

# 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. 2.8.8, 20 July, 2001

## Copyright Notice

copyright ©1998 – 2001 Adam Di Carlo  
copyright ©1997, 1998 Christian Schwarz

This manual is free software; you may redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

This is distributed in the hope that it will be useful, but *without any warranty*; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.

A copy of the GNU General Public License is available as `/usr/share/common-licenses/GPL` in the Debian GNU/Linux distribution or on the World Wide Web at the GNU website (<http://www.gnu.org/copyleft/gpl.html>). You can also obtain it by writing to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

# Contents

<b>1</b>	<b>Scope of This Document</b>	<b>1</b>
<b>2</b>	<b>Applying to Become a Maintainer</b>	<b>3</b>
2.1	Getting started . . . . .	3
2.2	Registering as a Debian developer . . . . .	3
2.3	Debian Mentors . . . . .	5
<b>3</b>	<b>Debian Developer's Duties</b>	<b>7</b>
3.1	Maintaining Your Debian Information . . . . .	7
3.2	Maintaining Your Public Key . . . . .	7
3.3	Going On Vacation Gracefully . . . . .	8
3.4	Coordination With Upstream Developers . . . . .	8
3.5	Managing Release Critical Bugs . . . . .	8
3.6	Quality Assurance Effort . . . . .	9
3.7	Retiring Gracefully . . . . .	9
<b>4</b>	<b>Mailing Lists, Servers, and Other Machines</b>	<b>11</b>
4.1	Mailing lists . . . . .	11
4.2	Debian servers . . . . .	12
4.2.1	The master server . . . . .	12
4.2.2	The ftp-master server . . . . .	12
4.2.3	The WWW server . . . . .	13

---

4.2.4	The CVS server . . . . .	13
4.2.5	Mirrors of Debian servers . . . . .	13
4.3	Other Debian Machines . . . . .	14
<b>5</b>	<b>The Debian Archive</b>	<b>15</b>
5.1	Overview . . . . .	15
5.2	Sections . . . . .	17
5.3	Architectures . . . . .	18
5.4	Subsections . . . . .	18
5.5	Packages . . . . .	18
5.6	Distribution directories . . . . .	19
5.6.1	Stable, testing, unstable, and sometimes frozen . . . . .	19
5.6.2	Experimental . . . . .	20
5.7	Release code names . . . . .	21
<b>6</b>	<b>Package uploads</b>	<b>23</b>
6.1	Announcing new packages . . . . .	23
6.2	Uploading a package . . . . .	24
6.2.1	Generating the changes file . . . . .	24
6.2.2	Picking a distribution . . . . .	24
6.2.3	Checking the package prior to upload . . . . .	26
6.2.4	Uploading to ftp-master . . . . .	26
6.2.5	Uploading to non-US (pandora) . . . . .	27
6.2.6	Uploads via chiark . . . . .	27
6.2.7	Uploads via erlangen . . . . .	28
6.2.8	Other Upload Queues . . . . .	28
6.3	Announcing package uploads . . . . .	28
6.4	Notification that a new package has been installed . . . . .	29
6.4.1	The override file . . . . .	29

---

<b>7</b>	<b>Non-Maintainer Uploads (NMUs)</b>	<b>31</b>
7.1	Terminology . . . . .	31
7.2	Who can do an NMu . . . . .	32
7.3	When to do a source NMu . . . . .	32
7.4	How to do a source NMu . . . . .	33
7.4.1	Source NMu version numbering . . . . .	33
7.4.2	Source NMUs must have a new changelog entry . . . . .	34
7.4.3	Source NMUs and the Bug Tracking System . . . . .	34
7.4.4	Building source NMUs . . . . .	35
<b>8</b>	<b>Porting and Being Ported</b>	<b>37</b>
8.1	Being Kind to Porters . . . . .	37
8.2	Guidelines for Porter Uploads . . . . .	38
8.2.1	When to do a source NMu if you are a porter . . . . .	38
8.3	Tools for Porters . . . . .	39
8.3.1	quinn-diff . . . . .	39
8.3.2	builddd . . . . .	39
8.3.3	dpkg-cross . . . . .	40
<b>9</b>	<b>Moving, Removing, Renaming, Adopting, and Orphaning Packages</b>	<b>41</b>
9.1	Moving packages . . . . .	41
9.2	Removing packages . . . . .	42
9.2.1	Removing packages from Incoming . . . . .	42
9.3	Replacing or renaming packages . . . . .	42
9.4	Orphaning a package . . . . .	42
9.5	Adopting a package . . . . .	43
<b>10</b>	<b>Handling Bugs</b>	<b>45</b>
10.1	Monitoring bugs . . . . .	45
10.2	Submitting Bugs . . . . .	45

