Référence du développeur Debian

Developer's Reference Team, Andreas Barth, Adam Di Carlo, Raphaël Hertzog, Lucas Nussbaum, Christian Schwarz, et Ian Jackson

5 novembre 2009

Référence du développeur Debian
by Developer's Reference Team, Andreas Barth, Adam Di Carlo, Raphaël Hertzog, Lucas Nussbaum, Christian Schwarz, et Ian Jackson

Published 2009-09-17

Si vous désirez imprimer cette référence, vous devriez utiliser la version PDF. Cette page est également disponible en anglais.

Table des matières

1 Portée de ce document **1**

2 Devenir responsable Debian **3**
2.1 Pour commencer . . . 3
2.2 Mentors et parrains Debian . . . 3
2.3 Registering as a Debian developer . . . 4

3 Les charges du responsable Debian **7**
3.1 Mise à jour de vos références Debian . . . 7
3.2 Gérer votre clé publique . . . 7
3.3 Voting . . . 7
3.4 Going on vacation gracefully . . . 8
3.5 Coordination with upstream developers . . . 8
3.6 Managing release-critical bugs . . . 8
3.7 Retiring . . . 9

4 Ressources pour le responsable Debian **11**
4.1 Les listes de diffusion . . . 11
4.1.1 Basic rules for use . . . 11
4.1.2 Core development mailing lists . . . 11
4.1.3 Special lists . . . 11
4.1.4 Requesting new development-related lists . . . 12
4.2 IRC channels . . . 12
4.3 Documentation . . . 12
4.4 Debian machines . . . 12
4.4.1 The bugs server . . . 13
4.4.2 The ftp-master server . . . 13
4.4.3 The www-master server . . . 13
4.4.4 The people web server . . . 13
4.4.5 The VCS servers . . . 14
4.4.6 chroots to different distributions . . . 14
4.5 The Developers Database . . . 14
4.6 The Debian archive . . . 14
4.6.1 Sections . . . 16
4.6.2 Architectures . . . 16
4.6.3 Packages . . . 16
4.6.4 Distributions . . . 17
4.6.4.1 Stable, testing, and unstable . . . 17
4.6.4.2 More information about the testing distribution . . . 17
4.6.4.3 Experimental . . . 18
4.6.5 Release code names . . . 18
4.7 Debian mirrors . . . 18
4.8 The Incoming system . . . 19
4.9 Package information . . . 19
4.9.1 On the web . . . 19
4.9.2 The **dak ls** utility . . . 19
4.10 The Package Tracking System . . . 20
4.10.1 The PTS email interface . . . 20
4.10.2 Filtering PTS mails . . . 21
4.10.3 Forwarding VCS commits in the PTS . . . 22
4.10.4 The PTS web interface . . . 22
4.11 Developer's packages overview . . . 23
4.12 Debian's GForge installation: Alioth . . . 23
4.13 Goodies for Developers . . . 23

4.13.1 LWN Subscriptions . . . 23
4.13.2 Gandi.net Hosting Discount . . . 23

5 Gestion des paquets **25**
5.1 Nouveaux paquets . . . 25
5.2 Recording changes in the package . . . 26
5.3 Testing the package . . . 26
5.4 Layout of the source package . . . 26
5.5 Picking a distribution . . . 27
5.5.1 Special case: uploads to the `stable` and `oldstable` distributions . . . 27
5.5.2 Special case: uploads to `testing/testing-proposed-updates` . . . 28
5.6 Uploading a package . . . 28
5.6.1 Uploading to `ftp-master` . . . 28
5.6.2 Delayed uploads . . . 28
5.6.3 Security uploads . . . 28
5.6.4 Other upload queues . . . 28
5.6.5 Notification that a new package has been installed . . . 28
5.7 Specifying the package section, subsection and priority . . . 29
5.8 Handling bugs . . . 29
5.8.1 Monitoring bugs . . . 29
5.8.2 Responding to bugs . . . 30
5.8.3 Bug housekeeping . . . 30
5.8.4 When bugs are closed by new uploads . . . 31
5.8.5 Handling security-related bugs . . . 32
5.8.5.1 The Security Tracker . . . 32
5.8.5.2 Confidentiality . . . 32
5.8.5.3 Security Advisories . . . 33
5.8.5.4 Preparing packages to address security issues . . . 33
5.8.5.5 Uploading the fixed package . . . 34
5.9 Moving, removing, renaming, adopting, and orphaning packages . . . 34
5.9.1 Moving packages . . . 35
5.9.2 Removing packages . . . 35
5.9.2.1 Removing packages from `Incoming` . . . 36
5.9.3 Replacing or renaming packages . . . 36
5.9.4 Orphaning a package . . . 36
5.9.5 Adopting a package . . . 36
5.10 Porting and being ported . . . 37
5.10.1 Being kind to porters . . . 37
5.10.2 Guidelines for porter uploads . . . 38
5.10.2.1 Recompilation or binary-only NMU . . . 38
5.10.2.2 When to do a source NMU if you are a porter . . . 38
5.10.3 Porting infrastructure and automation . . . 39
5.10.3.1 Mailing lists and web pages . . . 39
5.10.3.2 Porter tools . . . 39
5.10.3.3 `wanna-build` . . . 39
5.10.4 When your package is *not* portable . . . 40
5.11 Non-Maintainer Uploads (NMUs) . . . 40
5.11.1 When and how to do an NMU . . . 40
5.11.2 NMUs and debian/changelog . . . 41
5.11.3 Using the `DELAYED/` queue . . . 42
5.11.4 NMUs from the maintainer's point of view . . . 42
5.11.5 Source NMUs vs Binary-only NMUs (binNMUs) . . . 42
5.11.6 NMUs vs QA uploads . . . 42
5.12 Collaborative maintenance . . . 43
5.13 The testing distribution . . . 43
5.13.1 Basics . . . 43
5.13.2 Updates from unstable . . . 43
5.13.2.1 out-of-date . . . 44
5.13.2.2 Removals from testing . . . 44

