Debian 打包教程

Lucas Nussbaum
packaging-tutorial@packages.debian.org

version 0.30 – 2024-03-16

关于该教程

- 目标：**告诉你关于 Debian 打包必须知道的知识**
  - 修改现有软件包
  - 创建你自己的软件包
  - 在 Debian 社区内与人交流
  - 成为 Debian 高手
- 涵盖了大部分要点，但不是全部
  - 你还需要看更多的说明文档
- 绝大部分内容同样适用于 Debian 的各个衍生版本
  - 包括 Ubuntu

内容概括

内容概括

Debian

- **是一款 GNU/Linux 发行版**
- 第一款遵循“GNU 开放精神”而开发的大型发行版
- **非商业用途**，由超过 1000 名志愿者合作构建
- 3 项主要特性：
  - textbf 高质量——技术至上主义
    尽善尽美才可发行
  - **自由**——开发者和用户以社会契约的方式联系在一起，
    共同推广 1993 年开始的自由软件精神
  - **独立**——没有（单独一家）公司在照看 Debian
    我们集思广益群策群力（高行动力 + 民主决策）
- **业余爱好者**们的理想状态：与同好们一起用爱发电

Debian 软件包

- **.deb** 文件（程序文件包）
- 可靠又方便的软件发行方式
- 两种应用最广泛的软件包格式之一（另一种是 RPM）
- 概况：
  - Debian 有超过 30,000 个软件包
    → 绝大多数可用的自由软件都被打包进了 Debian！
  - 覆盖 12 种架构，包括两款非 Linux 架构（Hurd; KFreeBSD）
  - 同时被用于 120 款 Debian 衍生系统

Deb 软件包格式

- .deb 文件：其实是一种 ar 压缩包，里面包含了以下文件：

```
$ 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 文件格式的版本，"2.0\n"
  - control.tar.gz: 软件包的元数据 (metadata)，包括 control, md5sums, (pre|post)(rm|inst), triggers, shlibs, …
  - data.tar.gz: 软件包的数据文件

- 你可以手动创建你自己的 .deb 文件

  http://tldp.org/HOWTO/html_single/Debian-Binary-Package-Building-HOWTO/

- 但大家一般不这么做

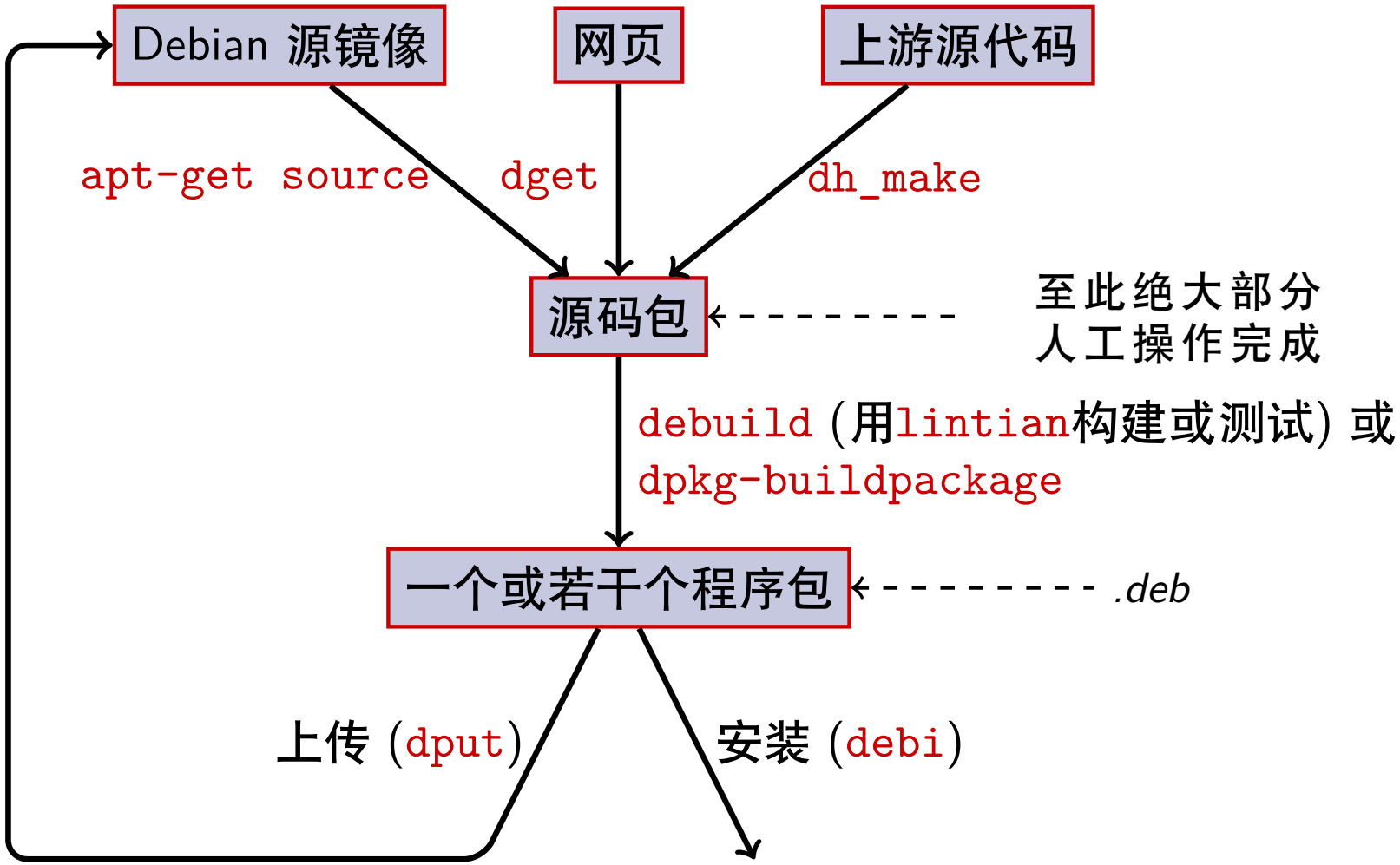
此教程会教你：如何用 Debian 的方式创建 Debian 软件包

你需要准备的工具

- Debian（或 Ubuntu）操作系统（且你有 root 权限）
- 某些软件包：
  - **build-essential**：包含了在开发机上使用的依赖包（不需要逐一把这些依赖包在你软件包的命令控制参数 `Build-Depends:` 里列出来）
    - 也包含了 **dpkg-dev** 的依赖包。**dpkg-dev** 里有基本的 Debian 专用打包工具。
  - **devscripts**：里面有很多对 Debian 维护者很有用的脚本。

之后还会提到很多其他工具，譬如 **debhelper**, **cdbs**, **quilt**, **pbuilder**, **sbuild**, **lintian**, **svn-buildpackage**, **git-buildpackage**, ...
在你需要的时候可以安装它们。

通用打包流程


Debian 源镜像
网页
上游源代码
apt-get source
dget
dh_make
源码包
至此绝大部分
人工操作完成
debuild (用lintian构建或测试) 或
dpkg-buildpackage
一个或若干个程序包
.deb
上传 (dput)
安装 (debi)

范例：重构 dash

1. 安装构建 dash 需要的软件包，以及 devscripts
   `sudo apt-get build-dep dash`
   (需要在 `/etc/apt/sources.list`) 文件里加上 `deb-src` 的相关地址内容
   `sudo apt-get install --no-install-recommends devscripts fakeroot`
2. 新建工作目录文件夹，并进入
   `mkdir /tmp/debian-tutorial ; cd /tmp/debian-tutorial`
3. 获取 `dash` 的源码包
   `apt-get source dash`
   (需要在 `/etc/apt/sources.list` 文件里加上 `deb-src` 的相关地址内容)
4. 构建软件包
   `cd dash-*`
   `debuild -us -uc`　(`-us -uc` 指不将软件包标记为 GPG)
5. 检查命令是否正常运作
   - 上级目录下应该会出现几个新的 `.deb` 文件
6. 查看 `debian/` 目录
   - 打包完成后的软件包位于此目录

内容概括

源码包

- 一个源码包可以生成若干个程序文件包
  例如 `libtar` 源代码可以生成 `libtar0` 和 `libtar-dev` 程序文件包
- 两种软件包：（如果不确定是哪种，选非原生）
  - 原生软件包：专为 Debian 定制开发的软件 (*dpkg*, *apt*)
  - 非原生软件包：在 Debian 之外开发的软件
- 主要文件：`.dsc`（元数据）
- 与源码格式版本相关的其他文件
  - 1.0 或 3.0（原生）：`package_version.tar.gz`
  - 1.0（非原生）：
    - `pkg_ver.orig.tar.gz`：上游源代码
    - `pkg_debver.diff.gz`：专为 Debian 准备的更新补丁
  - 3.0（quilt）
    - `pkg_ver.orig.tar.gz`：上游源代码
    - `pkg_debver.debian.tar.gz`：包含 Debian 补丁的压缩包

详见 `dpkg-source(1)`

源码包范例（wget_1.12-2.1.dsc）

