

Debian 套件打包教學指南

Lucas Nussbaum

`packaging-tutorial@packages.debian.org`

version 0.24 – 2019-03-13



關於此教學指南

- ▶ 目標: 瞭解 **Debian** 套件打包的相關知識
 - ▶ 修改既有套件
 - ▶ 新增自有套件
 - ▶ 和 Debian 社群進行交流
 - ▶ 成為 Debian 進階使用者
- ▶ 這份教學指南針對重要功能進行介紹, 但也許會有疏漏之處
 - ▶ 所以你需要閱讀更多文件
- ▶ 文件大部份的內容也適用於 Debian 衍生的 Linux 發行版
 - ▶ 其中包含 Ubuntu



大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練



大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練



Debian

- ▶ **GNU/Linux 發行版**
- ▶ 第一個以 GNU 開源精神進行開發的主要發行版
- ▶ 非營利, 由超過1000個志願者協同開發而成
- ▶ 三個主要特色
 - ▶ 品質 – 卓越的技術文化
準備好才會發行
 - ▶ 自由 – 開發者以及使用者皆遵循 社會契約
並發揚自1993年起開始倡導的自由軟體文化
 - ▶ 獨立 – Debian 不被任何一間公司所擁有
決策過程一切透明化 (行動至上 + 民主)
- ▶ 對於業餘開發者 來說最大的意義: 因本身喜愛而去完成這些事



Debian 套件

- ▶ **.deb** 檔案 (二進制套件)
- ▶ 以有效且合宜的方法來向使用者發佈軟體
- ▶ 是兩個最常見的套件格式之一 (另一個為 RPM)
- ▶ 一般來說
 - ▶ Debian 有 30,000 個二進制套件
→ 絕大部份的自由軟體皆已經打包並放進 Debian
 - ▶ 支援 12 種 CPU 架構, 其中包含 2 個非 Linux 相關(Hurd; KFreeBSD)
 - ▶ Debian 被其衍生120個發行版所使用



Deb 套件格式

- ▶ .deb 檔案: 是 ar 靜態函式庫格式

```
$ ar tv wget_1.12-2.1_i386.deb
rw-r--r-- 0/0      4 Sep  5 15:43 2010 debian-binary
rw-r--r-- 0/0    2403 Sep  5 15:43 2010 control.tar.gz
rw-r--r-- 0/0  751613 Sep  5 15:43 2010 data.tar.gz
```

- ▶ debian-binary: deb檔案格式的版本, "2.0\n"
 - ▶ control.tar.gz: 描述套件相關資訊的檔案
control, md5sums, (pre|post)(rm|inst), triggers, shlibs, ...
 - ▶ data.tar.gz: 套件裡的資料
- ▶ 你可以手動製作 .deb 檔案
http://tldp.org/HOWTO/html_single/Debian-Binary-Package-Building-HOWTO/
- ▶ 但大部份的人不會使用那種方法

這份教學指南: 以 **Debian** 風格來製作 **Debian** 套件



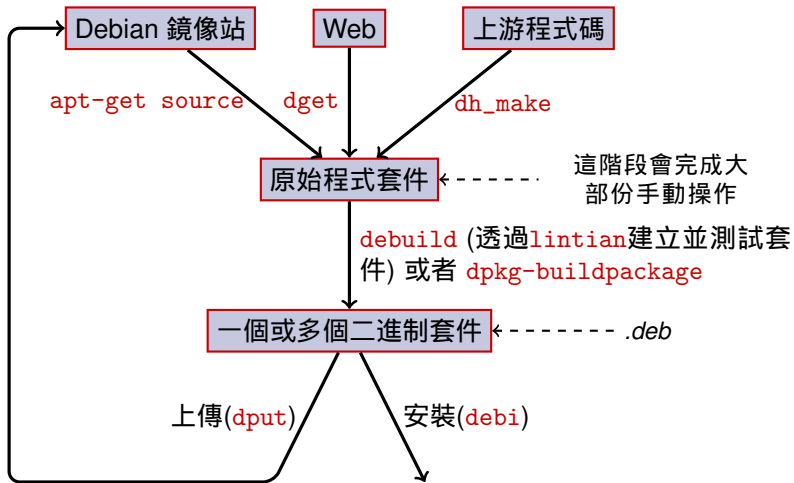
開發必需之工具

- ▶ 一台能以 root 權限存取的 Debian (or Ubuntu) 系統
- ▶ 一些必需套件
 - ▶ **build-essential**: 和 Debian 套件有相依性並假定已安裝在開發者的機器上 (不需在套件中的 control 欄位特別指定 Build-Depends:)
 - ▶ 相依 **dpkg-dev** 套件, 其中包含基本 Debian 特定工具, 以便於製作 Debian 套件
 - ▶ **devscripts**: 此套件提供許多便於開發的腳本給 Debian 維護者

還有一些接著會提到的工具, 比如說 **debhelper**, **cdb**s, **quilt**, **pbuilder**, **sbuid**, **lintian**, **svn-buildpackage**, **git-buildpackage**, ...
當需要使用時, 可安裝上述套件



一般打包套件的流程



範例: 重包 dash 套件

- 1 安裝重包 dash 套件所需的必要套件以及 devscripts 套件

```
sudo apt-get build-dep dash
```

(需要將deb-src 設定到 /etc/apt/sources.list)

```
sudo apt-get install --no-install-recommends devscripts  
fakeroot
```

- 2 創建並進入一個工作目錄:

```
mkdir /tmp/debian-tutorial ; cd /tmp/debian-tutorial
```

- 3 把 dash 原始碼套件下載下來

```
apt-get source dash
```

(務必確保 deb-src 已加入 /etc/apt/sources.list)

- 4 構建套件

```
cd dash-*
```

```
debuild -us -uc (-us -uc 代表不使用GPG來簽署套件)
```

- 5 確認已正常運行

- ▶ 會看到一些新產生的 .deb 檔案在上一層資料夾中

- 6 接著看debian/ 目錄

- ▶ 這是實際打包套件的地方



大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練



原始碼套件

- ▶ 一個原始碼套件可產生多個二進制套件
舉例來說 `libtar` 原始碼會產出 `libtar0` 跟 `libtar-dev` 二進制套件
- ▶ 套件分成兩種類型: (若不確定套件種類屬於哪一種, 請使用非原生)
 - ▶ 原生套件: 通常指的是在 Debian 上固有的特定軟體(`dpkg`, `apt`)
 - ▶ 非原生套件: 在 Debian 以外發展的軟體
- ▶ 主要檔案: `.dsc` (描述資料內容)
- ▶ 其他檔案則相依於原始碼格式版本
 - ▶ 1.0 or 3.0 (原生): `package_version.tar.gz`
 - ▶ 1.0 (非原生):
 - ▶ `pkg_ver.orig.tar.gz`: 上游原始碼
 - ▶ `pkg_debver.diff.gz`: 針對 Debian 環境而進行客制修改的補丁
 - ▶ 3.0 (quilt):
 - ▶ `pkg_ver.orig.tar.gz`: 上游原始碼
 - ▶ `pkg_debver.debian.tar.gz`: 針對 Debian 環境而進行客制修改的 tarball 檔

(可參考`dpkg-source(1)` 有更詳盡的解說)



