄(~/gentoo)：

- gentoo_0.9.12.orig.tar.gz

  這是原始的原始碼tarball，由dh_make -f ../gentoo-0.9.12.tar.gz 命令重命名以符合Debian 的標準。

- gentoo_0.9.12-1.dsc

  這是一個從control 檔案生成的原始碼概要，可用於dpkg-source(1) 程式。這個檔案是使用GPG 簽署過的，以便別人可以確信它確實是你所提供的。

- gentoo_0.9.12-1.debian.tar.gz

  這個壓縮的Tar 歸檔包含你的debian 目錄。其他對於原始碼的修改都由quilt 補丁存儲於debian/patches 中。

  如果其他人想要重新構建你的套件，他們可以使用以上三個檔案很容易地完成。只需複製三個檔案，再運行dpkg-source -x gentoo_0.9.12-1.dsc。[1]

- gentoo_0.9.12-1_i386.deb

  這是你的二進位套件，可以使用dpkg 程式安裝或反安裝它，就像其他套件一樣。

- gentoo_0.9.12-1_i386.changes

  這個檔案描述了當前版本的套件中的全部變更，它被Debian FTP 倉庫維護程式用於安裝二進位和原始碼套件。它的是部分從changelog 和.dsc 檔案生成的，並使用GPG 簽名以便別人可以確信它確實是你提供的。

  隨你不斷完善這個套件，程式的行會發生變化，也會有更多新特性添加進來。下載你套件的人可以看這個檔案來快速找到有那些變化，Debian 倉庫維護程式還會把它的內容發表至debian-devel-announce@lists.debian.org (http://lists.debian.org/debian-devel-announce/) 郵件列表。

.dsc 和.changes 檔案中很長的數字串是其中提及檔案的MD5/SHA1/SHA256 校驗和。下載你套件的人可以使用md5sum(1)、sha1sum(1) 或sha256sum(1) 來進行覈對。如果校驗和不符，則說明檔案已被損壞或偷換。

6.2 自動編譯系統

Debian 通過在各種不同構架的計算機上運行buildd 守護進程組成的autobuilder network (http://www.debian.org/devel/buildd/) 來支持多種ports (http://www.debian.org/ports/)。儘管這些工作不需要全由你完成，我們現在仍然要大致看看在各種構架上你的套件是如果被構建的。[2]

對於Architecture: any 的套件，自動編譯系統重構建它們，它確保系統中已經安裝：

[1] 你可以使用--skip-patches 選項來在正常的是否操作後避免應用3.0 (quilt) 原始碼套件格式中的quilt 補丁。你也可以在正常解壓後使用quilt pop -a 恢復這些補丁對源碼的修改。

[2] 實際的自動編譯系統涉及到比這篇文件所敘述的複雜許多的預設，這些細節已經超出了本文件的範圍。

- build-essential 套件；
- 列於Build-Depends 欄位的套件(參考'control 檔案',21Δ)。

然後在原始碼目錄中執行以下命令：

```
$ dpkg-buildpackage -B
```

這樣會自動完成從原始碼套件構建平臺依賴二進位套件的工作，包括：

- 清理原始碼樹(debian/rules clean)
- 構建程式(debian/rules build)
- 構建平臺依賴二進位套件(fakeroot debian/rules binary-arch)
- 使用gpg 簽署.dsc 檔案
- 使用dpkg-genchanges 和gpg 創建並簽署上傳用的.changes 檔案

這就是你看到你的套件在其他平臺上可用的原因。

儘管通常的打包工作中Build-Depends-indep 欄位中列出的套件都需要安裝(參考'完整的(重)構建',49Δ)，但是在編譯平臺依賴二進位套件時它們無需在自動編譯系統上安裝。[3]通常打包和自動編譯系統的這種不同你指出如何考慮必須的套件應如何放在debian/control 檔案的Build-Depends 或Build-Depends-indep 欄位中(參考'control 檔案',21Δ)。

6.3 在上傳時包含orig.tar.gz 檔案

第一次向倉庫上傳套件時要包含orig.tar.gz 原始碼歸檔。如果changelog 中的最後一個條目是這個上游版本的第一次上傳，那麼應dpkg-buildpackage 命令傳遞-sa 選項。相反，-sd 選項會強制將原始的orig.tar.gz 原始碼歸檔排除在外。