```
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
```

从现成源码包中获取

- 从 Debian 源代码库里获取
  - `apt-get source` *package*
  - `apt-get source` *package=version*
  - `apt-get source` *package/release*

  (需要在 `sources.list` 文件里加上 `deb-src` 的相关地址内容)

- 从互联网上获取：
  - `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` 提供了自 2005 年起 Debian 的所有软件包)

- 从（已声明的）版本控制系统中获取：
  - `debcheckout` *package*

- 下载完毕后，用该命令解压缩：`dpkg-source -x` *file.dsc*

创建基础源码包

- 下载上游源代码
  (上游源代码 = 软件开发者提供的源代码)
- 将源代码包重命名为
  `<source_package>_<upstream_version>.orig.tar.gz`
  (例如: `simgrid_3.6.orig.tar.gz`)
- 解压缩
- 重命名目录为 `<source_package>-<upstream_version>`
  (例如: `simgrid-3.6`)
- `cd <source_package>-<upstream_version> && dh_make`
  (利用 **dh-make** 软件包)
- 如果要用其他的 `dh_make` 命令来处理某些特定的源码包种类，还需要安装对应的软件包，如 **dh-make-perl**, **dh-make-php**, ...
- 这时 `debian/` 目录就会被创建，里面有一大堆文件

debian/ 目录下的文件

所有的打包工作都应该通过修改 `debian/` 目录下的文件来进行

- 主要文件：
  - **control**——软件包的元数据（依赖包，之类）
  - **rules**——规定了如何构建软件包
  - **copyright**——软件包的版权信息
  - **changelog**——Debian 软件包的更新历史记录
- 其他文件：
  - 兼容文件
  - 监测文件
  - dh_install* 的保存目录

    *.dirs, *.docs, *.manpages, ...
  - 维护脚本

    *.postinst, *.prerm, ...
  - 源码/格式
  - 补丁/ ——如果你需要修改上游源代码
- 有几个文件使用基于 RFC 8222 的格式（邮件 header 格式）

debian/changelog

- Debian 打包更新历史记录
- 列出软件包的当前版本

$$\underbrace{1.2.1.1}_{\text{上游版本}}\text{-}\underbrace{5}_{\substack{\text{Debian}\\ \text{版本}}}$$

- 手动编辑或使用dch命令
  - 为新发行的软件创建更新日志：`dch -i`
- 自动关闭 Debian 或 Ubuntu bug 的特殊字符：
  Debian: `Closes: #595268`; Ubuntu: `LP: #616929`
- 安装在 /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

- 软件包 metadata
  - 源码包的元数据
  - 从源代码构建出来的每个程序文件包的元数据
- 软件包名称，分类，优先级，维护者，上传者，build 依赖包，本体依赖包，描述介绍，主页，…
- 说明文档：Debian 政策第 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
```

架构 (Architecture)：全部 (all) 或任意 (any)

有两种程序文件包：

- 每种 Debian 架构上有不同内容的软件包
  - 例如：C 语言编程
  - `debian/control` 文件内 `Architecture: any`
    - 或者，仅在某几种架构下可用：
      `Architecture: amd64 i386 ia64 hurd-i386`
  - buildd.debian.org：会在你上传软件包时自动为你构建所有其他架构
  - 命名为 *package_version_architecture*.deb

- 所有架构下内容都相同的软件包
  - 举例：Perl 函数库
  - `debian/control` 文件内 `Architecture: all`
  - 命名为 *package_version*_all.deb

一个源码包可以生成既有 `Architecture: any` 程序又有 `Architecture: all` 程序的文件包。

debian/rules

- 用于编译源代码生成文件 (Makefile)
- 用于构建 Debian 软件包的接口
- 说明文档在 Debian 政策，章节 4.8
  `https://www.debian.org/doc/debian-policy/ch-source#s-debianrules`
- 用到的目标参数：
  - `build`，`build-arch`，`build-indep`: 这几项包括了所有配置和编译方法。
  - `binary`，`binary-arch`，`binary-indep`: 构建程序文件包
    - `dpkg-buildpackage` 命令可以调用参数 `binary` 来构建所有的软件包，也可以调用参数 `binary-arch` 只构建 `Architecture: any` 的软件包
  - `clean`: 清除源目录下的文件

打包助手——debhelper

- 你可以直接在 `debian/rules` 文件里写 shell 代码
- 更好的打包实操方法（大多数软件包都用此方法）：使用打包助手
- 最流行的打包助手：**debhelper**（98% 的软件包用它打包）
- 目标：
  - 归纳出用标准打包工具完成打包任务时适用于所有软件包的通用操作
  - 一次性解决所有软件包中共同存在的打包 bug

dh_installdirs, dh_installchangelogs, dh_installdocs, 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, ...