5.13.2.3 circular dependencies . . . 45
5.13.2.4 influence of package in testing . . . 45
5.13.2.5 details . . . 45
5.13.3 Direct updates to testing . . . 45
5.13.4 Frequently asked questions . . . 46
5.13.4.1 What are release-critical bugs, and how do they get counted? . . . 46
5.13.4.2 How could installing a package into `testing` possibly break other packages? . . . 46

6 Les meilleurs pratiques pour la construction des paquets 47
6.1 Les meilleures pratiques pour le fichier `debian/rules` . . . 47
6.1.1 Scripts d'assistance . . . 47
6.1.2 Séparation des modifications (« patches ») en plusieurs fichiers . . . 48
6.1.3 Paquets binaires multiples . . . 48
6.2 Meilleures pratiques pour `debian/control` . . . 48
6.2.1 Conseils généraux pour les descriptions de paquets . . . 48
6.2.2 Le résumé, ou description courte, d'un paquet . . . 49
6.2.3 La description longue . . . 49
6.2.4 Page d'accueil amont . . . 50
6.2.5 Emplacement du système de gestion de versions . . . 50
6.2.5.1 Vcs-Browser . . . 50
6.2.5.2 Vcs-* . . . 50
6.3 Meilleures pratiques pour `debian/changelog` . . . 51
6.3.1 Écrire des entrées de journalisation utiles . . . 51
6.3.2 Erreur communes pour les entrées de journaux de modifications . . . 51
6.3.3 Erreurs usuelles dans les entrées du journal des modifications . . . 52
6.3.4 Compléter les journaux de modifications avec des fichiers NEWS.Debian . . . 52
6.4 Meilleures pratiques pour les scripts du responsable . . . 53
6.5 Gestion de la configuration avec `debconf` . . . 54
6.5.1 N'abusez pas de `debconf` . . . 54
6.5.2 Recommandations générales pour les auteurs et les traducteurs . . . 54
6.5.2.1 Écrivez en anglais correct . . . 54
6.5.2.2 Pensez aux traducteurs . . . 54
6.5.2.3 Correction (« unfuzzy ») des traductions complètes lors des corrections de fautes de frappe ou d'orthographe . . . 55
6.5.2.4 Ne faites pas de suppositions sur les interfaces utilisateurs . . . 56
6.5.2.5 N'utilisez pas la première personne . . . 56
6.5.2.6 Restez neutre en genre . . . 56
6.5.3 Définition de champs des modèles (« templates ») . . . 56
6.5.3.1 Type . . . 56
6.5.3.1.1 string: . . . 56
6.5.3.1.2 password: . . . 57
6.5.3.1.3 boolean: . . . 57
6.5.3.1.4 select: . . . 57
6.5.3.1.5 multiselect: . . . 57
6.5.3.1.6 note: . . . 57
6.5.3.1.7 text: . . . 57
6.5.3.1.8 error: . . . 57
6.5.3.2 Description: description courte et étendue . . . 57
6.5.3.3 Choices . . . 58
6.5.3.4 Default . . . 58
6.5.4 Guide de style spécifique à certains modèles . . . 58
6.5.4.1 Champ type . . . 58
6.5.4.2 Champ Description . . . 58
6.5.4.2.1 Modèles string et password . . . 58
6.5.4.2.2 Modèles « boolean » . . . 58
6.5.4.2.3 Select/Multiselect . . . 58
6.5.4.2.4 Notes . . . 59
6.5.4.3 Champ Choices . . . 59

6.5.4.4 Champ Default . . . 59
6.5.4.5 Champ Default . . . 59
6.6 Internationalisation . . . 60
6.6.1 Gestion des traductions debconf . . . 60
6.6.2 Documentation internatonalisée . . . 60
6.7 Situation usuelle de gestion de paquets . . . 60
6.7.1 Paquets qui utilisent **autoconf/automake** . . . 60
6.7.2 Bibliothèques . . . 60
6.7.3 Documentation . . . 61
6.7.4 Type particuliers de paquets . . . 61
6.7.5 Données indépendantes de l'architecture . . . 61
6.7.6 Nécessitant des paramètres régionaux spécifiques lors de la construction . . . 61
6.7.7 Rendre les paquets de transition conformes à deborphan . . . 62
6.7.8 Les meilleures pratiques pour les fichiers `orig.tar.gz` . . . 62
6.7.8.1 Sources originelles (« pristine ») . . . 62
6.7.8.2 Source amont reconstruite . . . 62
6.7.8.3 Changing binary files in `diff.gz` . . . 63
6.7.9 Best practices for debug packages . . . 64