原始碼套件範例

```
Format: 3.0 (quilt)
Source: wget
Binary: wget
Architecture: any
Version: 1.12-2.1
Maintainer: Noel Kothe <noel@debian.org>
Homepage: http://www.gnu.org/software/wget/
Standards-Version: 3.8.4
Build-Depends: debhelper (>> 5.0.0), gettext, texinfo,
    libssl-dev (>= 0.9.8), dpatch, info2man
Checksums-Sha1:
    50d4ed2441e67[...]1ee0e94248 2464747 wget_1.12.orig.tar.gz
    d4c1c8bbe431d[...]dd7cef3611 48308 wget_1.12-2.1.debian.tar.gz
Checksums-Sha256:
    7578ed0974e12[...]dcba65b572 2464747 wget_1.12.orig.tar.gz
    1e9b0c4c00eae[...]89c402ad78 48308 wget_1.12-2.1.debian.tar.gz
Files:
    141461b9c04e4[...]9d1f2abf83 2464747 wget_1.12.orig.tar.gz
    e93123c934e3c[...]2f380278c2 48308 wget_1.12-2.1.debian.tar.gz
```

取回一個既有的原始碼套件

- ▶ 從 Debian 檔案庫裡面下載
 - ▶ `apt-get source package`
 - ▶ `apt-get source package=version`
 - ▶ `apt-get source package/release`(需要加入 `deb-src` 的設定到 `sources.list`)
- ▶ 透過網路下載
 - ▶ `dget url-to.dsc`
 - ▶ `dget http://snapshot.debian.org/archive/debian-archive/20090802T004153Z/debian/dists/bo/main/source/web/wget_1.4.4-6.dsc`
(`snapshot.d.o` 提供自2005年開始的 Debian 套件)
- ▶ 透過 Debian 版本控制系統下載
 - ▶ `debcheckout package`
- ▶ 下載後, 使用 `dpkg-source -x file.dsc` 解壓縮即可



創建一個基本原始碼套件

- ▶ 從上游下載原始碼
(上游原始碼 = 來自軟體原始開發者)
- ▶ 將檔案改名成 `<source_package>_<upstream_version>.orig.tar.gz`
(範例: `simgrid_3.6.orig.tar.gz`)
- ▶ 解開此 tar 檔
- ▶ 將目錄改名為 `<source_package>-<upstream_version>`
(範例: `simgrid-3.6`)
- ▶ `cd <source_package>-<upstream_version> && dh_make`
(來自 **dh-make** 套件)
- ▶ 對於特定套件, 有除了 `dh_make` 以外的方法, 舉例為: **dh-make-perl**, **dh-make-php**, ...
- ▶ 已新增 `debian/` 目錄, 裡面有許多檔案



在 **debian/** 裡的檔案

透過修改 **debian/** 中的檔案來定義所有打包的動作

- ▶ 主要的檔案：
 - ▶ **control** – 描述套件相關資訊的檔案 (相依性, 等等...)
 - ▶ **rules** – 定義如何構建此套件
 - ▶ **copyright** – 關於此套件的版權宣告
 - ▶ **changelog** – 有關此 Debian 套件的修改歷程
- ▶ 其他檔案：
 - ▶ **compat**
 - ▶ **watch**
 - ▶ **dh_install* targets**
*.dirs, *.docs, *.manpages, ...
 - ▶ 維護者所使用的腳本
*.postinst, *.prerm, ...
 - ▶ **source/format**
 - ▶ **patches/** – 如果你需要修改上游原始碼
- ▶ 許多檔案的格式都是基於 RFC 822 (信的表頭)



debian/changelog

- ▶ 列出 Debian 套件的修改歷程
- ▶ 訂定現在的套件版本

1.2.1.1-5

上游版本 Debian
修訂版

- ▶ 可以手動編輯或者透過 `dch`
 - ▶ 為一個新的發行版創建一條歷程記錄 `dch -i`
- ▶ 具備特殊格式並自動將 Debian 或 Ubuntu 的缺陷結案
Debian: Closes: #595268; Ubuntu: LP: #616929
- ▶ 安裝在 `/usr/share/doc/package/changelog.Debian.gz`

```
mpich2 (1.2.1.1-5) unstable; urgency=low
```

- ```
* Use /usr/bin/python instead of /usr/bin/python2.5. Allow
to drop dependency on python2.5. Closes: #595268
* Make /usr/bin/mpdroot setuid. This is the default after
the installation of mpich2 from source, too. LP: #616929
+ Add corresponding lintian override.
```

```
-- Lucas Nussbaum <lucas@debian.org> Wed, 15 Sep 2010 18:13:44 +0200
```

# debian/control

- ▶ 描述套件相關資訊的檔案
  - ▶ 對於原始碼套件本身
  - ▶ 對於每個從原始碼套件建置而成的二進制套件
- ▶ 套件名稱, 種類, 優先度, 維護者, 上傳者, 建置相依性, 相依性, 相關描述, 網頁
- ▶ 相關文件: Debian Policy 第 5 章  
<https://www.debian.org/doc/debian-policy/ch-controlfields>

---

```
Source: wget
Section: web
Priority: important
Maintainer: Noel Kothe <noel@debian.org>
Build-Depends: debhelper (> 5.0.0), gettext, texinfo,
 libssl-dev (>= 0.9.8), dpatch, info2man
Standards-Version: 3.8.4
Homepage: http://www.gnu.org/software/wget/

Package: wget
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: retrieves files from the web
 Wget is a network utility to retrieve files from the Web
```



# Architecture: all or any

有兩種二進制套件:

- ▶ 套件內容會隨著不同的 Debian 平台架構而有所差異
  - ▶ 範例: C 程式
  - ▶ debian/control 中描述 Architecture: any
    - ▶ 或者只運行在某些特定的平台架構中:  
Architecture: amd64 i386 ia64 hurd-i386
  - ▶ buildd.debian.org: 上傳後會自動構建其他架構的套件
  - ▶ 命名為 `package_version_architecture.deb`
- ▶ 套件內容在不同的 Debian 平台架構上皆相同
  - ▶ 範例: Perl 函式庫
  - ▶ debian/control 中描述 Architecture: all
  - ▶ 命名為 `package_version_all.deb`

一個原始碼套件可以產生多個平台架構 Architecture: any 以及 Architecture: all 二進制套件



# debian/rules

- ▶ Makefile
- ▶ 構建 Debian 套件的介面
- ▶ 詳細文件在 Debian Policy, 章節 4.8 中  
<https://www.debian.org/doc/debian-policy/ch-source#s-debianrules>
- ▶ 必要的 targets:
  - ▶ build, build-arch, build-indep: 必須執行所有設定並進行編譯
  - ▶ binary, binary-arch, binary-indep: 構建二進制套件
    - ▶ dpkg-buildpackage 會呼叫 binary 構建所有套件, 或者呼叫 binary-arch 構建特定 Architecture: any 套件
  - ▶ clean: 清理原始碼目錄



# 協助打包的幫手 – debhelper

- ▶ 可以直接在 `debian/rules` 中撰寫 shell code
  - ▶ 範例可參考 `rsync` 套件
- ▶ 然而更好的作法 (被大部份套件所採用): 使用 打包幫手
- ▶ 最常見的方法: **debhelper** (大約被 98% 的套件所採用)
- ▶ 目標:
  - ▶ 將常使用的任務拆解, 並轉化為標準工具, 最後適用於所有套件上
  - ▶ 當修正打包錯誤時, 可以適用於所有的套件

`dh_installdirs`, `dh_installchangelogs`, `dh_installdocs`, `dh_installexamples`, `dh_install`,  
`dh_installdebconf`, `dh_installinit`, `dh_link`, `dh_strip`, `dh_compress`, `dh_fixperms`, `dh_perl`,  
`dh_makeshlibs`, `dh_installdeb`, `dh_shlibdeps`, `dh_gencontrol`, `dh_md5sums`, `dh_builddeb`, ...

- ▶ 由 `debian/rules` 所呼叫
- ▶ 透過 `debian/` 中的指令參數或檔案來進行配置的動作

`package.docs`, `package.examples`, `package.install`, `package.manpages`, ...

