Debian Developer's Reference

Adam Di Carlo, current maintainer <aph@debian.org>
Christian Schwarz <schwarz@debian.org>
Ian Jackson <ijackson@gnu.ai.mit.edu>

ver. 3.0, 14 June, 2002

Copyright Notice

Contents

Chapter 1

Scope of This Document

The purpose of this document is to provide an overview of the recommended procedures and the available resources for Debian developers.

The procedures discussed within include how to become a maintainer ('Applying to Become a Maintainer' on page 3); how to upload new packages ('Package uploads' on page 27); how and when to do ports and interim releases of other maintainers' packages ('Non-Maintainer Uploads (NMUs)' on page 35); how to move, remove, or orphan packages ('Moving, Removing, Renaming, Adopting, and Orphaning Packages' on page 44); and how to handle bug reports ('Handling package bugs' on page 47).

The resources discussed in this reference include the mailing lists ('Mailing lists' on page 11) and servers ('Debian servers' on page 13); a discussion of the structure of the Debian archive ('The Debian archive' on page 15); explanation of the different servers which accept package uploads ('Uploading to `ftp-master`' on page 31); and a discussion of resources which can help maintainers with the quality of their packages ('Overview of Debian Maintainer Tools' on page 59).

It should be clear that this reference does not discuss the technical details of the Debian package nor how to generate Debian packages. Nor does this reference detail the standards to which Debian software must comply. All of such information can be found in the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`).

Furthermore, this document is *not an expression of formal policy*. It contains documentation for the Debian system and generally agreed-upon best practices. Thus, it is what is called a "normative" document.

Chapter 2

Applying to Become a Maintainer

2.1 Getting started

So, you've read all the documentation, you've gone through the Debian New Maintainers' Guide (`http://www.debian.org/doc/maint-guide/`), understand what everything in the `hello` example package is for, and you're about to Debianize your favorite piece of software. How do you actually become a Debian developer so that your work can be incorporated into the Project?

Firstly, subscribe to `<debian-devel@lists.debian.org>` if you haven't already. Send the word `subscribe` in the *Subject* of an email to `<debian-devel-REQUEST@lists.debian.org>`. In case of problems, contact the list administrator at `<listmaster@lists.debian.org>`. More information on available mailing lists can be found in 'Mailing lists' on page 11. `<debian-devel-announce@lists.debian.org>` is another list which is mandatory for anyone who wish to follow Debian's development.

You should subscribe and lurk (that is, read without posting) for a bit before doing any coding, and you should post about your intentions to work on something to avoid duplicated effort.

Another good list to subscribe to is `<debian-mentors@lists.debian.org>`. See 'Debian mentors and sponsors' on page 5 for details. The IRC channel `#debian` on the Linux People IRC network (e.g., `irc.debian.org`) can also be helpful.

When you know how you want to contribute to Debian GNU/Linux, you should get in contact with existing Debian maintainers who are working on similar tasks. That way, you can learn from experienced developers. For example, if you are interested in packaging existing software for Debian you should try to get a sponsor. A sponsor will work together with you on your package and upload it to the Debian archive once he is happy with the packaging work you have done. You can find a sponsor by mailing the `<debian-mentors@lists.debian.org>` mailing list, describing your package and yourself and asking for a sponsor (see 'Sponsoring packages' on page 57 for more information on sponsoring). On the other hand, if you are interested in porting Debian to alternative architectures or kernels you can subscribe to port specific mailing lists and ask there how to get started. Finally, if you are interested in documentation or Quality Assurance (QA) work you can join maintainers already working on these tasks and submit patches and improvements.

2.2 Registering as a Debian developer

Before you decide to register with Debian GNU/Linux, you will need to read all the information available at the New Maintainer's Corner (`http://www.debian.org/devel/join/newmaint`). It describes exactly the preparations you have to do before you can register to become a Debian developer. For example, before you apply, you have to to read the Debian Social Contract (`http://www.debian.org/social_contract`). Registering as a developer means that you agree with and pledge to uphold the Debian Social Contract; it is very important that maintainers are in accord with the essential ideas behind Debian GNU/Linux. Reading the GNU Manifesto (`http://www.gnu.org/gnu/manifesto.html`) would also be a good idea.

The process of registering as a developer is a process of verifying your identity and intentions, and checking your technical skills. As the number of people working on Debian GNU/Linux has grown to over 800 people and our systems are used in several very important places we have to be careful about being compromised. Therefore, we need to verify new maintainers before we can give them accounts on our servers and let them upload packages.

Before you actually register you should have shown that you can do competent work and will be a good contributor. You can show this by submitting patches through the Bug Tracking System or having a package sponsored by an existing maintainer for a while. Also, we expect that contributors are interested in the whole project and not just in maintaining their own packages. If you can help other maintainers by providing further information on a bug or even a patch, then do so!

Registration requires that you are familiar with Debian's philosophy and technical documentation. Furthermore, you need a GnuPG key which has been signed by an existing Debian maintainer. If your GnuPG key is not signed yet, you should try to meet a Debian maintainer in person to get your key signed. There's a GnuPG Key Signing Coordination page (`http://nm.debian.org/gpg.php`) which should help you find a maintainer close to you (If you cannot find a Debian maintainer close to you, there's an alternative way to pass the ID check. You can send in a photo ID signed with your GnuPG key. Having your GnuPG key signed is the preferred way, however. See the identification page (`http://www.debian.org/devel/join/nm-step2`) for more information about these two options.)

If you do not have an OpenPGP key yet, generate one. Every developer needs a OpenPGP key in order to sign and verify package uploads. You should read the manual for the software you are using, since it has much important information which is critical to its security. Many more security failures are due to human error than to software failure or high-powered spy techniques. See 'Maintaining your public key' on page 7 for more information on maintaining your public key.

Debian uses the `GNU Privacy Guard` (package `gnupg` version 1 or better) as its baseline standard. You can use some other implementation of OpenPGP as well. Note that OpenPGP is an open standard based on RFC 2440 (`http://www.gnupg.org/rfc2440.html`).

The recommended public key algorithm for use in Debian development work is the DSA (sometimes call "DSS" or "DH/ElGamal"). Other key types may be used however. Your key length must be at least 1024 bits; there is no reason to use a smaller key, and doing so would be much less secure. Your key must be signed with at least your own user ID; this prevents user ID tampering. `gpg` does this automatically.

If your public key isn't on public key servers such as `pgp5.ai.mit.edu`, please read the documentation available locally in `/usr/share/doc/pgp/keyserv.doc`. That document contains instructions on how to put your key on the public key servers. The New Maintainer Group will put your public key on the servers if it isn't already there.

Some countries restrict the use of cryptographic software by their citizens. This need not impede one's activities as a Debian package maintainer however, as it may be perfectly legal to use cryptographic products for authentication, rather than encryption purposes. Debian GNU/Linux does not require the use of cryptography *qua* cryptography in any manner. If you live in a country where use of cryptography even for authentication is forbidden then please contact us so we can make special arrangements.

To apply as a new maintainer, you need an existing Debian maintainer to verify your application (an *advocate*). After you have contributed to Debian for a while, and you want to apply to become a registered developer, an existing developer with whom you have worked over the past months has to express his belief that you can contribute to Debian successfully.

When you have found an advocate, have your GnuPG key signed and have already contributed to Debian for a while, you're ready to apply. You can simply register on our application page (`http://nm.debian.org/newnm.php`). After you have signed up, your advocate has to confirm your application. When your advocate has completed this step you will be assigned an Application Manager who will go with you through the necessary steps of the New Maintainer process. You can always check your status on the applications status board (`http://nm.debian.org/`).

For more details, please consult New Maintainer's Corner (`http://www.debian.org/devel/join/newmaint`) at the Debian web site. Make sure that you are familiar with the necessary steps of the New Maintainer process before actually applying. If you are well prepared, you can save a lot of time later on.

2.3 Debian mentors and sponsors

The mailing list `<debian-mentors@lists.debian.org>` has been set up for novice maintainers who seek help with initial packaging and other developer-related issues. Every new developer is invited to subscribe to that list (see 'Mailing lists' on page 11 for details).

Those who prefer one-on-one help (e.g., via private email) should also post to that list and an experienced developer will volunteer to help.

In addition, if you have some packages ready for inclusion in Debian, but are waiting for your new maintainer application to go through, you might be able find a sponsor to upload your package for you. Sponsors are people who are official Debian maintainers, and who are willing to criticize and upload your packages for you. Those who are seeking a sponsor can request one at `http://www.internatif.org/bortzmeyer/debian/sponsor/`.

If you wish to be a mentor and/or sponsor, more information is available in 'Interacting with prospective Debian developers' on page 57.

Chapter 3

Debian Developer's Duties

3.1 Maintaining your Debian information

There's a LDAP database containing many informations concerning all developers, you can access it at https://db.debian.org/. You can update your password (this password is propagated to most of the machines that are accessible to you), your address, your country, the latitude and longitude of the point where you live, phone and fax numbers, your preferred shell, your IRC nickname, your web page and the email that you're using as alias for your debian.org email. Most of the information is not accessible to the public, for more details about this database, please read its online documentation that you can find at http://db.debian.org/doc-general.html.

You have to keep the information available there up-to-date.

3.2 Maintaining your public key

Be very careful with your private keys. Do not place them on any public servers or multiuser machines, such as `master.debian.org`. Back your keys up; keep a copy offline. Read the documentation that comes with your software; read the PGP FAQ (http://www.cam.ac.uk.pgp.net/pgpnet/pgp-faq/).

If you add signatures to your public key, or add user identities, you can update the debian key ring by sending your key to the key server at `keyring.debian.org`. If you need to add a completely new key, or remove an old key, send mail to <keyring-maint@debian.org>. The same key extraction routines discussed in 'Registering as a Debian developer' on page 4 apply.

You can find a more in-depth discussion of Debian key maintenance in the documentation of the `debian-keyring` package.

3.3 Voting

Even if Debian is not always a real democracy, Debian has democratic tools and uses a democratic process to elect its leader or to approve a general resolution. Those processes are described in the Debian Constitution (`http://www.debian.org/devel/constitution`).

Democratic processes work well only if everybody take part in the vote, that's why you have to vote. To be able to vote you have to subscribe to `<debian-devel-announce@lists.debian.org>` since call for votes are sent there. If you want to follow the debate preceding a vote, you may want to subscribe to `<debian-vote@lists.debian.org>`.

The list of all the proposals (past and current) is available on the web at `url-vote`. You will find there additional information about how to make a vote proposal.

3.4 Going on vacation gracefully

Most developers take vacations, and usually this means that they can't work for Debian and they can't be reached by email if any problem occurs. The other developers need to know that you're on vacation so that they'll do whatever is needed when such a problem occurs. Usually this means that other developers are allowed to NMU (see 'Non-Maintainer Uploads (NMUs)' on page 35) your package if a big problem (release critical bugs, security update, etc.) occurs while you're on vacation.

In order to inform the other developers, there's two things that you should do. First send a mail to `<debian-private@lists.debian.org>` giving the period of time when you will be on vacation. You can also give some special instructions on what to do if any problem occurs. Be aware that some people don't care for vacation notices and don't want to read them; you should prepend "[VAC] " to the subject of your message so that it can be easily filtered.

Next you should update your information available in the Debian LDAP database and mark yourself as "on vacation" (this information is only accessible to debian developers). Don't forget to remove the "on vacation" flag when you come back!

3.5 Coordination with upstream developers

A big part of your job as Debian maintainer will be to stay in contact with the upstream developers. Debian users will sometimes report bugs to the Bug Tracking System that are not specific to Debian. You must forward these bug reports to the upstream developers so that they can be fixed in a future release. It's not your job to fix non-Debian specific bugs. However, if you are able to do so, you are encouraged to contribute to upstream development of the package by providing a fix for the bug. Debian users and developers will often submit patches to fix upstream bugs, and you should evaluate and forward these patches upstream.

If you need to modify the upstream sources in order to build a policy compliant package, then you should propose a nice fix to the upstream developers which can be included there, so that you won't have to modify the sources of the next upstream version. Whatever changes you need, always try not to fork from the upstream sources.

3.6 Managing release-critical bugs

Release-critical bugs (RCB) are all bugs that have severity *critical*, *grave* or *serious*. Those bugs can delay the Debian release and/or can justify the removal of a package at freeze time. That's why these bugs need to be corrected as quickly as possible. You must be aware that some developers who are part of the Debian Quality Assurance (`http://qa.debian.org/`) effort are following those bugs and try to help you whenever they are able. But if you can't fix such bugs within 2 weeks, you should either ask for help by sending a mail to the Quality Assurance (QA) group `<debian-qa@lists.debian.org>`, or explain your difficulties and present a plan to fix them by sending a mail to the proper bug report. Otherwise, people from the QA group may want to do a Non-Maintainer Upload (see 'Non-Maintainer Uploads (NMUs)' on page 35) after trying to contact you (they might not wait as long as usual before they do their NMU if they have seen no recent activity from you in the BTS).

3.7 Retiring

If you choose to leave the Debian project, you should make sure you do the following steps:

1. Orphan all your packages, as described in 'Orphaning a package' on page 46.
2. Send an email about how you are leaving the project to `<debian-private@lists.debian.org>`.
3. Notify the Debian key ring maintainers that you are leaving by emailing to `<keyring-maint@debian.org>`.

