

Debian 新维护人员手册

版权声明

版权所有 © 1998–2002 Josip Rodin.

本文档可以在 GNU 通用公众授权的第 2 或更高的版本下使用。

This document was made using with these two documents as examples:

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

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

目录

- 1 从一条正确的路开始
 - 1.1 开发时需要的软件
 - 1.2 其它信息
- 2 第一步
 - 2.1 选择你的程序
 - 2.2 获得程序，并且试用它
 - 2.3 软件包名称和版本
 - 2.4 首次“Debian 化”
- 3 修改源代码
 - 3.1 在一个子目录中安装
 - 3.2 不一样的库名称
- 4 debian/目录中必需的内容
 - 4.1 “control” 文件
 - 4.2 “copyright” 文件
 - 4.3 “changelog” 文件
 - 4.4 “rules” 文件
- 5 debian/中的其它文件
 - 5.1 README.Debian
 - 5.2 conffiles.ex
 - 5.3 cron.d.ex
 - 5.4 dirs
 - 5.5 docs
 - 5.6 emacs-en-*.ex
 - 5.7 init.d.ex
 - 5.8 manpage.1.ex, manpage.sgml.ex
 - 5.9 menu.ex
 - 5.10 watch.ex
 - 5.11 ex.package.doc-base

- 5.12 postinst.ex, preinst.ex, postrm.ex, prerm.ex
- 6 构建软件包
 - 6.1 完整的 rebuild
 - 6.2 快速 rebuild
 - 6.3 debuild 命令
 - 6.4 dpatch 系统
 - 6.5 在上传时包含 orig.tar.gz
- 7 检查软件包中的错误
 - 7.1 lintian 软件包
 - 7.2 mc 命令
 - 7.3 debdiff 命令
 - 7.4 interdiff 命令
 - 7.5 debi 命令
 - 7.6 pbuilder 包
- 8 上传软件包
 - 8.1 上传到 Debian 档案库
 - 8.2 上传到私有的档案库
- 9 更新软件包
 - 9.1 新的 Debian 修订版
 - 9.2 新的上游版本(基本)
 - 9.3 新的上游版本 (实际的)
 - 9.4 orig.tar.gz 文件
 - 9.5 cvs-buildpackage 命令和 similes
 - 9.6 校验软件包的升级
- 10 在哪里可以找到帮助
- A 例子
 - A.1 简单打包例子
 - A.2 用 dpatch 和 pbuilder 打包

Debian 新维护人员手册

第 1 章 - 从一条正确的路开始

这篇文章为普通的 Debian 用户和希望能够对 Debian 安装包有所了解的开发人员讲述 了制作 Debian 安装包的方法。它使用了非常通用的语言，并且通过一个可以工作的 例子进行了演示。有一句古老的罗马谚语说的好：Longum iter est per praecepta, breve et efficax per exempla! (通过理论要讲述很久的问题， 可以很快地用例子说明白。)

Debian 能够成为一个高质量的 Linux 发行版的重要原因之一就是它的安装包系统。 尽管已经存在相当大量的用 Debian 格式打包的软件，有时你还是需要安装一些不是 这一格式的软件。可能你会为如何制作自己的安装包而彷徨，而且也许你会认为这 是一个非常困难的任务。是的，如果你是一个 Linux 初学者，那么这的确很难，不过 如果你真的是一个新手，现在你也就不会来读这个文档了。:-) 你的确需要 对 Unix 的编程有所了解，但你并不需要是这方面的天才。

有一件事情是非常明确的：如果你希望创建并维护一个 Debian 的安装包，那将花费 你数个小时的时间。作为一个维护人员，为了能够不犯错误，让我们的系统很好地 工作，必须有良好的技术基础且非常勤奋。

这篇文档将会讲述每一个细节(开始时也许给人感觉毫不相关)的步骤，并且帮助你 创建第一个安装包，从而让你学习到可以帮助你制作它的下一个版本或者其它安装 包的的经验。

这个文档的更新版本可以在 <http://www.debian.org/doc/maint-guide/>和 “maint-guide” 安 装包中找到。这个文档的中文翻译版本也可以在 “maint-guide-xy” 安装 包中找到。

1.1 开发时需要的软件

在开始之前，你需要确认你是否已经正确安装了一些附加的在开发时需要的安装 包。注意这里列出的软件都没有标记为 “essential” 或者是 “required” ——我们希望 你能够安装好这些软件。

这个文档的当前版本已经为 Debian 2.2(“potato”)和 3.0(“Woody”)更新过了。

下面列出的这些软件在 Debian 的标准安装中已经有了，因此它们在你的机器上应 当已经安装好了(也包括它们依赖的其它软件包)。然而，你还是应该 用 “dpkg -s <package>” 来检查一下。

- dpkg-dev - 这个软件包包括了在解开、制作、上传 Debian 源文件包时 需要用到的工具。(参考 dpkg-source(1))
- file - 这个小程序可以检测文件的类型。(参考 file(1))
- gcc - GNU C 语言编译器，如果你的程序是和其它很多程序一样用 C 语 言编写，那么就需要这个软件包。(参考 gcc(1))这 个软件包还会 “pull in” 其它几个软件包，比如包括汇编(assembly)和链 接(link)目标文件的程序的软件包 binutils(参 考

binutils-doc 软件包中的“info binutils”)和 C 预处理器(preprocessor)cpp(参考 cpp(1))。

- g++ - GNU C++语言编译器，如果你的程序是用 C++语言写的那就需要它。(参考 g++(1))
- libc6-dev - gcc 需要的用于链接和创建目标文件的 C 函数库和头文件。(参考 glibc-doc 软件包中的“info libc”)
- make - 通常创建一个程序的过程需要经过好几步，为了能够不把同样的命令一遍又一遍的输入，你可以用这个程序通过创建“Makefile”来使这个过程自动化。(参考“info make”)
- patch - 这是一个非常有用的工具，它可以把(用 diff 程序生成的)包 含有一个差别清单的文件应用到原先的文件上去，从而生成一个补丁版本。(参考 patch(1))
- perl - Perl 是在当今的 Unix 类系统上应用得非常广泛的解释型脚本语言之一，它通常被称作“Unix 的瑞士军刀”。(参考 perl(1))

你也很可能会想要安装下面的软件包：

- autoconf 和 automake - 很多新的程序在这一类工具的帮助 下来配置脚本文件并对 Makefiles 进行预处理。(参考“info autoconf”和“info automake”)
- dh-make 和 debhelper - 在例子中创建我们的软件包时需要 使用 dh-make 来创建它的骨架，并且要使用到一些 debhelper 软件包中的工具。他们对于创建软件包不是最基本的，但对于新的维护者则是强 烈推荐使用的。它们使得整个过程的开始变得很容易且使后续的过程容 易控制。(参考 dh_make(1)、debhelper(1)和 /usr/share/doc/debhelper/README)
- devscripts - 这个软件包中包括了一些非常好的且很有用的脚本程 序，它们对于维护者是很有用的，但它们对创建软件包也不是必需的。(参 考 /usr/share/doc/devscripts/README.gz)
- fakeroot - 这个工具使你可以模拟变成 root 用户，这在创建软件包的 过程的一些部分是必要的。(参考 fakeroot(1))
- gnupg - 一个可以让你对你制作的软件包进行数字签名的工 具。如果你希望把它发布给其他人，这个步骤是非常重要的，并且当你所做的工作被加入到 Debian 发行版中时就必需进行这一步。(参考 gpg(1))
- g77 - GNU Fortran 77 语言编译器，如果你的程序是用 Fortran 语言编 写的就需要它了。(参考 g77(1))
- gpc - GNU Pascal 语言编译器，如果你的程序是用 Pascal 语言写的， 就需要它了。这里值得一提的是软件包 fp-compiler，自由 Pascal 编 译器(the Free Pascal Compiler)，它也是一个完成这一任务的好选择。(参 考 gpc(1)和 ppc386(1))
- xutils - 一些程序，通常是 为 X11 编写的程序，也使用这些程序通过 一套宏来生成 Makefile 文件。(参考 imake(1)和 xmkmf(1))
- lintian - 这是一个 Debian 软件包检查器，它可以在你创建软件包后 为你找出一些常见的错误，并解释这些错误。(参考 lintian(1)和 /share/doc/lintian/lintian.html/index.html)

- pbuilder – 这个软件包中包含了用于创建和维护 chroot 环境的程序。在此 chroot 环境中构建 Debian 可以检查构建软件包的依赖关系的正确性并防止 FTBFS 错误发生。(参考 pbuilder(8) 和 pdebuild(1))

下面列出的这些文档都非常重要，你在阅读本文档时也应当阅读它们：

- debian-policy – 政策(Policy)文件中包含了对很多内容的解释，比如 Debian 档案的结构及内容、几个关于操作系统设计的问题、文件系统层次标准(讲述了每个文件和目录应该存在的地方)等等。对你来说，最重要的是它描述了在加入每个软件包到发行版中时，它们必须满足的要求。(参考 </usr/share/doc/debian-policy/policy.html/index.html>)
- developers-reference – 包含了全部的参考资料，其内容不只是打包软件的技术细节，比如文档的结果、如何改名、孤儿、选择软件包、如何做 NMUs、如何管理 bugs、最佳打包实践以及在什么时间将软件包上传到什么位置等等。(参考 </usr/share/doc/developers-reference/index.en.html>)

上面的简短描述只是对每一个软件包进行了一下简单的介绍。在继续后面的工作前，请完整的阅读每一个程序的文档，至少要了解基本的用法。现在看来也许是很繁重的任务，不过以后你会非常高兴的去阅读它们的。

注意：debmake 软件包中包含了一些和 dh-make 作用相似的程序，但它的详细用法并没有包含在这份文档中，因为它已经不再推荐使用。要得到更多的信息，请参考 [the Debmake manual](#)。

1.2 其它信息

你可以制作的软件包有两种，源文件版本和可执行版本。源文件版本的软件包包含了可以被编译成程序的源代码。可执行版本的软件包只包含编译好的文件。不要把程序源文件和程序的源文件版本软件包混在一起！如果你需要更详细的关于这些词汇的资料，请参考阅读其它的手册。

在 Debian 中，“维护者(maintainer)”一词指的是制作软件包的人，“上游作者(upstream author)”指的是编写程序的人，而“上游维护者(upstream maintainer)”是指在 Debian 项目之外维护着程序的人。通常情况下作者和上游维护者是同一个人——有时维护者甚至也是同一个人。如果你编写了一个程序并且希望它被包含到 Debian 中，那么你可以提交你的程序从而成为一个维护者。

在你创建了你的软件包(或则正在做这件事情)，若你希望它能够被加入到下一个发行版中(如果你的程序非常有用，为什么不呢？)，那么你必须成为一个正式的 Debian 维护者。这一过程在开发人员参考(Developer's Reference)中解释了。请阅读它。