- ▶ 有第三方小幫手可以協助打包套件: **python-support**, **dh\_ocaml**, ...
- ▶ Gotcha: `debian/compat`: Debhelper 相容性版本 (use "7")



## debian/rules 使用 debhelper (1/2)

```
#!/usr/bin/make -f
```

```
Uncomment this to turn on verbose mode.
```

```
#export DH_VERBOSE=1
```

```
build:
```

```
$(MAKE)
```

```
#docbook-to-man debian/package.sgml > package.1
```

```
clean:
```

```
dh_testdir
```

```
dh_testroot
```

```
rm -f build-stamp configure-stamp
```

```
$(MAKE) clean
```

```
dh_clean
```

```
install: build
```

```
dh_testdir
```

```
dh_testroot
```

```
dh_clean -k
```

```
dh_installdirs
```

```
Add here commands to install the package into debian/package
```

```
$(MAKE) DESTDIR=$(CURDIR)/debian/package install
```



## debian/rules using debhelper (2/2)

```
Build architecture-independent files here.
```

```
binary-indep: build install
```

```
Build architecture-dependent files here.
```

```
binary-arch: build install
```

```
dh_testdir
```

```
dh_testroot
```

```
dh_installchangelogs
```

```
dh_installdocs
```

```
dh_installexamples
```

```
dh_install
```

```
dh_installman
```

```
dh_link
```

```
dh_strip
```

```
dh_compress
```

```
dh_fixperms
```

```
dh_installdeb
```

```
dh_shlibdeps
```

```
dh_gencontrol
```

```
dh_md5sums
```

```
dh_builddeb
```

```
binary: binary-indep binary-arch
```

```
.PHONY: build clean binary-indep binary-arch binary install configure
```



# CDBS

- ▶ 即使在套件裡使用 debhelper, 還是會有冗餘的事情要處理
- ▶ 第二層的小幫手能夠拆解共同的功能
  - ▶ 舉例: 使用 `./configure && make && make install` 或者 CMake 進行建構
- ▶ CDBS:
  - ▶ 源自 2005, 基於 *advanced GNU make magic*
  - ▶ 文件: `/usr/share/doc/cdb/`
  - ▶ 可支援 Perl, Python, Ruby, GNOME, KDE, Java, Haskell, ...
  - ▶ 但有些使用者不喜歡使用
    - ▶ 有時難以產生客製化套件:  
"makefile以及環境參數相當複雜"
    - ▶ 比明文的 debhelper還慢 (會有許多不必要 `dh_*`相關的呼叫)

---

```
#!/usr/bin/make -f
include /usr/share/cdb/1/rules/debhelper.mk
include /usr/share/cdb/1/class/autotools.mk

add an action after the build
build/mypackage::
 /bin/bash debian/scripts/foo.sh
```





# Dh (又名 Debhelper 7, 或者 dh7)

- ▶ 源自2008年, 預期 取代 *CDBS*
- ▶ **dh** 指令呼叫 `dh_*`
- ▶ 簡易化 *debian/rules*, 只列出需覆蓋的地方
- ▶ 比 *CDBS* 更容易進行客製化
- ▶ 文件: `manpages (debhelper(7), dh(1))` + DebConf9 的簡報投影片  
<http://kitenet.net/~joey/talks/debhelper/debhelper-slides.pdf>

---

```
#!/usr/bin/make -f
%:
 dh $@

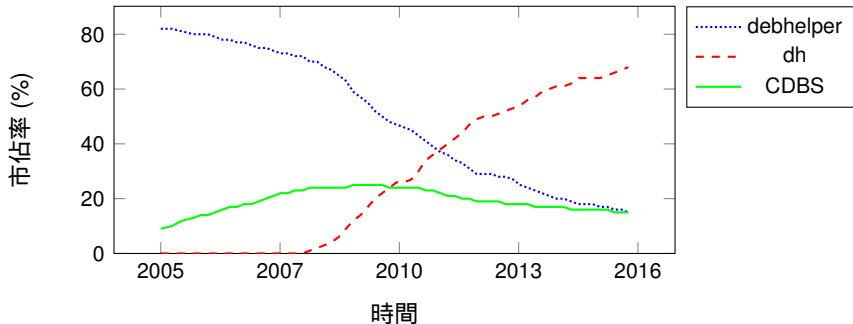
override_dh_auto_configure:
 dh_auto_configure -- --with-kitchen-sink

override_dh_auto_build:
 make world
```



# Classic debhelper vs CDBS vs dh

- ▶ 心佔率:  
Classic debhelper: 15%   CDBS: 15%   dh: 68%
- ▶ 究竟該採用哪一種方法呢？
  - ▶ 或許每一種都需要瞭解
  - ▶ 必須瞭解 debhelper 以使用 dh 以及 CDBS
  - ▶ 你也許需要修改 CDBS 套件
- ▶ 對於新套件, 該使用哪一種方法呢？
  - ▶ **dh** (只有它的心佔率是呈現上升)



# 大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件**
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練



# 構建套件

- ▶ `apt-get build-dep mypackage`  
安裝 *build-dependencies* (適用於套件已在 Debian 中)  
或者 `mk-build-deps -ir` (適用於套件尚未上傳到 Debian 中)
- ▶ `debuild`: 透過 `lintian` 進行構建, 測試, 並使用 GPG 進行簽署
- ▶ 又或者可以直接呼叫 `dpkg-buildpackage`
  - ▶ 通常使用 `dpkg-buildpackage -us -uc`
- ▶ 在小型且乾淨環境中構建套件較為適合
  - ▶ `pbuilder` – 使用 `chroot` 來協助構建套件好的文件:  
<https://wiki.ubuntu.com/PbuilderHowto>  
(最佳化: `cowbuilder ccache distcc`)
  - ▶ `schroot` 和 `sbuild`: 被 Debian build daemons 所使用  
(不若 `pbuilder` 簡單, 但能使用 LVM 快照功能  
參照: <https://help.ubuntu.com/community/SbuildLVMHowto>)
- ▶ 產生 `.deb` 檔案以及 `.changes` 檔案
  - ▶ `.changes`: 描述構建哪些東西; 用來上傳套件



# 安裝以及測試套件

- ▶ 本地安裝套件: `debi` (使用 `.changes` 來得知需安裝哪些)
- ▶ 列出套件的內容: `debc` `../mypackage<TAB>.changes`
- ▶ 和前一版的套件比較:  
`debdiff` `../mypackage_1_*.changes` `../mypackage_2_*.changes`  
或比較原始碼差異:  
`debdiff` `../mypackage_1_*.dsc` `../mypackage_2_*.dsc`
- ▶ 透過 `lintian` (靜態分析工具): 來確認套件  
`lintian` `../mypackage<TAB>.changes`  
`lintian -i`: 提示更多錯誤訊息  
`lintian -EviIL +pedantic`: 顯示更多問題
- ▶ 上傳套件到 Debian (`dput`) (需要設定)
- ▶ 可透過 `reprepro` 的指令管理私有 Debian 檔案庫  
文件: <https://mirrorer.alioth.debian.org/>



# 大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練



# 實際演練 1: 修改 `grep` 套件

- ❶ 前往 `http://ftp.debian.org/debian/pool/main/g/grep/` 並且下載版本 2.12-2 的套件
  - ▶ 如果原始碼套件沒有自動解開, 請使用下列指令  
`dpkg-source -x grep_*.dsc`
- ❷ 注意 `debian/` 中的檔案
  - ▶ 原始碼套件產生多少二進制套件?
  - ▶ 這個套件使用哪一種打包小幫手?
- ❸ 構建套件
- ❹ 先修改套件. 並在 `changelog` 中添加一個條目並且遞增版本號碼.
- ❺ 接著將 `perl-regexp` 功能移除 (位於 `./configure` 中的選項)
- ❻ 重新構建套件
- ❼ 使用 `debdiff` 來比較原始和新套件中差異
- ❽ 安裝新構建的套件