7 Au-delà de l'empaquetage **65**
7.1 Rapporter des bogues . . . 65
7.1.1 Reporting lots of bugs at once (mass bug filing) . . . 65
7.1.1.1 Usertags . . . 66
7.2 Quality Assurance effort . . . 66
7.2.1 Daily work . . . 66
7.2.2 Bug squashing parties . . . 66
7.3 Contacting other maintainers . . . 67
7.4 Dealing with inactive and/or unreachable maintainers . . . 67
7.5 Interacting with prospective Debian developers . . . 68
7.5.1 Sponsoring packages . . . 68
7.5.2 Managing sponsored packages . . . 68
7.5.3 Advocating new developers . . . 68
7.5.4 Handling new maintainer applications . . . 69

8 Internationalization and Translations **71**
8.1 How translations are handled within Debian . . . 71
8.2 I18N & L10N FAQ for maintainers . . . 72
8.2.1 How to get a given text translated . . . 72
8.2.2 How to get a given translation reviewed . . . 72
8.2.3 How to get a given translation updated . . . 72
8.2.4 How to handle a bug report concerning a translation . . . 72
8.3 I18N & L10N FAQ for translators . . . 72
8.3.1 How to help the translation effort . . . 72
8.3.2 How to provide a translation for inclusion in a package . . . 73
8.4 Best current practice concerning l10n . . . 73

A Aperçu des outils du responsable Debian **75**
A.1 Outils de base . . . 75
A.1.1 `dpkg-dev` . . . 75
A.1.2 `debconf` . . . 75
A.1.3 `fakeroot` . . . 75
A.2 Package lint tools . . . 76
A.2.1 `lintian` . . . 76
A.2.2 `debdiff` . . . 76
A.3 Helpers for `debian/rules` . . . 76
A.3.1 `debhelper` . . . 76
A.3.2 `debmake` . . . 76
A.3.3 `dh-make` . . . 77
A.3.4 `yada` . . . 77

A.3.5 equivs . . . 77
A.4 Package builders . . . 77
A.4.1 cvs-buildpackage . . . 77
A.4.2 debootstrap . . . 77
A.4.3 pbuilder . . . 77
A.4.4 sbuild . . . 78
A.5 Package uploaders . . . 78
A.5.1 dupload . . . 78
A.5.2 dput . . . 78
A.5.3 dcut . . . 78
A.6 Maintenance automation . . . 78
A.6.1 devscripts . . . 78
A.6.2 autotools-dev . . . 78
A.6.3 dpkg-repack . . . 78
A.6.4 alien . . . 79
A.6.5 debsums . . . 79
A.6.6 dpkg-dev-el . . . 79
A.6.7 dpkg-depcheck . . . 79
A.7 Porting tools . . . 79
A.7.1 quinn-diff . . . 79
A.7.2 dpkg-cross . . . 79
A.8 Documentation and information . . . 79
A.8.1 docbook-xml . . . 80
A.8.2 debiandoc-sgml . . . 80
A.8.3 debian-keyring . . . 80
A.8.4 debian-maintainers . . . 80
A.8.5 debview . . . 80

Chapitre 1

Portée de ce document

Le but de ce document est de donner une vue d'ensemble des procédures à suivre et des ressources mises à la disposition des développeurs Debian.

Les procédures décrites ci-après expliquent comment devenir responsableDebian (Chapitre 2), comment créer de nouveaux paquets(Section 5.1) et comment les installer dans l'archive (Section 5.6), comment gérer les rapports de bogues (Section 5.8), comment déplacer, effacer ou abandonner un paquet(Section 5.9), comment faire le portage d'un paquet (Section 5.10), quand et comment faire la mise à jour du paquet d'unautre responsable (Section 5.11).

Les ressources présentées dans ce manuel incluent les listes dediffusion (Section 4.1) et les serveurs (Section 4.4), une présentation de la structure de l'archiveDebian (Section 4.6), des explications sur les serveurs quiacceptent les mises à jour de paquets (Section 5.6.1) etune présentation des outils qui peuvent aider un responsable à améliorerla qualité de ses paquets (Annexe A).

Ce manuel de référence ne présente pas les aspects techniques liés auxpaquets Debian, ni comment les créer. Il ne décrit pas non plus lesrègles que doivent respecter les paquets Debian. Cette information estdisponible dans la charte Debian.

De plus ce document *n'est pas l'expression d'une politiqueofficielle*. Il contient de la documentation sur le système Debianet des conseils pratiques largement suivis. Ce n'est donc pas une sortede guide de normes.

Chapitre 2

Devenir responsable Debian

2.1 Pour commencer

Vous avez lu toute la documentation, vous avez examiné le guide du nouveau responsable, vous-comprenez l'intérêt de tout ce qui se trouve dans le paquet d'exemple`hello` et vous vous apprêtez à mettre en paquet votrelogiciel préféré. Comment devenir responsable Debian et intégrer votretravail au projet?