Debian 新维护人员手册

第 2 章 - 第一步

2.1 选择你的程序

你大概已经选好了你要制作的软件包。首先要做的事情是检查它是否已经在发行版中了。如果你使用的是“稳定”发行版，那么你最好到[软件包查询页面](#)查一下。如果你使用的是当前的“不稳定”发行版，可以用下面的命令来检查：

```
dpkg -s program
dpkg -l '*program*'
```

如果软件包已经存在了，那么好，安装它！:-) 如果他碰巧是个孤儿——如果它的维护者成为了“Debian QA Group”的成员，你就应该可以重新维护它。查询[孤儿软件包列表](#)和[打算收养的软件包列表](#)可以确认软件包是否真的需要领养。

如果你获准收养一个软件包，那就获取它们的源代码(用 `apt-get source packagename` 一类的命令)并检查它们。很不幸，这份文档中并不包含关于收养软件包的详细信息。值得庆幸的是，在收养软件包时，你不用花费很多时间在找出如何让其工作，因为已经有人为你做了初始的设置工作。尽管如此，也请继续读下去，下面的很多建议也会对你所处的情况有用。

如果软件包是新的，并且你已经决定让它出现在 Debian 中，请按照下面的步骤来做：

- 到[正在制作中的软件包列表](#)检查是否没有其他人正在为打包同一个软件而工作。如果已经有人正在做了，并且你觉得它对你很重要，就请和他们取得联系。否则——找另一个没人维护的有趣程序吧。
- 每一个软件都必须有授权，如果有可能最好是象 [Debian 自由软件指导方针](#)中说的那样属于自由软件。如果它并不遵守这些规则但仍然可以以任意形式发布，它也可以被加入到“contrib”或者“non-free”部分中。如果你不确定它究竟应该被放到哪里，可以把它的授权文字发到 debian-legal@lists.debian.org 问一下该怎么做。
- 程序的确不应当以 `setuid root` 的方式运行，或者最好它应该不需要 `setuid` 或 `setgid` 成为其它任何东西。
- 程序不能是一个守护程序，且它不应该放到 `*/sbin` 目录中去，也不该以 `root` 身份打开一个端口。
- 程序最终应当是二进制可执行的形式，库处理起来要困难一些。
- 它应当有很好的文档，最好连源代码也是容易理解的(比如不混乱)。
- 你应该与程序的作者取得联系问一下他是否同意程序被打包。能够向作者咨询关于程序的任何问题是非常重要的，不要试着去打包一个没有人维护的软件。

- 最后的但并不是不重要的，你必须知道它确实可以工作并且已经试着使用了一段时间。

当然，这些问题都是只是为了安全，并试着让你不至于在比如 `setuid` 的守护进程 等问题上犯错误而激怒了用户。当你有了在打包软件方面的更多经验时，你就可以处理那种软件包了，但即便是富有经验的开发人员在他们疑惑时也会发邮件 到 `debian-mentors` 邮件列表咨询。那里的人们会很乐意提供帮助的。

要获得关于这些内容的更多帮助，请参考开发者参考手册。

2.2 获得程序，并且试用它

第一件要做的事情就是找到并下载原始的软件包。我假定你已经从作者的主页上 找到它的源文件了。免费的 Unix 程序的源文件通常是以 `tar/gzip` 格式提供的，它 的文件扩展名是 `.tar.gz`，并且通常还包含了以 `program-version` 形式命名的子 目录，里面放着全部的源文件。如果你的程序源文件是以一些其它的形式提供 的(比如，文件名是以 `“.Z”` 或 `“.zip”` 结尾的)，那么就用适当的工具把它解包，或 者如果你不清楚应当如何正确把它解包，就在 `debian-mentors` 邮件列表上问一 下。(提示：可以用命令 `file archive.extension`)

作为一个例子，我将会使用程序 `“gentoo”`，它是一个基于 X GTK+ 的文件管理器。 需要注意的是，这个程序已经被打包好了，并且从写这篇文档之初到现在它已经 发生了很大的变化。

在你的 `home` 目录中创建一个名为 `“debian”` 或者 `“deb”` 或者任何你喜欢的名字的目 录(比如在这个例子中 `~/gentoo/` 就可以了)。把下载的文件放到这个目录中，然后 将其解包(用命令 `“tar xzf gentoo-0.9.12.tar.gz”`)。确认在这个过程中没有发 生错误，即便是一点 `“不恰当”` 的也不行，因为当在别人的系统上解包这些文件的 时候，如果它们的工具并不忽略这些反常的现象，那就会有有了。

现在又有了一个新的子目录，名叫 `“gentoo-0.9.12”`。进入这个目录并且彻 底的读完其中的文档。通常情况下在目录里面会有名 叫 `README*`、`INSTALL*`、`*.lsm` 或者 `*.html` 的文 件。你必需找到如何正确编译并安 装程序的指导。(最有可能的是它们会假设你希望把程序 安装到 `/usr/local/bin` 目 录中；你不需要这样做，但在后面的在一个子目录中安装，第 3.1 节中需要做很多事情。)

安装的过程对于不同的软件是不同的，但很多现代的程序都带有一个 `“configure”` 脚本文 件，这个文件配制你系统上的源文件，并确认你的系统已经可以编译它了。 在通过 `“./configure”` 命令配制之后，通常可以通过 `“make”` 来编译程序。有一些程 序还会支持通过 `“make check”` 命令来进行自检。把程序安装到目标目录中的命令 通常是 `“make install”`。

现在可以试着编译并运行你的程序了，从而确定它可以很好的工作并且在它安装 或工作时不会破坏其它程序的运行。

另外，通常你还可以通过 `“make clean”` (或者更好的 `“make distclean”`) 命令来清 理 `build` 目录。有时还会有一个 `“make uninstall”` 命令来删除所有已经安装的文件。

2.3 软件包名称和版本

在开始打包时，源程序目录应当是绝对干净(原始)的，或者直接从刚刚解包的源代码目录开始。

为了让软件包能够正确地制作，你必须把程序原有的名字改成小写(如果它不是的话)，并且你应该把源代码目录的名字改成<packagename>-<version>的形式。

如果程序的名字是由多于一个的英文单词组成的，那就把它改成一个单词，或者缩写的形式。例如，程序“John’s little editor for X”软件包可以改成 johnledx 或者 jle4x，或者随便什么你认为合适的，只要它符合一些很合理的限制，比如在 20 个字符以内。

另外要做的一件事情就是检查一下被包装在软件包里的程序的精确版本号(它将被包含在软件包的版本号中)。如果包装的软件并不是以 X.Y.Z 的方式来命名它的版本的，而是用比如日期一类的方式，那么就用那个日期来做版本号好了，只要在前面加上“0.0.”就可以了(直到上游的人决定发布一个好的版本比如 1.0 等时候)。因此，如果发行版或者 snapshot 的日期是 1998 年 12 月 19 日，你就可以使用 0.0.19981219 作为版本号。

有一些程序根本就没有数字的版本，在这种情况下，你就需要和上游维护者取得联系，看看他们是不是使用了什么别的版本跟踪方法。

2.4 首次“Debian 化”

确定你在程序的原代码目录中，然后执行这个命令：

```
dh_make -e your.maintainer@address -f ../gentoo-0.9.12.tar.gz
```

当然，要用你的 E-mail 地址换掉字符串“your.maintainer@address”，并用你的源代码文档的名字替换掉上面的文件名。你的这个 E-mail 地址将会被包含在 changlog 项目和其它的文件中。参考 dh_make(1) 获得详细的信息。

在执行这个命令后，你将会看到一些信息，它会问你你需要创建那种类型的软件包。

Gentoo 是一个单二进制软件包——它只创建一个二进制形式的软件包，也就是说只有一个 deb 文件——所以我们按“s”键选择第一个选项，检查屏幕上的信息，然后按<enter>键确认。

在此次运行 dh_make 之后，上游的软件包将会被打包为 gentoo_0.9.12.orig.tar.gz 并放在父目录中，以便用 diff.gz 创建非 Debian 固有的源代码包。请注意文件名中的两个关键点：

- 包名称和版本是以“_”分割的。
- 在之前“tar.gz”有“orig.”。

再说一下，作为一个新的维护者，我们不鼓励你创建复杂的软件包，譬如：

- 生成多个二进制包的，
- 库软件包，
- 源文件格式不是 tar.gz 也不是 tar.bz2，或者
- 在源码包中包含的是不可发布的内容。

要说清这些问题并不是很难，但你确实需要了解更多一些的知识，因此在这里讲述关于它们的全部内容。

请注意你只能运行一次 `dh_make` 程序，如果你再次在同一个已经“Debian 化”的目录中运行它，它将不能正常运行。这也意味着当你发布你的软件的下一个版本时，你需要使用一些不同的方法。以后你会在 更新软件包，第 9 章 一部分中读到更多关于这个问题的内容。

Debian 新维护人员手册

第 3 章 - 修改源代码

通常情况下，程序会把它自己安装到 `/usr/local` 子目录中。但是 Debian 的软件包绝对不能使用那个目录，因为它被保留给系统管理员(或者用户)使用。这就是说你必需要仔细看一下你的程序的构造系统(build system)，通常从 Makefile 开始。它是 `make(1)` 将会使用的用于自动构造程序的脚本。要了解更多关于 Makefiles 的内容，请参考“rules”文件，第 4.4 节。

注意如果你的程序使用了 GNU 的 `automake(1)` 和/或 `autoconf(1)`，也就意味着源代码是在 `Makefile.am` 和/或 `Makefile.in` 文件中，相应的，你需要修改这些文件。这是因为在每一次 `automake` 的调用中，`Makefile.in` 等文件中的信息将会通过 `Makefile.am` 等文件来重新产生，并且每次调用 `./configure` 时，类似的操作会执行在 `Makefile` 等文件上，它们会被根据 `Makefile.in` 文件重新产生。修改 `Makefile.am` 文件需要一些关于 `automake` 的知识，你可以阅读 `automake` 的 info 项目，然而修改 `Makefile.in` 文件和修改 `Makfile` 文件是差不多的，不过要注意一下变量，例如，任何被“@”包围的字符串如 `@CFLAGS@` 或 `@LN_S@` 将会在每次 `./configure` 调用时用实际的值替换掉。

还需要注意的是，在这里我们没有地方讨论所有的修改上游源代码的细节，这里我们只有一些人们经常会遇到的问题。

3.1 在一个子目录中安装

大多数的程序都能够以某种方式把自己安装到系统现有的目录结构上，所以它们的可执行文件已经存在于你的 `$PATH` 中，并且你也可以在通常的位置找到它们的文档和手册。然而，如果你这样做，这些程序将会和你系统上的其它程序混合在一起。这样的话，对于打包工具而言要想把它们同不属于这个软件包的程序区分开就很困难了。