6.4 debuild 命令

你可以使用debuild 命令來進一步自動化dpkg-buildpackage 的構建過程。參考debuild(1)

對debuild 命令的自定義可以通過/etc/devscripts.conf 或~/.devscripts 實現。我建議至少添加以下幾項：

[3]和在pbuilder 中不同，自動編譯系統使用的sbuild 所維護的chroot 不強制要求最小化的編譯系統，並可能保持很多套件始終安裝在其中。

```
DEBSIGN_KEYID="Your_GPG_keyID"
DEBUILD_LINTIAN=yes
DEBUILD_LINTIAN_OPTS="-i -I --show-overrides"
```

這樣套件會使用指定的ID 的GPG 私鑰簽署(對於sponsor 套件有好處)，並使用lintian 命令對套件做最大程度的檢。

在普通用户帳號中可以使用以下這樣簡單的命令清理原始碼並重構建套件：

```
$ debuild
```

請注意用於包含orig.tar.gz 原始碼歸檔的dpkg-buildpackage 參數-sa 可以這樣傳遞：

```
$ debuild -sa
```

還可以簡單地清理原始碼樹：

```
$ debuild clean
```

6.5 pbuilder 套件

對於使用淨室(chroot)編譯環境來驗證編譯依賴而言，pbuilder 套件是非常有用的。[4]它確保了套件在不同構架上的sid 發行版環境中的自動編譯器中能成功編譯，避免了總是被歸類於RC (Release Critical，影響發佈)的嚴重FTBFS (Fails To Build From Source，從原始碼編譯失敗) Bug。參考http://buildd.debian.org/ 以更多瞭解Debian 套件自動編譯系統。

我們通過以下操作來定製pbuilder 套件。

- 設置/var/cache/pbuilder/result 對用户可寫。
- 創建一個對用户可寫的目錄保存鉤子腳本，例如*/var/cache/pbuilder/hooks*
- 在~/.pbuilderrc 或/etc/pbuilderrc 中添加以下內容：

  ```
  AUTO_DEBSIGN=yes
  HOOKDIR="/var/cache/pbuilder/hooks"
  ```

這使你可以使用~/.gnupg/ 目錄中的GPG 私鑰簽署生成的套件。

使用以下命令初始化pbuilder chroot 系統：

```
$ sudo pbuilder create
```

[4]pbuilder 套件仍然在改進中，你可以看最新的官方文件來檢實際的設定狀況。

如果你已經創建了原始碼套件，在包含*foo*.orig.tar.gz、*foo*.debian.tar.gz和*foo*.dsc 檔案的目錄中執行下面的命令來更新pbuilder chroot 系統以便執行構建。

```
$ sudo pbuilder --update
$ sudo pbuilder --build foo.dsc
```

請注意用於包含orig.tar.gz 原始碼歸檔的dpkg-buildpackage 參數-sa 可以這樣傳遞：

```
$ sudo pbuilder --build --debbuildopts "-sa" foo.dsc
```

新構建的套件將被放置於/var/cache/pbuilder/result/ 且屬主不是root。

如果你已經更新了原始碼樹但沒有生成對應的原始碼套件，在存放debian 目錄的目錄裏執行：

```
$ sudo pbuilder --update
$ pdebuild
```

請注意用於包含orig.tar.gz 原始碼歸檔的dpkg-buildpackage 參數-sa 可以這樣傳遞：

```
$ pdebuild --debbuildopts "-sa"
```

你可以使用pbuilder --login --save-after-login 命令登錄到這個chroot 環境中並按照需要對其進行設定。通過^D (Control-D)離開這個shell 時環境會被保存。

最新版的lintian 命令可以通過設置鉤子腳本*/var/cache/pbuilder/hooks*/B90lintian 在chroot 環境中運行。腳本內容如下：[5]

```
#!/bin/sh
set -e
install_packages() {
    apt-get -y --force-yes install "$@"
    }
install_packages lintian
echo "+++ lintian output +++"
su -c "lintian -i -I --show-overrides /tmp/buildd/*.changes" - pbuilder
# use this version if you don't want lintian to fail the build
#su -c "lintian -i -I --show-overrides /tmp/buildd/*.changes; :" - pbuilder
echo "+++ end of lintian output +++"
```