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 Section 4.1 . debian-devel-announce@lists.debian.org is another list which is mandatory for anyone who wishes to follow Debian's development.

Vous suivrez les discussions de cette liste (sans poster) pendantquelque temps avant de coder quoi que ce soit et vous informerez laliste de votre intention de travailler sur quelque chose pour éviter dedupliquer le travail d'un autre.

Une autre liste intéressante est debian-mentors@lists.debian.org. Voir lasection Section 2.2 pour les détails. Le canal IRC`#debian` pourra aussi être utile ; voir Section 4.2.

When you know how you want to contribute to Debian GNU/Linux, you should get in contact with existing Debian maintainers who are working on similar tasks. That way, you can learn from experienced developers. For example, if you are interested in packaging existing software for Debian, you should try to get a sponsor. A sponsor will work together with you on your package and upload it to the Debian archive once they are happy with the packaging work you have done. You can find a sponsor by mailing the debian-mentors@lists.debian.org mailing list, describing your package and yourself and asking for a sponsor (see Section 7.5.1 and `http://people.debian.org/~mpalmer/debian-mentors_FAQ.html` for more information on sponsoring). On the other hand, if you are interested in porting Debian to alternative architectures or kernels you can subscribe to port specific mailing lists and ask there how to get started. Finally, if you are interested in documentation or Quality Assurance (QA) work you can join maintainers already working on these tasks and submit patches and improvements.

One pitfall could be a too-generic local part in your mailadress: Terms like mail, admin, root, master should be avoided, please see `http://www.debian.org/MailingLists/` for details.

2.2 Mentors et parrains Debian

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 Section 4.1 for details).

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

In addition, if you have some packages ready for inclusion in Debian, but are waiting for your new maintainer application to go through, you might be able find a sponsor to upload your package for you. Sponsors are people who are official Debian Developers, and who are willing to criticize and upload your packages for you. Please read the unofficial debian-mentors FAQ at `http://people.debian.org/~mpalmer/debian-mentors_FAQ.html` first.

If you wish to be a mentor and/or sponsor, more information is available in Section 7.5 .

2.3 Registering as a Debian developer

Before you decide to register with Debian GNU/Linux, you will need to read all the information available at the New Maintainer's Corner. It describes in detail the preparations you have to do before you can register to become a Debian developer. For example, before you apply, you have to read the Debian 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 would also be a good idea.

The process of registering as a developer is a process of verifying your identity and intentions, and checking your technical skills. As the number of people working on Debian GNU/Linux has grown to over 900 and our systems are used in several very important places, we have to be careful about being compromised. Therefore, we need to verify new maintainers before we can give them accounts on our servers and let them upload packages.

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

Registration requires that you are familiar with Debian's philosophy and technical documentation. Furthermore, you need a GnuPG key which has been signed by an existing Debian maintainer. If your GnuPG key is not signed yet, you should try to meet a Debian Developer in person to get your key signed. There's a GnuPG Key Signing Coordination page which should help you find a Debian Developer close to you. (If there is no Debian Developer close to you, alternative ways to pass the ID check may be permitted as an absolute exception on a case-by-case-basis. See the identification page for more information.)

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 Section 3.2 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.

You need a version 4 key for use in Debian Development. 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. [1]

If your public key isn't on a public key server such as `subkeys.pgp.net`, please read the documentation available at NM Step 2: Identification. That document contains instructions on how to put your key on the public key servers. The New Maintainer Group will put your public key on the servers if it isn't already there.

Some countries restrict the use of cryptographic software by their citizens. This need not impede one's activities as a Debian package maintainer however, as it may be perfectly legal to use cryptographic products for authentication, rather than encryption purposes. 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.

[1] Version 4 keys are keys conforming to the OpenPGP standard as defined in RFC 2440. Version 4 is the key type that has always been created when using GnuPG. PGP versions since 5.x also could create v4 keys, the other choice having beein pgp 2.6.x compatible v3 keys (also called legacy RSA by PGP).

Version 4 (primary) keys can either use the RSA or the DSA algorithms, so this has nothing to do with GnuPG's question about which kind of key do you want: (1) DSA and Elgamal, (2) DSA (sign only), (5) RSA (sign only). If you don't have any special requirements just pick the default.

The easiest way to tell whether an existing key is a v4 key or a v3 (or v2) key is to look at the fingerprint: Fingerprints of version 4 keys are the SHA-1 hash of some key material, so they are 40 hex digits, usually grouped in blocks of 4. Fingerprints of older key format versions used MD5 and are generally shown in blocks of 2 hex digits. For example if your fingerprint looks like `5B00 C96D 5D54 AEE1 206B  AF84 DE7A AF6E 94C0 9C7F` then it's a v4 key.

Another possibility is to pipe the key into **pgpdump**, which will say something like Public Key Packet - Ver 4.

Also note that your key must be self-signed (i.e. it has to sign all its own user IDs; this prevents user ID tampering). All modern OpenPGP software does that automatically, but if you have an older key you may have to manually add those signatures.

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

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

For more details, please consult New Maintainer's Corner at the Debian web site. Make sure that you are familiar with the necessary steps of the New Maintainer process before actually applying. If you are well prepared, you can save a lot of time later on.