Chapter 4

Resources for Debian Developers

In this chapter you will find a very brief road map of the Debian mailing lists, the main Debian servers, other Debian machines which may be available to you as a developer, and all the other resources that are available to help you in your maintainer work.

4.1 Mailing lists

The mailing list server is at `lists.debian.org`. Mail `debian-`*`foo`*`-REQUEST@lists.debian.org`, where `debian-`*`foo`* is the name of the list, with the word `subscribe` in the *Subject* to subscribe to the list or `unsubscribe` to unsubscribe. More detailed instructions on how to subscribe and unsubscribe to the mailing lists can be found at `http://www.debian.org/MailingLists/subscribe`, `ftp://ftp.debian.org/debian/doc/mailing-lists.txt` or locally in `/usr/share/doc/debian/mailing-lists.txt` if you have the `doc-debian` package installed.

When replying to messages on the mailing list, please do not send a carbon copy (`CC`) to the original poster unless they explicitly request to be copied. Anyone who posts to a mailing list should read it to see the responses.

The following are the core Debian mailing lists: `<debian-devel@lists.debian.org>`, `<debian-policy@lists.debian.org>`, `<debian-user@lists.debian.org>`, `<debian-private@lists.debian.org>`, `<debian-announce@lists.debian.org>`, and `<debian-devel-announce@lists.debian.org>`. All developers are expected to be subscribed to at least `<debian-devel-announce@lists.debian.org>`. There are other mailing lists available for a variety of special topics; see `http://www.debian.org/MailingLists/subscribe` for a list. Cross-posting (sending the same message to multiple lists) is discouraged.

`<debian-private@lists.debian.org>` is a special mailing list for private discussions amongst Debian developers. It is meant to be used for posts which for whatever reason should not be published publicly. As such, it is a low volume list, and users are urged not to use `<debian-private@`

`lists.debian.org>` unless it is really necessary. Moreover, do *not* forward email from that list to anyone. Archives of this list are not available on the web for obvious reasons, but you can see them using your shell account on `master.debian.org` and looking in the `~debian/archive/debian-private` directory.

`<debian-email@lists.debian.org>` is a special mailing list used as a grab-bag for Debian related correspondence such as contacting upstream authors about licenses, bugs, etc. or discussing the project with others where it might be useful to have the discussion archived somewhere.

As ever on the net, please trim down the quoting of articles you're replying to. In general, please adhere to the usual conventions for posting messages.

Online archives of mailing lists are available at `http://lists.debian.org/`.

4.2 IRC channels

Several IRC channels are dedicated to Debian's development. They are all hosted on the OpenProjects (`http://www.openprojects.net`) network. The `irc.debian.org` DNS entry is just an alias to `irc.openprojects.net`.

The main channel *#debian-devel* is very active since more than 150 persons are always logged in. It's a channel for people who work on Debian, it's not a support channel (there's *#debian* for that). It is however open to anyone who wants to lurk (and learn). Its topic is always full of interesting informations. Since it's an open channel, you should not speak there of issues that are discussed in `<debian-private@lists.debian.org>`. There's a key protected channel *#debian-private* for that purpose. The key is available in the archives of debian-private in `master.debian.org:~debian/archive/debian-private/`, just `zgrep` for *#debian-private* in all the files.

There are other additional channels dedicated to specific subjects. *#debian-bugs* is used for coordinating bug squash parties. *#debian-boot* is used to coordinate the work on the boot floppies (i.e. the installer). *#debian-doc* is occasionally used to work on documentation like the one you are reading. Other channels are dedicated to an architecture or a set of packages: *#debian-bsd*, *#debian-kde*, *#debian-sf* (SourceForge package), *#debian-oo* (OpenOffice package) . . .

Some non-English channels exist, for example *#debian-devel-fr* for French speaking people interested in Debian's development.

4.3 Documentation

This document contains many informations very useful to Debian developers, but it can not contain everything. Most of the other interesting documents are linked from The Developers' Corner (http://www.debian.org/devel/). Take the time to browse all the links, you will learn many more things.

4.4 Debian servers

Debian servers are well known servers which serve critical functions in the Debian project. Every developer should know what these servers are and what they do.

If you have a problem with the operation of a Debian server, and you think that the system operators need to be notified of this problem, please find the contact address for the particular machine at http://db.debian.org/machines.cgi. If you have a non-operating problems (such as packages to be remove, suggestions for the web site, etc.), generally you'll report a bug against a "pseudo-package". See 'Bug Reporting' on page 55 for information on how to submit bugs.

4.4.1 The master server

`master.debian.org` is the canonical location for the Bug Tracking System (BTS). If you plan on doing some statistical analysis or processing of Debian bugs, this would be the place to do it. Please describe your plans on `<debian-devel@lists.debian.org>` before implementing anything, however, to reduce unnecessary duplication of effort or wasted processing time.

All Debian developers have accounts on `master.debian.org`. Please take care to protect your password to this machine. Try to avoid login or upload methods which send passwords over the Internet in the clear.

If you find a problem with `master.debian.org` such as disk full, suspicious activity, or whatever, send an email to `<debian-admin@debian.org>`.

4.4.2 The ftp-master server

The ftp-master server, `ftp-master.debian.org` (or `auric.debian.org`), holds the canonical copy of the Debian archive (excluding the non-US packages). Generally, package uploads go to this server; see 'Package uploads' on page 27.

Problems with the Debian FTP archive generally need to be reported as bugs against the `ftp.debian.org` pseudo-package or an email to `<ftpmaster@debian.org>`, but also see the procedures in 'Moving, Removing, Renaming, Adopting, and Orphaning Packages' on page 44.

4.4.3 The WWW server

The main web server, `www.debian.org`, is also known as `klecker.debian.org`. All developers are given accounts on this machine.

If you have some Debian-specific information which you want to serve up on the web, you can do this by putting material in the `public_html` directory under your home directory. You should do this on `klecker.debian.org`. Any material you put in those areas are accessible via the URL `http://people.debian.org/~`*`user-id`*`/`. You should only use this particular location because it will be backed up, whereas on other hosts it won't. Please do not put any material on Debian servers not relating to Debian, unless you have prior permission. Send mail to `<debian-devel@lists.debian.org>` if you have any questions.

If you find a problem with the Debian web server, you should generally submit a bug against the pseudo-package, `www.debian.org`. First check whether or not someone else has already reported the problem on the Bug Tracking System (`http://bugs.debian.org/www.debian.org`).

4.4.4 The CVS server

`cvs.debian.org` is also known as `klecker.debian.org`, discussed above. If you need to use a publicly accessible CVS server, for instance, to help coordinate work on a package between many different developers, you can request a CVS area on the server.

Generally, `cvs.debian.org` offers a combination of local CVS access, anonymous client-server read-only access, and full client-server access through `ssh`. Also, the CVS area can be accessed read-only via the Web at `http://cvs.debian.org/`.

To request a CVS area, send a request via email to `<debian-admin@debian.org>`. Include the name of the requested CVS area, the Debian account that should own the CVS root area, and why you need it.

4.4.5 The Developers Database

The Developers Database, at `https://db.debian.org/`, is an LDAP directory for managing Debian developer attributes. You can use this resource to search the list of Debian developers. For information on keeping your entry the developer database up-to-date, see 'Maintaining your Debian information' on page 7. Part of this information is also available through the finger service on Debian servers, try `finger yourlogin@debian.org` to see what it reports.

4.5 Mirrors of Debian servers

The web and FTP servers have several mirrors available. Please do not put heavy load on the canonical FTP or web servers. Ideally, the canonical servers only mirror out to a first tier of mirrors, and all user access is to the mirrors. This allows Debian to better spread its bandwidth requirements over several servers and networks. Note that newer push mirroring techniques ensure that mirrors are as up-to-date as they can be.

The main web page listing the available public FTP (and, usually, HTTP) servers can be found at `http://www.debian.org/distrib/ftplist`. More information concerning Debian mirrors can be found at `http://www.debian.org/mirror/`. This useful page includes information and tools which can be helpful if you are interested in setting up your own mirror, either for internal or public access.

Note that mirrors are generally run by third-parties who are interested in helping Debian. As such, developers generally do not have accounts on these machines.

4.6 Other Debian developer machines

There are other Debian machines which may be made available to you. You can use these for Debian-related purposes as you see fit. Please be kind to system administrators, and do not use up tons and tons of disk space, network bandwidth, or CPU without first getting the approval of the local maintainers. Usually these machines are run by volunteers. Generally, these machines are for porting activities.

Aside from the servers mentioned in 'Debian servers' on page 13, there is a list of machines available to Debian developers at `http://db.debian.org/machines.cgi`.

4.7 The Debian archive

The Debian GNU/Linux distribution consists of a lot of packages (`.deb`'s, currently around 9000) and a few additional files (such documentation and installation disk images).

Here is an example directory tree of a complete Debian archive:

```
dists/stable/main/
dists/stable/main/binary-i386/
dists/stable/main/binary-m68k/
dists/stable/main/binary-alpha/
     ...
```

```
dists/stable/main/source/
     ...
dists/stable/main/disks-i386/
dists/stable/main/disks-m68k/
dists/stable/main/disks-alpha/
     ...

dists/stable/contrib/
dists/stable/contrib/binary-i386/
dists/stable/contrib/binary-m68k/
dists/stable/contrib/binary-alpha/
     ...
dists/stable/contrib/source/

dists/stable/non-free/
dists/stable/non-free/binary-i386/
dists/stable/non-free/binary-m68k/
dists/stable/non-free/binary-alpha/
     ...
dists/stable/non-free/source/

dists/testing/
dists/testing/main/
     ...
dists/testing/contrib/
     ...
dists/testing/non-free/
     ...

dists/unstable
dists/unstable/main/
     ...
dists/unstable/contrib/
     ...
dists/unstable/non-free/
     ...

pool/
pool/main/a/
pool/main/a/apt/
     ...
pool/main/b/
```

```
pool/main/b/bash/
     ...
pool/main/liba/
pool/main/liba/libalias-perl/
     ...
pool/main/m/
pool/main/m/mailx/
     ...
pool/non-free/n/
pool/non-free/n/netscape/
     ...
```

As you can see, the top-level directory contains two directories, `dists/` and `pool/`. The latter is a "pool" in which the packages actually are, and which is handled by the archive maintenance database and the accompanying programs. The former contains the distributions, *stable*, *testing* and *unstable*. Each of those distribution directories is divided in equivalent subdirectories purpose of which is equal, so we will only explain how it looks in stable. The `Packages` and `Sources` files in the distribution subdirectories can reference files in the `pool/` directory.

`dists/stable` contains three directories, namely `main`, `contrib`, and `non-free`.

In each of the areas, there is a directory for the source packages (`source`) and a directory for each supported architecture (`binary-i386`, `binary-m68k`, etc.).

The `main` area contains additional directories which holds the disk images and some essential pieces of documentation required for installing the Debian distribution on a specific architecture (`disks-i386`, `disks-m68k`, etc.).

4.7.1 Sections

The *main* section of the Debian archive is what makes up the **official Debian GNU/Linux distribution**. The *main* section is official because it fully complies with all our guidelines. The other two sections do not, to different degrees; as such, they are **not** officially part of Debian GNU/Linux.

Every package in the main section must fully comply with the Debian Free Software Guidelines (`http://www.debian.org/social_contract#guidelines`) (DFSG) and with all other policy requirements as described in the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`). The DFSG is our definition of "free software." Check out the Debian Policy Manual for details.

Packages in the *contrib* section have to comply with the DFSG, but may fail other requirements. For instance, they may depend on non-free packages.

Packages which do not conform to the DFSG are placed in the *non-free* section. These packages are not considered as part of the Debian distribution, though we support their use, and we provide infrastructure (such as our bug-tracking system and mailing lists) for non-free software packages.

The Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`) contains a more exact definition of the three sections. The above discussion is just an introduction.

The separation of the three sections at the top-level of the archive is important for all people who want to distribute Debian, either via FTP servers on the Internet or on CD-ROMs: by distributing only the *main* and *contrib* sections, one can avoid any legal risks. Some packages in the *non-free* section do not allow commercial distribution, for example.

On the other hand, a CD-ROM vendor could easily check the individual package licenses of the packages in *non-free* and include as many on the CD-ROMs as he's allowed to. (Since this varies greatly from vendor to vendor, this job can't be done by the Debian developers.)

4.7.2 Architectures

In the first days, the Linux kernel was only available for the Intel i386 (or greater) platforms, and so was Debian. But when Linux became more and more popular, the kernel was ported to other architectures, too.

The Linux 2.0 kernel supports Intel x86, DEC Alpha, SPARC, Motorola 680x0 (like Atari, Amiga and Macintoshes), MIPS, and PowerPC. The Linux 2.2 kernel supports even more architectures, including ARM and UltraSPARC. Since Linux supports these platforms, Debian decided that it should, too. Therefore, Debian has ports underway; in fact, we also have ports underway to non-Linux kernel. Aside from *i386* (our name for Intel x86), there is *m68k*, *alpha*, *powerpc*, *sparc*, *hurd-i386*, *arm*, *ia64*, *hppa*, *s390*, *mips*, *mipsel* and *sh* as of this writing.