---

10.3	Responding to Bugs . . . . .	46
10.4	When bugs are closed by new uploads . . . . .	46
10.5	Lintian reports . . . . .	47
10.6	Reporting lots of bugs at once . . . . .	47
<b>11</b>	<b>Interaction with Prospective Developers</b>	<b>49</b>
11.1	Sponsoring packages . . . . .	49
11.2	Advocating new developers . . . . .	49
11.3	Handling new maintainer applications . . . . .	50
<b>12</b>	<b>Overview of Debian Maintainer Tools</b>	<b>51</b>
12.1	dpkg-dev . . . . .	51
12.2	lintian . . . . .	51
12.3	debconf . . . . .	52
12.4	debhelper . . . . .	52
12.5	debmake . . . . .	52
12.6	yada . . . . .	52
12.7	equivs . . . . .	52
12.8	cvs-buildpackage . . . . .	53
12.9	dupload . . . . .	53
12.10	fakeroot . . . . .	53
12.11	devscripts . . . . .	53
12.12	debget . . . . .	53

# 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 23); how and when to do ports and interim releases of other maintainers' packages ('Non-Maintainer Uploads (NMUs)' on page 31); how to move, remove, or orphan packages ('Moving, Removing, Renaming, Adopting, and Orphaning Packages' on page 41); and how to handle bug reports ('Handling Bugs' on page 45).

The resources discussed in this reference include the mailing lists and servers ('Mailing Lists, Servers, and Other Machines' on page 11); 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 26); and a discussion of resources which can help maintainers with the quality of their packages ('Overview of Debian Maintainer Tools' on page 51).

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 understand what everything in the `hello` example package is for, and you're about to Debianize your favourite 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.

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' on page 5 for details. The IRC channel `#debian` on the Linux People IRC network (e.g., `irc.debian.org`) can also be helpful.

### 2.2 Registering as a Debian developer

Before you decide to register with the Debian Project, you will need to read the Debian Social Contract ([http://www.debian.org/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. 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 letting them upload packages.

Registration requires that the following information be sent in appropriate steps described at Checklist for applicants (<http://www.debian.org/devel/join/nm-checklist>) after the initial contact to `<new-maintainer@debian.org>`:

- Your name.
- Your preferred login name on master (eight characters or less), as well as the email address at which you'd prefer to be subscribed to `<debian-private@lists.debian.org>` (typically this will be either your primary mail address or your new `debian.org` address).
- A phone number where we can call you. Remember that the new maintainer team usually calls during evening hours to save on long distance tolls. Please do not give a work number, unless you are generally there in the evening.
- A statement of intention, that is, what package(s) you intend to work on, which Debian port you will be assisting, or how you intend to contribute to Debian.
- A statement that you have read and agree to uphold the Debian Social Contract ([http://www.debian.org/social\\_contract](http://www.debian.org/social_contract)).
- Some mechanism by which we can verify your real-life identity. For example, any of the following mechanisms would suffice:
  - An OpenPGP key signed by any well-known signature, such as:
    - \* Any current Debian developer you have met *in real life*.
    - \* Any formal certification service (such as Verisign, etc.) that verifies your identity. A certification that verifies your email address, and not your identity, is not sufficient.
  - Alternatively, you may identify yourself with a scanned (or physically mailed) copy of any formal documents certifying your identity (such as a birth certificate, national ID card, U.S. Driver's License, etc.). If emailed, please sign the mail with your OpenPGP key.

If you do not have an OpenPGP key yet, generate one. Every developer needs an 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.

Also remember that one of the names on your key must match the email address you list as the official maintainer for your packages. For instance, I set the maintainer of the `developers-reference` package to “Adam Di Carlo <aph@debian.org>”; therefore, one of the user IDs on my key is that same value, “Adam Di Carlo <aph@debian.org>”.