Chapitre 3

Les charges du responsable Debian

3.1 Mise à jour de vos références Debian

There's a LDAP database containing information about Debian developers at https://db.debian.org/. You should enter your information there and update it as it changes. Most notably, make sure that the address where your debian.org email gets forwarded to is always up to date, as well as the address where you get your debian-private subscription if you choose to subscribe there.

For more information about the database, please see Section 4.5 .

3.2 Gérer votre clé publique

Be very careful with your private keys. Do not place them on any public servers or multiuser machines, such as the Debian servers (see Section 4.4). Back your keys up; keep a copy offline. Read the documentation that comes with your software; read the PGP FAQ.

You need to ensure not only that your key is secure against being stolen, but also that it is secure against being lost. Generate and make a copy (best also in paper form) of your revocation certificate; this is needed if your key is lost.

If you add signatures to your public key, or add user identities, you can update the Debian key ring by sending your key to the key server at `keyring.debian.org`.

If you need to add a completely new key or remove an old key, you need to get the new key signed by another developer. If the old key is compromised or invalid, you also have to add the revocation certificate. If there is no real reason for a new key, the Keyring Maintainers might reject the new key. Details can be found at http://keyring.debian.org/replacing_keys.html.

The same key extraction routines discussed in Section 2.3 apply.

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

3.3 Voting

Even though Debian isn't really a democracy, we use a democratic process to elect our leaders and to approve general resolutions. These procedures are defined by the Debian Constitution.

Other than the yearly leader election, votes are not routinely held, and they are not undertaken lightly. Each proposal is first discussed on the debian-vote@lists.debian.org mailing list and it requires several endorsements before the project secretary starts the voting procedure.

You don't have to track the pre-vote discussions, as the secretary will issue several calls for votes on debian-devel-announce@lists.debian.org (and all developers are expected to be subscribed to that list). Democracy doesn't work well if people don't take part in the vote, which is why we encourage all developers to vote. Voting is conducted via GPG-signed/encrypted email messages.

The list of all proposals (past and current) is available on the Debian Voting Information page, along with information on how to make, second and vote on proposals.

3.4 Going on vacation gracefully

It is common for developers to have periods of absence, whether those are planned vacations or simply being buried in other work. The important thing to notice is that other developers need to know that you're on vacation so that they can do whatever is needed if a problem occurs with your packages or other duties in the project.

Usually this means that other developers are allowed to NMU (see Section 5.11) your package if a big problem (release critical bug, security update, etc.) occurs while you're on vacation. Sometimes it's nothing as critical as that, but it's still appropriate to let others know that you're unavailable.

In order to inform the other developers, there are two things that you should do. First send a mail to debian-private@lists.debian.org with [VAC] prepended to the subject of your message[1] and state the period of time when you will be on vacation. You can also give some special instructions on what to do if a problem occurs.

The other thing to do is to mark yourself as on vacation in the Debian developers' LDAP database (this information is only accessible to Debian developers). Don't forget to remove the on vacation flag when you come back!

Ideally, you should sign up at the GPG coordination site when booking a holiday and check if anyone there is looking for signing. This is especially important when people go to exotic places where we don't have any developers yet but where there are people who are interested in applying.

3.5 Coordination with upstream developers

A big part of your job as Debian maintainer will be to stay in contact with the upstream developers. Debian users will sometimes report bugs that are not specific to Debian to our bug tracking system. You have to forward these bug reports to the upstream developers so that they can be fixed in a future upstream release.

While it's not your job to fix non-Debian specific bugs, you may freely do so if you're able. When you make such fixes, be sure to pass them on to the upstream maintainers as well. Debian users and developers will sometimes submit patches to fix upstream bugs — you should evaluate and forward these patches upstream.

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

If you find that the upstream developers are or become hostile towards Debian or the free software community, you may want to re-consider the need to include the software in Debian. Sometimes the social cost to the Debian community is not worth the benefits the software may bring.

3.6 Managing release-critical bugs

Generally you should deal with bug reports on your packages as described in Section 5.8 . However, there's a special category of bugs that you need to take care of — the so-called release-critical bugs (RC bugs). All bug reports that have severity `critical`, `grave` or `serious` are considered to have an impact on whether the package can be released in the next stable release of Debian. These bugs can delay the Debian release and/or can justify the removal of a package at freeze time. That's why these bugs need to be corrected as quickly as possible.

Developers who are part of the Quality Assurance group are following all such bugs, and trying to help whenever possible. If, for any reason, you aren't able fix an RC bug in a package of yours within 2 weeks, you should either ask for help by sending a mail to the Quality Assurance (QA) group debian-qa@lists.debian.org, or explain your difficulties and present a plan to fix them by sending a mail to the bug report. Otherwise, people from the QA group may want to do a Non-Maintainer Upload (see Section 5.11) after trying to contact you (they might not wait as long as usual before they do their NMU if they have seen no recent activity from you in the BTS).

[1] This is so that the message can be easily filtered by people who don't want to read vacation notices.