[5]此處默認HOOKDIR="/var/cache/pbuilder/hooks"。你可以在/usr/share/doc/pbuilder/examples 目錄中找到很多鉤子腳本的例子。

sid 編譯套件需要使用sid 環境。在現實中sid 存在很多問題以至於你不願意將整個系統都遷移到其上。pbuilder 可以在這種情況下很好地解決問題。

你可能需要通過stable-proposed-updates、stable/updates 等升級你的stable 套件。[6]對於這類情況，"我正在運行sid 系統"並不是你不它們進行升級的充分理由。pbuilder 套件可以幫助你使用到相同CPU 構架下幾乎所有Debian 和Debian 衍生版系統。

參考http://www.netfort.gr.jp/~dancer/software/pbuilder.html、pdebuild(1)、pbuilderrc(5) 和pbuilder(8)

6.6 git-buildpackage 和相似命令

如果你的上游對原始碼使用版本控制系統(VCS (http://www.debian.org/doc/manuals/debian-reference/ch10.en.html#_version_control_systems))，你也應該考慮使用它們。這會使得合併和提取上游補丁更加簡單。在Debian 有多個不同VCS 設計的腳本套件來協助Debian 套件構建。

- git-buildpackage：幫助維護Git 倉庫中套件的套件。
- topgit：一個Git 補丁隊列管理器。
- svn-buildpackage：幫助維護Subversion 倉庫中套件的程式。
- cvs-buildpackage：CVS 原始碼樹設計的Debian 套件腳本集。

這些套件需要自動化 編譯套件的資深打包人員提供了相比於手工使用quilt 命令更加整潔的工作環境，本文件裏不會對此進行更多敘述。[7]

6.7 快速重構建

對於很大的套件，在調試debian/rules 的過程中你可能不想對整個套件進行重構建。僅用於測試目的，你可以不重新構建原始碼套件而使用以下的方法創建.deb 檔案[8]：

[6]升級你的stable 套件有規定限制。

[7]對於資深打包人員，有以下幾個可用的資源：

- 位於/usr/share/doc/git-buildpackage/manual-html/gbp.html 的Building Debian Packages with git-buildpackage
- "debian packages in git (https://honk.sigxcpu.org/piki/development/debian_packages_in_git/)"
- "Using Git for Debian Packaging (http://www.eyrie.org/~eagle/notes/debian/git.html)"
- "Using TopGit to generate quilt series for Debian packaging (http://git.debian.org/?p=collab-maint/topgit.git;a=blob_plain;f=debian/HOWTO-tg2quilt;hb=HEAD)"
- "git-dpm: Debian packages in Git manager (http://git-dpm.alioth.debian.org/)"

[8]常規情形下被設定好的環境變量在此時不會被自動設置。永遠不要將使用這個**快速** 方法構建的套件上傳到任何地方。

```
$ fakeroot debian/rules binary
```

或者可以僅看它是否能通過編譯：

```
$ fakeroot debian/rules build
```

一旦完成了調試，記住要按照前面所將的正常過程重構建你的套件。你可能無法正常上傳用此種方法構建的.deb 檔案。

Chapter 7

檢套件中的錯誤

在上傳套件到公共倉庫前，你還需要自己做一些檢套件錯誤的例行工作。

不僅在自己的機器上測試總是一個好主意。你必須謹慎地對待以下敘述的測試中顯示的任何一個警告或錯誤訊息。

7.1 檢驗套件安裝過程

你必須測試你的套件看是否存在安裝問題。`debi(1)` 命令可以幫助你測試生成的二進位套件。

```
$ sudo debi gentoo_0.9.12-1_i386.changes
```

你必須使用從Debian 倉庫下載的`Contents-i386` 檔案校驗是否在不同系統上存在檔案覆蓋的問題。`apt-file` 命令正適合完成這個任務。如果存在問題，請通過與受影響的套件合作使用alternatives 機制來避免實際問題(參考`update-alternatives(1)`)或在`debian/control` 檔案中設置`Conflicts` 條目。