# 大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題**
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練





# debian/copyright

- ▶ 原始碼以及打包的著作權和授權條款資訊
- ▶ 一般都是以 text 檔案格式編寫
- ▶ 新的機器可讀格式

<https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/>

---

```
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: X Solitaire
Source: ftp://ftp.example.com/pub/games
```

```
Files: *
Copyright: Copyright 1998 John Doe <jdoe@example.com>
License: GPL-2+
 This program is free software; you can redistribute it
 [...]
 .
 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'.
```

```
Files: debian/*
Copyright: Copyright 1998 Jane Smith <jsmith@example.net>
License:
 [LICENSE TEXT]
```



# 修改上游的原始碼

通常發生於

- ▶ 需要修正缺陷或者新增 Debian 客製化修改
- ▶ 從新的上游發行版中移值補丁回來

列出可實行的方法

- ▶ 直接修改檔案
  - ▶ 簡單
  - ▶ 但是沒有修改歷程等相關文件可供追蹤
- ▶ 使用補丁系統
  - ▶ 讓修改更容易的回到上游
  - ▶ 能夠分享給相關衍生性版本
  - ▶ 讓修改更具有揭露性

<http://patch-tracker.debian.org/> (目前無法使用)



# 補丁系統

- ▶ 原則: 修改需以補丁的型式存放在 `debian/patches/`
- ▶ 在構建時可選擇是否引入補丁
- ▶ 以往有數種實作方法 – *simple-patchsys* (*cdb*s), *dpatch*, **quilt**
  - ▶ 每一個皆支援兩個 `debian/rules` targets:
    - ▶ `debian/rules patch`: 引入所有補丁
    - ▶ `debian/rules unpatch`: 卸載所有補丁
  - ▶ 更多文件可參照: <https://wiki.debian.org/debian/patches>
- ▶ 新的原始碼套件格式和內建補丁系統: **3.0 (quilt)**
  - ▶ 建議的解決方案
  - ▶ 你需要學習 *quilt*  
<https://perl-team.pages.debian.net/howto/quilt.html>
  - ▶ 補丁系統相容工具 `devscripts: edit-patch`



# 補丁的相關文件

- ▶ 補丁一開頭即為標準檔頭
- ▶ DEP-3 文件 - Patch Tagging Guidelines  
<http://dep.debian.net/deps/dep3/>

---

```
Description: Fix widget frobnication speeds
 Frobnicating widgets too quickly tended to cause explosions.
Forwarded: http://lists.example.com/2010/03/1234.html
Author: John Doe <johndoe-guest@users.alioth.debian.org>
Applied-Upstream: 1.2, http://bZR.foo.com/frobnicator/revision/123
Last-Update: 2010-03-29
```

```
--- a/src/widgets.c
+++ b/src/widgets.c
@@ -101,9 +101,6 @@ struct {
```



# 安裝以及卸載的相關動作

- ▶ 只解壓縮套件, 有時會略感不足
- ▶ 新增/ 刪除系統使用者, 啟動/停止服務, 控制 *alternatives*
- ▶ 在 *maintainer scripts* 完成

```
preinst, postinst, prerm, postrm
```

  - ▶ 有些共同的動作可以被 *debhelper* 所生成
- ▶ 文件
  - ▶ Debian 維護手冊, 第6章  
<https://www.debian.org/doc/debian-policy/ch-maintainerscripts>
  - ▶ Debian 開發者參考, 第6.4章  
<https://www.debian.org/doc/developers-reference/best-pkging-practices.html>
  - ▶ <https://people.debian.org/~srivasta/MaintainerScripts.html>
- ▶ 提示使用者
  - ▶ 務必使用 **debconf**
  - ▶ 文件: `debconf-devel(7)` (`debconf-doc` 套件)



# 監看上游版本

- ▶ 在 `debian/watch` 中指定要監看的位址 (參考 `uscan(1)`)

```
version=3
```

```
http://tmrc.mit.edu/mirror/twisted/Twisted/(\d\.\d)/ \
Twisted-([\d\.]*)\.tar\.bz2
```

- ▶ 有一些可自動追蹤上游版本的系統, 會透過 `dashboards` 的方式來通知維護者. 例: <https://tracker.debian.org/> 以及 <https://udd.debian.org/dmd/>
- ▶ `uscan`: 執行手動確認
- ▶ `uupdate`: 讓你的套件更新到最新的上游版本



# 使用版本控制系統來進行打包

- ▶ 有數種工具可以來協助管理 branches and tags 以進行打包的動作:  
svn-buildpackage, git-buildpackage

- ▶ 範例: git-buildpackage

- ▶ upstream 分支使用 upstream/version 標籤來追蹤上游
- ▶ master 分支追蹤 Debian 套件
- ▶ debian/version 會對每個上傳進行標籤
- ▶ pristine-tar 分支能夠重新構建上游壓縮檔

文件: <http://honk.sigxcpu.org/projects/git-buildpackage/manual-html/gbp.html>

- ▶ Vcs-\* 在 debian/control 的欄位能夠定位目錄

- ▶ <https://wiki.debian.org/Alioth/Git>
- ▶ <https://wiki.debian.org/Alioth/Svn>

Vcs-Browser: <http://anonscm.debian.org/gitweb/?p=collab-maint/devscripts.git>  
Vcs-Git: <git://anonscm.debian.org/collab-maint/devscripts.git>

Vcs-Browser: <http://svn.debian.org/viewsvn/pkg-perl/trunk/libwww-perl/>  
Vcs-Svn: <svn://svn.debian.org/pkg-perl/trunk/libwww-perl>

- ▶ VCS 相容介面: debcheckout, debcommit, debrelease

- ▶ debcheckout grep → 從Git checks out 原始碼套件出來



# 向前移值套件

- ▶ 目標: 在舊有系統上使用較新的套件版本  
例: 使用 *mutt* from Debian *unstable* on Debian *stable*
- ▶ 一般來說
  - ▶ 從 Debian unstable 抓原始碼套件
  - ▶ 透過修改讓套件可以構建且正常運行在 Debian stable 上
    - ▶ 有時簡單 (不需任何修改)
    - ▶ 有時困難
    - ▶ 有時不可行 (有太多無法解決的相依性問題)
- ▶ Debian project 提供部份向前相容的套件  
<http://backports.debian.org/>



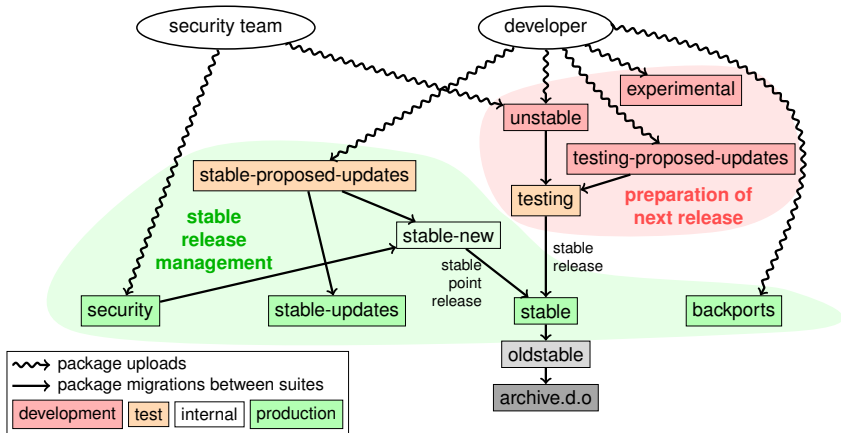


# 大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練



# Debian 檔案庫和 suites



基於 Antoine Beaupré 的圖片 <https://salsa.debian.org/debian/package-cycle>



# 開發用的 Suites

- ▶ 每個套件之新版本會被上傳至 **unstable** (**sid**)
- ▶ 套件會基於幾項準則從 **unstable** 移至 **testing** (例如，在 **unstable** 已經有 10 天而且沒有發生 regressions)
- ▶ 新套件也可被上傳至：
  - ▶ **experimental** (給 *experimental* 套件所使用, 像是當新版本還沒準備要取代在 **unstable** 的版本時)
  - ▶ **testing-proposed-updates**, 更新 **testing** 的版本而不透過 **unstable** (這很少被使用)