3.7 Retiring

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

1. Orphan all your packages, as described in Section 5.9.4 .
2. Send an gpg-signed email about why you are leaving the project to debian-private@lists.debian.org.
3. Notify the Debian key ring maintainers that you are leaving by opening a ticket in Debian RT by sending a mail to keyring@rt.debian.org with the words 'Debian RT' somewhere in the subject line (case doesn't matter).

Chapitre 4

Ressources pour le responsable Debian

Dans ce chapitre, vous trouverez une brève description des listes dediffusion, des machines Debian qui seront à votre disposition en tantque responsable Debian ainsi que toutes les autres ressources à votredisposition pour vous aider dans votre travail de responsable.

4.1 Les listes de diffusion

Much of the conversation between Debian developers (and users) is managed through a wide array of mailing lists we host at `lists.debian.org`. To find out more on how to subscribe or unsubscribe, how to post and how not to post, where to find old posts and how to search them, how to contact the list maintainers and see various other information about the mailing lists, please read `http://www.debian.org/MailingLists/`. This section will only cover aspects of mailing lists that are of particular interest to developers.

4.1.1 Basic rules for use

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.

Cross-posting (sending the same message to multiple lists) is discouraged. 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.

Please read the code of conduct for more information. The Debian Community Guidelines are also worth reading.

4.1.2 Core development mailing lists

The core Debian mailing lists that developers should use are:

- debian-devel-announce@lists.debian.org, used to announce important things to developers. All developers are expected to be subscribed to this list.
- debian-devel@lists.debian.org, used to discuss various development related technical issues.
- debian-policy@lists.debian.org, where the Debian Policy is discussed and voted on.
- debian-project@lists.debian.org, used to discuss various non-technical issues related to the project.

There are other mailing lists available for a variety of special topics; see `http://lists.debian.org/` for a list.

4.1.3 Special lists

debian-private@lists.debian.org is a special mailing list for private discussions amongst Debian developers. It is meant to be used for posts which for whatever reason should not be published publicly. As such, it is a low volume list, and users are urged not to use debian-private@lists.debian.org unless it is really necessary. Moreover, do *not* forward email from that list to anyone. Archives of this list are not

available on the web for obvious reasons, but you can see them using your shell account on `master.debian.org` and looking in the `~debian/archive/debian-private/` directory.

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

4.1.4 Requesting new development-related lists

Before requesting a mailing list that relates to the development of a package (or a small group of related packages), please consider if using an alias (via a .forward-aliasname file on master.debian.org, which translates into a reasonably nice *you-aliasname@debian.org* address) or a self-managed mailing list on Alioth is more appropriate.

If you decide that a regular mailing list on lists.debian.org is really what you want, go ahead and fill in a request, following the HOWTO.

4.2 IRC channels

Several IRC channels are dedicated to Debian's development. They are mainly hosted on the Open and free technology community (OFTC) network. The `irc.debian.org` DNS entry is an alias to `irc.oftc.net`.

The main channel for Debian in general is `#debian`. This is a large, general-purpose channel where users can find recent news in the topic and served by bots. `#debian` is for English speakers; there are also `#debian.de`, `#debian-fr`, `#debian-br` and other similarly named channels for speakers of other languages.

The main channel for Debian development is `#debian-devel`. It is a very active channel since usually over 150 people are always logged in. It's a channel for people who work on Debian, it's not a support channel (there's `#debian` for that). It is however open to anyone who wants to lurk (and learn). Its topic is commonly full of interesting information for developers.

Since `#debian-devel` is an open channel, you should not speak there of issues that are discussed in debian-private@lists.debian.org. There's another channel for this purpose, it's called `#debian-private` and it's protected by a key. This key is available in the archives of debian-private in `master.debian.org:~debian/archive/debian-private/`, just **zgrep** for `#debian-private` in all the files.

There are other additional channels dedicated to specific subjects. `#debian-bugs` is used for coordinating bug squashing parties. `#debian-boot` is used to coordinate the work on the debian-installer. `#debian-doc` is occasionally used to talk about documentation, like the document you are reading. Other channels are dedicated to an architecture or a set of packages: `#debian-kde`, `#debian-dpkg`, `#debian-jr`, `#debian-edu`, `#debian-oo` (OpenOffice package) ...

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

Channels dedicated to Debian also exist on other IRC networks, notably on the freenode IRC network, which was pointed at by the `irc.debian.org` alias until 4th June 2006.

To get a cloak on freenode, you send Jörg Jaspert <joerg@debian.org> a signed mail where you tell what your nick is. Put cloak somewhere in the Subject: header. The nick should be registered: Nick Setup Page. The mail needs to be signed by a key in the Debian keyring. Please see Freenodes documentation for more information about cloaks.

4.3 Documentation

This document contains a lot of information which is useful to Debian developers, but it cannot contain everything. Most of the other interesting documents are linked from The Developers' Corner. Take the time to browse all the links, you will learn many more things.

4.4 Debian machines

Debian has several computers working as servers, most of which serve critical functions in the Debian project. Most of the machines are used for porting activities, and they all have a permanent connec-

tion to the Internet.

Some of the machines are available for individual developers to use, as long as the developers follow the rules set forth in the Debian Machine Usage Policies.

Generally speaking, you can use these machines 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 system administrators. Usually these machines are run by volunteers.