7.2 檢驗*maintainer scripts*

所有*maintainer scripts*，包括`preinst`、`prerm`、`postinst` 和`postrm` 檔案，除非是由`debhelper` 程式自動生成的，都是測試的重點。如果你是新維護人員則不要使用它們(參考'`{post|pre}{inst|rm}` 檔案',45Δ)。

如果套件使用了這些需要嚴格測試的*maintainer scripts*，請確保不僅測試install，還要測試remove、purge 和upgrade。很多*maintainer scripts* 的Bug 都顯現於反安裝或徹底刪除套件時。使用`dpkg` 命令來測試它們。

```
$ sudo dpkg -r gentoo
$ sudo dpkg -P gentoo
$ sudo dpkg -i gentoo_version-revision_i386.deb
```

整個測試過程應按照以下序列操作：

- 如果可能，安裝前一個版本的套件；
- 從前一個版本升級套件；
- 降級套件到前一個版本(可選)；
- 徹底刪除該套件；
- 全新安裝該套件；
- 反安裝該套件；
- 再次安裝該套件。
- 徹底刪除該套件；

如果這是你的第一個套件，你應該使用其他版本號創建一個測試用的套件來完成升級測試，這樣可以避免將來的問題。

請牢記如果你的套件已經在以往版本的Debian 中發佈，人們通常會從最近發佈的Debian 發佈裏的版本升級，所以也要測試從那個版本升級到當前的版本。

儘管降級沒有被正式支持，支持它也總是不錯的。

7.3 lintian 套件

使用lintian(1) 檢你的.changes 檔案。lintian 命令會運行很多測試腳本來檢常見的打包錯誤。[1]

```
$ lintian -i -I --show-overrides gentoo_0.9.12-1_i386.changes
```

當然，要使用你自己套件所生成的.changes 檔案的檔案名。lintian 命令的輸出常帶有以下幾種標記：

- E: 代表錯誤：確定違反了Debian Policy 或是一個肯定的打包錯誤。
- W: 代表警告：可能違反了Debian Policy 或是一個可能的打包錯誤。
- I: 代表訊息：對於特定打包類別的訊息。
- N: 代表註釋：幫助你調試的詳細訊息。
- O: 代表已覆蓋：一個被lintian-overrides 檔案覆蓋的訊息，但由於使用--show-overrides 選項而顯示。

[1]如果你按照'debuild 命令',51Δ 中的敘述定義了/etc/devscripts.conf 或~/.devscripts 檔案，就不需要再添加lintian 選項-i -I --show-overrides。

對於警告，你應該改進套件或者檢 警告是否的確無意義。如果確定沒有意義，則按照'{*package*.|source/}lintian-overrides 檔案' ,42Δ 中的敘述使用lintian-overrides 檔案將其覆蓋。

你可以使用debuild(1) 或pdebuild(1) 將dpkg-buildpackage 與在其後運行lintian 合併到一個命令裏。

7.4 debc 命令

你可以使用debc(1) 命令列出一個二進位Debian 套件中的檔案。

```
$ debc package.changes
```

7.5 debdiff 命令

你可以使用debdiff(1) 命令比較兩個Debian 原始碼套件的內容。

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

你還可以使用debdiff(1) 命令比較兩個Debian 二進位套件的檔案列表。

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

這個命令對於檢原始碼套件中哪些檔案被修改了非常有用，還可以發現二進位套件中是否有檔案被意外替換或刪除，或在更新二進位套件時檢是否有其他沒有被提及的變更。

7.6 interdiff 命令

你可以使用interdiff(1) 命令比較兩個diff.gz 檔案。這對於更新使用舊的1.0 原始碼格式的套件時，檢是否有意外的變更非常有用。

```
$ interdiff -z old-package.diff.gz new-package.diff.gz
```

7.7 mc 命令

很多檔案檢操作可以通過使用類似mc(1) 的檔案管理器來完成，它可以幫助你直接看*.deb 檔案的內容，除此之外還可以用於*.udeb、*.debian.tar.gz、*.diff.gz 和*.orig.tar.gz 檔案。