因此，你还必须要做一些其它的事情：把程序安装到一个临时的子目录中，从这里打包工具可以创建一个可以工作的 .deb 软件包。在这个目录中的所有内容都将会被安装到用户的系统中，当他们安装你的软件包时，唯一的不同是 `dpkg` 将会把这些文件安装到文件系统的根目录上。

这个临时文件目录通常创建在源代码目录的 `debian/` 子目录中。通常情况下，它的名字是 `debian/packagename`。

有一件事情请记住，尽管你要把程序安装到 `debian/packagename` 目录中，但在安装 .deb 文件的时候，它仍应当可以正常地安装到根目录中。所以你绝对不能让软件包的构造系统把类似于 `/home/me/deb/gentoo-0.9.12/usr/share/gentoo` 的字符串写入到软件包中。

对于使用 GNU `autoconf` 的程序而言，这个工作是非常简单的。大多数这样的程序的 `makefile` 脚本缺省状态下就允许程序被安装到任何的子目录中，并以(例如) `/usr` 作为它的典型前缀。当检测到你的程序使用了 `autoconf` 时，`dh_make` 发现将会自动设定命令完成这

一任务，因此你可以跳过下面的部分。但对于其它的程序，你极 有可能不得不检查并修改 Makefile 文件。

这里是 gentoo 的 Makefile 文件的相应部分：

```
# Where to put binary on 'make install'?
BIN      = /usr/local/bin

# Where to put icons on 'make install'?
ICONS    = /usr/local/share/gentoo
```

我们发现这个文件被设定成为安装到/usr/local 目录下。将这些路 径改为：

```
# Where to put binary on 'make install'?
BIN      = $(DESTDIR)/usr/bin

# Where to put icons on 'make install'?
ICONS    = $(DESTDIR)/usr/share/gentoo
```

但为什么要在这个目录中而不是其它的呢？这是因为 Debian 绝不会把文件安装 到 /usr/local 目录中——那个目录是留给系统管理员用的。这些文件 在 Debian 系统上都会被安装到/usr 目录下。

在文件系统层次标准中描述了更多的关于二进制、图标、文档等文件放置位置的 信息(请参考/usr/share/doc/debian-policy/fhs/)。我建议你浏览一下其中可能 与你的软件包有关的部分。

因此，我们应该把二进制文件安装在/usr/bin 目录中而不是/usr/local/bin 目录 中，把手册安装在/usr/share/man/man1 目录中而不是/usr/local/man/man1 目录 中。也许你注意到在 gentoo 的 makefile 中并未涉及到手册文件，但 Debian 政策要 求每个程序都要有一篇手册，因此我们稍后会制作一份并把它安装 到/usr/share/man/man1 中。

有一些程序并不像这样使用 makefile 的变量来定义其路径。这就意味着你不得不 去修改一些 C 源程序来使其能够在正确的位置找到文件。但到哪里去找又改找些什 么呢？你可以使用这样的命令：

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

grep 会递归地搜索整个源代码目录树，并在找到相应的字符串时告诉你它所在文 件的名字和在文件中所处行的行号。

修改那些文件，并用 usr/*替换掉原来的/usr/local/*以及所有相关的内容。注意 不要为了修改这些地方而把代码的其它部分搞乱。 :-)

之后，你应该找到 install 目标(查找以“install:”开始的行)并修改所有对于目录 的引用，使其和在 Makefile 的开始部分定义的一致。最初的时候，gentoo 的 install 目标是下面的样子：

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

在我们修改以后它变成了这个样子：

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

你一定已经注意到在其它命令之前有一个 `install -d` 命令。原来的 `makefile` 脚本没有它是因为通常情况下在运行“`make install`”时，`/usr/local/bin` 和其它的目录都已经存在于文件系统上了。然而，我们是要把文件安装到我们的空的(或者是根本不存在的)目录中，因此我们不得不首先创建每一个目录。

在 `rule` 文件的结尾，我们还可以加入其它的内容，比如安装上游作者忽略掉的附加文档，如下所示：

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

细心的读者应该已经注意到我把“`install:`”一行中的“`gentoo`”改成了“`gentoo-target`”。这被称为无关 bug 修复 :-)

当你做了一些并不特定地与 Debian 软件包相关的修改时，请一定要把它们发送给上游的维护者，这样这些修改就可以被包含在软件的下一个版本中，这样会对其他人非常有用。还要记住不要使你的修改只是针对 Debian 或者 Linux(甚至是 Unix!)，在发送它们之前——让它们具有可移植性。这将会使你的修改更容易被接受。

注意，你不需要把 `debian/*` 文件也发送给上游的人。

3.2 不一样的库名称

有一个非常普遍的问题：在不同的平台上链接库通常是不一样的。例如，`Makefile` 中包含了对一个库的引用，但 Debian 系统上并没有这个库。在这种情况下，我们需要把它修改成为一个在 Debian 中确实存在并且完成相同功能的库。

因此，如果在你的程序的 `Makefile`(或者 `Makefile.in`)中有类似于下面的一行(并且使你的程序无法编译了)：

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

可以把它改成这样，通常情况下它都能工作：

```
LIBS = -lncurses -lsomething -lsomethingelse
```

(作者已经注意到这并不是最好的例子，因为我们现在使用的 `libncurses` 软件包在发布的时候包含了一个 `libcurses.so` 的符号链接，但他没能想到更好的。欢迎你提些建议 :-)

Debian 新维护人员手册

第 4 章 - debian/目录中必需的内容

在程序的源代码目录中有一个名叫“debian”的新子目录。在这个子目录中有很多 我们需要文件，通过修改这些文件可以定制软件包的行为。其中最为重要的是“control”、“changelog”、“copyright”和“rules”，它们对于所有的软件包都是必需的。

4.1 “control” 文件

在这个文件中包含了很多变量，dpkg、dselect 和其它软件包管理工具通过它们来管理软件包。

dh_make 为我们创建的 control 文件如下所示：

```
1 Source: gentoo
2 Section: unknown
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (> 3.0.0)
6 Standards-Version: 3.5.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Description: <insert up to 60 chars description>
12 <insert long description, indented with spaces>
```

(我为它增加了行号。)

1-6 行是源程序形式软件包的控制信息。

第 1 行是源程序包的名字。

第 2 行是源程序包在发行版中所属部分。

你也许已经注意到了，Debian 被分成许多不同的部分：包括 main(自由软件)、non-free(非自由软件)和 contrib(基于非自由软件的自由软件)。在这些部分中，还有子分类，这些子分类以简短的方式说明了软件包的用途。比如“admin”是只有系统管理员才能使用的程序，“base”是基本的工具，“devel”是 给程序员使用的工具，“doc”是文档，“libs”是函数库，“mail”是邮件阅读工具 和守护程序，“net”是网络应用程序和守护程序，“x11”是不属于以上各个部分的 X11 程序，还有更多这里就不一一叙述了。

我们把它改成 x11。(“main/”是缺省的前缀，因此我们可以忽略它。)

第 3 行描述了在用户安装系统时此软件包的重要程度。参考政策手册中相应的指导 可以知道应当把它设置成什么。“optional”的优先级对于新软件包通常是合适的。

所属部分和优先级对 dselect 等前端软件是有用的，它们在排序和选择缺省的软件包时会用到这些变量。当你把软件包上传到 Debian 以后，这两个字段的值可以被文档维护员修改，在这种情况下，你会收到一封通知电子邮件。

因为这是一个普通级别的软件包，并且它不和其它任何软件包冲突，我们让它保留原来的“optional”。

第四行是维护者的姓名和电子邮件地址。一定要保证这个字段包含有一个合法的电子邮件“To:”字段，因为在你把软件包上传以后，bug 跟踪系统将会使用这个地址来传递通知 bug 信息的电子邮件给你。不要使用逗号、“&”符号和括号。

第五行包括了要构建你的软件包需要的软件包列表。包括 gcc 和 make 在内的一些软件包是不需要列出来的，关于此内容的详细信息可以参考软件包 build-essential。如果在构造你的软件包时需要一些非标准的编译器或者是其它的工具，你就需要把它们加到“Build-Depends”这一行上。多个项目之间用逗号隔开；要了解关于这个项目的语法的更多信息，请阅读关于二进制文件依赖性的解释。

你还可以在这里加入 Build-Depends-Indep、Build-Conflicts 和其它一些字段。Debian 的软件包自动构造系统将会使用这些数据为其它的计算机平台创建二进制软件包。可以参考政策手册中关于 build-dependencies 的部分和程序员参考手册，里面包含有关于其它平台(体系结构)以及如何把软件移植到上面的更多信息。

要想知道你的软件在编译的时候需要用到哪一个软件包，可以通过下面的方法：

```
strace -f -o /tmp/log ./configure
# or make instead of ./configure, if the package doesn't use autoconf
for x in `dpkg -S $(grep open /tmp/log | \
    perl -pe 's!.* open\(("[^"]*)).*!$1!' | \
    grep "^/" | sort | uniq | \
    grep -v "^(/tmp|/dev|/proc)" ) 2>/dev/null | \
    cut -f1 -d":" | sort | uniq` ; \
do \
    echo -n "$x (>= " `dpkg -s $x | grep ^Version | cut -f2 -d":"` " ), "; \
done
```

要准确地找到构建/usr/bin/foo 所需要的软件包，执行：

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

而要列出每一个库，如 libfoo.so.6，执行：

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

现在你已经安装了“Build-deps”一项列出的每一个-dev 软件包。如果你使用 ldd 来完成这个任务，它会把并非直接使用的库也报告出来，导致过多的构造依赖。

Gentoo 还需要软件包 xlibs-dev、libgtk1.2-dev 和 libglib1.2-dev 才能够构造，因此我们把它加到 debhelper 的后面。

第 6 行是这个软件包遵循的 Debian 政策标准的版本，也就是你在制作这个软件包时读的政策手册的那个版本。

第 8 行是二进制软件包的名字。它通常和源文件软件包有一样的名字，但实际上并不一定得是这样。

第 9 行描述了可以使用这个二进制软件包的 CPU 类型。我们让它保持原来的“any”值，因为 dpkg-gencontrol(1) 会在为任何一种机器编译这个软件包时自动为这个字段填写合适的值。

如果你的软件包是体系结构无关的(比如一个 shell 或 Perl 脚本，或者是文档)，就把这个字段修改成“all”，另外在稍后还要仔细看一下“rules”文件，第 4.4 节中关于用“binary-indep”规则来代替“binary-arch”规则的内容。

第 10 行显示了 Debian 软件包系统的一个强大功能。软件包可以通过多种不同的方式和其它的软件包相关连。除了 Depends: 之外，还有其它的关联字段，它们是 Recommends:、Suggests:、Pre-Depends:、Conflicts:、Provides: 和 Replaces:。

在管理这些软件包的关联时，所有的软件包管理工具通常的行为都是一样的；如果不是这样的话，它将会给出解释。(参考 dpkg(8)、dselect(8)、apt(8) 和 aptitude(1) 等。)

下面给出每一种软件包依存性的含义：

- Depends:

除非把此软件包所倚赖的所有其它软件包安装好，否则软件包将不会被安装。你可以在除非提供了一个其它的软件包，否则你的软件包绝对不能运行(或者会导致严重的 breakage)时使用这种关联。

- Recommends:

dselect 或者是 aptitude 等前端工具在安装你的软件包的时候，它们会问你是否将与该软件包以推荐的方式相关联的软件包一起安装；dselect 甚至会坚持这样做。而 dpkg 和 apt-get 会忽略这个字段。这个字段可以被用于那些并不是严格需要却经常会和你的软件包一起使用的软件包。

- Suggests:

在一个用户安装你的软件时，所有的前端工具都会询问他是否要安装被建议的软件包。dpkg 和 apt-get 不会这样做。这个字段可以被用于那些可以和你的程序非常好地一起工作但并不是必需的软件包。

- Pre-Depends:

它的要求比 Depends: 更强。除非它需要的软件包已经安装并且正确配制好，它才会被安装。使用这个标签是非常 sparingly 的，要使用它一定要先在 debian-devel 邮件列表上讨论完才可以。读一遍这句话：绝对不要使用它。 :-)

- Conflicts:

除非与这个软件包冲突的软件包都已经被删除了，否则它不会被安装。如果一个软件包存在时你的程序不能被运行或者会出现严重的错误，就使用这个标签。

- Provides:

当多个软件包提供同一个功能时，可以定义一些虚拟的软件包名称。你可以在 /usr/share/doc/debian-policy/virtual-package-names-list.txt.gz 文件中 找到一个完整的虚拟软件包列表。当你的软件包提供一个已经存在的虚拟软件包 所需要的功能时，可以使用这个字段。

- Replaces:

当你的软件包会替换一些其它软件包的文件或者是整个软件包(与 Conflicts: 联用)时, 可以使用这个字段。这里提到的软件包中的文件将会被你的软件包中的文件覆盖。

所有这些字段使用统一的语法格式: 用逗号分隔的一系列软件包名称。这里的软件包名称可以用竖线符号“|”分开的一系列可相互替换(alternative)的软件包名称。

对于每一个软件包的特定版本的要求也可以在这个字段中限制。只要在软件包的名称后写上括号并在括号中写明版本列表并在每一个版本号前注明当前软件包和它的关系就可以了。这里提到的关系可以是: <<、<=、=、>=和>>, 它们分别表示先于、先于或等于、等于、晚于和晚于或等于。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)
```

最后一个你需要知道的功能是\${shlibs:Depends}。在你的软件包被创建并且安装到临时目录中以后, dh_shlibdeps(1)将会扫描其中的二进制文件和库文件, 检测它们对共享库的依赖性, 以及这些共享库所在的软件包, 如 libc6、xlib6g 等。它将会把结果列表传递给 dh_gencontrol(1), 后者将会把这些信息填写到正确的位置上, 你就不用为它操心了。

说了这么多, 我们现在可以继续了, 让 Depends: 这一行保持原状, 并在它后面插入一行, 在这一行中写上 Suggests: file, 因为 gentoo 可以使用这个程序/软件包提供的一些功能。

第 11 行是一个简短的描述。大多数人的屏幕都是 80 列宽的, 所以描述文字不能超过大概 60 个英文字符长。我把它改成“fully GUI configurable X file manager using GTK+”。

第 12 行是一个比较长的描述。在这里可以写关于这个软件包的详细情况的一段话。每行的第 1 列应该是空白的。两行之间不能有空白行, 如果真的想留一个空白行, 可以通过写一个单独的小数点符号实现。还有, 在长描述之后, 不能有超过一行的空白。

最后, 我们给出一个修改好的 control 文件:

```
1 Source: gentoo
2 Section: x11
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (>> 3.0.0), xlibs-dev, libgtk1.2-dev, libglib1.2-dev
6 Standards-Version: 3.5.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Suggests: file
12 Description: fully GUI configurable X file manager using GTK+
13 gentoo is a file manager for Linux written from scratch in pure C. It
14 uses the GTK+ toolkit for all of its interface needs. gentoo provides
15 100% GUI configurability; no need to edit config files by hand and re-
16 start the program. gentoo supports identifying the type of various
17 files (using extension, regular expressions, or the 'file' command),
```



```
18 and can display files of different types with different colors and icons.
19 .
20 gentoo borrows some of its look and feel from the classic Amiga file
21 manager "Directory OPUS" (written by Jonathan Potter).
```

(我为它增加了行号。)

4.2 “copyright” 文件

这个文件中包含着关于软件包的来自于上游的资源、版权和授权信息。它的格式 在政策中并未规定，关于它的内容是(12.6 “Copyright information”)。

dh_make 将会创建一个缺省的，其内容如下：

```
1 This package was debianized by Josip Rodin <joy-mg@debian.org> on
2 Wed, 11 Nov 1998 21:02:14 +0100.
3
4 It was downloaded from <fill in ftp site>
5
6 Upstream Author(s): <put author(s) name and email here>
7
8 Copyright:
9
10 <Must follow here>
```

(我为它增加了行号。)

在这个文件中需要增加的重要信息是你获得软件包的位置以及它原有的版权提示 和授权协议。如果它的授权协议不是通用的自由软件授权协议 如 GNU 的 GPL 或 LGPL、BSD 或者 Artistic 协议，你就必需把它的协议包含在这个文件中。而当它使用上述自由软件授权协议时，你可以直接引用 /usr/share/common-licenses/ 目录中的相应文件，它们已经存在于 Debian 系统 中了。

简而言之，下面是 gentoo 的 copyright 文件：

```
1 This package was debianized by Josip Rodin <joy-mg@debian.org> on
2 Wed, 11 Nov 1998 21:02:14 +0100.
3
4 It was downloaded from: ftp://ftp.obsession.se/gentoo/
5
6 Upstream author: Emil Brink <emil@obsession.se>
7
8 This software is copyright (c) 1998-99 by Emil Brink, Obsession
9 Development.
10
11 You are free to distribute this software under the terms of
12 the GNU General Public License.
13 On Debian systems, the complete text of the GNU General Public
14 License can be found in the file `/usr/share/common-licenses/GPL'.
```

(我为它增加了行号。)

4.3 “changelog” 文件

这是一个必需的文件，它的格式已经在政策文件的 4.4 节 “debian/changelog” 中说明了。dpkg 和其它需要获取你的软件包的版本号、修订号、发行版和紧急度的程序会需要使用这个格式。

对你而言，它也是非常重要的，因为它可以让你写下所有你所做的所有变更。这可以帮助下载了你的软件包的人们了解软件包中是否有它们应该知道的问题。在二进制版本的软件包中，它会被保存在 “/usr/share/doc/gentoo/changelog.Debian.gz” 文件中。

dh_make 创建了一个缺省的，如下所示：

```
1 gentoo (0.9.12-1) unstable; urgency=low
2
3  * Initial Release.
4
5  -- Josip Rodin <joy-mg@debian.org> Wed, 11 Nov 1998 21:02:14 +0100
6
```

(我为它增加了行号。)

第 1 行是软件包的名称、版本、发行版和紧急度。这里的名称必需和源代码包的名称相同，发行版应当是 “unstable” (或者 “experimental”)，urgency 应当改成任何比 “low” 高一些级别的内容。:-)

第 3 到 5 行是一个很长的项目，在这里你可以写下你对这个版本的软件说做的修改(不包括上游修改——有专门的由作者创建的文件来记录它们，稍后你将会把它安装到 /usr/share/doc/gentoo/changelog.gz)。新的内容必需插入在最上方的星号 (“*”) 前。你可以用 dch(1) 来做或者用一个文本编辑器手工修改。

最后，你的文件应该是下面的样子：

```
1 gentoo (0.9.12-1) unstable; urgency=low
2
3  * Initial Release.
4  * This is my first Debian package.
5  * Adjusted the Makefile to fix $DESTDIR problems.
6
7  -- Josip Rodin <joy-mg@debian.org> Wed, 11 Nov 1998 21:02:14 +0100
8
```

(我为它增加了行号。)

在后面的更新更新软件包，第 9 章中你可以找到更多关于更新 changelog 的内容。

4.4 “rules” 文件

现在我们需要来看看 dpkg-buildpackage(1) 用来创建软件包的精确规则了。这个实际上是另一个 Makefile 脚本，但同上游源代码中的那个不同。与 debian/ 目录中的其它文件不同的是这个文件有可执行标记。

就象其它的 Makefile 一样，每个“rules”文件都有一些用来指导如何处理源代码的规则。每个规则都由目标、文件名或者是需要执行的操作的名称(如“build:”、“install:”)组成。当你要执行一个规则时可以在命令行参数上加入参数(比如“./debian/rules build”或者“make -f rules install”)。在目标名称之后，你可以写这个规则对程序或文件的依赖性。之后，可以有任意数目的命令，这些命令要用<tab>符号缩进。新的规则以位于第一列上目标声明开始。空白行和以“#”(井字号)开始的行将会被作为注释忽略掉。

现在你可能已经听糊涂了，但只要看一下 dh_make 为我们创建的缺省的“rules”文件你就能明白了。另外你也应该读一下 info 中的“make”项目来获得更多的信息。

有一个关于 dh_make 所创建的 rules 文件的重要问题是你应该知道的：它只是一个建议版本。对于一些简单的软件包它可以工作，但对于稍为复杂一些的，应当敢于增加或者删减其内容使其符合你的需求。唯一你不能修改的就是规则的名称，因为政策手册中提到的所有工具都将使用它们。

这里是 dh_make 为我们产生的缺省的 debian/rules 文件：

```
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-make.
8  # Uncomment this to turn on verbose mode.
9  #export DH_VERBOSE=1
10 configure: configure-stamp
11 configure-stamp:
12     dh_testdir
13     # Add here commands to configure the package.
14     touch configure-stamp
15 build: build-stamp
16 build-stamp: configure-stamp
17     dh_testdir
18     # Add here commands to compile the package.
19     $(MAKE)
20     #docbook-to-man debian/testpack.sgml > testpack.1
21     touch $@
22 clean:
23     dh_testdir
24     dh_testroot
25     rm -f build-stamp configure-stamp
26     # Add here commands to clean up after the build process.
27     $(MAKE) clean
28     dh_clean
29 install: build
30     dh_testdir
31     dh_testroot
32     dh_clean -k
33     dh_installdirs
34     # Add here commands to install the package into debian/testpack.
35     $(MAKE) DESTDIR=$(CURDIR)/debian/testpack install
36 # Build architecture-independent files here.
```