  - 调用 `debian/rules`
  - 可以使用命令行参数或者修改 `debian/` 目录内的文件来进行配置

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

- 特定软件包种类的第三方助手：**python-support**, **dh_ocaml**, ...
- `debian/compat`: Debhelper compatibility version
  - Defines precise behaviour of dh_*
  - New syntax: `Build-Depends: debhelper-compat (= 13)`

用到 debhelper 的 debian/rules 文件写法 (1/2)

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

# Uncomment this to turn on verbose mode.
#export DH_VERBOSE=1

build:
        $(MAKE)
        #docbook-to-man debian/packagename.sgml > packagename.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/packagename
        $(MAKE) DESTDIR=$(CURDIR)/debian/packagename install
```

用到 debhelper 的 debian/rules 文件写法（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

- 即使用了 debhelper，打不同的包依然有许多重复劳动
- 需要更进一步的归纳通用操作的助手软件
  - 例如：用 `./configure && make && make install` 或者 CMake 构建
- CDBS:
  - 出现在 2005 年，基于进阶 *GNU make* 功能
  - 说明文档：`/usr/share/doc/cdbs/`
  - 支持 Perl, Python, Ruby, GNOME, KDE, Java, Haskell, ...
  - 但也有人很讨厌它：
    - 有时候会让自定义软件包构建变得困难：
      “搞得人头昏脑胀的文件生成和环境变量”
    - 比普通的 debhelper 要慢（这是对 `dh_*` 的大量无效调用导致的）

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

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

Dh（也叫 Debhelper 7，或 dh7）

- 在 2008 年以 *CDBS killer* 的名义被公布
- **dh** 命令调用 `dh_*`
- 简易的 *debian/rules* 文件，仅仅列出覆盖内容
- 比 CDBS 更容易定制
- 说明文档: man 手册 (`debhelper(7)`, `dh(1)`) + DebConf9 的幻灯片文档
  `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
```

经典 debhelper vs CDBS vs dh

- 份额比例:
  经典 debhelper: 15%　CDBS:15%　dh: 68%
- 我该学用哪个？
  - 最好全学一点
  - 你得先搞懂 debhelper 才能用好 dh 和 CDBS
  - 你可能需要修改 CDBS 软件包
- 打包新软件用哪个?
  - **dh** (份额比例不断上升的不二选择)
  - See `https://trends.debian.net/#build-systems`

内容概括

构建软件包

- `apt-get build-dep mypackage`
  安装 *build-dependencies*（构建依赖库）(如果该包已经在 Debian 源内)
  或者 `mk-build-deps -ir` (如果该包还没被上传过)

- `debuild`: 使用 `lintian` 构建、测试，使用 GPG 签名

- 也可以直接调用 `dpkg-buildpackage`
  - 通常使用命令 `dpkg-buildpackage -us -uc`

- 尽可能在干净而精简的系统环境中构建软件包
  - `pbuilder` – 在 *chroot* 中构建软件包的助手不错的说明文档:
    `https://wiki.ubuntu.com/PbuilderHowto`
    (优化: `cowbuilder ccache distcc`)
  - `schroot` 和 `sbuild`: 用于 Debian 构建后台
    (不如 `pbuilder` 简单，但允许使用 LVM 快照
    详见: `https://help.ubuntu.com/community/SbuildLVMHowto`)

- 生成 `.deb` 文件和 `.changes` 文件
  - `.changes`: 说明构建了哪些东西；上传软件包的时候用到

安装和测试软件包

- 在本机安装软件包: `debi` (查看 `.changes` 来了解要安装哪些东西)
- 列出软件包内容: `debc ../mypackage< 标签 >.changes`
- 与一版本的软件包进行对比:
  `debdiff ../mypackage_1_*.changes ../mypackage_2_*.changes`
  或者直接对比源代码
  `debdiff ../mypackage_1_*.dsc ../mypackage_2_*.dsc`
- 用 `lintian` (静态检测工具) 检查软件包:
  `lintian ../mypackage< 标签 >.changes`
  `lintian -i`: 反馈更多错误信息
  `lintian -EviIL +pedantic`: 显示更多问题
- 把软件包上传到 Debian (`dput`) (需要配置)
- 用 `reprepro` 或 `aptly` 管理个人 Debian 档案库
  说明文档: `https://wiki.debian.org/HowToSetupADebianRepository`

内容概括

实操练习环节 1：修改 grep 软件包

1. 访问 `http://ftp.debian.org/debian/pool/main/g/grep/` 并下载 2.12-2 版本的 grep 软件包
   - 如果源码包没有自动解包，用 `dpkg-source -x grep_*.dsc` 命令解包
2. 查看 `debian/` 目录下的文件。
   - 此源码包生成了多少个程序文件包？
   - 此软件包用了哪个打包助手？
3. 构建软件包
4. 现在我们要修改这个软件包。添加一条更新记录并且增加一位版本号。
5. 然后停用 perl-regexp 的支持（这是 `./configure` 里的一条选项）
6. 重构软件包
7. 用 debdiff 对比原版和新版软件包的区别
8. 安装新构建的软件包

内容概括

debian/copyright

- 源代码和打包的版权及证书信息
- 以前都是写成文本文件
- 现在用的可被机器识别的格式:
  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]
```

修改上游源代码

常用于：

- 修复 bug 或添加 Debian 专属定制内容
- 从新的上游版本回注（backport）各种修复

有几种方法

- 直接修改文件
  - 简单
  - 但不能追踪和记录各种更新
- 使用补丁系统
  - 简化了将更新提交到上游源代码的操作
  - 与其他衍生系统分享你的修复内容
  - 让更多人知道你的更新
    `http://patch-tracker.debian.org/` (暂不可用)

补丁系统

- 原则：更新以补丁形式存储在 `debian/patches/`
- 在构建中实装或卸装
- 以前：几种实现方式——*simple-patchsys* (*cdbs*), *dpatch*, ***quilt***
  - 每一种都支持两个 `debian/rules` 目标参数：
    - `debian/rules patch`: 实装所有补丁
    - `debian/rules unpatch`: 卸装所有补丁
  - 更多说明文档：`https://wiki.debian.org/debian/patches`
- **包含内建补丁系统的新源码包格式：3.0（quilt）**
  - 推荐使用这个方案
  - 你得先学用一下 *quilt*
    `https://perl-team.pages.debian.net/howto/quilt.html`
  - 补丁工具在 `devscripts`: `edit-patch`

补丁说明文档

- 补丁开头要有标准的头标（header）
- 说明文档在 DEP-3—补丁标签引导
  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 {
```