If your public key isn’t on public key servers such as `pgp5.ai.mit.edu`, please read the documentation available locally in `/usr/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.

Due to export restrictions by the United States government some Debian packages, including `gnupg`, are located on ftp sites outside of the United States. You can find the current locations of those packages at [ftp://ftp.debian.org/debian/README.non-US](http://ftp.debian.org/debian/README.non-US).

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 (as is the case in France). The Debian Project 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.

Once you have all your information ready, and your public key is available on public key servers, send a message to `<new-maintainer@debian.org>` to register as an official Debian developer so that you will be able to upload your packages. This message must contain your name and your valid e-mail address. All the information discussed above is required after your Application Manager is assigned. Application Manager is your agent in the registration process, and you can always ask him about the status of your application. You can check the Application StatusBoard (<http://nm.debian.org/>) as well.

For more details, please consult New Maintainer’s Corner (<http://www.debian.org/devel/join/newmaint>) at the Debian web site.

Once this information is received and processed, you should be contacted with information about your new Debian maintainer account. If you don’t hear anything within a month, please send a followup message asking if your original application was received. Do *not* re-send your original application, that will just confuse the New Maintainer Group. Please be patient, especially near release points; mistakes do occasionally happen, and people do sometimes run out of volunteer time.

## 2.3 Debian Mentors

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.

## 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 here : <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 keyring 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 3 apply.

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

### 3.3 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 31) your package if a big problem (release critical bugs, security update, ...) 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. 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.

### 3.4 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 conformant 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.

### 3.5 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 those bugs need to be corrected as fast 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 each time they can. 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 justify yourself and present your plan to fix it by sending a mail to the bug concerned report. Otherwise people from the QA group may want to do a Non-Maintainer Upload (see 'Non-Maintainer Uploads (NMUs)' on page 31) 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 on the BTS).

## 3.6 Quality Assurance Effort

Even though there is a dedicated group of people for Quality Assurance, QA duties are not reserved solely to 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 47) as possible. If you think that it's quite impossible, then you should consider orphaning (see 'Orphaning a package' on page 42) some of your packages so that you can do a good job with the other packages that you maintain. 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>).

## 3.7 Retiring Gracefully

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 42.
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

# Mailing Lists, Servers, and Other Machines

In this chapter you will find a very brief road map of the Debian mailing lists, the main Debian servers, and other Debian machines which may be available to you as a developer.

### 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/mailling-lists.txt> or locally in `/usr/doc/debian/mailling-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-private@lists.debian.org>` and `<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.

`<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 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 'Submitting Bugs' on page 45 for information on how to submit bugs.

### 4.2.1 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>`.

### 4.2.2 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-U.S. packages). Generally, package uploads go to this server; see 'Package uploads' on page 23.

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 41.

### 4.2.3 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/~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>).

### 4.2.4 The CVS server

`cvs.debian.org` is also known as `klecker.debian.org`, discussed above. If you need to use a publically 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, Debian account should own the CVS root area, and why you need it.

### 4.2.5 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.

### 4.3 Other Debian 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 12, there is a list of machines available to Debian developers at <http://db.debian.org/machines.cgi>.

## Chapter 5

# The Debian Archive

### 5.1 Overview

The Debian GNU/Linux distribution consists of a lot of Debian packages (.deb's, currently around 6800) and a few additional files (documentation, installation disk images, etc.).

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

```
dists/stable/main/  
dists/stable/main/binary-all/  
dists/stable/main/binary-all/admin/  
dists/stable/main/binary-all/base/  
dists/stable/main/binary-all/comm/  
dists/stable/main/binary-all/devel/  
...  
dists/stable/main/binary-i386/  
dists/stable/main/binary-i386/admin/  
dists/stable/main/binary-i386/base/  
...  
dists/stable/main/binary-m68k/  
dists/stable/main/binary-m68k/admin/  
dists/stable/main/binary-m68k/base/  
...  
dists/stable/main/source/  
dists/stable/main/source/admin/  
dists/stable/main/source/base/  
...  
dists/stable/main/disks-i386/
```

```
dists/stable/main/disks-m68k/
...

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

dists/stable/non-free/
dists/stable/non-free/binary-all/
dists/stable/non-free/binary-i386/
dists/stable/non-free/binary-m68k/
...
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/a/
pool/a/apt/
...
pool/b/
pool/b/bash/
...
pool/liba/
pool/liba/libalias-perl/
```

```
...  
pool/m/  
pool/m/mailx/  
...
```

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 with the source packages (`source`), a directory for each supported architecture (`binary-i386`, `binary-m68k`, etc.), and a directory for architecture independent packages (`binary-all`).