```

37 binary-indep: build install
38 # We have nothing to do by default.
39 # Build architecture-dependent files here.
40 binary-arch: build install
41     dh_testdir
42     dh_testroot
43     dh_installchangelogs
44     dh_installdocs
45     dh_installexamples
46 #     dh_install
47 #     dh_installmenu
48 #     dh_installdebconf
49 #     dh_installogrotate
50 #     dh_installemacsen
51 #     dh_installpam
52 #     dh_installmime
53 #     dh_python
54 #     dh_installinit
55 #     dh_installcron
56 #     dh_installinfo
57     dh_installman
58     dh_link
59     dh_strip
60     dh_compress
61     dh_fixperms
62 #     dh_perl
63 #     dh_makeshlibs
64     dh_installdeb
65     dh_shlibdeps
66     dh_gencontrol
67     dh_md5sums
68     dh_builddeb
69 binary: binary-indep binary-arch
70 .PHONY: build clean binary-indep binary-arch binary install configure

```

(我为它增加了行号。在实际的 `debian/rules` 文件中，行首的空白是制表符 TAB。)

对于第 1 行你一定很熟悉，因为它和 shell 及 Perl 脚本很象。它告诉操作系统这个文件应当交给 `/usr/bin/make` 来处理。

第 6 到 9 行上提到的 `DH_*` 变量应当被简短明确的说明。要想知道关于 `DH_COMPAT` 的更多信息，可以阅读 `debhelper(1)` 手册中关于“`Debhelper compatibility levels`”一节。

第 11 到 16 行是一个支持 `DEB_BUILD_OPTIONS` 的骨架，它的描述可以在政策文档的 第 10.1 节“`Binaries`”中找到。简单的说，它们控制着在构造二进制文件的时候是否要加入调试符号，是否要在安装的时候进行裁减。再重复一下，这只是一个骨架，一个你应当做这件事的提示。你需要找出上游的构建系统是如何处理调试符号和 `install-strip` 的，然后自己实现它们。

一般情况下，你可以通过 `CFLAGS` 变量来让 `gcc` 在编译的时候使用“`-g`”选项——如果这是你的软件包的情况，你可以把 `CFLAGS="$(CFLAGS)"` 附加到 `$(MAKE)` 调用的后面来 propagate 这个变量(看下面)。还有另外一种方法，如果你的软件包使用了 `autoconf` 脚本，你可以通过给 `./configure` 调用加上前缀来把它传递给构建规则。

关于裁减的问题，一般情况下程序自己的安装配制都不会进行裁减，而且通常也不会包含一个选项让你来做这件事情。幸运的是，你有 `dh_strip(1)`，而且当你设定了 `DEB_BUILD_OPTIONS=nostrip` 时，他会安静地退出。

第 18 到 26 行描述了“build”（和“build-stamp”）规则，它们运行应用程序自己的 Makefile 来编译它。如果你的软件包使用 GNU 配置工具来构造，请一定要阅读 `/usr/share/doc/autotools-dev/README.Debian.gz`。稍后在 `manpage.1.ex`，`manpage.sgml.ex`，第 5.8 节中我们会讨论 `docbook-to-man` 的例子。

第 28 到 36 行的“clean”规则会清除所有不需要的二进制文件和自动产生的东西，在每次构建软件包的时候都会首先执行。这个规则必须在所有的时候都能正常工作（即便源代码目录已经是清理被清理好的），所以请使用强制选项（比如对于 `rm` 是“-f”），或者通过在命令的名字前加上“-”让 `make` 忽略返回值（失败）。

“install”规则从第 38 行开始，它指导了安装过程。它通常去执行软件自己的 Makefile 中的“install”规则，但会把软件包安装在 `$(CURDIR)/debian/gentoo` 目录中——这就是为什么我们要在 `gentoo` 的 Makefile 中指定 `$(DESTDIR)` 作为安装的跟目录。

就象注释中说明的那样，第 48 行上的“binary-indep”规则是用来构建体系结构无关的软件包的。因为这里我们并没有这样的软件包，所以什么都不用做了。

在 52-79 行上是下一条规则“binary-arch”，在这里我们运行了好几个 `debhelper` 软件包中的小工具，它们将会在你的软件包上执行不同的操作来使其符合政策。

如果你的软件包是“Architecture: all”的，那么你需要在“binary-indep”中包含所有的命令，而让“binary-arch”保持空白。

`debhelper` 程序都是以 `dh_` 开始的，剩下的部分描述了这个工具具体的作用。其实它们的名字都已经说的很清楚了，但这里我们还是给出一些额外的解释：

- `dh_testdir(1)` 检查你是不是在正确的目录中（比如源代码目录的最上层）；
- `dh_testroot(1)` 检查你是否拥有在“binary-arch”、“binary-indep”和“clean”时需要用到的 `root` 权限；
- `dh_installman(1)` 把手册页文件复制到正确的目标目录中你不需要告诉它究竟相对于最高层源代码目录的那个位置是哪里；
- `dh_strip(1)` 从可执行文件和库文件中裁减掉调试信息，使它们更小一些；
- `dh_compress(1)` 用 `gzip(1)` 压缩所有大于 4 kB 的手册页和文档；
- `dh_installdeb(1)` 把与软件包相关的所有文件（例如维护脚本）复制到 `debian/gentoo/DEBIAN` 目录中；
- `dh_shlibdeps(1)` 计算库文件和可执行文件对共享库的倚赖性；
- `dh_gencontrol(1)` 在控制文件插入一个已经格式化（fine-tuned）好的 `debian/gentoo/DEBIAN` 文件；
- `dh_md5sums(1)` 为软件包中的所有文件产生 MD5 校验码。

要了解更多完整的关于这些 `dh_*` 脚本究竟会做什么的信息以及它们其它的选项，请阅读它们相应的手册。还有一些可能是非常有用的 `dh_*` 脚本文件在这里没有提及。如果你需要使用它们，请阅读 `debhelper` 的文档。

在 `binary-arch` 一节中，你必须注释掉或者删除掉你不需要的功能调用。对于 `gentoo`，我把关于 `examples`、`cron`、`init`、`man` 和 `info` 的行注释掉了，因为 `gentoo` 根本不需要它们。而且在第 68 行上，我把 “`ChangeLog`” 换成了 “`FIXES`”，因为那是上游的 `changelog` 文件的真实名字。

最后的两行(和其它这里未曾解释的地方)是需要的，在 `make` 的手册和 `Debian` 政策文件都是可以找到。现在知道它们的作用并不重要。

Debian 新维护人员手册

第 5 章 - debian/中的其它文件

你会看到在 debian/子目录中还有几个已经存在的文件，它们大多数都是以“.ex”结尾的，表明它们只是例子。仔细看一下它们。如果你希望使用其中任何一个功能，你需要做的事情是：

- 看一下相关的文档(提示：Debian 政策手册)，
- 如果需要就修改文件使其符合你的需求，
- 修改文件名如果有“.ex”后缀就去掉它，
- 修改文件名如果有“ex.”前缀就去掉它，
- 如果需要就修改“rules”文件。

在下面给出了一些经常会被用到的文件的解释。

5.1 README.Debian

所有的在原来的软件包和你的 debian 版本的软件包之间的细节及差异需要写在这里。

dh_make 创建了一个缺省的，如下所示：

```
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
```

由于我们不需要在这里写任何东西，我们就把它删掉了。

5.2 conffiles.ex

关于软件，有一件事情是很烦人的，那就是当你花费了很多的时间和精力定制好一个程序，只要一升级，你所有的定制就都会被丢弃了。Debian 解决这个问题的方法是标注出配制文件，这样当你升级一个软件包时，你将会被询问是否要保留你原来的配制文件。

如果希望对让一个软件包可以做到这点，只要把每一个配制文件(通常在/etc)目录下的完整路径逐行加入到一个名叫 conffiles 的文件里面就可以了。Gentoo 有一个配制文件，/etc/gentoorc，我们把他加入到文件 conffiles 中。

如果你自己的程序有一个配制文件并且会自己覆盖它，那么最好就不要把它标记为配制文件了，因为这样 dpkg 就总会询问用户是否要修改它。

如果你的软件包总要求每个用户修改自己的配制文件，最好也不要吧配制文件标记出来了。

你可以用“maintainer scripts”来处理例子配制文件，要了解更多的信息请参考 [postinst.ex](#), [preinst.ex](#), [postrm.ex](#), [prerm.ex](#), 第 5.12 节。

如果你的程序根本没有 conffiles，对你来说从 debian/ 目录中删除 conffiles 就是很安全的了。

5.3 cron.d.ex

如果你的软件包需要计划任务正常运行才能够正常操作，你可以在这个文件中设置它。

注意这里并不包括定期清除日志的任务；关于它，请参考 [dh_installlogrotate\(1\)](#) 和 [logrotate\(8\)](#)。

如果你不需要，那就把它删了吧。

5.4 dirs

这个文件里指出了我们需要的但常规的安装过程(`make install`)并不会自动创建的目录。

缺省情况下，它的内容如下所示：

```
usr/bin
usr/sbin
```

注意最前面是没有斜线的。我们通常把它改成下面的样子：

```
usr/bin
usr/man/man1
```

但这些目录在 Makefile 中已经创建了，所以我们不需要这个文件，并且打算把它删了。

5.5 docs

在这个文件中，我们可以指定一些让 `dh_installdocs` 帮我们安装到临时目录中的文档的文件名。

缺省的情况下，他会包含所有已经存在于源代码目录最高层目录中的名为“BUGS”、“README*”、“TODO”等的文件。

对于 gentoo，我们还加入了一些其它的内容：

```
BUGS
CONFIG-CHANGES
CREDITS
ONEWS
README
```


README.gtkrc
TODO

我们可以删掉这个文件，取而代之的是在 rules 文件中 的 dh_installdocs 命令后列出这里提到的文件的名字，如下所示：