Debian GNU/Linux 1.3 is only available as *i386*. Debian 2.0 shipped for *i386* and *m68k* architectures. Debian 2.1 ships for the *i386*, *m68k*, *alpha*, and *sparc* architectures. Debian 2.2 added support for the *powerpc* and *arm* architectures. Debian 3.0 adds support of five new architectures: *ia64*, *hppa*, *s390*, *mips* and *mipsel*.

Information for developers or uses about the specific ports are available at the Debian Ports web pages (`http://www.debian.org/ports/`).

4.7.3 Packages

There are two types of Debian packages, namely *source* and *binary* packages.

Source packages consist of either two or three files: a `.dsc` file, and either a `.tar.gz` file or both an `.orig.tar.gz` and a `.diff.gz` file.

If a package is developed specially for Debian and is not distributed outside of Debian, there is just one `.tar.gz` file which contains the sources of the program. If a package is distributed elsewhere too, the `.orig.tar.gz` file stores the so-called *upstream source code*, that is the source code that's distributed from the *upstream maintainer* (often the author of the software). In this case, the `.diff.gz` contains the changes made by the Debian maintainer.

The `.dsc` file lists all the files in the source package together with checksums (`md5sums`) and some additional info about the package (maintainer, version, etc.).

4.7.4 Distribution directories

The directory system described in the previous chapter is itself contained within *distribution directories*. Each distribution is actually contained in the `pool` directory in the top-level of the Debian archive itself.

To summarize, the Debian archive has a root directory within an FTP server. For instance, at the mirror site, `ftp.us.debian.org`, the Debian archive itself is contained in `/debian`, which is a common location (another is `/pub/debian`).

A distribution is comprised of Debian source and binary packages, and the respective `Sources` and `Packages` index files, containing the header information from all those packages. The former are kept in the `pool/` directory, while the latter are kept in the `dists/` directory of the archive (for backwards compatibility).

Stable, testing, and unstable

There are always distributions called *stable* (residing in `dists/stable`), one called *testing* (residing in `dists/testing`), and one called *unstable* (residing in `dists/unstable`). This reflects the development process of the Debian project.

Active development is done in the *unstable* distribution (that's why this distribution is sometimes called the *development distribution*). Every Debian developer can update his or her packages in this distribution at any time. Thus, the contents of this distribution changes from day-to-day. Since no special effort is done to make sure everything in this distribution is working properly, it is sometimes literally unstable.

The testing distribution is generated automatically by taking packages from unstable if they satisfy certain criteria. Those criteria should ensure a good quality for packages within testing. 'The testing scripts' on page 23 are launched each day after the new packages have been installed.

After a period of development, once the release manager deems fit, the *testing* distribution is frozen, meaning that the policies which control how packages move from *unstable* to testing are tightened. Packages which are too buggy are removed. No changes are allowed into *testing* except for bug fixes. After some time has elapsed, depending on progress, the *testing* distribution goes into a 'deep freeze', when no changes are made to it except those needed for the installation system. This is called a "test cycle", and it can last up to two weeks. There can be several test cycles, until the distribution is prepared for release, as decided by the release manager. At the end of the last test cycle, the *testing* distribution is renamed to *stable*, overriding the old *stable* distribution, which is removed at that time (although it can be found at `archive.debian.org`).

This development cycle is based on the assumption that the *unstable* distribution becomes *stable* after passing a period of being in *testing*. Even once a distribution is considered stable, a few bugs inevitably remain — that's why the stable distribution is updated every now and then. However, these updates are tested very carefully and have to be introduced into the archive individually to reduce the risk of introducing new bugs. You can find proposed additions to *stable* in the `proposed-updates` directory. Those packages in `proposed-updates` that pass muster are periodically moved as a batch into the stable distribution and the revision level of the stable distribution is incremented (e.g., '3.0' becomes '3.0r1', '2.2r4' becomes '2.2r5', and so forth).

Note that development under *unstable* continues during the freeze period, since the *unstable* distribution remains in place in parallel with *testing*.

Experimental

The *experimental* distribution is a special distribution. It is not a full distribution in the same sense as 'stable' and 'unstable' are. Instead, it is meant to be a temporary staging area for highly experimental software where there's a good chance that the software could break your system, or software that's just too unstable even for the *unstable* distribution (but there is a reason to package it nevertheless). Users who download and install packages from *experimental* are expected to have been duly warned. In short, all bets are off for the *experimental* distribution.

If there is a chance that the software could do grave damage to a system, it is likely to be better to put it into *experimental*. For instance, an experimental compressed file system should probably go into *experimental*.

Whenever there is a new upstream version of a package that introduces new features but breaks a lot of old ones, it should either not be uploaded, or be uploaded to *experimental*. A new, beta, version of some software which uses completely different configuration can go into *experimental*, at the maintainer's discretion. If you are working on an incompatible or complex upgrade situation, you can also use *experimental* as a staging area, so that testers can get early access.

Some experimental software can still go into *unstable*, with a few warnings in the description, but that isn't recommended because packages from *unstable* are expected to propagate to *testing* and thus to *stable*. You should not be afraid to use *experimental* since it does not cause any pain to the ftpmasters, the experimental packages are automatically removed once you upload the package in *unstable* with a higher version number.

New software which isn't likely to damage your system can go directly into *unstable*.

An alternative to *experimental* is to use your personal web space on `people.debian.org` (`klecker.debian.o`

4.7.5 Release code names

Every released Debian distribution has a *code name*: Debian 1.1 is called 'buzz'; Debian 1.2, 'rex'; Debian 1.3, 'bo'; Debian 2.0, 'hamm'; Debian 2.1, 'slink'; Debian 2.2, 'potato'; and Debian 3.0, 'woody'. There is also a "pseudo-distribution", called 'sid', which is the current 'unstable' distribution; since packages are moved from 'unstable' to 'testing' as they approach stability, 'sid' itself is never released. As well as the usual contents of a Debian distribution, 'sid' contains packages for architectures which are not yet officially supported or released by Debian. These architectures are planned to be integrated into the mainstream distribution at some future date.

Since Debian has an open development model (i.e., everyone can participate and follow the development) even the 'unstable' and 'testing' distributions are distributed to the Internet through the Debian FTP and HTTP server network. Thus, if we had called the directory which contains the release candidate version 'testing', then we would have to rename it to 'stable' when the version is released, which would cause all FTP mirrors to re-retrieve the whole distribution (which is quite large).

On the other hand, if we called the distribution directories *Debian-x.y* from the beginning, people would think that Debian release *x.y* is available. (This happened in the past, where a CD-ROM vendor built a Debian 1.0 CD-ROM based on a pre-1.0 development version. That's the reason why the first official Debian release was 1.1, and not 1.0.)

Thus, the names of the distribution directories in the archive are determined by their code names and not their release status (e.g., 'slink'). These names stay the same during the development period and after the release; symbolic links, which can be changed easily, indicate the currently released stable distribution. That's why the real distribution directories use the *code names*, while symbolic links for *stable*, *testing*, and *unstable* point to the appropriate release directories.

4.8 The Incoming system

The Incoming system is responsible of collecting updated packages and installing them in the Debian archive. It consists of a set of directories and scripts that are installed both on `ftp-master.debian.org` and `non-us.debian.org`.

Packages are uploaded by all the maintainers into an `unchecked` directory. This directory is scanned every 15 minutes by the katie script that verifies the integrity of the package and the cryptographic signature. If the package is considered ready to be installed, it is moved into an `accepted` directory. If it is the first upload of the package then it is moved in a `new` directory waiting an approval of the ftpmasters. If the package contains files to be installed "by-hand" is is moved in the byhand directory waiting a manual installation by the ftpmasters. Otherwise, if any error has been detected, the package is refused and is moved in the reject directory.

Once the package is accepted the system sends a confirmation mail to the maintainer, closes all the bugs marked as fixed by the upload and the auto-builders may start recompiling it. The package is now publicly accessible at http://incoming.debian.org (there is no such URL for packages in the non-US archive) until it is really installed in the Debian archive. This happens only once a day, the package is then removed from incoming and installed in the pool along with all the other packages. Once all the other updates (generating new Packages and Sources index files for example) have been made, a special script is called to ask all the primary mirrors to update themselves.

All debian developers have write access to the unchecked directory in order to upload their packages, they also have that access to the reject directory in order to remove their bad uploads or to move some files back in the unchecked directory. But all the other directories are only writable by the ftpmasters, that is why you can not remove an upload once it has been accepted.

4.8.1 Delayed incoming

The unchecked directory has a special DELAYED subdirectory. It is itself subdivided in nine directories called 1-day to 9-day. Packages which are uploaded in one of those directories will be moved in the real unchecked directory after the corresponding number of days. This is done by a script that is run each day and which moves the packages between the directories. Those which are in “1-day” are installed in unchecked while the others are moved in the adjacent directory (for example, a package in 5-day will be moved in 4-day). This feature is particularly useful for people who are doing non-maintainer uploads. Instead of waiting before uploading a NMU, it is uploaded as soon as it is ready but in one of those DELAYED/*x*-day directories. That leaves the corresponding number of days to the maintainer in order to react and upload himself another fix if he is not completely satisfied with the NMU. Alternatively he can remove the NMU by himself.

The use of that delayed feature can be simplified with a bit of integration with your upload tool. For instance, if you use dupload (see ‘dupload’ on page 61), you can add this snippet to your configuration file:

```
$delay = ($ENV{DELAY} || 7);
$cfg{’delayed’} = {
         fqdn => "ftp-master.debian.org",
         login => "yourdebianlogin",
         incoming => "/org/ftp.debian.org/incoming/DELAYED/$delay-day/",
         visibleuser => "yourdebianlogin",
         visiblename => "debian.org",
         fullname => "Your Full Name",
         dinstall_runs => 1,
          method => "scpb"
};
```

Once you've made that change, `dupload` can be used to easily upload a package in one of the delayed directories:

```
DELAY=5 dupload --to delayed <changes-file>
```

4.9 The testing scripts

The testing scripts are run each day after the installation of the updated packages. They generate the `Packages` files for the *testing* distribution, but they do so in an intelligent manner trying to avoid any inconsistency and trying to use only non-buggy packages.

The inclusion of a package from *unstable* is conditional on the following:

- The package must have been available in *unstable* for several days; the precise number depends on the upload's urgency field. It is 10 days for low urgency, 5 days for medium urgency and 2 days for high urgency. Those delays may be doubled during a freeze;
- It must have less release-critical bugs than the version available in *testing*;
- It must be available on all architectures on which it has been previously built. 'The `madison` utility' on the next page may be of interest to check that information;
- It must not break any dependency of a package that is already available in *testing*;
- The packages on which it depends must either be available in *testing* or they must be accepted into *testing* at the same time (and they will if they respect themselves all the criteria);

The scripts are generating some output files to explain why some packages are kept out of testing. They are available at `http://ftp-master.debian.org/testing/`. Alternatively, it is possible to use the `grep-excuses` program part of the `devscripts` package. It can be easily put in a crontab to keep someone informed of the progression of his packages in testing.

The `update_excuses` file does not always give the precise reason why the package is refused, one may have to find it by himself by looking what would break with the inclusion of the package. The testing FAQ (`http://people.debian.org/~jules/testingfaq.html`) gives some more information about the usual problems which may be causing such troubles.

Sometimes, some packages never enter testing because the set of inter-relationship is too complicated and can not be sorted out by the scripts. In that case, the release manager must be contacted, and he will force the inclusion of the packages.

4.10 Package's information

4.10.1 On the web

Each package has several dedicated web pages that contains many informations. `http://packages.debian.c` will display each version of the package available in the various distributions. The per-version detailed information includes the package description, the dependencies and links to download the package.

The bug tracking system sorts the bugs by package, you can watch the bugs of each package at `http://bugs.debian.org/`*`package-name`*.

4.10.2 The `madison` utility

`madison` is a command-line utility that is available on both `ftp-master.debian.org` and `non-us.debian.org`. It uses a single argument corresponding to a package name. In result it displays which version of the package is available for each architecture and distribution combination. An example will explain it better.

```
$ madison libdbd-mysql-perl
libdbd-mysql-perl |   1.2202-4 |        stable | source, alpha, arm, i386, m68k,
libdbd-mysql-perl |   1.2216-2 |       testing | source, arm, hppa, i386, ia64,
libdbd-mysql-perl | 1.2216-2.0.1 |       testing | alpha
libdbd-mysql-perl |   1.2219-1 |      unstable | source, alpha, arm, hppa, i386,
```

In this example, you can see that the version in unstable differs from the version in testing and that there has been a binary-only NMU of the package for the alpha architecture. Each time the package has been recompiled on most of the architectures.

4.11 The Package Tracking System

The Package Tracking System (PTS) is basically a tool to track by mail the activity of a source package. You just have to subscribe to a source package to start getting the mails related to it. You get the same mails than the maintainer. Each mail sent through the PTS is classified and associated to one of the keyword listed below. This will let you select the mails that you want to receive.

By default you will get:

`bts` All the bug reports and following discussions.

`bts-control` The control mails notifying a status change in one of the bugs.