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.).

The *binary-\** and *source* directories are divided further into *subsections*.

## 5.2 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\hyper@hashguidelines](http://www.debian.org/social_contract\hyper@hashguidelines)) (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 apply 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.)

### 5.3 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*, and *arm*, 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 adds support for the *powerpc* and *arm* architectures.

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

### 5.4 Subsections

The sections *main*, *contrib*, and *non-free* are split into *subsections* to simplify the installation process and the maintainance of the archive. Subsections are not formally defined, except perhaps the 'base' subsection. Subsections simply exist to simplify the organization and browsing of available packages. Please check the current Debian distribution to see which sections are available.

Note however that with the introduction of package pools (see the top-level *pool/* directory), the subsections in the form of subdirectories will eventually cease to exist. They will be kept in the packages' 'Section' header fields, though.

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

## 5.6 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 (because of backwards compatibility).

### 5.6.1 Stable, testing, unstable, and sometimes frozen

There is always a distribution 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 change from day-to-day. Since no special effort is done to make sure everything in this distribution is working properly, it is sometimes “unstable.”

Packages get copied from *unstable* to *testing* if they satisfy certain criteria. To get into *testing* distribution, a package needs to be in the archive for two weeks and not have any release critical bugs. After that period, it will propagate into *testing* as soon as anything it depends on is also added. This process is automatic.

After a period of development, once the release manager deems fit, the *testing* distribution is renamed to *frozen*. Once that has been done, no changes are allowed to that distribution except bug fixes; that's why it's called “frozen.” After another month or a little longer, depending on the progress, the *frozen* 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 *frozen* distribution is renamed to *stable*, overriding the old *stable* distribution, which is removed at that time.

This development cycle is based on the assumption that the *unstable* distribution becomes *stable* after passing a period of testing as *frozen*. 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., ‘1.3’ becomes ‘1.3r1’, ‘2.0r2’ becomes ‘2.0r3’, and so forth).

Note that development under *unstable* continues during the “freeze” period, since the *unstable* distribution remains in place when the *testing* is moved to *frozen*. Another wrinkle is that when the *frozen* distribution is officially released, the old stable distribution is completely removed from the Debian archives (although they do live on at `archive-host` ;).

In summary, there is always a *stable*, a *testing* and an *unstable* distribution available, and a *frozen* distribution shows up for a couple of months from time to time.

### 5.6.2 Experimental

The *experimental* distribution is a specialty 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. 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.

Developers should be very selective in the use of the *experimental* distribution. Even if a package is highly unstable, it could still go into *unstable*; just state a few warnings in the description. However, if there is a chance that the software could do grave damage to a system, it might be better to put it into *experimental*.

For instance, an experimental encrypted file system should probably go into *experimental*. A new, beta, version of some software which uses completely different configuration might go into *experimental* at the maintainer’s discretion. New software which isn’t likely to damage your system can go into *unstable*. 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.

However, using *experimental* as a personal staging area is not always the best idea. You can’t replace or upgrade the files in there on your own (it is done with Debian archive maintenance software). Additionally, you’ll have to remember to ask the archive maintainers to delete the package once you have uploaded it to *unstable*. Using your personal web space on `klecker.debian.org` is generally a better idea, so that you put less strain on the Debian archive maintainers.

## 5.7 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’; and Debian 2.2, ‘potato’. 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*, *unstable*, and *frozen* point to the appropriate release directories.



## Chapter 6

# Package uploads

### 6.1 Announcing 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 ('Submitting Bugs' on page 45) 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 46).

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.

## 6.2 Uploading a package

### 6.2.1 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 46. Only the Distribution field is discussed in this section, since it relates to the archive maintenance policies.

