第 2 章 第一步

目录

让我们来创建一个自己的软件包 (更好的是, "领养"一个已存在的软件包)。

2.1. Debian 软件包制作流程

如果你的工作基于某个上游程序, 那么典型的 Debian 软件包制作流程就会需要如下几个特定的文件:

- 获取一份上游软件的拷贝, 它通常为压缩过的 tar 格式。
  - ***package*-*version*.tar.gz**
- 在上游源码的 **debian** 目录下添加 Debian 打包的专用文件 (构建 Debian 软件包它们会被读取), 同时以 **3.0 (quilt)** 格式创建一个非本地源码包。
  - ***package*_*version*.orig.tar.gz**
  - ***package*_*version*-*revision*.debian.tar.gz** [4]
  - ***package*_*version*-*revision*.dsc**
- 用 Debian 源码包构建 Debian 二进制包;这些二进制包的格式通常是 **.deb** (或者 **.udeb**, Debian Installer 专用)
  - ***package*_*version*-*revision*_*arch*.deb**

请注意一个细节, 在 Debian 软件包的文件名中, 分隔 ***package*** 和 ***version*** 的字符从 tarball 名称中的 - (连字符)换成了 **_** (下划线)。

在上述的文件名中, 请根据 Debian Policy, 将 ***package*** 这个部分替换为 **package name**, 将 ***version*** 这个部分替换为 **upstream version**, 将***revision*** 这个部分替换为 **Debian revision**, 以及将 ***arch*** 这个部分替换为 **package architecture**。[5]

本概要中的每一步, 我们都会在后续的章节中辅以详细的例子进行解释。

2.2. 挑一个你喜欢的程序

可能你已经想好了要制作的软件包。那此时第一件要做的事, 就是检查它是否已经被别人打包, 并已经安置于发行版仓库中。这里有几种检查方法供您参考:

- **aptitude** 命令
- the Debian packages 页面
- Debian 软件包追踪系统 网页

如果软件包已经存在于仓库中, 那直接装上它就可以了呀！:-) 如果它是被 **甩掉了(orphaned)** 的 —— 也就是说它的维护者被设置为 Debian QA Group, 这时候你可以尝试接手维护它。另外, 你也可以"领养"那些维护者发送过"Request for Adoption"(**RFA**)请求的软件包。[6]

软件包归属状态有这几种:

- **wnpp-alert** 命令, 来自 **devscripts** 软件包
- Work-Needing and Prospective Packages
- Debian Bug report logs: Bugs in pseudo-package **wnpp** 位于 **unstable**
- Debian Packages that Need Lovin'
- Browse **wnpp** bugs based on debtags

作为旁注必须指出, Debian 已经拥有了海量各种类型的软件包, 而且处在仓库中软件包的数量也远远超过了拥有上传权限的贡献者数量。因此, 为已经在仓库中的软件包贡献力量是非常受其他开发者欢迎的 (且更容易获得 sponsorship)[7]。有非常多的方式可以实现这一目的:

- 接手那些已经被甩掉, 而仍然有很多人在使用的软件包
- 加入一些 打包小组
- 帮忙为某些常用软件分类 Bug
- 在必要时, 帮忙准备 QA 或 NMU 上传

如果你有能力"领养"那个软件包, 那就先下载 (使用 **apt-get source *packagename*** 或其他类似的工具)并分析它的源代码。这篇文档不会详细说明如何领养软件包, 不过幸运的是, 领养软件包时, 打包的起始工作已经有人完成, 接手的工作应比从头开始轻松得多。尽管如此也请您不要轻敌, 请继续阅读, 下面给出的建议会对你很有帮助。

如果您要制作的软件包是完全崭新的, 而您又希望它出现在 Debian 中, 那请遵循以下的步骤:

- 首先, 你必须知道这个软件能不能干活, 而且你需要亲自试用一段时间来确保其可用性。
- 一定要在 Work-Needing and Prospective Packages 上仔细查看, 以确定并没有其他人已经开始相关的工作。如果没有的话, 则可以提交一份 ITP (Intent To Package, 打包意向) Bug 报告给 **wnpp** 虚包 (可以使用 **reportbug**)。如果你看到已经有人在处理相关事宜, 则在有需要的情况下再联系他(们)。如果他(们)不需要你的帮助, 那就寻找其他你感兴趣, 而且没有人维护的软件包咯。
- 软件 **必须有许可证**。
  - 对于仓库中 **main** 区的软件, Debian Policy 要求其 **完全兼容 Debian Free Software Guidelines (Debian 自由软件准则)** (DFSG) 并且它 **不能要求使用 main** 区以外的软件来编译或执行。这即是最理想的状况。
  - 对于仓库中 **contrib** 区的软件, 其许可证必须满足 DFSG 的全部条件, 不同于 main 区软件的一点是, 它们可以依赖于 **main** 之外的软件包来完成编译或运行。
  - 对于仓库中 **non-free** 区的软件, 其许可证可以不满足 DFSG 中的一部分条件。其中坚决不能违背的一点是, 该软件 **必须是可分发的**。
  - 如果你不确定你的软件应该归入仓库的哪一个区, 那你可以把许可证文本发送到 debian-legal@lists.debian.org 邮件列表上寻求意见。
- 程序 **不能** 给 Debian 系统带来安全或维护问题。
  - 程序应当带有良好的文档, 最好是源代码也容易理解 (比如, 不混乱)。
  - 你应该与程序的作者取得联系, 问问他是否认为程序应当被打包, 以及他是否对 Debian 持友好态度。能够询问作者关于程序的任何问题都非常重要, 所以千万不要尝试打包一个无人维护的软件 (遇到麻烦找谁去呢)。
  - 程序一定 **不能** setuid 到 root 。如果它不 setuid 或 setgid 到任何用户或组的话, 那就再好不过了。
  - 程序不应该是守护进程, 也不应该进入任何一个 ***/sbin** 目录, 不应该以 root 权限打开任何端口。

当然, 这些都是为了安全, 并试图避免你在, 比如 setuid 守护进程等问题上犯错误而激怒了用户 ... 当你在打包方面有了更多经验时, 就能够处理这样的软件包了, 不必着急。

我们鼓励你, 作为一个新维护者, 选择易于打包和维护的软件, 而不鼓励选择复杂的软件包。

- 简单软件包
  - 生成单个二进制包, arch = all (比如像壁纸那样的资料集)
  - 生成单个二进制包, arch = all (用解释型语言编写的可执行脚本文件, 比如 POSIX shell 语言)
- 中等复杂软件包
  - 生成单个二进制包, arch = any (用C/C++等语言编写的 ELF 二进制可执行文件)
  - 生成多个二进制包, arch = any + all (包含ELF二进制可执行程序+文档)
  - 既不是 **tar.gz** 也不是 **tar.bz2** 格式的上游源代码包
  - 包含不可分发的内容物的源码包
- 高复杂度软件包
  - 被其他软件包使用的解释器模块包
  - 被其他软件包使用的ELF库文件包

- 生成多个二进制包，其中包含括ELF库文件
- 有多个上游的源码包
- 内核模块包
- 内核补丁包
- 含有关键维护者脚本的软件包

打包高复杂软件包并非难如登天，但这个过程需要用到更多知识。你应该针对每一个所谓的复杂特性来搜寻相应的指南。比如，一些语言有它们自己的子策略文档：

- Perl policy
- Python policy
- Java policy

还有一句拉丁谚语：*fabricando fit faber*（熟能生巧）。我们 *强烈* 建议你在阅读这篇教程的时候，用一个简单的软件包来对所有的 Debian 打包步骤进行实验。跟随下边的步骤您就可以创建一个微不足道的软件包 **hello-sh-1.0.tar.gz**，而且这会是一个非常好的开端：[8]

```
$ mkdir -p hello-sh/hello-sh-1.0; cd hello-sh/hello-sh-1.0
$ cat > hello <<EOF
#!/bin/sh
# (C) 2011 Foo Bar, GPL2+
echo "Hello!"
EOF
$ chmod 755 hello
$ cd ..
$ tar -cvzf hello-sh-1.0.tar.gz hello-sh-1.0
```

2.3. 获取程序并试用

这里第一件事是找到并下载原始的源代码。我们假定你已经在作者的主页上找到了程序的源代码。一般来说，Unix 下的自由软件源代码是以 **tar+gzip** 格式(扩展名为 **.tar.gz**)或 **tar+bzip2** 格式(扩展名为 **.tar.bz2**)或**tar+xz** (扩展名为 **.tar.xz**) 的文件格式提供的。有时候，归档文件中包含了一个名为 ***package-version*** 的子目录，在那里边有全部的源代码。

如果最新版本的源代码可通过像 Git，Subversion，CVS 这样的版本控制系统获得的话，你可以用 **git clone**， **svn co**，或 **cvs co** 来下载它，然后将它重新打包压缩为 **tar+gzip** 格式，同时别忘了 **--exclude-vcs** 选项哟。

如果你的程序源代码是以其他文件格式提供的 (比如文件名以 **.Z** 或 **.zip** 结尾 [9])，则应当使用对应和合适的工具将其解压，再重新归档为 Tarball。