安装或移除过程中的操作

- 仅仅解压缩软件包通常是不够的
- 新建/移除系统用户，开始/停止服务，管理马甲命令（*alternatives*）
- 用维护脚本（*maintainer scripts*）完成
  `preinst`, `postinst`, `prerm`, `postrm`
  - debhelper 可以生成通用操作的重复脚本
- 说明文档:
  - Debian 政策说明, 第 6 章
    `https://www.debian.org/doc/debian-policy/ch-maintainerscripts`
  - Debian 开发者手册, 第 6.4 章
    `https://www.debian.org/doc/developers-reference/best-pkging-practices.html`
  - `https://people.debian.org/~srivasta/MaintainerScripts.html`
- 用户指引
  - 需要用 **debconf** 做好用户指引
  - 说明文档: `debconf-devel(7)` (`debconf-doc` package)

监测上游版本

- 在 debian/watch 文件中指定监测目标 (详见 uscan(1))

```
version=3

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

- 有网站会自动追踪上游软件的新版本，且会通过控制台通知维护者，譬如 https://tracker.debian.org/ 和 https://udd.debian.org/dmd/
- uscan: 手动检查命令
- uupdate: 尝试将你的软件包更新到最新的上游版本

用版本控制系统打包

- 有些工具可以帮你在打包工作中管理分支版本（branch）和标签：
  svn-buildpackage, git-buildpackage
- 例如：git-buildpackage
  - upstream 分支会通过 upstream/*version* 标签追踪上游版本
  - master 分支用于追踪 Debian 软件包
  - debian/*version* 标签用于每次上传
  - pristine-tar 分支用于重构上游压缩包

  文档：http://honk.sigxcpu.org/projects/git-buildpackage/manual-html/gbp.html
- Vcs-* debian/control 文件里用于定位软件仓库的代码块
  - https://wiki.debian.org/Salsa

```
Vcs-Browser: https://salsa.debian.org/debian/devscripts
Vcs-Git: https://salsa.debian.org/debian/devscripts.git

Vcs-Browser: https://salsa.debian.org/perl-team/modules/packages/libwww-perl
Vcs-Git: https://salsa.debian.org/perl-team/modules/packages/libwww-perl.git
```

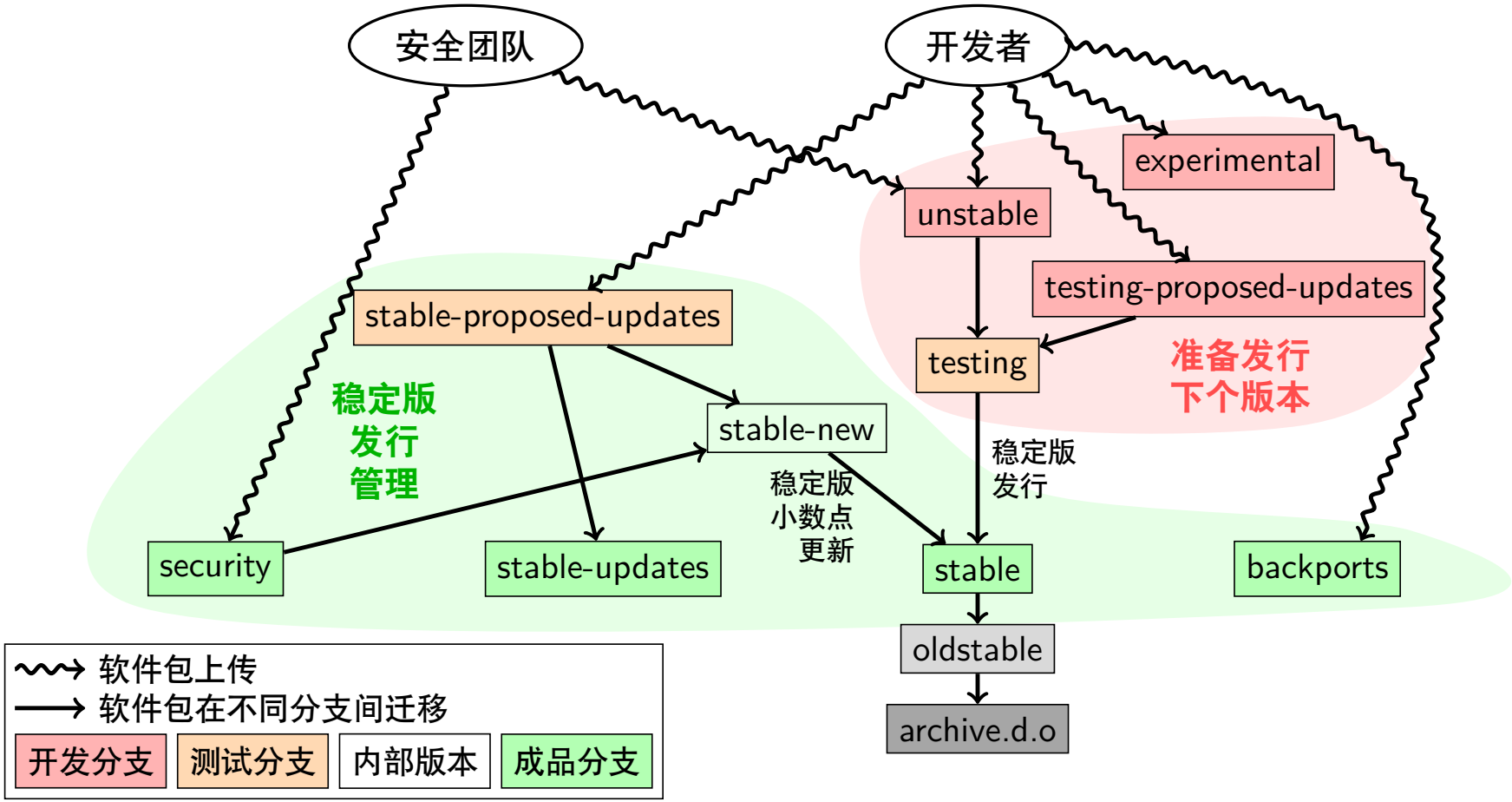
- VCS-交互接口：debcheckout, debcommit, debrelease
  - debcheckout grep → 检查来自 Git 的源码包

回注（Backport）软件包

- 目标：在老系统版本里使用新版软件包
  例如，在 Debian*stable* 分支使用 Debian *unstable* 分支的 *mutt* 软件包
- 通用方法：
  - 直接从 Debian unstable（开发）分支里照搬源码包
  - 修改并构建使之正常用于 Debian stable（稳定）分支
    - 有时候没啥必要（不用改动代码）
    - 有时候很麻烦
    - 有时候不可能（有很多不可用的依赖库）
- 有些软件包回注是由以下这个 Debian 项目来提供和支持的
  `http://backports.debian.org/`

内容概括

Debian 档案库和分支



该图参考 Antoine Beaupré 的图表绘制。https://salsa.debian.org/debian/package-cycle