### 6.2.2 Picking a distribution

Notably, the `Distribution` field, which originates from the `debian/changelog` file, indicates which distribution the package is intended for. There are four possible values for this field: ‘stable’, ‘unstable’, ‘frozen’, or ‘experimental’; these values can also be combined. Or, if Debian has been frozen, and you want to get a bug-fix release into *frozen*, you would set the distribution to ‘frozen unstable’. (See ‘Uploading to *frozen*’ on the facing page for more information on when to upload to *frozen*.) Note that it never makes sense to combine the *experimental* distribution with anything else.

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). Also note that setting the distribution 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*. The Release Team (which can be reached at <debian-release@lists.debian.org>) will decide if your package can be included in stable, therefore if your changelog entry is not clear enough, you may want to explain them why you uploaded your package to stable by sending them a short explication.

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 behaviour 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. If there is some reason why this is not the case, the new version of the original source should be uploaded, possibly by using the `-sa` flag.

### Uploading to *frozen*

The Debian freeze is a crucial time for Debian. It is our chance to synchronize and stabilize our distribution as a whole. Therefore, care must be taken when uploading to *frozen*.

It is tempting to always try to get the newest release of software into the release. However, it’s much more important that the system as a whole is stable and works as expected.

The watchword for uploading to *frozen* is **no new code**. This is a difficult thing to quantify, so here are some guidelines:

- Fixes for bugs of severity *critical*, *grave*, or *serious* severity are always allowed for those packages that must exist in the final release
- *critical*, *grave*, and *serious* bug fixes are allowed for non-necessary packages but only if they don’t add any new features
- important, normal and minor bug fixes are allowed (though discouraged) on all packages if and only if there are no new features
- wishlist fixes are not allowed (they are, after all, not really bugs)
- documentation bug fixes are allowed, since good documentation is important

Experience has shown that there is statistically a 15% chance that every bug fix will introduce a new bug. The introduction and discovery of new bugs either delays release or weakens the final product. There is little correlation between the severity of the original bug fixed and the severity of the bug newly introduced by the fix.

### 6.2.3 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.changes`. 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 51.

- Downgrade the package to the previous version (if one exists) – this tests the `postrm` and `prerm` scripts.
- Remove the package, then reinstall it.

### 6.2.4 Uploading to ftp-master

To upload a package, you need a personal account on [ftp-master.debian.org](http://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/`.

*Note:* Do not upload to `ftp-master` packages containing software that is export-controlled by the United States government, nor to the overseas upload queues on `chiark` or `erlangen`. This prohibition covers almost all cryptographic software, and even sometimes software that contains “hooks” to cryptographic software, such as electronic mail readers that support PGP encryption and authentication. Uploads of such software should go to `non-us` (see ‘Uploading to non-US (pandora)’ on the facing page). If you are not sure whether U.S. export controls apply to your package, post a message to `<debian-devel@lists.debian.org>` and ask.

You may also find the Debian package `dupload` useful when uploading packages. This handy program is distributed with defaults for uploading via `ftp` to `ftp-master`, `chiark`, and `erlangen`. It can also be configured to use `ssh` or `rsync`. See `dupload(1)` and `dupload(5)` for more information.