```
dh_installdocs BUGS CONFIG-CHANGES CREDITS ONEWS README \  
                README.gtkrc TODO
```

也许并不像你看到的这样，你自己的软件包可能根本就没有这些文件。在这种情况下，对你来说删掉这个文件是很安全的。但是不要删掉 rules 文件中 的 dh_installdocs，因为它还要被用于安装 copyright 和其它的一些 东西。

5.6 emacs-*.*.ex

如果你的软件包提供了一些可以在安装时进行字节编译的 Emacs 文件，你可以使用 这些文件来设置它们。

通过命令 dh_installemacs(1)可以把它们安装到临时文件中，所以如果你要使用它不要忘了去掉 rules 文件中那一行上的注释。

如果你不需要这些，删掉它们。

5.7 init.d.*.ex

如果你的软件包是一个需要在系统启动时运行的守护程序，那么很显然你没有采纳我在开始时的建议，不是吗？ :-)

这是一个/etc/init.d/脚本的通用骨架，所以你不得不对它进行大规模的修改。通过 dh_installinit(1)可以把它安装 到临时目录中。

如果你不需要这些，删掉这个文件。

5.8 manpage.1.*.ex, manpage.sgml.*.ex

你的程序应该有一个手册页。如果它们没有，这里的每一个文件都是一个模板， 你只要把它填好就可以了。

手册页通常用 nroff(1)写成。manpage.1.*.ex 这个例子也使用 nroff 写的。在 man(7)的手册页中可以 找到一个关于如何编写这样一个文件的简短说明。

如果你更希望些 SGML 而不时 nroff，那么你可以使用 manpage.sgml.*.ex 模板。 如果是这样，那么你就必需做下面这些事情：

- 安装软件包 docbook-to-man
- 把 docbook-to-man 加到 control 文件的 Build-Depends 一行。

- 删除 rules 文件 “build” 规则中 docbook-to-man 调用前的注释

另外还要记得把文件名改成类似于 gentoo.sgml 的样子！

最后，手册页文件的名字应该包含它所描述的程序的名称。所以我们需要把 “manpage” 改成 “gentoo”。这个文件明中还以一个 “.1” 作为后缀，这表明它是一个 关于用户命令的手册。一定要确保这个节编号是正确的。这里有一个关于手册页 各个节的简短列表：

Section	Description	Notes
1	User commands	Executable commands or scripts.
2	System calls	Functions provided by the kernel.
3	Library calls	Functions within system libraries.
4	Special files	Usually found in /dev
5	File formats	E.g. /etc/passwd's format
6	Games	Or other frivolous programs
7	Macro packages	Such as man macros.
8	System administration	Programs typically only run by root.
9	Kernel routines	Non-standard calls and internals.

所以 gentoo 的手册页应该叫做 gentoo.1。在原来的源程序中，并没有 gentoo.1 的手册页，所以我利用例子和上游文档给出的信息写了一个。

5.9 menu.ex

X 窗口系统的用户通常都会使用支持菜单的窗口管理器，这些菜单可以被定制用于 启动程序。如果它们安装了 Debian 的 menu 软件包，那么系统就会创建 一套包含有系统上每一个程序的菜单。

这里有一个缺省的由 dh_make 创建的 menu.ex 文件：

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

在冒号之后的第一个字段是 “needs”，他指明了程序需要什么样的界面。可以把这个字段改成一个合适的值，比如 “text” 或者 “X11”。

下面是 “section”，它指出这个项目应当出现的菜单和子菜单。当前的可选节被列在文件 /usr/share/doc/debian-policy/menu-policy.html/ch2.html#s2.1 中。

“title” 字段是程序的名称。如果你喜欢，可以用大写字母开头。但一定要使它保持简短。

最后，“command” 字段用于运行程序。

现在我们要把菜单项改成下面的样子：

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

你还可以加入其它的字段，比如 “longtitle”、“icon” 和 “hints” 等。参考 menufile(5)、update-menus(1) 和 /usr/share/doc/debian-policy/menu-policy.html/可以了解更多信息。

5.10 watch.ex

这个文件用于配制程序 uscan(1) 和 uupdate(1) (在软件包 devscripts 中)。它们可以用于监视你下载源代码的站点。

这里是我的配制：

```
# watch control file for uscan
# Site      Directory Pattern      Version Script
ftp.obsession.se /gentoo  gentoo-(.*)\.tar\.gz  debian  uupdate
```

提示：在创建了这个文件后，可以连接到 Internet，并且试着在程序目录中运行“uscan”命令。还要读手册哦！：)

5.11 ex.package.doc-base

如果你的软件包还有除了手册页以外的其它普通文档和 info 文档，你需要使用“doc-base”文件来注册它们，这样用户就能够用如 dhhelp(1)、dwww(1) 或者 doccentral(1) 等工具来找到它们。

这通常包括 /usr/share/doc/package/ 中的 HTML、PS 和 PDF 文件。

gentoo 的 doc-base 文件 gentoo.doc-base 如下所示：

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

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

关于安装的文件格式，请参考 install-docs(8) 和 /usr/share/doc/doc-base/doc-base.html/ 中的 doc-base 手册。

5.12 postinst.ex, preinst.ex, postrm.ex, prerm.ex

这些文件叫做维护者脚本。它们位于软件包的控制区域中，并且在你安装、升级或删除软件包时，dpkg 会调用它们。

现在，你应该尽可能避免修改任何维护者脚本，因为它们有一些复杂。要了解关于它们的更多信息请参考政策手册的第 6 章，而且还应该看看 dh_make 所提供的例子文件。

Debian 新维护人员手册

第 6 章 - 构建软件包

现在我们已经为构建软件包做好了准备。

6.1 完整的 rebuild

进入程序的主目录然后运行如下命令：

```
dpkg-buildpackage -rfakeroot
```

这将为你做好每件事情。它会：

- 清理源代码目录树 (debian/rules clean)，需要使用 fakeroot
- 构建源代码软件包 (dpkg-source -b)
- 构建程序 (debian/rules build)
- 构建二进制软件包 (debian/rules binary)，需要使用 fakeroot
- 用文件 .dsc 给源代码签名，需要使用 gnupg
- 创建上传文件 .changes 并给它签名，需要使用 dpkg-genchanges 和 gnupg

唯一需要你输入的是你的 GPG 密钥的密码，两次。

当完成所有这些，你会在上一层目录 (~ / gentoo /) 中看到下面的文件：

- gentoo_0.9.12.orig.tar.gz

这是原来的源程序的压缩包，为了遵守 Debian 的标准，修改了它的文件名。注意 它是我们通过在开始时运行带有 “-f” 参数的 dh_make 命令创建的。

- gentoo_0.9.12-1.dsc

这时对源代码的一个总结。这个程序是利用你的 “control” 文件创建的，并且在用 dpkg-source (1) 命令解包源代码时将会用到。这个 文件已经有了 GPG 签名，这样人们就可以确认他是你发布的。

- gentoo_0.9.12-1.diff.gz

这个文件中包含了每一个你对原始源代码所做的每一个修改，它的格式是 “unified diff”。他是由 dpkg-source (1) 程序创建的，而且这个程序还要使用它。警告：如果你没有把原始的源代码压缩包的名字改成 packagename_version.orig.tar.gz，dpkg-source 将不能正确地创建这个 .diff.gz 文件。

如果其它人希望重头重新构造你的软件包，它们可以用上面的三个文件很容易地做到。解包的过程很简单：只要把这三个文件复制到一个别的什么地方，然后运行 dpkg-source -x gentoo_0.9.12-1.dsc。

- gentoo_0.9.12-1_i386.deb

这是你的完整的二进制软件包。你可以象对待其它软件包一样用 dpkg 命令 安装和删除它。

- gentoo_0.9.12-1_i386.changes

这个文件描述了描述了所有对当前版本的修订版所作的改动，Debian FTP 文档维护程序在安装二进制版本软件包和源代码版本的软件包时将会使用到它。它的一部分是通过“changelog”文件和.dsc 文件创建的。这个文件已经有了 GPG 签名，这样人们可以确信它确实是你的。

因为你会继续花精力在这个软件包上，它的行为可能会改变，还有可能会增加一些新的功能。下载了你的软件包的人们可以通过阅读这个文件从而快速的了解到什么东西发生了变化。Debian 的文档维护程序也会把这个文件的内容发送到 debian-devel-changes 邮件列表上。

.dsc 和 .changes 文件中的长数字字符串是上面提到的文件的 MD5 校验码。下载了你的文件的人可以用 md5sum(1) 来检查这些数字是否相同，这样它们就可以知道文件是不是损坏了，或者是否被篡改了。

6.2 快速 rebuild

对于一个很大的软件包，你可能不希望你调整了 debian/rules 文件的一些细节后都从头来构建它。为了测试，你可以只制作一个 .deb 文件而不重新构造上游源代码，具体的作法如下所示：

```
fakeroot debian/rules binary
```

一旦你完成了调整，记得要根据上面的内容从头以正确的顺序重新构建软件包。如果你想上传一个以这种方式制作的 .deb 文件时可能会遇到错误。

6.3 debuild 命令

使用 debuild 命令你可以让后面的构造软件包的过程自动完成。参考 debuild(1)。

对 debuild 命令的地址可以通过修改/etc/devscripts.conf 或者 ~/.devscripts 完成。我建议至少修改以下内容：

```
DEBSIGN_KEYID="Your_GPG_keyID"
DEBUILD_DPKG_BUILDPACKAGE_OPTS="-i -ICVS -I.svn"
```

使用这个配置，你就可以总是使用你的 GPG 密钥来构造软件包并避免不希望的部件。(这对于发起人也是很好的。)譬如，使用一个用户帐号清空源码并重新构造软件包会非常简单：

```
debuild clean
debuild
```

6.4 dpatch 系统

简单地使用 `dh_make` 和 `dpkg-buildpackage` 命令会创建一个大的 `diff.gz` 文件，这个文件中会包含 `debian/` 目录中的文件和源码补丁。当日后要检查和理解每一处对源码的修改时，这样的包会难以处理。这样总不是很好。 [1]

目前已经有多种管理多个补丁的方法用于 Debian 中。dpatch 系统是其中最简单的一个，此外还有 `dbp`、`cdbs` 等。

一个用 dpatch 系统打包的软件包，其软件包的修改记录在 `debian/patches/` 目录下的一个文档清晰的补丁集中。`debian/` 目录之外的源码树并没有被修改。如果你需要其它人来帮你上传软件包，通过上述方法将你所做的修改清晰地分离出来并加上文档是非常重要的，它可以方便别人检查。dpatch 程序的用法在 `dpatch(1)` 中解释清楚了。

此后如果当某人(包括你自己)为你提供了一个源码补丁，在 dpatch 下修改源码包是非常容易的：

- 修改补丁，使其成为对源码树的 `-p1` 补丁。
- 用 `dpatch patch-template` 命令加入文件头。
- 将其放入 `debian/patches` 目录
- 将此 dpatch 文件放入 `debian/patches/00list` 文件中

此外 dpatch 还可以通过使用 CPP 宏让补丁对体系结构无关。

6.5 在上传时包含 `orig.tar.gz`

当你第一次上传了软件包时，你需要包含原始的 `orig.tar.gz` 源码包。如果包的 Debian 修正版本号不是 `-0` 或 `-1`，你在使用 `dpkg-buildpackage` 命令需要加上 `“-sa”` 选项。换句话说，`“-sd”` 选项将会除去 `orig.tar.gz` 文件。

Debian 新维护人员手册

第 7 章 - 检查软件包中的错误

7.1 lintian 软件包

在你的 .changes 文件上运行 lintian(1)；它们会检查出其中很多常见的错误。通常使用命令：