還要檢在二進位套件和原始碼套件中是否有不需要的檔案或者空檔案。這些檔案經常沒有被正確清理，如果存在這種情況，要調整rules 檔案進行處理。

Chapter 8

上傳套件

當你完成了對套件的徹底測試，在技術上你已經滿足了申請Debian 新維護人員的要求，詳細的描述參考http://www.debian.org/devel/join/newmaint

8.1 上傳到Debian 倉庫

當你成正式的開發人員，你會需要把套件上傳到Debian 倉庫。你可以手工進行這項工作，但使用例如dupload(1) 或dput(1) 的自動化工具可以幫你更好地完成這項操作。在此我們將敘述如何使用dupload 操作。[1]

首先需要設置dupload 的設定檔案。你既可以編輯系統級的/etc/dupload.conf 檔案，也可以使用自己的~/.dupload.conf 檔案覆蓋一些需要修改的設置。

你可以閱讀dupload.conf(5) man 手冊頁來了解各選項的含義。

$default_host 選項決定了默認使用哪個上傳隊列，anonymous-ftp-master 是最基本的一個，但你很可能希望改用其他的。

連接到互聯網後，可以使用以下命令上傳你的套件：

```
$ dupload gentoo_0.9.12-1_i386.changes
```

dupload 會檢檔案的MD5/SHA1/SHA256 校驗和是否與.changes 檔案中的相匹配，如果不匹配它會做出警告。你應按照如'完整的(重)構建',49Δ 所述來重構建套件使得它可以被正常上傳。

如果你在ftp://ftp.upload.debian.org/pub/UploadQueue/ 遇到了上傳問題，你可以通過ftp 手工上傳一個用GPG 簽署的*.commands 檔案到那裏。[2]例如使用hello.commands：

[1] dput 套件提供了更多的特性，相比於dupload 也越來越受歡迎。它使用/etc/dput 檔案作全局設定檔案、~/.dput.cf 作用户設定檔案。它也直接支持Ubuntu 相關的服務。

[2] 參考ftp://ftp.upload.debian.org/pub/UploadQueue/README。另外你也可以使用來自dput 的dcut 命令。

```
-----BEGIN PGP SIGNED MESSAGE-----

Uploader: Foo Bar <Foo.Bar@example.org>
Commands:
 rm hello_1.0-1_i386.deb
 mv hello_1.0-1.dsx hello_1.0-1.dsc

-----BEGIN PGP SIGNATURE-----
Version: 2.6.3ia

iQCVAwUBNFiQSXVhJ0HiWnvJAQG58AP+IDJVeSWmDvzMUphScg1EK0mvChgnuD7h
BRiVQubXkB2DphLJW5UUSRnjw1iuFcYwH/lFpNpl7XP95LkLX3iFza9qItw4k2/q
tvylZkmIA9jxCyv/YB6zZCbHmbvUnL473eLRoxlnYZd3JFaCZMJ86B0Ph4GFNPAf
Z4jxNrgh7Bc=
=pH94
-----END PGP SIGNATURE-----
```

看mentors.debian.net (http://mentors.debian.net/cgi-bin/welcome) 來了解對於非Debian Developer 可用的公共上傳位置。

訪問Debian Reference, 2.7.12. 'Small public package archive' (http://www.debian.org/doc/manuals/debian-reference/ch02.en.html#_small_public_package_archive) 來看一個創建與現代的安全APT 系統兼容的公共套件倉庫的例子。