`upload-source` The confirmation mail from `katie` when an uploaded source package is accepted.

`katie-other` Other warning and error mails from `katie` (like the override disparity for the section or priority field).

`default` Any non-automatic mail sent to the PTS by people who wanted to contact the subscribers of the package.

`summary` In the future, you may receive regular summary mails to keep you informed of the package's status (bug statistics, porting overview, progression in testing, . . .).

You can also decide to receive some more information:

`upload-binary` The confirmation mail from `katie` when an uploaded binary package is accepted (to check that your package is recompiled for all architectures).

`cvs` CVS commits if the maintainer has setup a system to forward commit notification to the PTS.

4.11.1 The PTS email interface

You can control your subscription(s) to the PTS by sending various commands to `<pts@qa.debian.org>`.

`subscribe <srcpackage> [<email>]` Subscribes *email* to communications related to the source package *srcpackage*. Sender address is used if the second argument is not present. If *srcpackage* is not a valid source package, you'll get a warning. However if it's a valid binary package, the PTS will subscribe you to the corresponding source package.

`unsubscribe <srcpackage> [<email>]` Removes a previous subscription to the source package *srcpackage* using the specified email address or the sender address if the second argument is left out.

`which [<email>]` Lists all subscriptions for the sender or the email address optionally specified.

`keyword [<email>]` Tells you the keywords that you are accepting. Each mail sent through the Package Tracking System is associated to a keyword and you receive only the mails associated to keywords that you are accepting. Here is the list of available keywords:

- `bts`: mails coming from the Debian Bug Tracking System

- `bts-control`: reply to mails sent to `<control@bugs.debian.org>`
- `summary`: automatic summary mails about the state of a package
- `cvs`: notification of CVS commits
- `upload-source`: announce of a new source upload that has been accepted
- `upload-binary`: announce of a new binary-only upload (porting)
- `katie-other`: other mails from ftpmasters (override disparity, etc.)
- `default`: all the other mails (those which aren't "automatic")

`keyword <srcpackage> [<email>]` Same as previous item but for the given source package since you may select a different set of keywords for each source package.

`keyword [<email>] {+|-|=} <list of keywords>` Accept (+) or refuse (-) mails associated to the given keyword(s). Define the list (=) of accepted keywords.

`keyword <srcpackage> [<email>] {+|-|=} <list of keywords>` Same as previous item but overrides the keywords list for the indicated source package.

`quit | thanks | --` Stops processing commands. All following lines are ignored by the bot.

4.11.2 Filtering PTS mails

Once you are subscribed to a package, you will get the mails sent to `*srcpackage*@packages.qa.debian.org`. Those mails have special headers appended to let you filter them in a special mailbox with `procmail`. The added headers are `X-Loop`, `X-PTS-Package`, `X-PTS-Keyword` and `X-Unsubscribe`.

Here is an example of added headers for a source upload notification on the `dpkg` package:

```
X-Loop: dpkg@packages.qa.debian.org
X-PTS-Package: dpkg
X-PTS-Keyword: upload-source
X-Unsubscribe: echo 'unsubscribe dpkg' | mail pts@qa.debian.org
```

4.11.3 Forwarding CVS commits in the PTS

If you use a publicly accessible CVS repository for maintaining your Debian package you may want to forward the commit notification to the PTS so that the subscribers (possible co-maintainers) can closely follow the package's evolution.

It's very easy to setup. Once your CVS repository generates commit notifications, you just have to make sure it sends a copy of those mails to `*srcpackage*_cvs@packages.qa.debian.org`. Only people who accepts the *cvs* keyword will receive the notifications.

Chapter 5

Managing Packages

This chapter contains information related to creating, uploading, maintaining, and porting packages.

5.1 Package uploads

5.1.1 New packages

If you want to create a new package for the Debian distribution, you should first check the Work-Needing and Prospective Packages (WNPP) (`http://www.debian.org/devel/wnpp/`) list. Checking the WNPP list ensures that no one is already working on packaging that software, and that effort is not duplicated. Read the WNPP web pages (`http://www.debian.org/devel/wnpp/`) for more information.

Assuming no one else is already working on your prospective package, you must then submit a bug report ('Bug Reporting' on page 55) against the pseudo-package `wnpp` describing your plan to create a new package, including, but not limiting yourself to, a description of the package, the license of the prospective package and the current URL where it can be downloaded from.

You should set the subject of the bug to "ITP: *foo – short description*", substituting the name of the new package for *foo*. The severity of the bug report must be set to *wishlist*. If you feel it's necessary, send a copy to `<debian-devel@lists.debian.org>` by putting the address in the `X-Debbugs-CC:` header of the message (no, don't use `CC:`, because that way the message's subject won't indicate the bug number).

Please include a `Closes: bug#`*`nnnnn`* entry on the changelog of the new package in order for the bug report to be automatically closed once the new package is installed on the archive ('When bugs are closed by new uploads' on page 49).

There are a number of reasons why we ask maintainers to announce their intentions:

- It helps the (potentially new) maintainer to tap into the experience of people on the list, and lets them know if anyone else is working on it already.
- It lets other people thinking about working on the package know that there already is a volunteer, so efforts may be shared.
- It lets the rest of the maintainers know more about the package than the one line description and the usual changelog entry "Initial release" that gets posted to `debian-devel-changes`.
- It is helpful to the people who live off unstable (and form our first line of testers). We should encourage these people.
- The announcements give maintainers and other interested parties a better feel of what is going on, and what is new, in the project.

5.1.2 Adding an entry to `debian/changelog`

Changes that you make to the package need to be recorded in the `debian/changelog`. These changes should provide a concise description of what was changed, why (if it's in doubt), and note if any bugs were closed. They also record when the package was completed. This file will be installed in `/usr/share/doc/`*`package`*`/changelog.Debian.gz`, or `/usr/share/doc/`*`package`*`/changelog.gz` for native packages.

The `debian/changelog` file conforms to a certain structure, with a number of different fields. One field of note, the *distribution*, is described in 'Picking a distribution' on page 30. More information about the structure of this file can be found in the Debian Policy section titled "`debian/changelog`".

Changelog entries can be used to automatically close Debian bugs when the package is installed into the archive. See 'When bugs are closed by new uploads' on page 49.

It is conventional that the changelog entry notating of a package that contains a new upstream version of the software looks like this:

```
  * new upstream version
```

There are tools to help you create entries and finalize the `changelog` for release — see '`devscripts`' on page 62 and '`dpkg-dev-el`' on page 62.

5.1.3 Checking the package prior to upload

Before you upload your package, you should do basic testing on it. At a minimum, you should try the following activities (you'll need to have an older version of the same Debian package around):

- Install the package and make sure the software works, or upgrade the package from an older version to your new version if a Debian package for it already exists.
- Run `lintian` over the package. You can run `lintian` as follows: `lintian -v` *`package-version`*`.cha` This will check the source package as well as the binary package. If you don't understand the output that `lintian` generates, try adding the `-i` switch, which will cause `lintian` to output a very verbose description of the problem.

  Normally, a package should *not* be uploaded if it causes lintian to emit errors (they will start with `E`).

  For more information on `lintian`, see '`lintian`' on page 59.
- Downgrade the package to the previous version (if one exists) — this tests the `postrm` and `prerm` scripts.
- Remove the package, then reinstall it.

5.1.4 Generating the changes file

When a package is uploaded to the Debian FTP archive, it must be accompanied by a `.changes` file, which gives directions to the archive maintainers for its handling. This is usually generated by `dpkg-genchanges` during the normal package build process.

The changes file is a control file with the following fields:
- `Format`
- `Date`
- `Source`
- `Binary`
- `Architecture`
- `Version`
- `Distribution`
- `Urgency`
- `Maintainer`
- `Description`
- `Changes`
- `Files`

All of these fields are mandatory for a Debian upload. See the list of control fields in the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`) for the contents of these fields. You can close bugs automatically using the `Description` field, see 'When bugs are closed by new uploads' on page 49.

The original source tarball

The first time a version is uploaded which corresponds to a particular upstream version, the original source tar file should be uploaded and included in the `.changes` file. Subsequently, this very same tar file should be used to build the new diffs and `.dsc` files, and will not need to be re-uploaded.

By default, `dpkg-genchanges` and `dpkg-buildpackage` will include the original source tar file if and only if the Debian revision part of the source version number is 0 or 1, indicating a new upstream version. This behavior may be modified by using `-sa` to always include it or `-sd` to always leave it out.

If no original source is included in the upload, the original source tar-file used by `dpkg-source` when constructing the `.dsc` file and diff to be uploaded *must* be byte-for-byte identical with the one already in the archive.

Picking a distribution

The `Distribution` field, which originates from the first line of the `debian/changelog` file, indicates which distribution the package is intended for.

There are three possible values for this field: 'stable', 'unstable', and 'experimental'. Normally, packages are uploaded into *unstable*.

You should avoid combining 'stable' with others because of potential problems with library dependencies (for your package and for the package built by the build daemons for other architecture). See 'Uploading to *stable*' on this page for more information on when and how to upload to *stable*.

It never makes sense to combine the *experimental* distribution with anything else.

Uploading to *stable* Uploading to *stable* means that the package will be placed into the `proposed-updates` directory of the Debian archive for further testing before it is actually included in *stable*.

Extra care should be taken when uploading to *stable*. Basically, a package should only be uploaded to stable if one of the following happens:

- a security problem (e.g. a Debian security advisory)
- a truly critical functionality problem
- the package becomes uninstallable
- a released architecture lacks the package

It is discouraged to change anything else in the package that isn't important, because even trivial fixes can cause bugs later on. Uploading new upstream versions to fix security problems is deprecated; applying the specific patch from the new upstream version to the old one ("back-porting" the patch) is the right thing to do in most cases.

Packages uploaded to *stable* need to be compiled on systems running *stable*, so that their dependencies are limited to the libraries (and other packages) available in *stable*; for example, a package uploaded to *stable* that depends on a library package that only exists in unstable will be rejected. Making changes to dependencies of other packages (by messing with `Provides` or shlibs files), possibly making those other packages uninstallable, is strongly discouraged.

The Release Team (which can be reached at `<debian-release@lists.debian.org>`) will regularly evaluate the uploads in *proposed-updates* and decide if your package can be included in *stable*. Please be clear (and verbose, if necessary) in your changelog entries for uploads to *stable*, because otherwise the package won't be considered for inclusion.

5.1.5 Uploading a package

Uploading to `ftp-master`

To upload a package, you need a personal account on `ftp-master.debian.org`, which you should have as an official maintainer. If you use `scp` or `rsync` to transfer the files, place them into `/org/ftp.debian.org/incoming/`; if you use anonymous FTP to upload, place them into `/pub/UploadQueue/`. Please note that you should transfer the changes file last. Otherwise, your upload may be rejected because the archive maintenance software will parse the changes file and see that not all files have been uploaded. If you don't want to bother with transferring the changes file last, you can simply copy your files to a temporary directory on `ftp-master` and then move them to `/org/ftp.debian.org/incoming/`.

Note: Do not upload to `ftp-master` cryptographic packages which belong to *contrib* or *non-free*. Uploads of such software should go to `non-us` (see 'Uploading to `non-US` (pandora)' on the next page). Furthermore packages containing code that is patent-restricted by the United States government can not be uploaded to `ftp-master`; depending on the case they may still be uploaded to `non-US/non-free` (it's in non-free because of distribution issues and not because of the license of the software). If you can't upload it to `ftp-master`, then neither can you upload it to the overseas upload queues on `chiark` or `erlangen`. If you are not sure whether U.S. patent controls or cryptographic controls apply to your package, post a message to `<debian-devel@lists.debian.org>` and ask.

You may also find the Debian packages '`dupload`' on page 61 or '`dput`' on page 61 useful when uploading packages. These handy programs help automate the process of uploading packages into Debian.

After uploading your package, you can check how the archive maintenance software will process it by running `dinstall` on your changes file:

```
    dinstall -n foo.changes
```

. Note that `dput` can do this for you automatically.

Uploading to **`non-US`** (pandora)

As discussed above, export controlled software should not be uploaded to `ftp-master`. Instead, upload the package to `non-us.debian.org`, placing the files in `/org/non-us.debian.org /incoming/` (again, both '`dupload`' on page 61 and '`dput`' on page 61 can do this for you if invoked properly). By default, you can use the same account/password that works on `ftp-master`. If you use anonymous FTP to upload, place the files into `/pub/UploadQueue/`.

You can check your upload the same way it's done on `ftp-master`, with:

```
    dinstall -n foo.changes
```

Note that U.S. residents or citizens are subject to restrictions on export of cryptographic software. As of this writing, U.S. citizens are allowed to export some cryptographic software, subject to notification rules by the U.S. Department of Commerce. However, this restriction has been waived for software which is already available outside the U.S. Therefore, any cryptographic software which belongs in the *main* section of the Debian archive and does not depend on any package outside of *main* (e.g., does not depend on anything in *non-US/main*) can be uploaded to `ftp-master` or its queues, described above.

Debian policy does not prevent upload to non-US by U.S. residents or citizens, but care should be taken in doing so. It is recommended that developers take all necessary steps to ensure that they are not breaking current US law by doing an upload to non-US, *including consulting a lawyer*.