```
lintian -i gentoo_0.9.12-1_i386.changes
```

当然，要用为你的软件包产生的 .changes 文件的文件名替换掉上面的。如果这个 命令的运行结果显示在软件包中有错误(以 E:开始的行)，请仔细阅读关于错误的 说明(以 N:开始的行)，纠正错误，然后根据前文完整的 rebuild，第 6.1 节所述 重新构建软件包。如果在输入的信息中有以 W:开始的行，它们代表警告，那就要 调整软件包或者如果你确认这些警告是不是 spurious 的(让 Lintianoverride 它 们；请参考文档以获得更多的信息。)

你可以用 debuild(1)命令，它会首先 用 dpkg-buildpackage 构建软件包，接着运 行 lintian。

7.2 mc 命令

你可以用 dpkg-deb(1)解压*.deb 包中 的内容。你也可以用 debc(1)列出生成的 Debian 包中 的内容。

这也可以用如 mc(1)的文件管理器直接完成，使用它， 不仅可以浏览*.deb 文件的内容，还可以浏览*.diff.gz 和*.tar.gz 文件的内容。

请注意源码包和二进制包中没有用处的文件或零长度的文件。通常 cruft 都不能被 正确地清理；请调整你的 rules 文件来修复它们。

技巧：使用 “zgrep ^+++ ../gentoo_0.9.12-1.diff.gz” 命令可以得到 一系列对源文件进行的修改或增加。而 “dpkg-deb -c gentoo_0.9.12-1_i386.deb” 或 “debc gentoo_0.9.12-1_i386.changes” 会列出二进制包中的文件。

7.3 debdiff 命令

你可以用 debdiff(1)命令来比较两个 Debian 二进制软 件包中的文件列表。这对于核对是否有错误地放置或删除了文件和其它粗心大意 的修改是很有用的。你可以用 “debdiff old-package.change new-package.change” 检查一组*.deb 文件。

7.4 interdiff 命令

你可以用 `interdiff(1)` 命令比较两个 `diff.gz` 文件。这对于核对维护者在更新包时对于源码包是否有粗心大意地修改是很有用的。运行 “`interdiff -z old-package.diff.gz new-package.diff.gz`”。

7.5 debi 命令

自己安装你的软件包，比如用 `root` 的身份使用 `debi(1)` 命令。尝试在其它的机器上而不只是你自己的机器上安装并运行你的软件包，并仔细观察所有的在安装和运行时系统给出的错误信息。

7.6 pbuilder 包

对于净室(chroot)构造环境而言，要核对编译环境的依赖关系，`pbuilder` 软件包是很有用的。使用它可以确保在 `auto-builder` 中为不同的体系结构完全从源码完成编译，从而避免了很严重的 FTBFS(无法从源码编译)的 bug，而这种 bug 经常会出现在 RC(发布临界版)中。要了解 Debian 软件包 `auto-builder` 的更多信息，请参考 <http://buildd.debian.org/>。

最简单地使用 `pbuilder` 包的方法是直接以 `root` 身份使用 `pbuilder` 命令。例如，在包含了 `orig.tar.gz`、`.diff.gz` 和 `dsc` 的目录下使用下面的命令可以构造一个软件包。

```
root # pbuilder create # if second time, pbuilder update
root # pbuilder build foo.dsc
```

新构造的软件包可以在 `/var/cache/pbuilder/result/` 中找到，而它们的所有者都是 `root` 用户。

`pdebuild` 命令让你可以以普通用户的身份使用 `pbuilder` 包的功能。从源码树的根中，当其父目录中有 `orig.tar.gz` 时，你可以输入下面的命令：

```
$ sudo pbuilder create # if second time, sudo pbuilder update
$ pdebuild
```

新构造的软件包会在 `/var/cache/pbuilder/result/` 中，而其所有者将不再是 `root` 用户。
[2]

如果你希望增加新的 apt 源让 `pbuilder` 包，你可以设定 `OTHERMIRROR`、`~/.pbuilder` 和 `/etc/pbuilder` 且(对 `sarge`)可运行

```
$ sudo pbuilder update --distribution sarge --override-config
```

使用 `--override-config` 则需要更新 `chroot` 环境中的 apt 源。

参考 <http://www.netfort.gr.jp/~dancer/software/pbuilder.html>、`pdebuild(1)`、`pbuilder(5)` 和 `pbuilder(8)`。

Debian 新维护人员手册

第 8 章 - 上传软件包

现在你已经完整地测试过你的新软件包了，你可以开始 Debian 新维护者申请的过程，<http://www.debian.org/devel/join/newmaint> 讲述了这个过程。

8.1 上传到 Debian 档案库

一旦你成为正式的开发人员，你就可以上传软件包到 Debian 文档中了。你可以手工做这件事情，但如果使用一些已经提供的自动化工具(比如 `dupload(1)` 和 `dput(1)`)，这个过程将变得更容易。我们将仔细描述如何使用 `dupload` 来完成这一任务。

首先你必需设定 `dupload` 的配制文件。你可以修改会影响整个系统的文件 `/etc/dupload.conf`，或者是创建一个属于你自己的文件 `~/.dupload.conf`，来覆盖系统文件中一些你希望修改的部分。把下面的内容添加到文件中去：

```
package config;

$default_host = "anonymous-ftp-master";

$config{'anonymous-ftp-master'} = {
    fqdn => "ftp-master.debian.org",
    method => "ftp",
    incoming => "/pub/UploadQueue/",
    # files pass on to dinstall on ftp-master which sends emails itself
    dinstall_runs => 1,
};

1;
```

当然，要把我的个人设置改成你的，再阅读一下 `dupload.conf(5)` 的手册，搞懂这里的每一个选项是什么意思。

设定 `$default_host` 选项是最有窍门的——它会自动检查缺省情况下究竟使用哪一个上传序列。“anonymous-ftp-master”是一个主序列，但很有可能你会希望能够使用另外一个更快的。要了解关于上传序列(queues)的更多内容，请阅读位于 `/usr/share/doc/developers-reference/ch-pkgs.en.html#s-upload` 的开发人员参考中“Uploading a package”一节。

然后连接到你的 Internet 服务提供商，并且运行下面的命令：

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

`dupload` 会检查文件的 MD5 校验码是否和 `.changes` 文件中的相同，正因为此，它才会象在完整的 rebuild，第 6.1 节中所述警告你重新构建软件包，只有这样它才能正常上传软件包。

如果你在 <ftp://ftp-master.debian.org/pub/UploadQueue/> 上传时遇到问题，可以通过用 ftp 程序手动上传以 gnupg 签名的 *.commands 文件到 <ftp://ftp-master.debian.org/pub/UploadQueue/> 来解决这个问题。 [3] 例如，使用 hello.commands:

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

Uploader: Roman Hodek <Roman.Hodek@informatik.uni-erlangen.de>
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

iQCVAwUBNFiQSXVhJOHiWnvJAQG58AP+IDJVeSWmDvzMUpHScglEK0mvChgnuD7h
BRiVQubXkB2DphLJW5UUSRnjwliuFcYwH/1FpNp17XP95LkLX3iFza9qItw4k2/q
tvylZkmIA9jxCyv/YB6zZCbHmbvUnL473eLRoxlnYZd3JFaCZMJ86B0Ph4GFNPaf
Z4jxNrgh7Bc=
=pH94
-----END PGP SIGNATURE-----
```

8.2 上传到私有的档案库

如果你想要创建一个私有的档案库，并以开发者的身份简单使用 `dupload -t target_name` 命令将其放在 `URL="http://people.debian.org/~account_name"` 中， 你需要在 `/etc/dupload.conf` 文件中加入：

```
# Developer account
$cfg{'target_name'} = {
    fqdn => "people.debian.org",
    method => "scpb",
    incoming => "/home/account_name/public_html/package/",
    # I do not need to announce
    dinstall_runs => 1,
};
$cfg{'target_name'}{'preupload'}{'changes'} = "
    echo 'mkdir -p public_html/package' | ssh people.debian.org 2>/dev/null ;
    echo 'Package directory created!';";

$cfg{'target_name'}{'postupload'}{'changes'} = "
    echo 'cd public_html/package ;
    dpkg-scanpackages . /dev/null >Packages || true ;
    dpkg-scansources . /dev/null >Sources || true ;
    gzip -c Packages >Packages.gz ;
    gzip -c Sources >Sources.gz ' | ssh people.debian.org 2>/dev/null ;
    echo 'Package archive created!';";
```

这里，APT 档案库是使用一个快速但写的不太好的脚本通过远程 SSH 来完成的。`dpkg-scanpackages` 和 `dpkg-scansources` 需要的覆盖文件都使用了 `/dev/null`。非 Debian

开发者可以使用这一技术在他自己的 Web 站点上放置他自己的包。你也可以使用 apt-ftparchive 或其它脚本来创建 APT 档案库。

Debian 新维护人员手册

第 9 章 - 更新软件包

9.1 新的 Debian 修订版

现在我们假设有人提交了一个关于你的软件包的 bug 报告，第#54321 号，它描述了一个你可以解决的问题。要为你的软件包创建一个新的 Debian 修订版，你需要：

- 当然，必需更正软件包的源代码中的问题。
- 在 Debian changelog 文件中增加一个新的修订版，比如通过命令“`dch -i`”或者是“`dch -v <version>-<revision>`”。然后 用你喜欢的文本编辑器插入新的关于修改的注释信息。

小技巧：如何才能方便地以希望的格式得到日期呢？使用“`822-date`”或者是“`date -R`”。

- 在 changelog 的项目上包含一份对 bug 的简短描述以及解决方法，并将它附加 在“Closes: #54321”之后。这样，当你的软件包被 Debian 文档库接受的时候，这个 bug 报告将会被文档维护软件自动关闭。
 - 重复你在完整的 rebuild，第 6.1 节、检查软件包中的错误，第 7 章和上传软件包，第 8 章中所做的。不同的是这一次，原始的源代码将不会被包括，因为它们并没有被修改并且已经存在于 Debian 的文档库中了。
-

9.2 新的上游版本(基本)

现在我们来考虑另外一种情况，一种稍微复杂一点的情况——一个上游的版本发布了，当然你会希望它能够被打包。你需要做下面的事情：

- 下载新版本的源代码，并将它的压缩包(比如“`gentoo-0.9.13.tar.gz`”)放到原先的源代码目录树的上层目录中(比如`~/gentoo/`)。
- 进入旧的源代码目录，并且运行下面的命令：

```
uupdate -u gentoo-0.9.13.tar.gz
```

当然用你的新程序的文档名称替换掉这里的文件名。`uupdate(1)`将会修改这个压缩包的名字，还会试着将原先`.diff.gz`文件中的所有的修改应用到它上面，并且会更新新的 `debian/changelog` 文件。

- 更换到目录“`../gentoo-0.9.13`”中，它是新的软件包源码目录树，重复你在完整的 rebuild，第 6.1 节、检查软件包中的错误，第 7 章和上传软件包，第 8 章中所做的。

注意如果你已经如 watch.ex, 第 5.10 节所述建立了一个 “debian/watch” 文件, 那么你就可以通过运行 `uscan(1)` 来自 动检测新版本的源代码并下载它们, 然后运行 `uupdate`。

9.3 新的上游版本 (实际的)