如果你挑选的程序的源代码中包含一些不符合 DFSG 的内容，你应当解压后移除它们，然后将删减过的源码重新归档。因为 DFSG 而进行删减过的源码归档的上游版本号中，需要添加 **dfsg** 这个标识。

作为示例，本教程在这里使用一个名为 **gentoo** 的程序，它是一个 GTK+ 文件管理器。[10]

在你的家目录下创建一个子目录，命名为 **debian** 或 **deb** 或者依你的喜好。(本例中使用 **~/gentoo**)。把下载好的归档文件放在其中并解压 (使用 **tar xzf gentoo-0.9.12.tar.gz** 命令)。你需要确定这个归档在解压过程中没有任何发生错误，即便是有一点 *小小小问题* 也是不行的。因为在别人的系统上解压这些归档时，可能他们的工具并不会忽略这些非正常现象，于是就出问题了。一般的预期是，在你的终端屏幕上，应该能看到如下的情形。

```
$ 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
```

现在解压缩工具创建了一个新的子目录，名为 **gentoo-0.9.12**。接下来你需要进入该目录并 *彻底* 读完其中的文档。通常情况下文档会被命名为 **README***、**INSTALL***、***.lsm** 或 ***.html**。同时，你需要学会正确编译和安装这个程序 (最可能出现的一种情况是，软件会被默认安装到 **/usr/local/bin** 目录；但在 Debian 软件包制作过程中不可以那样做。详细的原因在 第 3.3 节 “文件安装” 中说明)。

开始打包时的源代码目录应当是绝对干净原始，没有做过任何修改的。一般来说直接使用刚刚解压得到的源代码即可。

2.4. 简单的编译系统

有些比较简单的程序源码自己带有 **Makefile**，这时你可以很容易地使用 **make** 命令来编译它。[11] 有一些软件的 Makefile 还支持 **make check**，这个命令可以完成一系列程序检验和测试。当程序编译好后即可用 **make install** 命令，将程序安装到目标目录。

现在尝试编译和运行你的程序，你需要确保它能正常工作，以及它在安装和运行时不会破坏别的东西。

你还可以运行 **make clean** (或更好的 **make distclean**) 来清理编译目录。Makefile 中有时还会支持 **make uninstall**，它被用来卸载已经安装了的程序文件。

2.5. 常见的可移植编译系统

有非常非常多的自由软件是使用 C 和 C++ 语言编写的。其中又有很多程序使用了 Autotools 或 CMake 来使其可以移植到不同平台上。这些工具被用于生成 **Makefile** 和其他必须的源文件。然后程序就可以使用正常的 **make; make install** 来编译和安装。

Autotools 是 GNU 编译系统工具集，包括 Autoconf、Automake、Libtool 和 gettext。你可以通过 **configure.ac**、**Makefile.am** 和 **Makefile.in** 等特征文件来识别使用 Autotools 作为编译系统的源代码。[12]

Autotools 工作流程的第一步是在上游作者在代码中运行 **autoreconf -i -f**，然后把生成的文件同源代码一起分发。

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

编辑 **configure.ac** 和 **Makefile.am** 等文件前，你需要一些关于 **autoconf** 和 **automake** 的知识。请参看 **info autoconf** 和 **info automake**。

Autotools 工作流程的第二步是，用户获得分发的源代码后在源码目录下运行 **./configure && make** 来将其编译成为 ***binary*** (二进制文件)。

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

你可以改变 **Makefile** 文件的许多设置，比如你可以修改默认的文件安装路径(使用 **./configure --prefix=/usr**)。

可选：若使用 **autoreconf -i -f** 来更新 **configure** 和其他相关文件，则有可能可以提高源代码的兼容性。[13]

CMake 是另一个备选的编译系统，你可以通过 **CMakeLists.txt** 这个特征文件来识别使用 CMake 的源代码。

2.6. 软件包名称和版本

如果上游源代码以 **gentoo-0.9.12.tar.gz** 这样的文件名来分发，你就可以用 **gentoo** 作为 **软件包名**，并用 **0.9.12** 作为 **上游版本**。它们会被 **debian/changelog** 这个文件用到；第 4.3 节 “**changelog**” 部分会详细描述这个文件。

虽然此法在大部分情况下能显灵，但你仍需要根据 Debian 政策(Debian Policy)以及约定俗成的做法来调整 **软件包名** 和 **上游版本**。

在 **软件包名** 里只能含有 小写字母 (**a-z**)，数字 (**0-9**)，加号 (**+**) 和 减号 (**-**)，以及 点号 (**.**)。软件包名最短长度两个字符；它必须以字母开头；它不能与仓库软件包名发生冲突。还有，把软件包名的长度控制在 30 字符以内是明智之举。[14]