Please take care to protect your Debian passwords and SSH keys installed on Debian machines. Avoid login or upload methods which send passwords over the Internet in the clear, such as telnet, FTP, POP etc.

Please do not put any material that doesn't relate to Debian on the Debian servers, unless you have prior permission.

The current list of Debian machines is available at `http://db.debian.org/machines.cgi`. That web page contains machine names, contact information, information about who can log in, SSH keys etc.

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, you can check the list of open issues in the DSA queue of our request tracker at `https://rt.debian.org/` (you can login with user "guest" and password "readonly"). To report a new problem, simply send a mail to admin@rt.debian.org and make sure to put the string "Debian RT" somewhere in the subject.

If you have a problem with a certain service, not related to the system administration (such as packages to be removed from the archive, suggestions for the web site, etc.), generally you'll report a bug against a ``pseudo-package''. See Section 7.1 for information on how to submit bugs.

Some of the core servers are restricted, but the information from there is mirrored to another server.

4.4.1 The bugs server

`bugs.debian.org` is the canonical location for the Bug Tracking System (BTS).

It is restricted; a mirror is available on `merkel`.

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.

4.4.2 The ftp-master server

The `ftp-master.debian.org` server holds the canonical copy of the Debian archive. Generally, package uploads go to this server; see Section 5.6.

It is restricted; a mirror is available on `merkel`.

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 Section 5.9 .

4.4.3 The www-master server

The main web server is `www-master.debian.org`. It holds the official web pages, the face of Debian for most newbies.

If you find a problem with the Debian web server, you should generally submit a bug against the pseudo-package, `www.debian.org`. Remember to check whether or not someone else has already reported the problem to the Bug Tracking System.

4.4.4 The people web server

`people.debian.org` is the server used for developers' own web pages about anything related to Debian.

If you have some Debian-specific information which you want to serve on the web, you can do this by putting material in the `public_html` directory under your home directory on `people.debian.org`. This will be accessible at the URL `http://people.debian.org/~your-user-id/`.

You should only use this particular location because it will be backed up, whereas on other hosts it won't.

Usually the only reason to use a different host is when you need to publish materials subject to the U.S. export restrictions, in which case you can use one of the other servers located outside the United States.

Send mail to debian-devel@lists.debian.org if you have any questions.

4.4.5 The VCS servers

If you need to use a Version Control System for any of your Debian work, you can use one the existing repositories hosted on Alioth or you can request a new project and ask for the VCS repository of your choice. Alioth supports CVS (alioth.debian.org), Subversion (svn.debian.org), Arch (tla/baz, both on arch.debian.org), Bazaar (bzr.debian.org), Darcs (darcs.debian.org), Mercurial (hg.debian.org) and Git (git.debian.org). Checkout `http://wiki.debian.org/Alioth/PackagingProject` if you plan to maintain packages in a VCS repository. See Section 4.12 for information on the services provided by Alioth.

Historically, Debian first used `cvs.debian.org` to host CVS repositories. But that service is deprecated in favor of Alioth. Only a few projects are still using it.

4.4.6 chroots to different distributions

On some machines, there are chroots to different distributions available. You can use them like this:

```
vore$ dchroot unstable
Executing shell in chroot: /org/vore.debian.org/chroots/user/unstable
```

In all chroots, the normal user home directories are available. You can find out which chroots are available via `http://db.debian.org/machines.cgi`.

4.5 The Developers Database

The Developers Database, at `https://db.debian.org/`, is an LDAP directory for managing Debian developer attributes. You can use this resource to search the list of Debian developers. Part of this information is also available through the finger service on Debian servers, try **finger yourlogin@db.debian.org** to see what it reports.

Developers can log into the database to change various information about themselves, such as:

- forwarding address for your debian.org email
- subscription to debian-private
- whether you are on vacation
- personal information such as your address, country, the latitude and longitude of the place where you live for use in the world map of Debian developers, phone and fax numbers, IRC nickname and web page
- password and preferred shell on Debian Project machines

Most of the information is not accessible to the public, naturally. For more information please read the online documentation that you can find at `http://db.debian.org/doc-general.html`.

Developers can also submit their SSH keys to be used for authorization on the official Debian machines, and even add new *.debian.net DNS entries. Those features are documented at `http://db.debian.org/doc-mail.html`.

4.6 The Debian archive

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

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

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

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

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

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

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

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

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

As you can see, the top-level directory contains two directories, `dists/` and `pool/`. The latter is a "pool" in which the packages actually are, and which is handled by the archive maintenance database and the accompanying programs. The former contains the distributions, `stable`, `testing` and `unstable`. The `Packages` and `Sources` files in the distribution subdirectories can reference files in the `pool/` directory. The directory tree below each of the distributions is arranged in an identical manner. What we describe below for `stable` is equally applicable to the `unstable` and `testing` distributions.

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

In each of the areas, there is a directory for the source packages (`source`) and a directory for each

supported architecture (binary-i386, binary-m68k, etc.).

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

4.6.1 Sections

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

Every package in the main section must fully comply with the Debian Free Software Guidelines (DFSG) and with all other policy requirements as described in the Debian Policy Manual. The DFSG is our definition of "free software." Check out the Debian Policy Manual for details.