开发分支

- 新的软件包版本一般都上传到 **unstable (sid)** 分支
- **unstable**（开发）分支中的软件在符合一定标准（例如在开发分支运行了 10 天且没有回滚过版本）后会从 **unstable** 分支迁移到 **testing**（测试）分支
- 也可以把新软件包上传到
  - **experimental**（实验）分支（用于某些实验性的软件包，如某个还没有确定将用于替换当前 unstable 分支版本的新版本）
  - **testing-proposed-updates**（用于测试的更新）分支，不经过 **unstable** 直接更新到 **testing** 分支（很少这么做）

冻结和发行

- 在发行周期中的某个时间点，发行团队会 *freeze*（冻结）testing 分支：停止软件在 **unstable** 和 **testing** 分支间的自动迁移，改为人工审核
- 当发行团队认为 **testing** 分支已经足够完善，可以对外发行时：
  - **testing** 分支变为新的 **stable**（稳定）分支
  - 同时，原来的 **stable** 分支成为 **oldstable**（旧稳定）分支
  - 不再提供支持的发行版则归入 `archive.debian.org`
- 详见 `https://release.debian.org/`

稳定版的分支和管理

- 我们使用若干分支来发行稳定版软件包
  - **stable**: 主分支
  - 安全团队在 `security.debian.org` 网站提供 **security** 分支更新，同时也会向订阅 `debian-security-announce` 的邮件列表推送更新通知
  - **stable-updates**: 与安全无关但很紧急的更新（等不到下一次小数点更新）：杀毒软件数据库，时区调整相关的软件包，等等。向订阅 `debian-stable-announce` 的邮件列表推送通知
  - **backports**: 从 **testing** 分支回注的新上游版本
- **stable** 分支通常每几个月会进行一次 *stable point releases* （小数点更新）(一般就是修修 bug)
  - 随下一次小数点更新一起更新的软件包会被上传到 **stable-proposed-updates** 分支，并且由发行团队审核
- **oldstable** 旧稳定版的分支系列保持不变

几种为 Debian 做贡献的方式

- **最糟糕的**的贡献方式：
  1. 打包你自己的应用
  2. 放进 Debian
  3. 不管了

- **更好的**贡献方式：
  - 加入打包团队
    - 许多专注于打包的团队需要人手
    - 团队列表 `https://wiki.debian.org/Teams`
    - 这也是向高手们请教学习的绝佳途径
  - 认领现有的未维护软件包（*orphaned packages* 遗弃软件包）
  - 为 Debian 提供新软件
    - 请务必只提供足够有趣/有用的软件，拜托
    - 注意是不是已经有类似软件被打包上传过了？

认领遗弃软件包

- Debian 里有很多未维护软件包
- 详细的列表 + 当前状态：`https://www.debian.org/devel/wnpp/`
- 你可以在自己系统里安装这两个包来查看：`wnpp-alert` 或 `how-can-i-help`
- 不同的软件包状态
  - **O**rphaned: 遗弃：该软件包无人维护
    可以直接认领
  - **RFA**: **R**equest **F**or **A**dopter 求认领
    现维护者正在找人接盘，有人接盘前会继续维护。
    可以直接认领，记得认领前给现维护者写一封邮件告知
  - **ITA**: **I**ntent **T**o **A**dopt 打算认领
    有人想要认领某软件包
    你可以去帮他!
  - **RFH**: **R**equest **F**or **H**elp 求帮忙
    维护者想要找人一起维护
- 有一些未维护软件包还没有被检测到 → 尚未被遗弃
- 如果你不确定，可以询问 `debian-qa@lists.debian.org` 或 `irc.debian.org` 的 `#debian-qa` 频道

认领软件包：范例