当为 Debain 档案库准备软件包时, 你必须仔细检查最终的软件包。下面是一个更加 实际的过程。

- 校验上游修改
- 阅读上游的 changelog、NEWS 及其它可能和新版本 一起发布的文档。
- 用 “`diff -urN`” 比较新旧上游软件包的差别, 从而对修改的范围有个概 略性的了解, 哪些工作已经完成了(同时因此在哪里可能会有新的 bug), 而且还要留心 观察任何值得怀疑的东西。
- 把旧 Debian 软件包升级为新版本。
- 解压源码包, 并修改源码树的根目录为 `<packagename>-<upstream_version>/` 并 “`cd`” 到此目录中。
- 复制父目录中的源码包并将其更名为 `<packagename>_<upstream_version>.orig.tar.gz`。
- 对新的源码树也进行与旧源码树一样的修改。可能的方法有:
 - “`zcat /path/to/<packagename>_<old-version>.diff.gz | patch -p1`” 命令,
 - “`uupdate`” 命令
 - “`svn merge`” 命令, 如果你使用 Subversion 仓库来管理源码, 或者
 - 直接将 debian/ 目录从旧源码树复制到新的源码树中, 如果它是 用 `dpatch` 打包的。
- 保留旧的修改日志项目(看上去不重要, 但其实随时会有意外...)
- 新软件包的版本应该由上游版本号加上 Debian 修订号-1 构成, 例 如 “0.9.13-1”。
- 在 `debian/changelog` 文件的顶部为此新版本添加修改日志项目 —— “新的上游版本”。例如 “`dch -v 0.9.13-1`”。
- 简要地说明新上游版本中修复的已报 bug 并在修改日志中关闭这些 bug。
- 简要地说明维护者为修复已报 bug 对新上游版本所做的修改并且在修改 日志中关闭这些 bug。
- 如果补丁或者合并的过程并不顺利, 就要仔细地查出哪些地方有错误(在 `.rej` 文件中会留下线索。)多数情况下是因为你使用的补丁已经整合到上游版本 中, 这样就不再需要那补丁了。
- 向新版本的升级应当是安静地且不会打扰到用户(除非是发现旧的 bug 已经被修复 或者增加了新的功能, 已有的用户应当不会注意到升级。) [4]

- 如果你处于某种原因需要已经删除的模板文件，你可以在同一个已经“debian 化”的目录中再次运行 `dh_make`，运行时加上 `-o` 选项。然后就可以 修改它了。
 - 应当对已经存在的 Debian 修改进行重新评估；除非有什么不得已的原因，都应当 删除掉上游作者已经合并的(无论何种形式)并继续保留上游作者并未合并的。
 - If any changes were made to the build system (hopefully you'd know from the step 1 and update the `debian/rules` and `debian/control` build dependencies if necessary.
 - 如 `debuild` 命令，第 6.3 节或 `pbuilder` 包，第 7.6 节所述构建新的软件包。最好是使用 `pbuilder`。
 - 核对新软件包是否构造正确。
 - 执行 检查软件包中的错误，第 7 章.
 - 执行 校验软件包的升级，第 9.6 节.
 - 在此检查是否有已经修复但在 Debian Bug 跟踪系统(BTS) 仍然为开启状态的 bug。
 - 检查 `.changes` 文件的内容以确认你把软件包上传到了正确的发行版中、已经关闭 的 bug 已经列出在“Closes:”域中，而“Maintainer:Check”和“Changed-By:”域可以匹配， 以及文件已经用 GPG 签署等等。
 - 如果为修正任何内容而修改了软件包，请返回到第 2 步直到满意。
 - 如果你需要别人帮助才可以上传，请一定要注意在构造软件包的时候使用特殊的 选项(如“`dpkg-buildpackage -sa -v ...`”), 同时请通知帮助你上传的人以便他/她能够正确构造软件包。
 - 如果你自己上传，执行上传软件包，第 8 章.
-

9.4 orig.tar.gz 文件

如果你构造软件时使用的源码树只有 `debian/` 目录而没有 `orig.tar.gz` 文件在其父目录中，最后你将得到一个 Debian 专用源码包， 而没有 `diff.gz` 文件。这种包装方式应当仅对那些 Debian 专用的软件适用，这些软件包在其它发行版中应当是完全没有用处的。 [5]

要获得由 `orig.tar.gz` 和 `diff.gz` 两个文件构成的 非 Debian 专用的源码包，你必须手工复制上游软件包到父目录中，并将其名称改 为

`<packagename>_<upstream_version>.orig.tar.gz`，如首次“Debian 化”，第 2.4 节所述由 `dh_make` 所做的那样。

9.5 cvs-buildpackage 命令和 similes

你应当考虑使用一些源码关系系统来管理软件包。有几个脚本已经被定制用于和 多数流行的系统一起工作。

- CVS

- cvs-buildpackage
- Subversion
 - svn-buildpackage
- Arch (tla)
 - tla-buildpackage
 - arch-buildpackage

这些命令也可以使对新版上游软件的打包自动化。

9.6 校验软件包的升级

当你创建了一个软件包的新版本，你必需做下面的事情来确认所有的人都可以 安全的升级：

- 从原先的版本升级，
- 降级到原先的版本并删除它，
- 安装新的软件包，
- 删除它然后重新安装它一遍，
- 彻底清楚它。

注意如果你以前的软件包已经被发布到 Debian，人们会通常会更新到 Debian 最新 的发布中的那个版本上，所以要记得测试从那个版本升级的情况。

Debian 新维护人员手册

第 10 章 - 在哪里可以找到帮助

在你决定要在一些公共地方提出你的问题时，请先 RTFM。包括在 `/usr/share/doc/dpkg`、`/usr/share/doc/debian`、`/usr/share/doc/autotools-dev/README.Debian.gz` 和 `/usr/share/doc/package/*` 等目录中的 文档和所有在文档中提及的程序的 `man/info` 页面。在 <http://nm.debian.org/> 和 http://people.debian.org/~mpalmer/debian-mentors_FAQ.html 可以看到详细的信息。

如果你有一些关于软件打包的问题，但却无法从文档中找出答案，你可以在 Debian Mentors 邮件列表上提出你的问题，邮件列表 在 debian-mentors@lists.debian.org。一些经验更丰富的 Debian 开发人员将会很高兴地帮助你，但在问问题前你要确定确实至少读过一些文档！

更多的关于这个邮件列表的信息可以在 <http://lists.debian.org/debian-mentors/> 找到。

当你收到一个 bug 报告(是的，真正的 bug 报告！)，就是需要你去 [Debian Bug Tracking System](#) 看看 并读一下那里的文档的时候了，这样你才能高效地处理 bug。我强烈建议你阅读一下开发人员手册中“Handling Bugs”一节，它位于 `/usr/share/doc/developers-reference/ch-pkgs.en.html#s-bug-handling`。

如果你仍然有问题，可以在 Debian 开发人员邮件列表上把它提出来，这个邮件列表位于 debian-devel@lists.debian.org。要想了解更多关于这个邮件列表的信息，请参考 <http://lists.debian.org/debian-devel/>。

即便所有的东西都工作正常，也到了你祈祷的时候了。为什么？因为在几小时(或者几天)之后，全球各地的 Debian 用户将会开始使用你的软件包，如果你犯了一些严重的错误，那么你的邮箱就会因受到太多愤怒的用户的抱怨而爆炸…… 笑笑吧。 :-)

放松一下然后为接受 bug 报告做准备吧，因为在你的软件包能够完全符合 Debian 的各种政策之前还有很多工作要做呢(再说一遍，一定要阅读文档原文来了解详细信息哦)。祝你好运！

Debian 新维护人员手册

附录 A - 例子

现在我们来给上游软件 gentoo-1.0.2.tar.gz 打包，并且上传所有的 软件包到 nm_target.

A.1 简单打包例子

```
$ mkdir -p /path/to # new empty directory
$ cd /path/to
$ tar -xvzf /path/from/gentoo-1.0.2.tar.gz # get source
$ cd gentoo-1.0.2
$ dh_make -e name@domain.dom -f /path/from/gentoo-1.0.2.tar.gz
... Answer prompts.
... Fix source tree
... If it is a script package, set debian/control to "Architecture: all"
... Do not erase ../gentoo_1.0.2.orig.tar.gz
$ debuild
... Make sure no warning happens.
$ cd ..
$ dupload -t nm_target gentoo_1.0.2-1_i386.changes
```

A.2 用 dpatch 和 pbuilder 打包

```
$ mkdir -p /path/to # new empty directory
$ cd /path/to
$ tar -xvzf /path/from/gentoo-1.0.2.tar.gz
$ cp -a gentoo-1.0.2 gentoo-1.0.2-orig
$ cd gentoo-1.0.2
$ dh_make -e name@domain.dom -f /path/from/gentoo-1.0.2.tar.gz
... Answer prompts.
... Fix source tree by editor
... Try building packages with "dpkg-buildpackage -rfakeroot -us -uc"
... Edit source to make source buildable.
... Do not erase ../gentoo_1.0.2.orig.tar.gz
$ cd ..
$ cp -a gentoo-1.0.2 gentoo-1.0.2-keep # safety backup
$ mv gentoo-1.0.2/debian debian
$ diff -Nru gentoo-1.0.2-orig gentoo-1.0.2 > patch-file
... You may overwrite gentoo-1.0.2 directory while doing this.
... Make sure to keep gentoo-1.0.2-keep for your safety
$ mkdir -p debian/patches
$ dpatch patch-template patch-file \
    -p "01_patchname" "patch-file description" \
    < patch-file > debian/patches/01_patchname.dpatch
$ cd debian/patches
```

```
$ echo 01_patchname.dpatch >00list
$ cd ../../ # back to /path/to
$ rm -rf gentoo-1.0.2
$ editor debian/rules
```

现在文件 `debian/rules` 的内容是：

```
config.status: configure
./configure --prefix=/usr --mandir=/usr/share
build: config.status
${MAKE}
clean:
$(testdir)
$(testroot)
${MAKE} distclean
rm -rf debian/imaginary-package debian/files debian/substvars
```

你可以用编辑器修改 `debian/rules` 文件为以下内容使其使用 `dpatch`：

```
config.status: patch configure
./configure --prefix=/usr --mandir=/usr/share
build: config.status
${MAKE}
clean: clean-patched unpatch
clean-patched:
$(testdir)
$(testroot)
${MAKE} distclean
rm -rf debian/imaginary-package debian/files debian/substvars
patch: patch-stamp
patch-stamp:
dpatch apply-all
dpatch call-all -a=pkg-info >patch-stamp

unpatch:
dpatch deapply-all
rm -rf patch-stamp debian/patched
```

现在你可以用 `dpatch` 系统重新打包了。

```
$ tar -xvzf gentoo_1.0.2.orig.tar.gz
$ cp -a debian/ gentoo-1.0.2/debian
$ cd gentoo-1.0.2
$ sudo pbuilder update
$ pdebuild
$ cd /var/cache/pbuilder/result/
$ dupload -t nm_target gentoo_1.0.2-1_i386.changes
```