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
```

### 6.2.5 Uploading to non-US (pandora)

As discussed above, export controlled software should not be uploaded to `ftp-master`. Instead, use `scp` or non-anonymous FTP to copy the package to [non-us.debian.org](http://non-us.debian.org), placing the files in `/org/non-us.debian.org/incoming/`. By default, you can use the same account/password that works on `ftp-master`.

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

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.

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 or contrib, developers should at least follow the procedure outlined by the US Government (<http://www.bxa.doc.gov/Encryption/PubAvailEncSourceCodeNotify.html>). Maintainers of non-US/non-free packages should further consult these 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.

### 6.2.6 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](http://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 the preceding page 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.

### 6.2.7 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 PGP-signed 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 26 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.

### 6.2.8 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/>.

## 6.3 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` ( $\geq 1.4.1.2$ ). The mail generated by the archive maintenance software will contain the PGP/GPG 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’, ‘experimental’, or ‘frozen’ (when present), the announcement will be posted to <debian-devel-changes@lists.debian.org> instead.

On occasion, it is necessary to upload a package to both the *stable* and *unstable* distributions; this is done by putting both distributions in the `Distribution:` line. In such a case the upload announcement will go to both of the above mailing lists.

The `dupload` program is clever enough to determine where the announcement should go, and will automatically mail the announcement to the right list. See ‘dupload’ on page 53.

## 6.4 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 archive maintenance tools ‘dak’ (also referred to as *katie* or *dinstall*). 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 email notification indicating that the package has been uploaded. Please examine this notification carefully. You may notice that the package didn’t go into the section you thought you set it to go into. Read on for why.

### 6.4.1 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*. Sometimes the *override file* needs correcting. Simply changing the package’s `control` file is not going to work. Instead, you should email <override-change@debian.org> or submit a bug against `ftp.debian.org`.

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



## Chapter 7

# 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, do NMUs as part of their normal porting activity (see ‘Porting and Being Ported’ on page 37). Another reason why NMUs are done is when a Debian developer 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 NMUs, which is explained in the next section.

### 7.1 Terminology

There are two new terms used throughout this section: “binary NMu” and “source NMu”. These terms are used with specific technical meaning throughout this document. Both binary 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.

A binary NMu is a recompilation and upload of a binary package for a new architecture. As such, it is usually part of a porting effort. A binary NMu is a non-maintainer uploaded binary version of a package (often for another architecture), 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 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, 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 I use the unqualified term “NMU”, I mean both source and binary NMUs.

## 7.2 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.

## 7.3 When to do a source NMU

Guidelines for when to do a source NMU depend on the target distribution, i.e., stable, unstable, or frozen. 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 38).

Only critical changes or security bug fixes make it into stable. When a security bug is detected, a fixed package should be uploaded as soon as possible. In this case, the Debian Security Managers should 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, the Security Manager may upload a fixed package (i.e., do a source NMU).

During the release freeze (see ‘Uploading to *frozen*’ on page 25), 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. Try the following steps first, and if they don't work, it is probably OK to do an NMU:

- Make sure that the package bug is in the Debian Bug Tracking System (BTS). If not, submit a bug.
- Email the maintainer, and offer to help fix the package bug. Give it a few days.
- Go ahead and fix the bug, submitting a patch to the right bug in the BTS. Build the package and test it as discussed in ‘Checking the package prior to upload’ on page 26. Use it locally.

- Wait a couple of weeks for a response.
- Email the maintainer, asking if it is OK to do an NMU.
- 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.
- Wait another week for a response.
- Go ahead and do the source NMU, as described in 'How to do a source NMU' on the current page.

## 7.4 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 38.

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.

### 7.4.1 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').

Remember, porters who are simply recompiling a package for a different architecture do not need to renumber. Porters should use new version numbers if and only if they actually have to modify the source package in some way, i.e., if they are doing a source NMU and not a binary NMU.

### 7.4.2 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
```

### 7.4.3 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? In this case, the process is different for porters than it is for non-porters, as mentioned above. If you are not a porter and are doing an NMU that simply requires a recompile (i.e., a new shared library is available to be linked against, a bug was fixed in `debhelper`), there must still be a changelog entry; therefore, there will be a patch. If you are a porter, you are probably just doing a binary NMU. (Note: this leaves out in the cold porters who have to do recompiles – chalk it up as a weakness in how we maintain our archive.)

If the source NMU (non-maintainer upload) fixes some existing bugs, the bugs in the Bug Tracking System which are fixed need to be *notified* but not actually *closed* by the non-maintainer. Technically, only the official package maintainer or the original bug submitter are allowed to close bugs. However, the person making the non-maintainer release must send a short message to the relevant bugs explaining that the bugs have been fixed by the NMU. Using `<control@bugs.debian.org>`, the party doing the NMU should also set the severity of the bugs fixed in the NMU to 'fixed'. This 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, open a bug with the patches needed to fix the problem, or make sure that one of the other (already open) bugs has the patches.

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.

#### 7.4.4 Building source NMUs

Source NMU packages are built normally. Pick a distribution using the same rules as found in 'Picking a distribution' on page 24. Just as described in 'Uploading a package' on page 24, a normal changes file, etc., will be built. In fact, all the prescriptions from 'Package uploads' on page 23 apply, including the need to announce the NMU to the proper lists.

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.



## Chapter 8

# 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 five more builds.

### 8.1 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 more 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).

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. 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.
2. 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`.