For packages in *non-US/main*, *non-US/contrib*, developers should at least follow the procedure outlined by the US Government (`http://www.bxa.doc.gov/Encryption/PubAvailEncSourceCodeNofify html`). Maintainers of *non-US/non-free* packages should further consult the rules on notification of export (`http://www.bxa.doc.gov/Encryption/`) of non-free software.

This section is for information only and does not constitute legal advice. Again, it is strongly recommended that U.S. citizens and residents consult a lawyer before doing uploads to non-US.

Uploads via `chiark`

If you have a slow network connection to `ftp-master`, there are alternatives. One is to upload files to `Incoming` via a upload queue in Europe on `chiark`. For details connect to `ftp://ftp.chiark.greenend.org.uk/pub/debian/private/project/README.how-to-upload`.

Note: Do not upload packages containing software that is export-controlled by the United States government to the queue on `chiark`. Since this upload queue goes to `ftp-master`, the prescription found in 'Uploading to `ftp-master`' on page 31 applies here as well.

The program `dupload` comes with support for uploading to `chiark`; please refer to the documentation that comes with the program for details.

Uploads via `erlangen`

Another upload queue is available in Germany: just upload the files via anonymous FTP to `ftp://ftp.uni-erlangen.de/pub/Linux/debian/UploadQueue/`.

The upload must be a complete Debian upload, as you would put it into `ftp-master`'s `Incoming`, i.e., a `.changes` files along with the other files mentioned in the `.changes`. The queue daemon also checks that the `.changes` is correctly signed with GnuPG or OpenPGP by a Debian developer, so that no bogus files can find their way to `ftp-master` via this queue. Please also make sure that the `Maintainer` field in the `.changes` contains *your* e-mail address. The address found there is used for all replies, just as on `ftp-master`.

There's no need to move your files into a second directory after the upload, as on `chiark`. And, in any case, you should get a mail reply from the queue daemon explaining what happened to your upload. Hopefully it should have been moved to `ftp-master`, but in case of errors you're notified, too.

Note: Do not upload packages containing software that is export-controlled by the United States government to the queue on `erlangen`. Since this upload queue goes to `ftp-master`, the prescription found in 'Uploading to `ftp-master`' on page 31 applies here as well.

The program `dupload` comes with support for uploading to `erlangen`; please refer to the documentation that comes with the program for details.

Other upload queues

Another upload queue is available which is based in the US, and is a good backup when there are problems reaching `ftp-master`. You can upload files, just as in `erlangen`, to `ftp://samosa.debian.org/pub/UploadQueue/`.

An upload queue is available in Japan: just upload the files via anonymous FTP to `ftp://master.debian.or.jp/pub/Incoming/upload/`.

5.1.6 Announcing package uploads

When a package is uploaded, an announcement should be posted to one of the “debian-changes” lists. This is now done automatically by the archive maintenance software when it runs (usually once a day). You just need to use a recent `dpkg-dev` (>= 1.4.1.2). The mail generated by the archive maintenance software will contain the OpenPGP/GnuPG signed `.changes` files that you uploaded with your package. Previously, `dupload` used to send those announcements, so please make sure that you configured your `dupload` not to send those announcements (check its documentation and look for “dinstall_runs”).

If a package is released with the `Distribution:` set to ‘stable’, the announcement is sent to `<debian-changes@lists.debian.org>`. If a package is released with `Distribution:` set to ‘unstable’, or ‘experimental’, the announcement will be posted to `<debian-devel-changes@lists.debian.org>` instead.

5.1.7 Notification that a new package has been installed

The Debian archive maintainers are responsible for handling package uploads. For the most part, uploads are automatically handled on a daily basis by the archive maintenance tools, `katie`. Specifically, updates to existing packages to the ‘unstable’ distribution are handled automatically. In other cases, notably new packages, placing the uploaded package into the distribution is handled manually. When uploads are handled manually, the change to the archive may take up to a month to occur. Please be patient.

In any case, you will receive an email notification indicating that the package has been added to the archive, which also indicates which bugs will be closed by the upload. Please examine this notification carefully, checking if any bugs you meant to close didn’t get triggered.

The installation notification also includes information on what section the package was inserted into. If there is a disparity, you will receive a separate email notifying you of that. Read on below.

The override file

The `debian/control` file’s `Section` and `Priority` fields do not actually specify where the file will be placed in the archive, nor its priority. In order to retain the overall integrity of the archive, it is the archive maintainers who have control over these fields. The values in the `debian/control` file are actually just hints.

The archive maintainers keep track of the canonical sections and priorities for packages in the *override file*. If there is a disparity between the *override file* and the package’s fields as indicated in `debian/control`, then you will receive an email noting the divergence when the package is installed into the archive. You can either correct your `debian/control` file for your next upload, or else you may wish to make a change in the *override file*.

To alter the actual section that a package is put in, you need to first make sure that the `debian/control` in your package is accurate. Next, send an email `<override-change@debian.org>` or submit a bug against `ftp.debian.org` requesting that the section or priority for your package be changed from the old section or priority to the new one. Be sure to explain your reasoning.

For more information about *override files*, see `dpkg-scanpackages(8)`, `/usr/share/doc/debian/bug-log-mailserver.txt`, and `/usr/share/doc/debian/bug-maint-info.txt`.

5.2 Non-Maintainer Uploads (NMUs)

Under certain circumstances it is necessary for someone other than the official package maintainer to make a release of a package. This is called a non-maintainer upload, or NMU.

Debian porters, who compile packages for different architectures, occasionally do binary-only NMUs as part of their porting activity (see 'Porting and Being Ported' on page 39). Another reason why NMUs are done is when a Debian developers needs to fix another developers' packages in order to address serious security problems or crippling bugs, especially during the freeze, or when the package maintainer is unable to release a fix in a timely fashion.

This chapter contains information providing guidelines for when and how NMUs should be done. A fundamental distinction is made between source and binary-only NMUs, which is explained in the next section.

5.2.1 Terminology

There are two new terms used throughout this section: "binary-only NMU" and "source NMU". These terms are used with specific technical meaning throughout this document. Both binary-only and source NMUs are similar, since they involve an upload of a package by a developer who is not the official maintainer of that package. That is why it's a *non-maintainer* upload.

A source NMU is an upload of a package by a developer who is not the official maintainer, for the purposes of fixing a bug in the package. Source NMUs always involves changes to the source (even if it is just a change to `debian/changelog`). This can be either a change to the upstream source, or a change to the Debian bits of the source. Note, however, that source NMUs may also include architecture-dependent packages, as well as an updated Debian diff.

A binary-only NMU is a recompilation and upload of a binary package for a given architecture. As such, it is usually part of a porting effort. A binary-only NMU is a non-maintainer uploaded binary version of a package, with no source changes required. There are many cases where porters must fix problems in the source in order to get them to compile for their target architecture; that would be considered a source NMU rather than a binary-only NMU. As you can see, we don't distinguish in terminology between porter NMUs and non-porter NMUs.

Both classes of NMUs, source and binary-only, can be lumped by the term “NMU”. However, this often leads to confusion, since most people think “source NMU” when they think “NMU”. So it’s best to be careful. In this chapter, if we use the unqualified term “NMU”, we refer to any type of non-maintainer upload NMUs, whether source and binary, or binary-only.

5.2.2 Who can do an NMU

Only official, registered Debian maintainers can do binary or source NMUs. An official maintainer is someone who has their key in the Debian key ring. Non-developers, however, are encouraged to download the source package and start hacking on it to fix problems; however, rather than doing an NMU, they should just submit worthwhile patches to the Bug Tracking System. Maintainers almost always appreciate quality patches and bug reports.

5.2.3 When to do a source NMU

Guidelines for when to do a source NMU depend on the target distribution, i.e., stable, unstable, or experimental. Porters have slightly different rules than non-porters, due to their unique circumstances (see ‘When to do a source NMU if you are a porter’ on page 42).

When a security bug is detected, a fixed package should be uploaded as soon as possible. In this case, the Debian security officers get in contact with the package maintainer to make sure a fixed package is uploaded within a reasonable time (less than 48 hours). If the package maintainer cannot provide a fixed package fast enough or if he/she cannot be reached in time, a security officer may upload a fixed package (i.e., do a source NMU).

During the release cycle (see ‘Stable, testing, and unstable’ on page 19), NMUs which fix serious or higher severity bugs are encouraged and accepted. Even during this window, however, you should endeavor to reach the current maintainer of the package; they might be just about to upload a fix for the problem. As with any source NMU, the guidelines found in ‘How to do a source NMU’ on the next page need to be followed.

Bug fixes to unstable by non-maintainers are also acceptable, but only as a last resort or with permission. The following protocol should be respected to do an NMU:

- Make sure that the package’s bug is in the Debian Bug Tracking System (BTS). If not, submit a bug.
- Wait a few days the response from the maintainer. If you don’t get any response, you may want to help him by sending the patch that fixes the bug. Don’t forget to tag the bug with the “patch” keyword.
- Wait a few more days. If you still haven’t got an answer from the maintainer, send him a mail announcing your intent to NMU the package. Prepare an NMU as described in ‘How to

do a source NMU' on this page, test it carefully on your machine (cf. 'Checking the package prior to upload' on page 28). Double check that your patch doesn't have any unexpected side effects. Make sure your patch is as small and as non-disruptive as it can be.

- Upload your package to incoming in `DELAYED/7-day` (cf. 'Delayed incoming' on page 22), send the final patch to the maintainer via the BTS, and explain him that he has 7 days to react if he wants to cancel the NMU.
- Follow what happens, you're responsible for any bug that you introduced with your NMU. You should probably use 'The Package Tracking System' on page 24 (PTS) to stay informed of the state of the package after your NMU.

5.2.4 How to do a source NMU

The following applies to porters insofar as they are playing the dual role of being both package bug-fixers and package porters. If a porter has to change the Debian source archive, automatically their upload is a source NMU and is subject to its rules. If a porter is simply uploading a recompiled binary package, the rules are different; see 'Guidelines for porter uploads' on page 41.

First and foremost, it is critical that NMU patches to source should be as non-disruptive as possible. Do not do housekeeping tasks, do not change the name of modules or files, do not move directories; in general, do not fix things which are not broken. Keep the patch as small as possible. If things bother you aesthetically, talk to the Debian maintainer, talk to the upstream maintainer, or submit a bug. However, aesthetic changes must *not* be made in a non-maintainer upload.

Source NMU version numbering

Whenever you have made a change to a package, no matter how trivial, the version number needs to change. This enables our packing system to function.

If you are doing a non-maintainer upload (NMU), you should add a new minor version number to the *debian-revision* part of the version number (the portion after the last hyphen). This extra minor number will start at '1'. For example, consider the package 'foo', which is at version 1.1-3. In the archive, the source package control file would be `foo_1.1-3.dsc`. The upstream version is '1.1' and the Debian revision is '3'. The next NMU would add a new minor number '.1' to the Debian revision; the new source control file would be `foo_1.1-3.1.dsc`.

The Debian revision minor number is needed to avoid stealing one of the package maintainer's version numbers, which might disrupt their work. It also has the benefit of making it visually clear that a package in the archive was not made by the official maintainer.

If there is no *debian-revision* component in the version number then one should be created, starting at '0.1'. If it is absolutely necessary for someone other than the usual maintainer to make a release based on a new upstream version then the person making the release should start with the *debian-revision* value '0.1'. The usual maintainer of a package should start their *debian-revision* numbering at '1'. Note that if you do this, you'll have to invoke `dpkg-buildpackage` with the `-sa` switch to force the build system to pick up the new source package (normally it only looks for Debian revisions of '0' or '1' — it's not yet clever enough to know about '0.1').

Source NMUs must have a new changelog entry

A non-maintainer doing a source NMU must create a changelog entry, describing which bugs are fixed by the NMU, and generally why the NMU was required and what it fixed. The changelog entry will have the non-maintainer's email address in the log entry and the NMU version number in it.

By convention, source NMU changelog entries start with the line

```
    * Non-maintainer upload
```

Source NMUs and the Bug Tracking System

Maintainers other than the official package maintainer should make as few changes to the package as possible, and they should always send a patch as a unified context diff (`diff -u`) detailing their changes to the Bug Tracking System.

What if you are simply recompiling the package? If you just need to recompile it for a single architecture, then you may do a binary-only NMU as described in 'Recompilation or binary-only NMU' on page 41 which doesn't require any patch to be sent. If you want the package to be recompiled for all architectures, then you do a source NMU as usual and you will have to send a patch.

If the source NMU (non-maintainer upload) fixes some existing bugs, these bugs should be tagged *fixed* in the Bug Tracking System rather than closed. By convention, only the official package maintainer or the original bug submitter are allowed to close bugs. Fortunately, Debian's archive system recognizes NMUs and thus marks the bugs fixed in the NMU appropriately if the person doing the NMU has listed all bugs in the changelog with the `Closes: bug#`*`nnnnn`* syntax (see 'When bugs are closed by new uploads' on page 49 for more information describing how to close bugs via the changelog). Tagging the bugs *fixed* ensures that everyone knows that the bug was fixed in an NMU; however the bug is left open until the changes in the NMU are incorporated officially into the package by the official package maintainer.