8.2 跳過的上傳

如果你在debian/changelog 創建了多個條目並跳過了上傳，你必須創建一個對應的*_.changes 檔案，其中包含自上次上傳以來的全部變更記錄。

這可以通過傳遞-v 選項給例如dpkg-buildpackage、debuild、pdebuild 等軟體構建程式來完成。

Chapter 9

更新套件

一旦你發佈了一個套件，在之後的某個時間裏就需要對它進行更新。

9.1 新的Debian 版本

假設針對你的套件報告的Bug 號#54321，它描述了一個你可以解決的問題。要創建套件的一個新Debian 版本，你需要：

- 如果要將它記錄於新的補丁中，這樣做：
  - quilt new *bugname.patch* 設置補丁名稱；
  - quilt add *buggy-file* 聲明檔案將被更改；
  - 修正套件程式碼中的上游Bug；
  - quilt refresh 將修改記錄到*bugname.patch*；
  - quilt header -e 添加對它的描述；
- 如果是更新一個已存在的補丁，這樣做：
  - quilt pop *foo.patch* 重現已存在的*foo.patch*；
  - 修正舊的*foo.patch* 中的問題；
  - quilt refresh 更新*foo.patch*；
  - quilt header -e 更新對它的描述；
  - while quilt push; do quilt refresh; done 應用所有補丁以確保它們邊界清晰；
- 在Debian changelog 檔案的頂部添加一個條目。例如可以使用dch -i 或用dch -v *version-revision* 來指定版本，然後用你喜歡的編輯器插入訊息。[1]

[1]要獲得需要的日期格式，使用LANG=C date -R。

- 在changlog 條目中簡要描述Bug 和相應的解決辦法，並在後面添加Closes: #54321。這樣Bug 報告會在你的套件被Debian 倉庫接受的同時被倉庫管理軟體自動關閉。
- 重複上述操作來修復更多的Bug，並在需要的時候使用dch 更新Debian changelog 檔案。
- 重複在'完整的(重)構建' ,49Δ、'檢套件中的錯誤' ,57Δ 和'上傳套件' ,61Δ 中進行的操作。惟一的區別是這次不需要再包含原始程式碼檔案，因它們沒有變化且已經存在於Debian 倉庫中。

9.2 檢新上游版本

在Debian 倉庫準備新上游版本的套件前，你必須首先檢它。

檢工作應從閱讀changelog、NEWS 以及所有隨新版本一同發佈的文件。

然後應按照以下步驟檢新舊版本之間的差別，看是否存在可疑的內容：

```
$ diff -urN foo-oldversion foo-newversion
```

對於Autotools 自動生成的檔案，例如missing、aclocal.m4、config.guess、config.h.in、config 和Makefile.in 是可以忽略的。你可以在運行diff 進行程式碼檢前刪除它們。

9.3 新上游版本

如果套件*foo* 是使用新的3.0 (native) 或3.0 (quilt) 格式打包的，製作新的上游版本時需要先把舊的debian 目錄移至新的原始碼內。這可以通過在新解壓的原始碼目錄裏運行tar xvzf /*path/to/foo_oldversion*.debian.tar.gz 完成。[2]當然，你需要做幾個很顯然的雜事。

- 創建一份上游原始碼的副本，命名foo_*newversion*.tar.gz
- 使用dch -v *newversion-1* 更新Debian changelog 檔案。
  - 添加一個條目，內容New upstream release。
  - 簡明地介紹在新上游版本中 上游修復和關閉的Bug。
  - 簡明地介紹維護者對此新上游版本 做出的修改，修復和關閉的Bug。
- 運行while quilt push; do quilt refresh; done 以應用全部補丁並使它們邊界清晰。

[2]如果套件*foo* 是使用舊的1.0 格式的，可以在新解壓的原始碼目錄裏運行zcat /*path/to/foo_oldversion*.diff.gz|patch -p1 來完成。

如果補丁沒有乾淨地被應用，檢原因(線索在.rej 檔案裏)。

- 如果你的補丁已經被上游接受：
  - 使用quilt delete 刪除它。
- 如果你的補丁與上游程式碼中的變更有衝突：
  - 使用quilt push -f 應用舊補丁，未應用的部分會被保存*baz*.rej。
  - 手工編輯*baz* 來在新的程式碼中實現*baz*.rej 中應有的效果。
  - 使用quilt refresh 更新補丁。
- 再次運行while quilt push; do quilt refresh; done。

這個過程可以通過使用uupdate(1) 來更自動化地完成：

```
$ apt-get source foo
...
dpkg-source: info: extracting foo in foo-oldversion
dpkg-source: info: unpacking foo_oldversion.orig.tar.gz
dpkg-source: info: applying foo_oldversion-1.debian.tar.gz
$ ls -F
foo-oldversion/
foo_oldversion-1.debian.tar.gz
foo_oldversion-1.dsc
foo_oldversion.orig.tar.gz
$ wget http://example.org/foo/foo-newversion.tar.gz
$ cd foo-oldversion
$ uupdate -v newversion ../foo-newversion.tar.gz
$ cd ../foo-newversion
$ while quilt push; do quilt refresh; done
$ dch
... document changes made
```