如果上游代码在它的名称中使用了一些通用术语比如 **test-suite**，那么你应当将其重命名，显式地说明其内容并避免污染命名空间。[15]

你应该让 **upstream version(上游版本号)** 只包含字母和数字 (**0-9A-Za-z**)，以及加号 (**+**)，波浪号 (**~**)，还有 点号(**.**)。它必须以数字开头 (**0-9**)。[16] 如果可能的话，最好把它的长度控制在8字符以内。[17]

如果上游不使用像 **2.30.32** 这样的常规版本格式，而是用类似 **11Apr29** 这样的日期作为版本，类似于随机的代号字符串，或者以VCS的哈希值作为版本号的一部分，那么请确认将其从 **upstream version** 中移除。为此作出的改动信息可以记录在 **debian/changelog** 文件中。如果你需要发明一个版本字符串，请使用 **YYYYMMDD** 这个格式作为上游版本，比如 **20110429**。这会确保 **dpkg** 在升级软件包时能够正确解读新版本。如果需要确保未来能够平滑过渡到类似 **0.1** 这样的版本号的话，那就请使用 **0~YYMMDD** 格式作为上游版本，例如 **0~110429**。

版本字符串 [18] 可以用 dpkg(1) 来进行比较：

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

版本比较规则可总结为以下几点：

- 字符串要从头到尾进行比较。

- 字母比数字大。
- 数字作为整数进行比较。
- 字母按照 ASCII 编码顺序进行比较。
- 对于点号 (.)，加号 (+)，以及波浪号 (~) 则有对应的特殊规则，具体如下：

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

有一种比较棘手的情况，当上游释出 **gentoo-0.9.12-ReleaseCandidate-99.tar.gz** 作为 **gentoo-0.9.12.tar.gz** 的预发布版本时，就需要确保升级工作妥当进行：重命名该上游源代码为 **gentoo-0.9.12~rc99.tar.gz**.

2.7. 设置 dh_make

首先设置两个 环境变量，**$DEBEMAIL** 和 **$DEBFULLNAME**，这样大多数 Debian 维护工具就能够正确识别你用于维护软件包的姓名和电子邮件地址。[19]

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

2.8. 初始化外来 Debian 软件包

一般来说，由上游程序产生的 Debian 软件包都是 外来 的。若你想要用上游源代码 **gentoo-0.9.12.tar.gz** 创建一个外来 Debian 软件包，你可以在它的基础上进行外来软件包初始化，这只需要调用 **dh_make** 命令：

```
$ cd ~/gentoo
$ wget http://example.org/gentoo-0.9.12.tar.gz
$ tar -xvzf gentoo-0.9.12.tar.gz
$ cd gentoo-0.9.12
$ dh_make -f ../gentoo-0.9.12.tar.gz
```

当然，这里请用你原版源码归档的名字来替换 filename (文件名)。[20] 详情请参见 dh_make(8) 。

你会看到一些输出，询问你想要创建什么类型的软件包。这里的 Gentoo 被规划为一个单一二进制包——它仅仅产生一个二进制包，亦即单个 **.deb** 文件——于是我们就选择第一项 (按 **s** 键)，认真阅读屏幕上的输出信息，然后按 ***ENTER*** 键来确认。[21]

执行 **dh_make** 后，上级目录中自动创建了一份上游 tarball 的副本，名为 **gentoo_0.9.12.orig.tar.gz**，这个文件和稍后要介绍的 **debian.tar.gz** 合在一起便满足了一部分 Debian 非本土源码包的要求。

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

请注意在 **gentoo_0.9.12.orig.tar.gz** 这个文件名中有两个关键点：

- 软件包名称 和 版本 中间以字符 **_**（下划线）来分隔。
- 后缀名 **.tar.gz** 前边插有 **.orig**。

你或许注意到了 **debian** 目录下已经有了许多模板。这些文件将在 第 4 章 *debian 目录中的必需内容* 和 第 5 章 *debian 目录下的其他文件* 中一一加以说明。另外，打包这件事情无法被完全自动化。因此你还得按照 第 3 章 *修改源代码* 中介绍的方法为 Debian 修改软件包。再接下来，按照 第 6 章 *构建软件包* 中叙述的合适的方法来构建 Debian 软件包，并根据 第 7 章 *检查软件包中的错误* 中的方法进行软件包测试，最后参考 第 9 章 *上传软件包* 这里的说明将其上传。所有的这些主要步骤本教程都会进行解释。