Also, after doing an NMU, you have to open a new bug and include a patch showing all the changes you have made. Alternatively you can send that information to the existing bugs that are fixed by your NMU. The normal maintainer will either apply the patch or employ an alternate method of fixing the problem. Sometimes bugs are fixed independently upstream, which is another good reason to back out an NMU's patch. If the maintainer decides not to apply the NMU's patch but to release a new version, the maintainer needs to ensure that the new upstream version really fixes each problem that was fixed in the non-maintainer release.

In addition, the normal maintainer should *always* retain the entry in the changelog file documenting the non-maintainer upload.

Building source NMUs

Source NMU packages are built normally. Pick a distribution using the same rules as found in 'Picking a distribution' on page 30. Just as described in 'Uploading a package' on page 31, a normal changes file, etc., will be built. In fact, all the prescriptions from 'Package uploads' on page 27 apply.

Make sure you do *not* change the value of the maintainer in the `debian/control` file. Your name as given in the NMU entry of the `debian/changelog` file will be used for signing the changes file.

5.2.5 Acknowledging an NMU

If one of your packages has been NMUed, you have to incorporate the changes in your copy of the sources. This is easy, you just have to apply the patch that has been sent to you. Once this is done, you have to close the bugs that have been tagged fixed by the NMU. You can either close them manually by sending the required mails to the BTS or by adding the required `closes: #nnnn` in the changelog entry of your next upload.

In any case, you should not be upset by the NMU. An NMU is not a personal attack against the maintainer. It is just the proof that someone cares enough about the package and was willing to help you in your work. You should be thankful to him and you may want to ask him if he would be interested to help you on a more frequent basis as co-maintainer or backup maintainer (see 'Collaborative maintenance' on page 43).

5.3 Porting and Being Ported

Debian supports an ever-increasing number of architectures. Even if you are not a porter, and you don't use any architecture but one, it is part of your duty as a maintainer to be aware of issues of portability. Therefore, even if you are not a porter, you should read most of this chapter.

Porting is the act of building Debian packages for architectures that is different from the original architecture of the package maintainer's binary package. It is a unique and essential activity. In fact, porters do most of the actual compiling of Debian packages. For instance, for a single *i386* binary package, there must be a recompile for each architecture, which amounts to 12 more builds.

5.3.1 Being kind to porters

Porters have a difficult and unique task, since they are required to deal with a large volume of packages. Ideally, every source package should build right out of the box. Unfortunately, this is often not the case. This section contains a checklist of "gotchas" often committed by Debian maintainers — common problems which often stymie porters, and make their jobs unnecessarily difficult.

The first and most important watchword is to respond quickly to bug or issues raised by porters. Please treat porters with courtesy, as if they were in fact co-maintainers of your package (which in a way, they are). Please be tolerant of succinct or even unclear bug reports, doing your best to hunt down whatever the problem is.

By far, most of the problems encountered by porters are caused by *packaging bugs* in the source packages. Here is a checklist of things you should check or be aware of.

1. Make sure that your `Build-Depends` and `Build-Depends-Indep` settings in `debian/control` are set properly. The best way to validate this is to use the `debootstrap` package to create an unstable chroot environment. Within that chrooted environment, install the `build-essential` package and any package dependencies mentioned in `Build-Depends` and/or `Build-Depends-Indep`. Finally, try building your package within that chrooted environment. These steps can be automated by the use of the `pbuilder` program which is provided by the package of the same name.

   See the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`) for instructions on setting build dependencies.

2. Don't set architecture to a value other than "all" or "any" unless you really mean it. In too many cases, maintainers don't follow the instructions in the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`). Setting your architecture to "i386" is usually incorrect.

3. Make sure your source package is correct. Do `dpkg-source -x` ***`package`***`.dsc` to make sure your source package unpacks properly. Then, in there, try building your package from scratch with `dpkg-buildpackage`.

4. Make sure you don't ship your source package with the `debian/files` or `debian/substvars` files. They should be removed by the 'clean' target of `debian/rules`.

5. Make sure you don't rely on locally installed or hacked configurations or programs. For instance, you should never be calling programs in `/usr/local/bin` or the like. Try not to

rely on programs be setup in a special way. Try building your package on another machine, even if it's the same architecture.

6. Don't depend on the package you're building already being installed (a sub-case of the above issue).

7. Don't rely on the compiler being a certain version, if possible. If not, then make sure your build dependencies reflect the restrictions, although you are probably asking for trouble, since different architectures sometimes standardize on different compilers.

8. Make sure your debian/rules contains separate "binary-arch" and "binary-indep" targets, as the Debian Policy Manual requires. Make sure that both targets work independently, that is, that you can call the target without having called the other before. To test this, try to run `dpkg-buildpackage -b`.

5.3.2 Guidelines for porter uploads

If the package builds out of the box for the architecture to be ported to, you are in luck and your job is easy. This section applies to that case; it describes how to build and upload your binary package so that it is properly installed into the archive. If you do have to patch the package in order to get it to compile for the other architecture, you are actually doing a source NMU, so consult 'How to do a source NMU' on page 37 instead.

For a porter upload, no changes are being made to the source. You do not need to touch any of the files in the source package. This includes `debian/changelog`.

The way to invoke `dpkg-buildpackage` is as `dpkg-buildpackage -B -m`*`porter-email`*. Of course, set *porter-email* to your email address. This will do a binary-only build of only the architecture-dependent portions of the package, using the 'binary-arch' target in `debian/rules`.

Recompilation or binary-only NMU

Sometimes the initial porter upload is problematic because the environment in which the package was built was not good enough (outdated or obsolete library, bad compiler, ...). Then you may just need to recompile it in an updated environment. However, you have to bump the version number in this case, so that the old bad package can be replaced in the Debian archive (`katie` refuses to install new packages if they don't have a version number greater than the currently available one). Despite the required modification of the changelog, these are called binary-only NMUs — there is no need in this case to trigger all other architectures to consider themselves out of date or requiring recompilation.

Such recompilations require special "magic" version numbering, so that the archive maintenance tools recognize that, even though there is a new Debian version, there is no corresponding source update. If you get this wrong, the archive maintainers will reject your upload (due to lack of corresponding source code).

The "magic" for a recompilation-only NMU is triggered by using the third-level number on the Debian part of the version. For instance, if the latest version you are recompiling against was version "2.9-3", your NMU should carry a version of "2.9-3.0.1". If the latest version was "3.4-2.1", your NMU should have a version number of "3.4-2.1.1".

When to do a source NMU if you are a porter

Porters doing a source NMU generally follow the guidelines found in 'Non-Maintainer Uploads (NMUs)' on page 35, just like non-porters. However, it is expected that the wait cycle for a porter's source NMU is smaller than for a non-porter, since porters have to cope with a large quantity of packages. Again, the situation varies depending on the distribution they are uploading to.

However, if you are a porter doing an NMU for 'unstable', the above guidelines for porting should be followed, with two variations. Firstly, the acceptable waiting period — the time between when the bug is submitted to the BTS and when it is OK to do an NMU — is seven days for porters working on the unstable distribution. This period can be shortened if the problem is critical and imposes hardship on the porting effort, at the discretion of the porter group. (Remember, none of this is Policy, just mutually agreed upon guidelines.)

Secondly, porters doing source NMUs should make sure that the bug they submit to the BTS should be of severity 'serious' or greater. This ensures that a single source package can be used to compile every supported Debian architecture by release time. It is very important that we have one version of the binary and source package for all architecture in order to comply with many licenses.

Porters should try to avoid patches which simply kludge around bugs in the current version of the compile environment, kernel, or libc. Sometimes such kludges can't be helped. If you have to kludge around compilers bugs and the like, make sure you `#ifdef` your work properly; also, document your kludge so that people know to remove it once the external problems have been fixed.

Porters may also have an unofficial location where they can put the results of their work during the waiting period. This helps others running the port have the benefit of the porter's work, even during the waiting period. Of course, such locations have no official blessing or status, so buyer, beware.

5.3.3 Tools for porters

There are several tools available for the porting effort. This section contains a brief introduction to these tools; see the package documentation or references for full information.

quinn-diff

`quinn-diff` is used to locate the differences from one architecture to another. For instance, it could tell you which packages need to be ported for architecture *Y*, based on architecture *X*.

buildd

The `buildd` system is used as a distributed, client-server build distribution system. It is usually used in conjunction with *auto-builders*, which are “slave” hosts which simply check out and attempt to auto-build packages which need to be ported. There is also an email interface to the system, which allows porters to “check out” a source package (usually one which cannot yet be auto-built) and work on it.

`buildd` is not yet available as a package; however, most porting efforts are either using it currently or planning to use it in the near future. It collects a number of as yet unpackaged components which are currently very useful and in use continually, such as `andrea`, `sbuild` and `wanna-build`.

Some of the data produced by `buildd` which is generally useful to porters is available on the web at `http://buildd.debian.org/`. This data includes nightly updated information from `andrea` (source dependencies) and `quinn-diff` (packages needing recompilation).

We are very excited about this system, since it potentially has so many uses. Independent development groups can use the system for different sub-flavors of Debian, which may or may not really be of general interest (for instance, a flavor of Debian built with gcc bounds checking). It will also enable Debian to recompile entire distributions quickly.

dpkg-cross

`dpkg-cross` is a tool for installing libraries and headers for cross-compiling in a way similar to `dpkg`. Furthermore, the functionality of `dpkg-buildpackage` and `dpkg-shlibdeps` is enhanced to support cross-compiling.

5.4 Collaborative maintenance

“Collaborative maintenance” is a term describing the sharing of Debian package maintenance duties by several people. This collaboration is almost a good idea, since it generally results in higher quality and faster bug fix turnaround time. It is strongly recommended that packages in which a priority of `Standard` or which are part of the base set have co-maintainers.

Generally there is a primary maintainer and one or more co-maintainers. The primary maintainer is the whose name is listed in the `Maintainer` field of the `debian/control` file. Co-maintainers are all the other maintainers.

In its most basic form, the process of adding a new co-maintainer is quite easy:

- Setup the co-maintainer with access to the sources you build the package from. Generally this implies you are using a network-capable version control system, such as `CVS` or `Subversion`.
- Add the co-maintainer's correct maintainer name and address to the `Uploaders` field in the global part of the `debian/control` file.
- Using the PTS ('The Package Tracking System' on page 24), the co-maintainers should subscribe themselves to the appropriate source package.

5.5 Moving, Removing, Renaming, Adopting, and Orphaning Packages

Some archive manipulation operations are not automated in the Debian upload process. These procedures should be manually followed by maintainers. This chapter gives guidelines in what to do in these cases.

5.5.1 Moving packages

Sometimes a package will change its section. For instance, a package from the 'non-free' section might be GPL'd in a later version, in which case, the package should be moved to 'main' or 'contrib'.[1]

If you need to change the section for one of your packages, change the package control information to place the package in the desired section, and re-upload the package (see the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`) for details). If your new section is valid, it will be moved automatically. If it does not, then contact the ftpmasters in order to understand what happened.

If, on the other hand, you need to change the *subsection* of one of your packages (e.g., "devel", "admin"), the procedure is slightly different. Correct the subsection as found in the control file of the package, and re-upload that. Also, you'll need to get the override file updated, as described in 'The override file' on page 34.

[1]See the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`) for guidelines on what section a package belongs in.

5.5.2 Removing packages

If for some reason you want to completely remove a package (say, if it is an old compatibility library which is no longer required), you need to file a bug against `ftp.debian.org` asking that the package be removed. Make sure you indicate which distribution the package should be removed from. Normally, you can only have packages removed from *unstable* and *experimental*. Packages are not removed from *testing* directly. Rather, they will be removed automatically after the package has been removed from *unstable* and no package in *testing* depends on it.

You also have to detail the reasons justifying that request. This is to avoid unwanted removals and to keep a trace of why a package has been removed. For example, you can provide the name of the package that supersedes the one to be removed.

Usually you only ask the removal of a package maintained by yourself. If you want to remove another package, you have to get the approval of its last maintainer.

If in doubt concerning whether a package is disposable, email `<debian-devel@lists.debian.org>` asking for opinions. Also of interest is the `apt-cache` program from the `apt` package. When invoked as `apt-cache showpkg` *`package`*, the program will show details for *package*, including reverse depends.

Once the package has been removed, the package's bugs should be handled. They should either be reassigned to another package in the case where the actual code has evolved into another package (e.g. `libfoo12` was removed because `libfoo13` supersedes it) or closed if the software is simply no more part of Debian.

Removing packages from `Incoming`

In the past, it was possible to remove packages from `incoming`. However, with the introduction of the new incoming system, this is no longer possible. Instead, you have to upload a new revision of your package with a higher version as the package you want to replace. Both versions will be installed in the archive but only the higher version will actually be available in *unstable* since the previous version will immediately be replaced by the higher. However, if you do proper testing of your packages, the need to replace a package should not occur too often anyway.

5.5.3 Replacing or renaming packages

Sometimes you made a mistake naming the package and you need to rename it. In this case, you need to follow a two-step process. First, set your `debian/control` file to replace and conflict with the obsolete name of the package (see the Debian Policy Manual (`http://www.debian.org/doc/debian-policy/`) for details). Once you've uploaded that package, and the package has moved into the archive, file a bug against `ftp.debian.org` asking to remove the package with the obsolete name. Do not forget to properly reassign the package's bugs at the same time.