# 凍結和發行

- ▶ 在發行週期的某些時刻, 發行團隊會決定進行 *freeze* 測試: 停止從 "unstable 到 testing 的自動移轉, 並且改用人工審核
- ▶ 當發行團隊認為 **testing** 已經準備發行時:
  - ▶ **testing** suite 會變成新的 **stable** suite
  - ▶ 同樣地, 舊的 **stable** 會變成 **oldstable**
  - ▶ 不支援的發行則會被移到 `archive.debian.org`
- ▶ 參考 <https://release.debian.org/>



# Stable release management

- ▶ 許多 suites 是用來提供穩定發行的套件：
  - ▶ **stable**: 主要的 suite
  - ▶ **security** 更新為 security.debian.org 所提供的 suite, 並被安全團隊所使用, 更新消息會公佈在 debian-security-announce mailing list
  - ▶ **stable-updates**: 跟安全無關, 但是需要被緊急安裝的更新(不需等待下次的小版本更新): 如防毒的資料庫, 時區相關套件等等, 會公佈在 debian-stable-announce mailing list
  - ▶ **backports**: 新的上游版本, 會基於在 **testing** 的版本
- ▶ **stable** suite 每幾個月會有 穩定小版本更新 (只有包含 bug 修復)
  - ▶ 目標是下次穩定小版本更新的套件會被上傳到 **stable-proposed-updates** 並被發行團隊審核
- ▶ **oldstable** 的發行也擁有相同的 suites 組合



# 一些為 Debian 貢獻的方法

- ▶ 不建議 提供貢獻的方法
  - ① 打包自有應用程式
  - ② 放進 Debian
  - ③ 消失
- ▶ 較建議 提供貢獻的方法:
  - ▶ 實際參與打包群組
    - ▶ 許多群組專注於特定套件, 且皆需要協助
    - ▶ 列出可用群組 <https://wiki.debian.org/Teams>
    - ▶ 從更有經驗的貢獻者身上學習
  - ▶ 認養現有但無人維護之套件 (*orphaned packages*)
  - ▶ 將新的軟體帶進 Debian
    - ▶ 必須是有用或者有趣的
    - ▶ Debian 原有套件是否就可以滿足需求?



# 認養無人維護的套件

- ▶ Debian 有許多尚待維護的套件
- ▶ 完整清單以及流程: <https://www.debian.org/devel/wnpp/>
- ▶ 將 wnpp-alert 安裝到機器上,並可更進一步的安裝 how-can-i-help
- ▶ 不同的狀態:
  - ▶ **Orphaned**: 此套件目前被遺棄中  
歡迎認領
  - ▶ **RFA: Request For Adopter**  
此套件維護中, 但維護者在尋找繼任者  
歡迎認養, 但禮貌上先以電子郵件知會現任維護者
  - ▶ **ITA: Intent To Adopt**  
有人意圖認養此套件  
但你依然可以提出協助的需求
  - ▶ **RFH: Request For Help**  
維護者尋求協助中
- ▶ 有些無人維護的套件沒有被偵測到→不是被遺棄的套件
- ▶ 若有任何疑問, 請到 [debian-qa@lists.debian.org](mailto:debian-qa@lists.debian.org) 提出  
或者前往 [irc.debian.org](https://irc.debian.org)裡的頻道 #debian-qa詢問



## 認領套件：範例

```
From: You <you@yourdomain>
To: 640454@bugs.debian.org, control@bugs.debian.org
Cc: Francois Marier <francois@debian.org>
Subject: ITA: verbiste -- French conjugator
```

```
retitle 640454 ITA: verbiste -- French conjugator
owner 640454 !
thanks
```

Hi,

I am using verbiste and I am willing to take care of the package.

Cheers,

You

- ▶ 有禮貌的連絡上一任維護者 (尤其套件狀態為 RFA 而不是 orphaned)
- ▶ 適時的和上游專案有所連繫





# 將你的套件放入 **Debian**

- ▶ 不需要具備任何官方身份就可以將你的套件放進 Debian
  - ① 使用 `reportbug wnpp` 送出 **ITP** bug (Intent To Package)
  - ② 準備好原始碼套件
  - ③ 找一位 Debian Developer 來協助確認你的套件
- ▶ 官方身份 (當你具備許多打包維護的經驗):
  - ▶ **Debian Maintainer (DM):**  
允許上傳你自己的套件  
參照 <https://wiki.debian.org/DebianMaintainer>
  - ▶ **Debian Developer (DD):**  
Debian project member; 能夠投票且上傳任何套件



# 發出協助需求前需確認事項

- ▶ Debian 非常重視品質
- ▶ 一般而言, 協助人員很忙又很難找
  - ▶ 請再三確認你的套件已經很完整, 再提出協助確認需求
- ▶ 需要確認之事項
  - ▶ 避免忘了加入構建相依性: 確認你的套件可以在乾淨的 *sid chroot* 中正常構建
    - ▶ 建議使用 `pbuilder`
  - ▶ 在你的套件上執行 `lintian -EviLL +pedantic`
    - ▶ 若出現 `Errors` 則必須被修復, 其他問題也需一併修復
  - ▶ 請在你的套件上進行大規模的測試驗證
- ▶ 若有疑問, 請提出問題以尋求協助



# 去哪邊尋求協助?

可針對下列事項進行協助

- ▶ 對於提出的問題, 給予建議或答案, 以及程式碼審查
- ▶ 協助確認並且上傳你已經準備好的套件

你可以從下列得到協助

- ▶ 其他打包部門的成員
  - ▶ 部門列表: <https://wiki.debian.org/Teams>
- ▶ The **Debian Mentors group** (如果你的套件找不到相對應的部門)
  - ▶ <https://wiki.debian.org/DebianMentorsFaq>
  - ▶ Mailing list: [debian-mentors@lists.debian.org](mailto:debian-mentors@lists.debian.org)  
(有時可以學到不錯的東西)
  - ▶ IRC: #debian-mentors on [irc.debian.org](http://irc.debian.org)
  - ▶ <http://mentors.debian.net/>
  - ▶ 文件: <http://mentors.debian.net/intro-maintainers>
- ▶ 當地 **mailing lists** (根據你的語言來尋求協助)
  - ▶ [debian-devel-{french,italian,portuguese,spanish}@lists.d.o](mailto:debian-devel-{french,italian,portuguese,spanish}@lists.d.o)
  - ▶ 完整清單: <https://lists.debian.org/devel.html>
  - ▶ 或者使用者清單: <https://lists.debian.org/users.html>



## 更多文件

- ▶ Debian Developers' Corner  
<https://www.debian.org/devel/>  
有許多開發 Debian 相關的資源連結
- ▶ Debian 維護者指南  
<https://www.debian.org/doc/manuals/debmake-doc/>
- ▶ Debian Developer's Reference  
<https://www.debian.org/doc/developers-reference/>  
比較多關於 Debian 的常規, 但也含有一些很棒的實際範例 (part 6)
- ▶ Debian Policy  
<https://www.debian.org/doc/debian-policy/>
  - ▶ 所有需求都必須在每個套件中被滿足
  - ▶ 對於 Perl, Java, Python, ... 特定的規則
- ▶ Ubuntu 打包指南  
<http://developer.ubuntu.com/resources/tools/packaging/>