```
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
```

- 记得联系一下上任维护者（特别是该软件包处于 RFA 状态而不是遗弃状态时），这是礼貌
- 最好也能联系一下上游开发者

把你的软件包放进 Debian

- 你不需要获得任何官方身份就能直接上传你的软件包到 Debian
  1. 用 `reportbug wnpp` 来提交 **ITP** (**I**ntent **T**o **P**ackage 有意向打包) 的 bug
  2. 准备好源码包
  3. 找一个肯帮你上传软件包的 Debian Developer（DD—Debian 开发者）

- 官方身份（如果你已经是一个熟练的软件包维护者）:
  - **Debian Maintainer (DM——Debian 维护者):**
    有权上传你自己的软件包
    详见 `https://wiki.debian.org/DebianMaintainer`
  - **Debian Developer (DD——Debian 开发者):**
    Debian 项目成员；有权投票或上传任何软件包

在找人上传前你需要检查的事项

- Debian **极重视软件质量**
- 通常，**维护者们都非常忙**
  - 在找人上传前务必确保你的软件包已经准备妥当
- 要检查的事项：
  - 不要遗漏构建依赖库：确保你的软件包是在干净的 *sid chroot* 环境里构建的
    - 推荐使用 `pbuilder`
  - 为你的软件包运行一下 `lintian -EviIL +pedantic`
    - 修复 Error 错误和各种其他问题
  - 当然还要做好各种延申测试
- 有问题就求助

哪里可以求助？

你可能需要的帮助：

- 针对你的疑问给出建议和解答，以及代码审核
- 在你准备好软件包后替你上传

你可以通过以下方式求助：

- **各打包团队的成员**
  - 打包团队列表：`https://wiki.debian.org/Teams`
- **Debian 导师组**（如果你的软件包没有团队接）
  - `https://wiki.debian.org/DebianMentorsFaq`
  - 邮件列表：`debian-mentors@lists.debian.org`
    (正好也是个学习的好途径)
  - IRC 聊天：`irc.debian.org` 上的 `#debian-mentors` 频道
  - `http://mentors.debian.net/`
  - 说明文档：`http://mentors.debian.net/intro-maintainers`
- **本地化团队邮件列表** (以你的母语求助)
  - `debian-devel-{french,italian,portuguese,spanish}@lists.d.o`
  - 完整列表：`https://lists.debian.org/devel.html`
  - 或者用户列表：`https://lists.debian.org/users.html`

更多说明文档

- Debian 开发者圈子
  `https://www.debian.org/devel/`
  很多 Debian 开发方面的资源链接
- Debian 维护者指南
  `https://www.debian.org/doc/manuals/debmake-doc/`
- Debian 开发者手册
  `https://www.debian.org/doc/developers-reference/`
  主要内容是关于 Debian 自身发展，但也有一些极好的打包实操 (part 6)
- Debian 政策
  `https://www.debian.org/doc/debian-policy/`
  - 这是每个软件包都必须达到的硬性要求
  - 针对 Perl，Java，Python，...都有专用条款
- Ubuntu 打包指南
  `https://packaging.ubuntu.com/html/`

Debian 维护者查询面板

- **源码包中心**:
  `https://tracker.debian.org/dpkg`
- **维护者/团队中心**：：Developer's Packages Overview (DDPO——开发者软件包概览)
  `https://qa.debian.org/developer.php?login=pkg-ruby-extras-maintainers@lists.alioth.debian.org`
- **待办事项表**：：Debian Maintainer Dashboard (DMD——Debian 维护者查询面板)
  `https://udd.debian.org/dmd/`

使用 Debian Bgu 追踪系统（BTS）

- 非常独特的 bug 管理方式
  - 用 web 接口查看 bug
  - 用 Email 接口更新 bug
- 给 bug 添加相关信息：
  - 发邮件给 123456@bugs.debian.org (不包括 bug 提交者，你得手动抄送给 123456-submitter@bugs.debian.org)
- 更新 bug 状态：
  - 给 control@bugs.debian.org 发邮件
  - 命令行接口：在 devscripts 里用 bts 命令
  - 说明文档：https://www.debian.org/Bugs/server-control
- 报告 bug：用 reportbug
  - 一般使用本地邮件服务器：安装 ssmtp 或者 nullmailer
  - 或者用 reportbug --template，然后 (手动) 发给 submit@bugs.debian.org

使用 BTS（bug 追踪系统）：范例

- 发邮件给 bug 和提交者：
  https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680822#10
- 更改标签和更新严重程度：
  https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680227#10
- 分配任务，更新严重度，重命名 ...：
  https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=680822#93
  - notfound, found, notfixed, fixed 是给 **version-tracking**（版本追踪）提供的标签
    详见 https://wiki.debian.org/HowtoUseBTS#Version_tracking
- 使用用户标签：https://bugs.debian.org/cgi-bin/bugreport.cgi?msg=42;bug=642267
  详见 https://wiki.debian.org/bugs.debian.org/usertags
- BTS 说明文档：
  - https://www.debian.org/Bugs/
  - https://wiki.debian.org/HowtoUseBTS

对 Ubuntu 更感兴趣？

- Ubuntu 主要管理与 Debian 的差异部分
- Ubuntu 并不关注特定的软件包，
  而是与 Debian 团队合作
- 通常情况下，推荐将新软件包先上传到 Debian
  `https://wiki.ubuntu.com/UbuntuDevelopment/NewPackages`
- 还有个更好的方案：
  - 加入 Debian 团队然后负责与 Ubuntu 沟通
  - 协助降低两者差异性，收集整理 Launchpad 里的 bug
  - 用好 Debian 的多种工具：
    - 开发者软件包概览（Developer’s packages overview）里的 Ubuntu 部分
    - 软件包追踪系统 (Package Tracking System) 里的 Ubuntu 内容
    - 通过 PTS 收取 launchpad 的 bugmail 邮件

内容概括

结论

- 虽然你已经对 Debian 打包有了一个完整的大致印象
- 但你还需要阅读更多说明文档
- 最好的打包实操方法经过了多年的优化
  - 如果不确定，就用 **dh** 打包助手，以及 **3.0 (quilt)** 格式

反馈：**packaging-tutorial@packages.debian.org**

合规信息

为该教程做出贡献

- 贡献方式：
  - `apt-get source packaging-tutorial`
  - `debcheckout packaging-tutorial`
  - `git clone https://salsa.debian.org/debian/packaging-tutorial.git`
  - `https://salsa.debian.org/debian/packaging-tutorial`
  - 已知 bugs: `bugs.debian.org/src:packaging-tutorial`
- 提供反馈：
  - `mailto:packaging-tutorial@packages.debian.org`
    - 还需要在教程里添加哪些内容？
    - 还有哪些内容需要优化？
  - `reportbug packaging-tutorial`

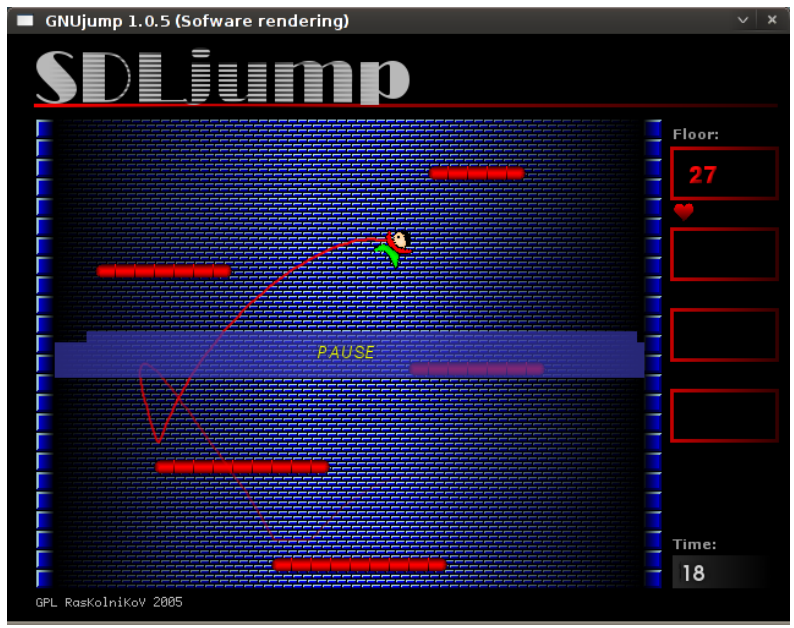
内容概括

实操练习环节 2：打包 GNUjump

1. 下载 GNUjump 1.0.8
   `http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz`

2. 创建 Debian 软件包
   - 安装构建依赖库，用于构建软件包
   - 修复 bug
   - 获得基础工作包
   - 填写 `debian/control` 和其他文件

3. 开搞



实操练习环节 2：打包 GNUjump（提示）

- 要获得基础工作包，使用 `dh_make`
- 刚上手的话，可以先建一个 *1.0* 格式的源码包，比 *3.0 (quilt)* 简单一点（在 `debian/source/format` 里修改格式）
- 要搜索缺失的构建依赖库，可以找到缺失文件，使用 `apt-file` 来查找缺失软件包
- 如果你遇到错误：