3. 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`.
4. 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.
5. Don't depend on the package you're building already being installed (a sub-case of the above issue).
6. Don't rely on `egcc` being available; don't rely on `gcc` being a certain version.
7. Make sure your `debian/rules` contains separate "binary-arch" and "binary-indep" targets, as the Debian Packaging 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`.

## 8.2 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 NMU 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 33 instead.

In a binary NMU, no real 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`.

Sometimes you need to recompile a package against other packages which have been updated, such as libraries. You do have to bump the version number in this case, so that the upgrade system can function properly. Even so, these are considered binary-only NMUs – there is no need in this case for all architectures to recompile. You should set the version number as in the case of NMU versioning, but add a ".0." before the the NMU version. For instance, a recompile-only NMU of the source package "foo\_1.3-1" would be numbered "foo\_1.3-1.0.1".

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

### 8.2.1 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 31, 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. Crucial fixes (i.e., changes need to get a source package to compile for a released-targeted architecture) can be uploaded with *no* waiting period for the ‘frozen’ distribution.

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.

## 8.3 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.

### 8.3.1 `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*.

### 8.3.2 `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 autobuilt) 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.

### 8.3.3 `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.

## Chapter 9

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

Some archive manipulation operation 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.

### 9.1 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’.<sup>1</sup>

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). Carefully examine the installation log sent to you when the package is installed into the archive. If for some reason the old location of the package remains, file a bug against `ftp.debian.org` asking that the old location be removed. Give details on what you did, since it might be a bug in the archive maintenance software.

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 reupload that. Also, you’ll need to get the override file updated, as described in ‘The override file’ on page 29.

---

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

## 9.2 Removing packages

If for some reason you want to completely remove a package (say, if it is an old compatibility library which is not 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.

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.

### 9.2.1 Removing packages from Incoming

If you decide to remove a package from Incoming, it is nice but not required to send a notification of that to the appropriate announce list (either `<debian-changes@lists.debian.org>` or `<debian-devel-changes@lists.debian.org>`).

## 9.3 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.

## 9.4 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*. 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.



## 9.5 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>.

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, send an email to <override-change@debian.org> so that bug reports will go to you right away.



## Chapter 10

# Handling Bugs

### 10.1 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.

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/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 maintainer-address" | mail request@bugs.debian.
```

Replace *maintainer-address* with you official Debian maintainer address.

### 10.2 Submitting 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.

We encourage you to file bugs when there are problems. 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).

### 10.3 Responding to Bugs

Make sure that any discussions you have about bugs are sent both to the original submitter of the bug, and the bug itself (e.g., <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 feedback on why the bug was closed.

### 10.4 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 manpage. Closes: #98725.
```

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

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

The author prefers the `(closes: Bug#XXX)` syntax, since it stands out from the rest of the changelog entries.

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.

## 10.5 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`.

## 10.6 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.



## Chapter 11

# Interaction with Prospective Developers

This chapter describes procedures that existing Debian developers should follow when it comes to dealing with wannabe developers.

### 11.1 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.

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 FTP admins 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 the how the applicant is handling the 'Tasks and Skills' part of their application.

### 11.2 Advocating new developers

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

## 11.3 Handling new maintainer applications

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



## Chapter 12

# Overview of Debian Maintainer Tools

This section contains a rough overview of the tools available to maintainers. These tools are meant to help convenience developers and free their time for critical tasks.

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.

### 12.1 `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.

### 12.2 `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 26 and ‘Lintian reports’ on page 47.

## 12.3 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.

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.

## 12.4 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 debmake, debhelper is broken into several small, granular commands which act in a consistent manner. As such, it allows a greater granularity of control than debmake.

## 12.5 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.

## 12.6 yada

yada is a new packaging helper tool with a slightly different philosophy. It uses a `debian/packages` file to auto-generate other necessary files in the `debian/` subdirectory.

Note that yada is still quite new and possibly not yet as robust as other systems.

## 12.7 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.

## 12.8 `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.

## 12.9 `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.

## 12.10 `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 say, i.e., `dpkg-buildpackage -rfakeroot` as a user.

## 12.11 `devscripts`

`devscripts` is a package containing a few wrappers and tools which you may find helpful for maintaining your Debian packages. Example scripts include `debchange`, which will manipulate your `debian/changelog` file from the command-line, and `debuild`, which is a wrapper around `dpkg-buildpackage`.

## 12.12 `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.