# Debian 給維護者的dashboards

- ▶ 原始碼套件中心: 套件追蹤系統 Package Tracking System (PTS)  
<https://packages.qa.debian.org/dpkg>
- ▶ 維護者/ 小組中心: 開發者套件概述 Developer's Packages Overview (DDPO)  
<https://qa.debian.org/developer.php?login=pkg-ruby-extras-maintainers@lists.alioth.debian.org>
- ▶ 待作清單列表: Debian Maintainer Dashboard (DMD)  
<https://udd.debian.org/dmd.cgi>



# 使用 Debian Bug 追蹤系統 (BTS)

- ▶ 使用特有的方法來管控缺陷
  - ▶ 使用 Web 的介面來查看缺陷
  - ▶ 使用 Email 的介面來處理缺陷
- ▶ 加入缺陷的資訊
  - ▶ 寫電子郵件到 `123456@bugs.debian.org` (若要包含submitter, 你需要加入 `123456-submitter@bugs.debian.org`)
- ▶ 更改缺陷狀態:
  - ▶ 送出指令到 `control@bugs.debian.org`
  - ▶ 指令列的介面: 在 `devscripts` 中的指令 `bts`
  - ▶ 文件: `https://www.debian.org/Bugs/server-control`
- ▶ 回報缺陷: 使用 `reportbug`
  - ▶ 通常使用本地郵件服務器: 可透過 `ssmtp` 或者 `nullmailer` 安裝
  - ▶ 或者使用 `reportbug --template`, 然後手動送出至 `submit@bugs.debian.org`



# 使用 **Debian Bug** 追蹤系統 (BTS): 範例

- ▶ 針對缺陷送出電子郵件給 submitter:  
`https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680822#10`
- ▶ 標記並且修改嚴重程度:  
`https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680227#10`
- ▶ 重新指定, 修改嚴重程度, 修改主旨 ...:  
`https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680822#93`
  - ▶ notfound, found, notfixed, fixed 是給 版本追蹤來使用  
參照 `https://wiki.debian.org/HowtoUseBTS#Version\_tracking`
- ▶ 使用使用者標籤: `https://bugs.debian.org/cgi-bin/bugreport.cgi?msg=42;bug=642267`  
參照 `https://wiki.debian.org/bugs.debian.org/usertags`
- ▶ Debian Bug 追蹤系統 (BTS) 文件:
  - ▶ `https://www.debian.org/Bugs/`
  - ▶ `https://wiki.debian.org/HowtoUseBTS`



# More interested in Ubuntu?

- ▶ Ubuntu 主要管控和 Debian 的差異
- ▶ 不針對於特定套件，  
而是和 Debian team 協同合作
- ▶ 一般來說會建議新套件要先上傳到 Debian  
<https://wiki.ubuntu.com/UbuntuDevelopment/NewPackages>
- ▶ 但有更好的方法
  - ▶ 參與 Debian team 並且和 Ubuntu 建立溝通橋樑
  - ▶ 協助減少差異並在 Launchpad 進行缺陷診斷
  - ▶ 許多 Debian 工具能提供協助：
    - ▶ Developer 的套件大綱中有 ubuntu 欄位
    - ▶ Ubuntu box on the 套件追蹤系統
    - ▶ 透過 PTS 收 launchpad bugmail





# 大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論**
- 8 深入淺出實際演練
- 9 深入淺出實際演練



# 結論

- ▶ 現在你對 Debian 打包已經有一個完整的概觀
- ▶ 但你需要閱讀更多的文件
- ▶ 典範實務是經年累月而成的
  - ▶ 如果不太確定, 使用 **dh** 套件小幫手, 以及 **3.0 (quilt)** 格式

回饋: [packaging-tutorial@packages.debian.org](mailto:packaging-tutorial@packages.debian.org)



# 法律相關

Copyright ©2011–2019 Lucas Nussbaum – [lucas@debian.org](mailto:lucas@debian.org)

**This document is free software:** you can redistribute it and/or modify it under either (at your option):

- ▶ The terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.  
<http://www.gnu.org/licenses/gpl.html>
- ▶ The terms of the Creative Commons Attribution-ShareAlike 3.0 Unported License.  
<http://creativecommons.org/licenses/by-sa/3.0/>



# 對此教學指南做出貢獻

## ▶ 貢獻:

- ▶ `apt-get source packaging-tutorial`
- ▶ `debcheckout packaging-tutorial`
- ▶ `git clone`  
`git://git.debian.org/collab-maint/packaging-tutorial.git`
- ▶ `http://git.debian.org/?p=collab-maint/packaging-tutorial.git`
- ▶ 回報缺陷: `bugs.debian.org/src:packaging-tutorial`

## ▶ 提供回饋:

- ▶ `mailto:packaging-tutorial@packages.debian.org`
  - ▶ 有哪些必須加入此教學指南?
  - ▶ 有哪些部份可以再優化?
- ▶ `reportbug packaging-tutorial`



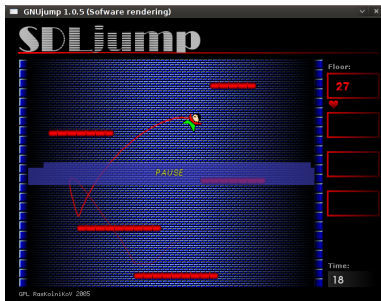
# 大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練**
- 9 深入淺出實際演練



## 實際演練 2: 打包 GNUjump

- 1 從 <http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz> 下載 GNUjump 1.0.8
- 2 建立一個 Debian 套件
  - ▶ 安裝 build-dependencies 以進行構建套件之先行必要動作
  - ▶ 修正問題
  - ▶ 產生一個基本可運作的套件
  - ▶ 完成填入 debian/control 以及其他檔案
- 3 享受吧



## 實際演練 2: 打包 GNUjump (小訣竅)

- ▶ 創建一個基本的套件: `dh_make`
- ▶ 一開始先創建一個 1.0 原始碼套件會比 3.0 (*quilt*) (透過修改 `debian/source/format`)簡單
- ▶ 先搜尋構建所需要的相依檔案,找到檔案後,使用`apt-file`來找到套件
- ▶ 如果你遇到錯誤

```
/usr/bin/ld: SDL_rotozoom.o: undefined reference to symbol 'ceil@@GLIBC_2.2.5'
//lib/x86_64-linux-gnu/libm.so.6: error adding symbols: DSO missing from command line
collect2: error: ld returned 1 exit status
Makefile:376: recipe for target 'gnujump' failed
```

代表你需要加入`-lm` 到linker指令列:

編輯 `src/Makefile.am`並且替換

```
gnujump_LDFLAGS = $(all_libraries)
```

成下列

```
gnujump_LDFLAGS = -Wl,--as-needed
gnujump_LDADD = $(all_libraries) -lm
```

接著執行 `autoreconf -i`



## 實際演練 3: 打包 Java library

### ❶ 先看一下如何打包 Java 相關文件:

- ▶ <https://wiki.debian.org/Java>
- ▶ <https://wiki.debian.org/Java/Packaging>
- ▶ <https://www.debian.org/doc/packaging-manuals/java-policy/>
- ▶ `/usr/share/doc/javahelper/tutorial.txt.gz`

### ❷ 從 <http://moepii.sourceforge.net/> 下載 IRClib

### ❸ 開始打包





## 實際演練 4: 打包 Ruby gem

### ❶ 先看一下如何打包 Ruby 相關文件:

- ▶ <https://wiki.debian.org/Ruby>
- ▶ <https://wiki.debian.org/Teams/Ruby>
- ▶ <https://wiki.debian.org/Teams/Ruby/Packaging>
- ▶ `gem2deb(1)`, `dh_ruby(1)` (`gem2deb` 套件)