```
/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
```

你需要在关联命令行后加上 `-lm`：
编辑 `src/Makefile.am` 并将

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

替换为以下内容

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

然后运行 `autoreconf -i`

实操练习环节 3：打包 Java 库

1. 快速浏览以下打包 Java 的说明文档：
   - 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
2. 从http://moepii.sourceforge.net/ 下载 IRClib
3. 打包

实操练习环节 4：打包 Ruby 库（Gem）

1. 快速浏览以下打包 Ruby 的说明文档：
   - `https://wiki.debian.org/Ruby`
   - `https://wiki.debian.org/Teams/Ruby`
   - `https://wiki.debian.org/Teams/Ruby/Packaging`
   - `gem2deb(1)`, `dh_ruby(1)` (在 `gem2deb` 软件包里)
2. 将 `peach` 库创建成基础 Debian 源码包：
   `gem2deb peach`
3. 把它优化成合格的 Debian 软件包

实操练习环节 5：打包 Perl 模组

1. 快速浏览以下打包 Perl 的说明文档：
   - `https://perl-team.pages.debian.net`
   - `https://wiki.debian.org/Teams/DebianPerlGroup`
   - `dh-make-perl(1)`, `dpt(1)` (在 `pkg-perl-tools` 软件包里)
2. 将 `Acme` 的 CPAN 发行版创建成基础 Debian 源码包：
   `dh-make-perl --cpan Acme`
3. 把它优化成合格的 Debian 软件包

内容概括

解答

实操练习环节

实操练习环节 1：修改 grep 软件包

1. 访问 `http://ftp.debian.org/debian/pool/main/g/grep/` 并下载 2.12-2 版本的 grep 软件包
2. 查看 `debian/` 目录下的文件。
   - 此源码包生成了多少个程序文件包？
   - 此软件包用了哪个打包助手？
3. 构建软件包
4. 现在我们要修改这个软件包。添加一条更新记录并且增加一位版本号。
5. 然后停用 perl-regexp 的支持（这是 `./configure` 里的一条选项）
6. 重构软件包
7. 用 debdiff 对比原版和新版软件包的区别
8. 安装新构建的软件包

获取源代码

1. 访问 http://ftp.debian.org/debian/pool/main/g/grep/ 并下载 2.12-2 版本的 grep 软件包

- 用 dget 命令下载.dsc 文件:
  dget http://cdn.debian.net/debian/pool/main/g/grep/grep_2.12-2.dsc
- 如果你的 Debian 版本里的 deb-src 有 2.12-2 版的 grep (可以在 https://tracker.debian.org/grep 查询)，你可以: apt-get source grep=2.12-2
  或 apt-get source grep/release (例如 grep/stable)
  或者，干脆直接: apt-get source grep
- grep 源码包由以下三个文件构成
  - grep_2.12-2.dsc
  - grep_2.12-2.debian.tar.bz2
  - grep_2.12.orig.tar.bz2

  这是典型的"3.0 (quilt)" 格式。
- 如果有需要可以解压缩源码文件
  dpkg-source -x grep_2.12-2.dsc

查看一下各个文件，然后构建软件包

2. 查看 debian/ 下的文件
   - 此源码包生成了多少个程序文件包？
   - 此软件包用了哪个打包助手？

- 根据 debian/control，该源码包只生成了一个程序文件包，名为 grep.
- 根据 debian/rules，该软件包用经典 debhelper 打包，而没有用 *CDBS* 或 *dh*。可以发现在 debian/rules 里调用了大量 dh_* 命令。

3. 构建软件包

- 先用 apt-get build-dep grep 命令获取构建依赖库
- 再用 debuild 或 dpkg-buildpackage -us -uc 命令（耗时约 1 分钟）

编辑更新日志

4. 现在我们要修改这个软件包。添加一条更新记录并且增加一位版本号。

- debian/changelog 是个文本文件。你可以手动编辑并添加条目。
- 或者你也可以用 dch -i 来添加条目并打开编辑器
- 姓名和邮件可以通过环境变量 DEBFULLNAME 和 DEBEMAIL 来定义
- 做完这些再重构一下软件包：这样新版本的软件包就构建好了
- 软件包的版本号使用规则在 Debian 政策 5.6.12 章节有详细阐述。
  https://www.debian.org/doc/debian-policy/ch-controlfields

停用 Perl regexp 支持并重构

5. 然后停用 perl-regexp 的支持（这是 `./configure` 里的一条选项）
6. 重构软件包

- 用`./configure --help` 命令查看：停用 Perl regexp 的选项是 `--disable-perl-regexp`
- 编辑 `debian/rules` 文件，找到`./configure` 所在行
- 在命令后添加`--disable-perl-regexp`
- 用 `debuild` 或 `dpkg-buildpackage -us -uc` 命令重构

对比和测试软件包

7. 用 debdiff 对比原版和新版软件包的区别
8. 安装新构建的软件包

- 对比程序文件包：`debdiff ../*changes`
- 对比源码包：`debdiff ../*dsc`
- 安装新构建的软件包：`debi`
  或 `dpkg -i ../grep_<TAB>`
- `grep -P foo` 已经失效！

重装上一版本的软件包：

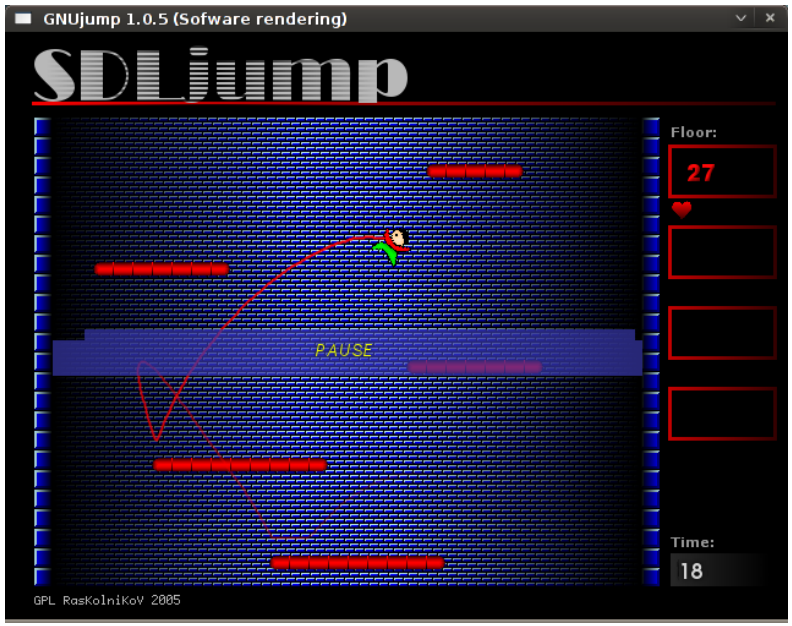
- `apt-get install --reinstall grep=2.6.3-3` (= 上个版本)

实操练习环节 2：打包 GNUjump

1. 下载 GNUjump 1.0.8
   `http://ftp.gnu.org/gnu/gnujump/gnujump-1.0.8.tar.gz`

2. 创建 Debian 软件包
   - 安装构建依赖库，用于构建软件包
   - 获得基础工作包
   - 填写 `debian/control` 和其他文件

3. 开搞



具体步骤...

- `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`
  - 当前软件包类型：单程序文件（暂时）