如果你在修改过程中不小心删除或玩坏了某些模板，你还可以使用 **dh_make** 加 **--addmissing** 参数来将其还原。(译注：也可以用dpkg -L dh-make 来寻找您想要的模板)

更新一个已存在的软件包可能有点复杂，因为它可能使用了旧的打包技术。在学习基本功的阶段，建议只创建全新的软件包；稍后的 第 8 章 *更新软件包* 中会细致地讲解更新现存软件包。

请注意，源代码中不必包含任何在 第 2.4 节 “简单的编译系统” 或 第 2.5 节 “常见的可移植编译系统” 中谈论到的编译系统。就算源码包仅仅是一组图像之类的也可以，这时候这些文件的安装可以用 **debhelper** 的配置文件来搞定，比如 **debian/install** (参见 第 5.11 节 “**install**”)。

---

[4] 对于老式的 **1.0** 格式非本地 Debian 源码包，应当使用 ***package_version-revision*.diff.gz** 这个命名规则。

[5] 参见 5.6.1 "Source"，5.6.7 "Package"，以及 5.6.12 "Version"。**package architecture** 遵循 Debian Policy Manual, 5.6.8 "Architecture" 并且会在软件包构建的过程中被自动分配。

[6] 参见 Debian Developer's Reference 5.9.5. "Adopting a package".

[7] 当然了，无论何时，总会有一些值得打包，而且并未进入 Debian 仓库的新软件。

[8] 不用担心失踪的 **Makefile**。你可以参照 第 5.11 节 “**install**”，简单地通过 **debhelper** 来安装 **hello** 程序，或者修改上游源代码来添加带有**install**目标的新**Makefile**，参照第 3 章 *修改源代码*。

[9] 当通过文件扩展名不足以判断文件类型时，可以使用 **file** 命令来判断。

[10] 这个程序已经被打包好了。当前的版本 使用 Autotools 作为其编译系统(build structure)，并且已经和下边的例子不一样了。下边的例子基于老的版本 0.9.12 。

[11] 许多新时代的程序都配有一个叫做 **configure** 的脚本。执行它的时候会生成一个为你的计算机专门定制的 **Makefile** 。

[12] Autotools 这个庞然大物显然已经超出本教程的讨论范围，毕竟本文主要提供关键字和提示。如果你需要使用 Autotools，请认真研读 Autotools Tutorial 以及 **/usr/share/doc/autotools-dev/README.Debian.gz** 的本地副本。

[13] 软件包 **dh-autoreconf** 可以帮助你将这个过程自动化。参见 第 4.4.3 节 “**定制 rules 文件**”.

[14] 在 **aptitude** 工具中，软件包名字段的默认最大长度为 30。而 90% 以上的软件包包名都少于 24 个字符。

[15] 如果你遵循 Debian Developer's Reference 5.1. "New packages"，那么在 ITP 过程中很容易遇到这样的问题。

[16] 这一条更严格的规则能帮助你避免混淆文件名。

[17] **aptitude** 的版本字段默认长度为10。通常其中的 Debian 修订号和前置的连字符会消耗2个字符位置。对 80% 以上的软件包来说，上游版本小于8字符，Debian 修订号小于2字符。对 90% 以上的软件包来说，上游版本小于10字符，Debian 修订号小于3字符。

[18] 版本字符串可以是 **upstream version** (***version***)，**Debian revision** (***revision***)，或者 **version** (***version*-*revision***)。关于 Debian 修订号如何增长，请参见 第 8.1 节 “新的 Debian 版本”**Debian revision** 。

[19] 以下内容默认你以 Bash 作为登陆 shell。如果你使用其他的 shell，例如 Z shell，那就使用它们的配置文件代替这里提到的 **~/.bashrc**。

[20] 如果上游源代码已经提供了有内容的 **debian** 目录，那么带上参数 **--addmissing** 来执行**dh_make** 命令。新的源码包格式 **3.0 (quilt)** 的鲁棒性(Robust)已经足够优秀，以不至于轻易损坏。另外，你可能需要修改上游提供的内容，以满足你的 Debian 软件包之需。

[21] 这里有这几种选择：**s** 代表单一二进制包，**i** 代表独立于体系结构的软件包，**m** 代表多个二进制包，**l** 代表共享库文件包，**k** 代表内核模块包，**n** 代表内核补丁包，**b** 代表 **cdbs** 软件包。本教程专注于使用 **dh** 命令 (来自 **debhelper** 软件包) 来创建单一二进制包，但也会涉及到创建 独立于体系结构 或 多个二进制软件包 相关的内容。软件包 **cdbs** 提供了 另一套可以代替 **dh** 命令的基础打包脚本，不过这个家伙已经超出了我们的讨论范围。