### ❷ 從 `peach gem`: `gem2deb peach` 建立基本的 Debian 原始碼套件

### ❸ 適時的優化, 讓它成為一個合適的 Debian 套件



## 實際演練 5: 打包 Perl 模組

### ❶ 先看一下如何打包 Perl 相關文件:

- ▶ <https://perl-team.pages.debian.net>
- ▶ <https://wiki.debian.org/Teams/DebianPerlGroup>
- ▶ `dh-make-perl(1)`, `dpt(1)` (in the `pkg-perl-tools` package)

### ❷ 從 Acme CPAN 發行版:

`dh-make-perl --cpan Acme` 建立基本的 Debian 原始碼套件

### ❸ 適時的優化, 讓它成為一個合適的 Debian 套件



# 大綱

- 1 介紹
- 2 製作原始碼套件
- 3 構建並測試套件
- 4 實際演練 1: 修改 grep 套件
- 5 進階打包主題
- 6 維護 Debian 套件
- 7 結論
- 8 深入淺出實際演練
- 9 深入淺出實際演練



# 深入淺出 實際演練



# 實際演練 1: 修改 `grep` 套件

- 1 前往 `http://ftp.debian.org/debian/pool/main/g/grep/` 並且下載版本 2.12-2 的套件
- 2 注意 `debian/` 中的檔案
  - ▶ 原始碼套件產生多少二進制套件？
  - ▶ 這個套件使用哪一種打包小幫手？
- 3 構建套件
- 4 先修改套件. 並在 `changelog` 中添加一個條目並且遞增版本號碼.
- 5 接著將 `perl-regexp` 功能移除 (位於 `./configure` 中的選項)
- 6 重新構建套件
- 7 使用 `debdiff` 來比較原始和新套件中差異
- 8 安裝新構建的套件



# 把原始碼抓下來

- ❶ 前往 <http://ftp.debian.org/debian/pool/main/g/grep/> 並且下載版本 2.12-2 的套件
  - ▶ 使用 dget 下載 .dsc 檔案:  
`dget https://cdn.debian.net/debian/pool/main/g/grep/grep_2.12-2.dsc`
  - ▶ 如果你在Debian發行版上有deb-src有grep版本2.12-2(可查看<https://tracker.debian.org/grep>),你可以使用`apt-get source grep=2.12-2`  
或者`apt-get source grep/release (例 grep/stable`  
又或者你可直接使用`apt-get source grep`
  - ▶ grep 原始碼套件主要有 3 個檔案所組成:
    - ▶ `grep_2.12-2.dsc`
    - ▶ `grep_2.12-2.debian.tar.bz2`
    - ▶ `grep_2.12-2.orig.tar.bz2`這些代表 "3.0 (quilt)" 格式
  - ▶ 若有需求, 解壓縮這些原始碼  
`dpkg-source -x grep_2.12-2.dsc`



# 瞧瞧並且構建套件包

## ② 注意 debian/ 中的檔案

- ▶ 原始碼套件產生多少二進制套件？
- ▶ 這個套件使用哪一種打包小幫手？
- ▶ 根據 debian/control 的設定, 此套件只會產生一個二進制套件, 取名為grep.
- ▶ 根據debian/rules內容所述, 此套件是採用 *classic* debhelper 來進行打包, 而不是使用 *CDBS* 或者 *dh*. 可以看到它在 debian/rules 呼叫許多 *dh\_\** 指令.

## ③ 構建套件

- ▶ 使用 `apt-get build-dep grep` 來取得構建相依性
- ▶ 然後使用 `debuild` 或者 `dpkg-buildpackage -us -uc` (大約需要一分鐘)



## 編輯修改歷程

- ④ 先修改套件. 並在changelog中添加一個條目並且遞增版本號碼.
  - ▶ debian/changelog 是一個文字檔. 你可以手動編輯或者創建一條歷程記錄.
  - ▶ 或者你可以使用 `dch -i`, 使用編輯器創建一條歷程記錄
  - ▶ 名字以及電子郵件的相關設定可透過 `DEBFULLNAME` and `DEBEMAIL` 環境變數來進行配置
  - ▶ 接著重新構建套件: 產生一個新版本套件
  - ▶ 套件版本規則被定義在 Debian policy 章節 5.6.12  
<https://www.debian.org/doc/debian-policy/ch-controlfields>





## 移除 Perl 中的正規表示法支援然後重新構建

- ⑤ 接著將 perl-regex 功能移除 (位於 ./configure 中的選項)
- ⑥ 重新構建套件
  - ▶ 確認 ./configure --help: 移除 Perl 正規表示法的選項為--disable-perl-regex
  - ▶ 編輯debian/rules 然後找到./configure這一行
  - ▶ 添加 --disable-perl-regex
  - ▶ 使用debuild 或者 dpkg-buildpackage -us -uc重新構建



# 比對以及測試套件

7 使用 debdiff 來比較原始和新套件中差異

8 安裝新構建的套件

▶ 比對二進制套件: `debdiff ../changes`

▶ 比對原始碼套件: `debdiff ../dsc`

▶ 安裝新構建成的套件: `debi`  
Or `dpkg -i ../grep_<TAB>`

▶ `grep -P foo` 已失效!

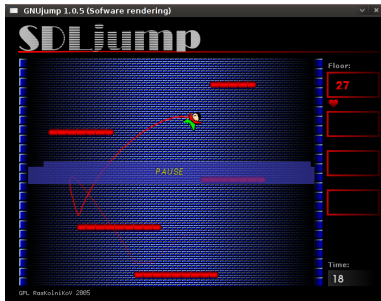
重新安裝前一版的套件:

▶ `apt-get install --reinstall grep=2.6.3-3 (= previous version)`



## 實際演練 2: 打包 GNUjump

- 1 從 <http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz> 下載 GNUjump 1.0.8
- 2 建立一個 Debian 套件
  - ▶ 安裝 build-dependencies 以進行構建套件之先行必要動作
  - ▶ 產生一個基本可運作的套件
  - ▶ 完成填入 debian/control 以及其他檔案
- 3 享受吧



## 循序漸進...

- ▶ `wget http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz`
- ▶ `mv gnujump-1.0.8.tar.gz gnujump_1.0.8.orig.tar.gz`
- ▶ `tar xf gnujump_1.0.8.orig.tar.gz`
- ▶ `cd gnujump-1.0.8/`
- ▶ `dh_make -f ../gnujump-1.0.8.tar.gz`
  - ▶ 套件種類: 單一二進制 (for now)

```
gnujump-1.0.8$ ls debian/
changelog gnujump.default.ex preinst.ex
compat gnujump.doc-base.EX prerm.ex
control init.d.ex README.Debian
copyright manpage.1.ex README.source
docs manpage.sgml.ex rules
emacsen-install.ex manpage.xml.ex source
emacsen-remove.ex menu.ex watch.ex
emacsen-startup.ex postinst.ex
gnujump.cron.d.ex postrm.ex
```



## 循序漸進...(2)

- ▶ 注意 `debian/changelog`, `debian/rules`, `debian/control` (由 **dh\_make** 自動生成)
- ▶ 在 `debian/control`:  
`Build-Depends: debhelper (>= 7.0.50 ), autotools-dev`  
列出 *build-dependencies* = `packages` 來滿足構建套件的需求
- ▶ 可透過 `debuild` (感謝 **dh**) 來構建套件
  - ▶ And add build-dependencies, until it builds
  - ▶ 提示: 使用 `apt-cache search` 以及 `apt-file` 來尋找套件
  - ▶ 範例:

```
checking for sdl-config... no
checking for SDL - version >= 1.2.0... no
[...]
configure: error: *** SDL version 1.2.0 not found!
```

→ 將 **libsdl1.2-dev** 加入 `Build-Depends` 並且安裝.