5.5.4 Orphaning a package

If you can no longer maintain a package, you need to inform the others about that, and see that the package is marked as orphaned. You should set the package maintainer to `Debian QA Group <packages@qa.debian.org>` and submit a bug report against the pseudo package `wnpp`. The bug report should be titled `O:` *`package -- short description`* indicating that the package is now orphaned. The severity of the bug should be set to *normal*. If you feel it's necessary, send a copy to `<debian-devel@lists.debian.org>` by putting the address in the X-Debbugs-CC: header of the message (no, don't use CC:, because that way the message's subject won't indicate the bug number).

If the package is especially crucial to Debian, you should instead submit a bug against `wnpp` and title it `RFA:` *`package -- short description`* and set its severity to *important*. `RFA` stands for *Request For Adoption*. Definitely copy the message to debian-devel in this case, as described above.

Read instructions on the WNPP web pages (`http://www.debian.org/devel/wnpp/`) for more information.

5.5.5 Adopting a package

A list of packages in need of a new maintainer is available at in the Work-Needing and Prospective Packages list (WNPP) (`http://www.debian.org/devel/wnpp/`). If you wish to take over maintenance of any of the packages listed in the WNPP, please take a look at the aforementioned page for information and procedures.

It is not OK to simply take over a package that you feel is neglected — that would be package hijacking. You can, of course, contact the current maintainer and ask them if you may take over the package. However, without their assent, you may not take over the package. Even if they ignore you, that is still not grounds to take over a package. If you really feel that a maintainer has gone AWOL (absent without leave), post a query to `<debian-private@lists.debian.org>`. You may also inform the QA group (cf. 'Dealing with unreachable maintainers' on page 56).

If you take over an old package, you probably want to be listed as the package's official maintainer in the bug system. This will happen automatically once you upload a new version with an updated `Maintainer:` field, although it can take a few hours after the upload is done. If you do not expect to upload a new version for a while, you can use 'The Package Tracking System' on page 24 to get the bug reports. However, make sure that the old maintainer is not embarrassed by the fact that he will continue to receive the bugs during that time.

5.6 Handling package bugs

Often as a package maintainer, you find bugs in other packages or else have bugs reported to your packages which need to be reassigned. The BTS instructions (`http://www.debian.org/Bugs/server-control.html`) can tell you how to do this. Some information on filing bugs can be found in 'Bug Reporting' on page 55.

5.6.1 Monitoring bugs

If you want to be a good maintainer, you should periodically check the Debian bug tracking system (BTS) (`http://www.debian.org/Bugs/`) for your packages. The BTS contains all the open bugs against your packages. You can check them by browsing this page: `http://bugs.debian.org/`*`yourlog`*

Maintainers interact with the BTS via email addresses at `bugs.debian.org`. Documentation on available commands can be found at `http://www.debian.org/Bugs/`, or, if you have installed the `doc-debian` package, you can look at the local files `/usr/share/doc/debian/bug-*`.

Some find it useful to get periodic reports on open bugs. You can add a cron job such as the following if you want to get a weekly email outlining all the open bugs against your packages:

```
# ask for weekly reports of bugs in my packages
0 17 * * fri   echo "index maint address" | mail request@bugs.debian.org
```

Replace *address* with your official Debian maintainer address.

5.6.2 Responding to bugs

Make sure that any discussion you have about bugs are sent both to the original submitter of the bug, and the bug itself (e.g., `<123@bugs.debian.org>`). If you're writing a new mail and you don't remember the submitter email address, you can use the `<123-submitter@bugs.debian.org>` email to contact the submitter *and* to record your mail within the bug log (that means you don't need to send a copy of the mail to `<123@bugs.debian.org>`).

You should *never* close bugs via the bug server `close` command sent to `<control@bugs.debian.org>`. If you do so, the original submitter will not receive any information about why the bug was closed.

5.6.3 Bug housekeeping

As a package maintainer, you will often find bugs in other packages or have bugs reported against your packages which are actually bugs in other packages. The BTS instructions (`http://www.`

`debian.org/Bugs/server-control.html`) document the technical operations of the BTS, such as how to file, reassign, merge, and tag bugs. This section contains some guidelines for managing your own bugs, based on the collective Debian developer experience.

Filing bugs for problems that you find in other packages is one of the "civic obligations" of maintainership, see 'Bug Reporting' on page 55 for details. However handling the bugs on your own packages is even more important.

Here's a list of steps that you may follow to handle a bug report:

1. Decide whether the report corresponds to a real bug or not. Sometimes users are just calling a program in the wrong way because they haven't read the documentation. If you diagnose this, just close the bug with enough information to let the user correct his problem (give pointers to the good documentation and so on). If the same report comes up again and again you may ask yourself if the documentation is good enough or if the program shouldn't detect its misuse in order to give an informative error message. This is an issue that may need to be brought to the upstream author.

   If the bug submitter disagree with your decision to close the bug, he may reopen it until you find an agreement on how to handle it. If you don't find any, you may want to tag the bug `wontfix` to let people know that the bug exists but that it won't be corrected. If this situation is unacceptable, you (or the submitter) may want to require a decision of the technical committee by reassigning the bug to `tech-ctte` (you may use the clone command of the BTS if you wish to keep it reported against your package).

2. If the bug is real but it's caused by another package, just reassign the bug the right package. If you don't know which package it should be reassigned to, you may either ask for help on `<debian-devel@lists.debian.org>` or reassign it to `debian-policy` to let them decide which package is in fault.

   Sometimes you also have to adjust the severity of the bug so that it matches our definition of the severity. That's because people tend to inflate the severity of bugs to make sure their bugs are fixed quickly. Some bugs may even be dropped to wishlist severity when the requested change is just cosmetic.

3. The bug submitter may have forgotten to provide some information, in that case you have to ask him the information required. You may use the `moreinfo` tag to mark the bug as such. Moreover if you can't reproduce the bug, you tag it `unreproducible`. Anyone who can reproduce the bug is then invited to provide more information on how to reproduce it. After a few months, if this information has not been sent by someone, the bug may be closed.

4. If the bug is related to the packaging, you just fix it. If you are not able to fix it yourself, then tag the bug as `help`. You can also ask for help on `<debian-devel@lists.debian.org>` or `<debian-qa@lists.debian.org>`. If it's an upstream problem, you have to forward it to the upstream author. Forwarding a bug is not enough, you have to check at each release

if the bug has been fixed or not. If it has, you just close it, otherwise you have to remind the author about it. If you have the required skills you can prepare a patch that fixes the bug and that you send at the same time to the author. Make sure to send the patch in the BTS and to tag the bug as `patch`.

5. If you have fixed a bug in your local copy, or if a fix has been committed to the CVS repository, you may tag the bug as `pending` to let people know that the bug is corrected and that it will be closed with the next upload (add the `closes:` in the `changelog`). This is particularly useful if you are several developers working on the same package.

6. Once a corrected package is available in the *unstable* distribution, you can close the bug. This can be done automatically, read 'When bugs are closed by new uploads' on the current page.

5.6.4 When bugs are closed by new uploads

If you fix a bug in your packages, it is your responsibility as the package maintainer to close the bug when it has been fixed. However, you should not close the bug until the package which fixes the bug has been accepted into the Debian archive. Therefore, once you get notification that your updated package has been installed into the archive, you can and should close the bug in the BTS.

If you are using a new version of `dpkg-dev` and you do your changelog entry properly, the archive maintenance software will close the bugs automatically. All you have to do is follow a certain syntax in your `debian/changelog` file:

```
acme-cannon (3.1415) unstable; urgency=low

  * Frobbed with options (closes: Bug#98339)
  * Added safety to prevent operator dismemberment, closes: bug#98765,
    bug#98713, #98714.
  * Added man page. Closes: #98725.
```

Technically speaking, the following Perl regular expression is what is used:

```
/closes:\s*(?:bug)?\#\s*\d+(?:,\s*(?:bug)?\#\s*\d+)*/ig
```

The author prefers the `closes:  #XXX` syntax, as one of the most concise and easiest to integrate with the text of the `changelog`.

If you want to close bugs the old fashioned, manual way, it is usually sufficient to mail the `.changes` file to `<XXX-done@bugs.debian.org>`, where *XXX* is your bug number.

5.6.5 Lintian reports

You should periodically get the new `lintian` from ‘unstable’ and check over all your packages. Alternatively you can check for your maintainer email address at the online lintian report (`http://lintian.debian.org/`). That report, which is updated automatically, contains `lintian` reports against the latest version of the distribution (usually from ’unstable’) using the latest `lintian`.

Chapter 6

Best Packaging Practices

Debian's quality is largely due to its Policy that all packages follow. But it's also because we accumulated years of experience in packaging; very talented people created great tools to make good packages without much troubles.

This chapter provides the best known solutions to common problems faced during packaging. It also lists various advice collected on several mailing lists. By following them, you will make Debian's quality even better.

6.1 Packaging tools and common cases

6.1.1 Helper scripts

To help you in your packaging effort, you can use helper scripts. The best scripts available are provided by `debhelper`. With `dh_make` (package `dh-make`), you can generate in a few seconds a package that is mostly ready. However that apparent simplicity is hiding many things done by the helper scripts. You have to know what is done by them, that's why you are strongly encouraged to read the corresponding manual pages, starting with `debhelper(1)`. That's required because you'll have to understand what is going on to be able to use them wisely and to fix bugs in a pretty way.

debhelper is very useful because it lets you follow the latest Debian policy without doing many modifications since the changes that can be automated are almost always automatically done by a debhelper script. Furthermore it offers enough flexibility to be able to use it in conjunction with some hand crafted shell invocations within the `rules` file.

You can however decide to not use any helper script, and still write some very good `rules` file. Many examples are available at `http://people.debian.org/~srivasta/rules`.

6.1.2 Package with multiple patches

Big packages tend to have many upstream bugs that you want to fix within the Debian package. If you just correct the bug in the source, all the fixes are directly integrated in the `.diff.gz` file and you can't easily differentiate the various patches that you applied. It gets very messy when you have to update the package to a new upstream version which integrates some of the fixes (but not all).

The good solution is to keep separate patches within the `debian/patches` directory and to apply them on the fly at build time. The package `dbs` provides an implementation of such a system, you just have to build-depend on dbs to be able to use its functionalities. The package `hello-dbs` is a simple example that demonstrates how to use `dbs`.

Additionally, dbs provides facilities to create the patches and to keep track of what they are for.

6.1.3 Multiple binary packages

A single source package will often build several binary packages, either to provide several flavors of the same software (examples are the vim-* packages) or to make several small packages instead of a big one (it's interesting if the user doesn't need all the packages and can thus save some disk space).

The second case can be easily managed by `dh_install` (from `debhelper`) to move files from the build directory to the package's temporary trees.

The first case is a bit more difficult since it involves multiple recompiles of the same software but with different configure options. The `vim` is an example of how to manage this with an hand crafted rules file.

6.1.4 Handling debconf translations

Like porters, translators have a difficult task. Since they work on many packages, they cannot keep track of every change in packages in order to be informed when a translated string is outdated. Fortunately `debconf` can automatically report outdated translations, if package maintainers follow some basic guidelines described below.

Translators can use `debconf-getlang` (package `debconf-utils`) to write a `templates.xx` file containing both English and localized fields (where *xx* is the language code, may be followed by a country code). This file can be put into the `debian` subdirectory without any change.

When building a binary package, `debian/templates.xx` files are merged along with `debian /templates` to generate the `templates` file contained in the binary package. This is automatically done by `dh_installdebconf` (package `debhelper`). If you do not use debhelper, you can do the same with `debconf-mergetemplate` (package `debconf-utils`).

When the package maintainer needs to update the templates file, he only changes `debian/templates`. When English strings in this file and in `debian/templates.xx` differ, translators do know that their translation is outdated.

Please see the page about localizing debconf templates files (`http://www.debian.org/intl/l10n/templates/hints`) at the Debian web site, it contains more detailed instructions, including a full example.

6.2 Specific packaging practices

6.2.1 Libraries

Libraries are always difficult to package for various reasons. The policy imposes many constraints to ease their maintenance and to make sure upgrades are as simple as possible when a new upstream version comes out. A breakage in a library can result in dozens of dependent packages to break...

Good practices for library packaging have been grouped in the library packaging guide (`http://www.netfort.gr.jp/~dancer/column/libpkg-guide/`).

6.2.2 Other specific packages

Several subsets of packages have special sub-policies and corresponding packaging rules and practices:

- Perl related packages have a perl policy (`http://www.debian.org/doc/packaging-manuals/perl-policy/`), some examples of packages following that policy are `libdbd-pg-perl` (binary perl module) or `libmldbm-perl` (arch independent perl module).
- Python related packages have their python policy: `/usr/share/doc/python/python-policy.txt.g` (in the python package).
- Emacs related packages have the emacs policy (`http://www.debian.org/doc/packaging-manuals/debian-emacs-policy/`).
- Java related packages have their java policy (`http://people.debian.org/~opal/java/policy.html/`).
- Ocaml related packages have their ocaml policy: `/usr/share/doc/ocaml/ocaml_packaging_polic` (in the `ocaml` package). A good example is the `camlzip` source package.