如果你按照'watch 檔案',45Δ 的敘述設置了debian/watch 檔案，你可以跳過這個wget 命令，轉而在*foo-oldversion* 目錄中運行uscan(1)，且無需再執行uupdate 命令。它會自動 找新的原始碼、下載並運行uupdate 命令。[3]

重複'完整的(重)構建',49Δ、'檢套件中的錯誤',57Δ 和'上傳套件',61Δ 中的操作，即可發佈此更新的套件。

[3]如果uscan 命令下載並更新了原始碼，但沒有運行uupdate 命令，你應該修正debian/watch 檔案，使URL 末尾後帶有debian uupdate。

9.4 更新打包風格

更新打包風格不是更新套件的必須步驟，但是這樣可以使你的套件得到對現代的debhelper系統和3.0 原始碼套件格式完整的兼容性。[4]

- 如果你需要重新添加已刪除的模板檔案，可以在同一個Debian 套件原始碼樹中運行dh_make，並添加--addmissing 選項。然後對模板進行相應的編輯。
- 如果套件的debian/rules 檔案沒有更新使用debhelper V7 的dh 語法，則更新它使用dh。在需要的時候更新debian/control 檔案。
- 如果你希望將使用cdbs 的Makefile 語法創建的rules 檔案更新dh 語法，參考/usr/share/doc/cdbs/cdbs-doc.html 並理解各DEB_* 設定變量。[5]
- 如果你有一個不帶有*foo*.diff.gz 檔案的1.0 格式的原始碼套件，你可以通過創建debian/source/format 檔案並在其中添加3.0 (native) 來將其更新新的3.0 (native) 原始碼套件格式。debian 目錄中的其他檔案可以直接複製過來。
- 如果你有一個帶有*foo*.diff.gz 檔案的1.0 格式的原始碼套件，你可以通過創建debian/source/format 檔案並在其中添加3.0 (quilt) 來將其更新新的3.0 (quilt) 原始碼套件格式。debian 目錄中的其他檔案可以直接複製過來。如果需要，把filterdiff -z -x '*/debian/*' *foo*.diff.gz > big.diff 生成的big.diff 檔案導入到quilt 系統。[6]
- 如果它使用了其他的補丁系統，例如dpatch、dbs 或cdbs，使用-p0、-p1 或-p2 級別，使用http://bugs.debian.org/581186 的deb3 命令將其轉換到quilt 系統。
- 如果它使用dh 命令的--with quilt 選項，或dh_quilt_patch 和dh_quilt_unpatch 命令，刪除它們並使其使用新的3.0 (native) 原始碼套件格式。

當然你還需要按照'新上游版本',64Δ 完成其他的步驟。

9.5 對更新套件的幾點提示

以下是對更新套件的幾點提示。

- 保留舊的changelog 條目(看似顯然，但是總有可能把dch -i 輸入dch)。
- 已存在的Debian 修改需要被重新校驗，去除上游已經接受的東西，除非有必要的原因，還要記錄尚未被上游接受的部分。

[4]如果你的sponsor 或其他維護者一定反對更新已有的打包風格，則不值得去此煩惱或爭論，總是有更重要的事要做。

[5]在cdbs (0.4.74)套件中有一些對於dh_make 創建的非cdbs 的rules 檔案的消極描述。不要擔心它們，它們僅是針對於lenny 中需要列出大串的dh_* 命令而言的。

[6]你可能使用splitdiff 命令將big.diff 分割多個有各自作用的補丁。

- 如果對編譯系統作出了修改(希望你已經在檢上游變更時瞭解了這些)，那麼要在必要時更新debian/rules 和debian/control 編譯依賴關係。
- 檢Debian Bug Tracking System (BTS) (http://www.debian.org/Bugs/) 是否有人某些仍然未修復的Bug 提供了補丁。
- 檢 .changes 檔案以確保你正要上傳到正確的發行版、正確的列於Closes 欄位的Bug、Maintainer 和Changed-By 欄位相匹配，且該檔案已經使用GPG 簽署等。