- ▶ 建議方法: 使用 **pbuilder** 來構建一個乾淨的環境



## 循序漸進...(3)

- ▶ 構建所需安裝的套件有libSDL1.2-dev, libSDL-image1.2-dev, libSDL-mixer1.2-dev
- ▶ 接著, 你可能會遇到其他錯誤:

```
/usr/bin/ld: SDL_rotozoom.o: undefined reference to symbol 'ceil@@GLIBC_2.2.5'
//lib/x86_64-linux-gnu/libm.so.6: error adding symbols: DSO missing from command line
collect2: error: ld returned 1 exit status
Makefile:376: recipe for target 'gnujump' failed
```

- ▶ 這個問題主要是因為bitrot: guijump需要照下列linker修改來調整.
- ▶ 如果你使用原始碼格式版本為**1.0**,你可以直接修改上游原始碼
  - ▶ 編輯src/Makefile.am並替換

```
gnujump_LDFLAGS = $(all_libraries)
```

成下列

```
gnujump_LDFLAGS = -Wl,--as-needed
gnujump_LDADD = $(all_libraries) -lm
```

- ▶ 接著執行 autoreconf -i



## 循序漸進...(4)

- ▶ 如果你使用原始碼格式的版本為 **3.0 (quilt)**, 請使用 `quilt` 來準備補丁. (可參照 <https://wiki.debian.org/UsingQuilt>)

- ▶ `export QUILT_PATCHES=debian/patches`
- ▶ `mkdir debian/patches`  
`quilt new linker-fixes.patch`  
`quilt add src/Makefile.am`

- ▶ 編輯 `src/Makefile.am` 並替換

```
gnujump_LDFLAGS = $(all_libraries)
```

成下列

```
gnujump_LDFLAGS = -Wl,--as-needed
gnujump_LDADD = $(all_libraries) -lm
```

- ▶ `quilt refresh`
- ▶ 由於 `src/Makefile.am` 已被修改, 所以在構建時必須要呼叫 `autoreconf`. 另外也可以使用 `dh` 來進行自動化, 可修改 `dh` 在 `debian/rules` 的指令. 從: `dh $ --with autotools-dev` 到: `dh $ --with autotools-dev --with autoreconf`



## 循序漸進...(5)

- ▶ 此套件理應完成正常構建
- ▶ 使用 `debc` 來列出套件的內容,並且使用 `debi`來進行安裝以及測試.
- ▶ 使用 `lintian`來測試套件
  - ▶ 雖不是強制性的要求, 但推薦使用 *lintian-clean*來上傳套件到Debian
  - ▶ 可透過 `lintian -EviIL +pedantic`列出更多問題
  - ▶ 小提示
    - ▶ 移除在`debian/`中不必要的檔案
    - ▶ 填好 `debian/control`
    - ▶ 透過覆蓋 `dh_auto_configure` 安裝執行檔到 `/usr/games`
    - ▶ 使用 *hardening* 編譯 flags 來增加安全性.  
參照 <https://wiki.debian.org/Hardening>





## 循序漸進... (6)

- ▶ 將你的套件和 Debian 上的套件進行差異比對
  - ▶ 它將檔案分開放到第二個套件中, 讓所有平台架構都可以一起共用(→ 可讓 Debian 檔案庫節省不必要的空間)
  - ▶ 它安裝 .desktop 檔案(給 GNOME/KDE 選單) 並且整合到 Debian 選單中
  - ▶ 透過補丁來修補一些次要的問題



## 實際演練 3: 打包 Java library

### ❶ 先看一下如何打包 Java 相關文件:

- ▶ <https://wiki.debian.org/Java>
- ▶ <https://wiki.debian.org/Java/Packaging>
- ▶ <https://www.debian.org/doc/packaging-manuals/java-policy/>
- ▶ `/usr/share/doc/javahelper/tutorial.txt.gz`

### ❷ 從 <http://moepii.sourceforge.net/> 下載 IRClib

### ❸ 開始打包



## 循序漸進...

- ▶ `apt-get install javahelper`
- ▶ 創建一個基本的原始碼套件: `jh_makepkg`
  - ▶ 函式庫
  - ▶ 無
  - ▶ Default Free compiler/runtime
- ▶ 修正 `debian/*`
- ▶ `dpkg-buildpackage -us -uc Or debuild`
- ▶ `lintian`, `debc`, 等等.
- ▶ 將你的套件和 `libirc-lib-java` 原始碼套件進行差異比對



## 實際演練 4: 打包 Ruby gem

### ❶ 先看一下如何打包 Ruby 相關文件:

- ▶ <https://wiki.debian.org/Ruby>
- ▶ <https://wiki.debian.org/Teams/Ruby>
- ▶ <https://wiki.debian.org/Teams/Ruby/Packaging>
- ▶ `gem2deb(1)`, `dh_ruby(1)` (`gem2deb` 套件)

### ❷ 從 `peach gem`:

`gem2deb peach` 建立基本的 Debian 原始碼套件

### ❸ 適時的優化, 讓它成為一個合適的 Debian 套件



## 循序漸進...

gem2deb peach:

- ▶ 從 [rubygems.org](http://rubygems.org) 下載 gem 回來
- ▶ 建立一個正確的 .orig.tar.gz 檔案,並且解壓縮
- ▶ 根據描述 gem 套件相關資訊的檔案來初始化 Debian 原始碼套件
  - ▶ 命名為 `ruby-gemname`
- ▶ 嘗試構建 Debian 二進制套件 (也許會失敗)

dh\_ruby ( 在 *gem2deb* 裡) 執行 Ruby 特定相關任務:

- ▶ 替每個 Ruby 版本皆構建 C 擴充功能
- ▶ 將檔案複製到每個目的端目錄
- ▶ 更新每一個執行腳本中的shebang符號(!)
- ▶ 執行定義在 `debian/ruby-tests.rb`, `debian/ruby-tests.rake` 中的測試, 或者其他種類的驗證 `debian/ruby-test-files.yaml`.



## 循序漸進...(2)

優化產生的套件:

- ▶ 執行 `debclean` 來清除原始碼樹工具. 請看 `debian/`.
- ▶ `changelog` 以及 `compat` 需要被修正
- ▶ 編輯 `debian/control`: 優化 `Description`
- ▶ 根據上游檔案來寫對應的 `copyright`
- ▶ 構建套件
- ▶ 將你的套件和 Debian 檔案庫中的 `ruby-peach` 套件比對



## 實際演練 5: 打包 Perl 模組

### ❶ 先看一下如何打包 Perl 相關文件:

- ▶ <https://perl-team.pages.debian.net>
- ▶ <https://wiki.debian.org/Teams/DebianPerlGroup>
- ▶ `dh-make-perl(1)`, `dpt(1)` (in the `pkg-perl-tools` package)

### ❷ 從 Acme CPAN 發行版:

`dh-make-perl --cpan Acme` 建立基本的 Debian 原始碼套件

### ❸ 適時的優化, 讓它成為一個合適的 Debian 套件



## 循序漸進...

`dh-make-perl --cpan Acme:`

- ▶ 從 CPAN 下載壓縮檔案
- ▶ 創建適合的 `.orig.tar.gz` 檔案, 並且解壓縮
- ▶ 根據描述發行版相關資訊的檔案初始化 Debian 原始碼套件
  - ▶ 取名為 `libdistname-perl`





## 循序漸進...(2)

優化產生的套件:

- ▶ `debian/changelog`, `debian/compat`, `debian/libacme-perl.docs`, 以及 `debian/watch` 這些內容需正確無誤
- ▶ 編輯 `debian/control`: 優化 `Description`, 並且移除底部的樣版
- ▶ 編輯 `debian/copyright`: 將置頂的樣版移除, 加入著作權的年份到 `Files: *` 的章節中



本教學指南由 SZ Lin (林上智) 翻譯成繁體中文

若有任何翻譯上的建議, 請發信至 `<debian-chinese-big5@lists.debian.org>`