6.3 Configuration management

6.3.1 The wise use of debconf

Debconf is a configuration management system, it is used by all the various packaging scripts (postinst mainly) to request feedback from the user concerning how to configure the package. Direct user interactions must now be avoided in favor of debconf interaction. This will enable non-interactive installations in the future.

Debconf is a great tool but it is often badly used ... many common mistakes are listed in the `debconf-devel(8)` man page. It is something that you must read if you decide to use debconf.

6.4 Miscellaneous advice

6.4.1 Writing useful descriptions

The description of the package (as defined by the corresponding field in the `control` file) is usually the first information available to the user before he installs it. As such, it should provide all the required information to let him decide whether to install the package.

For example, apart from the usual description that you adapt from the upstream `README`, you should include the URL of the web site if there's any. If the package is not yet considered stable by the author, you may also want to warn the user that the package is not ready for production use.

Last but not least, since the first user impression is based on that description, you should be careful to avoid English mistakes. Ensure that you spell check it. `ispell` has a special option (`-g`) for that:

```
ispell -d american -g debian/control
```

Chapter 7

Beyond Packaging

Debian is about a lot more than just packaging software and maintaining those packages. This chapter contains information about ways, often really critical ways, to contribute to Debian beyond simply creating and maintaining packages.

As a volunteer organization, Debian relies on the discretion of its members in choosing what they want to work on, and choosing what is the most critical thing to spend their time on.

7.1 Bug Reporting

We encourage you to file bugs as you find them in Debian packages. In fact, Debian developers are often the first line testers. Finding and reporting bugs in other developer's packages improves the quality of Debian.

Try to submit the bug from a normal user account at which you are likely to receive mail. Do not submit bugs as root.

Make sure the bug is not already filed against a package. Try to do a good job reporting a bug and redirecting it to the proper location. For extra credit, you can go through other packages, merging bugs which are reported more than once, or setting bug severities to 'fixed' when they have already been fixed. Note that when you are neither the bug submitter nor the package maintainer, you should not actually close the bug (unless you secure permission from the maintainer).

From time to time you may want to check what has been going on with the bug reports that you submitted. Take this opportunity to close those that you can't reproduce anymore. To find out all the bugs you submitted, you just have to visit `http://bugs.debian.org/from:<your-email-addr>`.

7.1.1 Reporting lots of bugs at once

Reporting a great number of bugs for the same problem on a great number of different packages — i.e., more than 10 — is a deprecated practice. Take all possible steps to avoid submitting bulk bugs at all. For instance, if checking for the problem can be automated, add a new check to `lintian` so that an error or warning is emitted.

If you report more than 10 bugs on the same topic at once, it is recommended that you send a message to `<debian-devel@lists.debian.org>` describing your intention before submitting the report. This will allow other developers to verify that the bug is a real problem. In addition, it will help prevent a situation in which several maintainers start filing the same bug report simultaneously.

Note that when sending lots of bugs on the same subject, you should send the bug report to `<maintonly@bugs.debian.org>` so that the bug report is not forwarded to the bug distribution mailing list.

7.2 Quality Assurance effort

Even though there is a dedicated group of people for Quality Assurance, QA duties are not reserved solely for them. You can participate in this effort by keeping your packages as bug-free as possible, and as lintian-clean (see 'Lintian reports' on page 50) as possible. If you do not find that possible, then you should consider orphaning some of your packages (see 'Orphaning a package' on page 46). Alternatively, you may ask the help of other people in order to catch up the backlog of bugs that you have (you can ask for help on `<debian-qa@lists.debian.org>` or `<debian-devel@lists.debian.org>`).

7.3 Dealing with unreachable maintainers

If you notice that a package is lacking maintenance, you should make sure the maintainer is active and will continue to work on his packages. Try contacting him yourself.

If you do not get a reply after a few weeks you should collect all useful information about this maintainer. Start by logging into the Debian Developer's Database (`https://db.debian.org/`) and doing a full search to check whether the maintainer is on vacation and when he was last seen. Collect any important package names he maintains and any Release Critical bugs filled against them.

Send all this information to `<debian-qa@lists.debian.org>`, in order to let the QA people do whatever is needed.

7.4 Contacting other maintainers

During your lifetime within Debian, you will have to contact other maintainers for various reasons. You may want to discuss a new way of cooperating between a set of related packages, or you may simply remind someone that a new upstream version is available and that you need it.

Looking up the email address of the maintainer for the package can be distracting. Fortunately, there is a simple email alias, `<package>@packages.debian.org`, which provides a way to email the maintainer, whatever their individual email address (or addresses) may be. Replace `<package>` with the name of a source or a binary package.

You may also be interested by contacting the persons who are subscribed to a given source package via 'The Package Tracking System' on page 24. You can do so by using the `<package-name>@packages.qa.de` email address.

7.5 Interacting with prospective Debian developers

Debian's success depends on its ability to attract and retain new and talented volunteers. If you are an experienced developer, we recommend that you get involved with the process of bringing in new developers. This section describes how to help new prospective developers.

7.5.1 Sponsoring packages

Sponsoring a package means uploading a package for a maintainer who is not able to do it on their own, a new maintainer applicant. Sponsoring a package also means accepting responsibility for it.

If you wish to volunteer as a sponsor, you can sign up at `http://www.internatif.org/bortzmeyer/debian/sponsor/`.

New maintainers usually have certain difficulties creating Debian packages — this is quite understandable. That is why the sponsor is there, to check the package and verify that it is good enough for inclusion in Debian. (Note that if the sponsored package is new, the ftpmasters will also have to inspect it before letting it in.)

Sponsoring merely by signing the upload or just recompiling is **definitely not recommended**. You need to build the source package just like you would build a package of your own. Remember that it doesn't matter that you left the prospective developer's name both in the changelog and the control file, the upload can still be traced to you.

If you are an application manager for a prospective developer, you can also be their sponsor. That way you can also verify how the applicant is handling the 'Tasks and Skills' part of their application.

7.5.2 Managing sponsored packages

By uploading a sponsored package to Debian, you are certifying that the package meets minimum Debian standards. That implies that you must build and test the package on your own system before uploading.

You can not simply upload a binary `.deb` from the sponsoree. In theory, you should only ask only for the diff file, and the location of the original source tarball, and then you should download the source and apply the diff yourself. In practice, you may want to use the source package built by your sponsoree. In that case you have to check that he hasn't altered the upstream files in the `.orig.tar.gz` file that he's providing.

Do not be afraid to write the sponsoree back and point out changes that need to be made. It often takes several rounds of back-and-forth email before the package is in acceptable shape. Being a sponsor means being a mentor.

Once the package meets Debian standards, build the package with

```
dpkg-buildpackage -us -uc
```

and sign it with

```
debsign -m <your-email-addr> <changes-file>
```

before uploading it to the incoming directory.

The Maintainer field of the `control` file and the `changelog` should list the person who did the packaging, i.e. the sponsoree. The sponsoree will therefore get all the BTS mail about the package.

If you prefer to leave a more evident trace of your sponsorship job, you can add a line stating it in the most recent changelog entry.

You are encouraged to keep tabs on the package you sponsor using 'The Package Tracking System' on page 24.

7.5.3 Advocating new developers

See the page about advocating a prospective developer (`http://www.debian.org/devel/join/nm-advocate`) at the Debian web site.

7.5.4 Handling new maintainer applications

Please see Checklist for Application Managers (`http://www.debian.org/devel/join/nm-amchecklist`) at the Debian web site.

Appendix A

Overview of Debian Maintainer Tools

This section contains a rough overview of the tools available to maintainers. The following is by no means complete or definitive, but just a guide to some of the more popular tools.

Debian maintainer tools are meant to help convenience developers and free their time for critical tasks. As Larry Wall says, there's more than one way to do it.

Some people prefer to use high-level package maintenance tools and some do not. Debian is officially agnostic on this issue; any tool which gets the job done is fine. Therefore, this section is not meant to stipulate to anyone which tools they should use or how they should go about with their duties of maintainership. Nor is it meant to endorse any particular tool to the exclusion of a competing tool.

Most of the descriptions of these packages come from the actual package descriptions themselves. Further information can be found in the package documentation itself. You can also see more info with the command `apt-cache show <package-name>`.

A.1 `dpkg-dev`

`dpkg-dev` contains the tools (including `dpkg-source`) required to unpack, build and upload Debian source packages. These utilities contain the fundamental, low-level functionality required to create and manipulated packages; as such, they are required for any Debian maintainer.

A.2 `lintian`

`Lintian` dissects Debian packages and reports bugs and policy violations. It contains automated checks for many aspects of Debian policy as well as some checks for common errors. The use of `lintian` has already been discussed in 'Checking the package prior to upload' on page 28 and 'Lintian reports' on page 50.

A.3 `debconf`

`debconf` provides a consistent interface to configuring packages interactively. It is user interface independent, allowing end-users to configure packages with a text-only interface, an HTML interface, or a dialog interface. New interfaces can be added modularly.

You can find documentation for this package in the `debconf-doc` package.

Many feel that this system should be used for all packages requiring interactive configuration. `debconf` is not currently required by Debian Policy, however, that may change in the future.

A.4 `debhelper`

`debhelper` is a collection of programs that can be used in `debian/rules` to automate common tasks related to building binary Debian packages. Programs are included to install various files into your package, compress files, fix file permissions, integrate your package with the Debian menu system.

Unlike some approaches, `debhelper` is broken into several small, granular commands which act in a consistent manner. As such, it allows a greater granularity of control than some of the other "debian/rules tools".

There are a number of little `debhelper` add-on packages, too transient to document. You can see the list of most of them by doing `apt-cache search ^dh-`.

A.5 `debmake`

`debmake`, a pre-cursor to `debhelper`, is a less granular `debian/rules` assistant. It includes two main programs: `deb-make`, which can be used to help a maintainer convert a regular (non-Debian) source archive into a Debian source package; and `debstd`, which incorporates in one big shot the same sort of automated functions that one finds in `debhelper`.

The consensus is that `debmake` is now deprecated in favor of `debhelper`. However, it's not a bug to use `debmake`.

A.6 `yada`

`yada` is another packaging helper tool. It uses a `debian/packages` file to auto-generate `debian/rules` and other necessary files in the `debian/` subdirectory.

Note that `yada` is called "essentially unmaintained" by it's own maintainer, Charles Briscoe-Smith. As such, it can be considered deprecated.

A.7 `equivs`

`equivs` is another package for making packages. It is often suggested for local use if you need to make a package simply to fulfill dependencies. It is also sometimes used when making "meta-packages", which are packages whose only purpose is to depend on other packages.

A.8 `cvs-buildpackage`

`cvs-buildpackage` provides the capability to inject or import Debian source packages into a CVS repository, build a Debian package from the CVS repository, and helps in integrating upstream changes into the repository.

These utilities provide an infrastructure to facilitate the use of CVS by Debian maintainers. This allows one to keep separate CVS branches of a package for *stable*, *unstable*, and possibly *experimental* distributions, along with the other benefits of a version control system.

A.9 `dupload`

`dupload` is a package and a script to automatically upload Debian packages to the Debian archive, to log the upload, and to send mail about the upload of a package. You can configure it for new upload locations or methods.

A.10 `dput`

The `dput` package and script does much the same thing as `dupload`, but in a different way. It has some features over `dupload`, such as the ability to check the GnuPG signature and checksums before uploading, and the possibility of running `dinstall` in dry-run mode after the upload.

A.11 fakeroot

fakeroot simulates root privileges. This enables you to build packages without being root (packages usually want to install files with root ownership). If you have fakeroot installed, you can build packages as a user: dpkg-buildpackage -rfakeroot.

A.12 debootstrap

The debootstrap package and script allows you to "bootstrap" a Debian base system into any part of your file-system. By "base system", we mean the bare minimum of packages required to operate and install the rest of the system.

Having a system like this can be useful in many ways. For instance, you can chroot into it if you want to test your build depends. Or, you can test how your package behaves when installed into a bare base system.

A.13 pbuilder

pbuilder constructs a chrooted system, and builds a package inside the chroot. It is very useful to check that a package's build-dependencies are correct, and to be sure that unnecessary and wrong build dependencies will not exist in the resulting package.

A.14 devscripts

devscripts is a package containing a few wrappers and tools which are very helpful for maintaining your Debian packages. Example scripts include debchange and dch, which manipulate your debian/changelog file from the command-line, and debuild, which is a wrapper around dpkg-buildpackage. The bts utility is also very helpful to update the state of bug reports on the command line, as is uscan to watch for new upstream versions of your packages. Check the devscripts(1) manual page for a complete list of available scripts.

A.15 dpkg-dev-el

dpkg-dev-el is an Emacs lisp package which provides assistance when editing some of the files in the debian directory of your package. For instance, when editing debian/changelog, there are handy functions for finalizing a version and listing the package's current bugs.

A.16 `debget`

`debget` is a package containing a convenient script which can be helpful in downloading files from the Debian archive. You can use it to download source packages, for instance (although `apt-get source <package-name>` does pretty much the same thing).