```
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
```

具体步骤...(2)

- 查看 `debian/changelog`, `debian/rules`, `debian/control`（由 **dh_make** 自动填写）
- `debian/control` 文件内容：

  `Build-Depends: debhelper (>= 7.0.50 ), autotools-dev`

  用 *build-dependencies* = 构建所需软件包名称的格式列出构建依赖库
- 尝试用 `debuild` 的“像 xx 一样”功能构建软件包（得益于 **dh** 的神奇功能）
  - 然后添加构建依赖库，开始构建
  - 提示：用 `apt-cache search` 和 `apt-file` 命令查找软件包
  - 例如：

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

    → 将 **libsdl1.2-dev** 添加到构建依赖库并安装。
  - 如果可以尽量用 **pbuilder** 在干净的系统环境里构建。

具体步骤...(3)

- 用到的构建依赖库有 `libsdl1.2-dev`, `libsdl-image1.2-dev`, `libsdl-mixer1.2-dev`
- 接下来你大概又会遇到如下错误:

```
/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
```

- 这个问题是 bitrot 产生的：gnujump 没有跟着调整文件连接
- 如果你用的是源码格式版本 **1.0**，你可以直接改上游源代码。
  - 编辑 `src/Makefile.am`，将

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

    替换为以下内容

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

  - 然后运行 `autoreconf -i`

具体步骤...(4)

- 如果你用的是源码格式版本 3.0 (quilt)，就用 quilt 命令制作补丁文件。(详见 https://wiki.debian.org/UsingQuilt)
  - export QUILT_PATCHES=debian/patches
  - mkdir debian/patches
    quilt new linker-fixes.patch
    quilt add src/Makefile.am
  - 编辑 src/Makefile.am，将

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

    替换为以下内容

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

  - quilt refresh
  - 因为修改了 src/Makefile.am，所以在构建时要调用一下 autoreconf。用 dh 命令可以自动调用，将 debian/rules 里的 dh 调用代码块从：
    dh $ --with autotools-dev
    改成：dh $ --with autotools-dev --with autoreconf

具体步骤...(5)

- 现在软件包已经能正常构建了
- 用 `debc` 列出所生成的软件包内容，用 `debi` 安装并测试。
- 用 `lintian` 测试软件包
  - 虽然不是必须，但强烈推荐上传到 Debian 的软件包都是 *lintian-clean*（lintian 测试无误）的
  - 更多问题可以用 `lintian -EviIL +pedantic` 命令列出来
  - 几条提醒：
    - 将 `debian/`里不需要的文件删掉
    - 填写 `debian/control`
    - 修改 `dh_auto_configure` 来将可执行文件安装到`/usr/games`
    - 利用 *hardening* compiler flags（硬化编译器标识）增加安全性。详见`https://wiki.debian.org/Hardening`

具体步骤...(6)

- 将你的软件包与 Debian 中现有的软件包进行对比
  - 你的软件包在所有架构下统一地将 data 数据文件分离到了另一个软件包里（→ 节约了 Debian 档案库的空间）
  - 你的软件包装有.desktop 文件（用于 GNOME/KDE 菜单）并将其整合进了 Debian 菜单
  - 你的软件包用补丁修复了几个小问题

实操练习环节 3：打包 Java 库

1. 快速浏览以下打包 Java 的说明文档：
   - 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
2. 从http://moepii.sourceforge.net/ 下载 IRClib
3. 打包

具体步骤...

- `apt-get install javahelper`
- 创建基础源码包：`jh_makepkg`
  - 库
  - 空
  - 默认免费编译器/运行环境
- 查看并修正 `debian/*`
- `dpkg-buildpackage -us -uc` 或 `debuild`
- `lintian`, `debc`, 等等
- 将你的结果与 `libirclib-java` 源码包进行对比

实操练习环节 4：打包 Ruby 库（Gem）

1. 快速浏览以下打包 Ruby 的说明文档：
   - `https://wiki.debian.org/Ruby`
   - `https://wiki.debian.org/Teams/Ruby`
   - `https://wiki.debian.org/Teams/Ruby/Packaging`
   - `gem2deb(1)`, `dh_ruby(1)` (在 `gem2deb` 软件包里)
2. 将 `peach` 库创建成基础 Debian 源码包：
   `gem2deb peach`
3. 把它优化成合格的 Debian 软件包

具体步骤...

`gem2deb peach`:

- 从 rubygems.org 下载 gem 库
- 创建合适的.orig.tar.gz 压缩文件，然后解压缩
- 基于 gem 库的元数据初始化一个 Debian 源码包
  - 命名为 `ruby-`*`gemname`*
- 尝试构建 Debian 程序文件包（可能会失败）

`dh_ruby` (包含在 *gem2deb* 内) 命令可以完成 Ruby 专属任务：

- 为每个 Ruby 版本构建 C 语言扩展
- 把文件复制到目标目录
- 更新可执行脚本的 shebang 开头
- 运行 `debian/ruby-tests.rb`, `debian/ruby-tests.rake`, 或 `debian/ruby-test-files.yaml`, 里定义地测试，以及其他各项检测

具体步骤...(2)

优化生成的软件包：

- 运行 `debclean` 清理源码结构，查看 `debian/`。
- `changelog` 和 `compat` 必须准确无误
- 编辑 `debian/control`：优化 `Description`
- 基于上游文件编写正确的 `copyright` 版权文件
- 构建软件包
- 将你的软件包与 Debian 档案库的 `ruby-peach` 软件包进行对比

实操练习环节 5：打包 Perl 模组

1. 快速浏览以下打包 Perl 的说明文档：
   - `https://perl-team.pages.debian.net`
   - `https://wiki.debian.org/Teams/DebianPerlGroup`
   - `dh-make-perl(1)`, `dpt(1)` (在 `pkg-perl-tools` 软件包里)
2. 将 `Acme` 的 CPAN 发行版创建成基础 Debian 源码包：
   `dh-make-perl --cpan Acme`
3. 把它优化成合格的 Debian 软件包

具体步骤...

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

- 从 CPAN 下载压缩包
- 创建合适的.orig.tar.gz 压缩文件，然后解压缩
- 基于分发文件的元数据初始化一份 Debian 源码包
  - 命名为 `lib`*`distname`*`-perl`

具体步骤...(2)

优化生成的软件包：

- 确保 `debian/changelog`, `debian/compat`, `debian/libacme-perl.docs`, 和 `debian/watch` 文件都准确无误
- 编辑 `debian/control`: 优化 `Description`, 删除底下的自动模板代码
- 编辑 `debian/copyright`: 删除顶上的样板代码，在 `Files: *` 那节添加版权年份

翻译

本教程由 Nessaj Wang 翻译为简体中文

若您想对翻译内容提出意见，请发邮件至
<debian-l10n-chinese@lists.debian.org>.