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

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

The Debian Policy Manual 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 it's allowed to. (Since this varies greatly from vendor to vendor, this job can't be done by the Debian developers.)

Note that the term section is also used to refer to categories which simplify the organization and browsing of available packages, e.g. admin, net, utils etc. Once upon a time, these sections (subsections, rather) existed in the form of subdirectories within the Debian archive. Nowadays, these exist only in the Section header fields of packages.

4.6.2 Architectures

In the first days, the Linux kernel was only available for Intel i386 (or greater) platforms, and so was Debian. But as 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 kernels. Aside from i386 (our name for Intel x86), there is m68k, alpha, powerpc, sparc, hurd-i386, arm, ia64, hppa, s390, mips, mipsel and sh as of this writing.

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

Information for developers and users about the specific ports are available at the Debian Ports web pages.

4.6.3 Packages

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

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

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

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

4.6.4 Distributions

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

4.6.4.1 Stable, testing, and unstable

There are always distributions called stable (residing in dists/stable), testing (residing in dists/testing), and 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 made to make sure everything in this distribution is working properly, it is sometimes literally unstable.

The testing distribution is generated automatically by taking packages from unstable if they satisfy certain criteria. Those criteria should ensure a good quality for packages within testing. The update to testing is launched twice each day, right after the new packages have been installed. See Section 5.13 .

After a period of development, once the release manager deems fit, the testing distribution is frozen, meaning that the policies which control how packages move from unstable to testing are tightened. Packages which are too buggy are removed. No changes are allowed into testing except for bug fixes. After some time has elapsed, depending on progress, the testing distribution is frozen even further. Details of the handling of the testing distribution are published by the Release Team on debian-devel-announce. After the open issues are solved to the satisfaction of the Release Team, the distribution is released. Releasing means that testing is renamed to stable, and a new copy is created for the new testing, and the previous stable is renamed to oldstable and stays there until it is finally archived. On archiving, the contents are moved to archive.debian.org).

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

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

4.6.4.2 More information about the testing distribution

Packages are usually installed into the testing distribution after they have undergone some degree of testing in unstable.

For more details, please see the information about the testing distribution.

4.6.4.3 Experimental

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

These are the sources.list(5) lines for `experimental`:

```
deb http://ftp.xy.debian.org/debian/ experimental main
deb-src http://ftp.xy.debian.org/debian/ experimental main
```

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

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

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

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

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

When uploading to `unstable` a package which had bugs fixed in `experimental`, please consider using the option `-v` to **dpkg-buildpackage** to finally get them closed.

4.6.5 Release code names

Every released Debian distribution has a `code name`: Debian 1.1 is called `buzz`; Debian 1.2, `rex`; Debian 1.3, `bo`; Debian 2.0, `hamm`; Debian 2.1, `slink`; Debian 2.2, `potato`; Debian 3.0, `woody`; Debian 3.1, `sarge`; Debian 4.0, `etch`; Debian 5.0, `lenny` and the next release will be called `squeeze`. There is also a ``pseudo-distribution'', called `sid`, which is the current `unstable` distribution; since packages are moved from `unstable` to `testing` as they approach stability, `sid` itself is never released. As well as the usual contents of a Debian distribution, `sid` contains packages for architectures which are not yet officially supported or released by Debian. These architectures are planned to be integrated into the mainstream distribution at some future date.

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

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

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

4.7 Debian mirrors

The various download archives and the web site have several mirrors available in order to relieve our canonical servers from heavy load. In fact, some of the canonical servers aren't public — a first tier of

mirrors balances the load instead. That way, users always access the mirrors and get used to using them, which allows Debian to better spread its bandwidth requirements over several servers and networks, and basically makes users avoid hammering on one primary location. Note that the first tier of mirrors is as up-to-date as it can be since they update when triggered from the internal sites (we call this push mirroring).

All the information on Debian mirrors, including a list of the available public FTP/HTTP servers, can be found at http://www.debian.org/mirror/. This useful page also 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.

4.8 The Incoming system

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

Packages are uploaded by all the maintainers into a directory called UploadQueue. This directory is scanned every few minutes by a daemon called **queued**, *.command-files are executed, and remaining and correctly signed *.changes-files are moved together with their corresponding files to the unchecked directory. This directory is not visible for most Developers, as ftp-master is restricted; it is scanned every 15 minutes by the **katie** script, which verifies the integrity of the uploaded packages and their cryptographic signatures. If the package is considered ready to be installed, it is moved into the accepted directory. If this is the first upload of the package (or it has new binary packages), it is moved to the new directory, where it waits for approval by the ftpmasters. If the package contains files to be installed by hand it is moved to the byhand directory, where it waits for manual installation by the ftpmasters. Otherwise, if any error has been detected, the package is refused and is moved to the reject directory.

Once the package is accepted, the system sends a confirmation mail to the maintainer and closes all the bugs marked as fixed by the upload, and the auto-builders may start recompiling it. The package is now publicly accessible at http://incoming.debian.org/ until it is really installed in the Debian archive. This happens only once a day (and is also called the `dinstall run' for historical reasons); the package is then removed from incoming and installed in the pool along with all the other packages. Once all the other updates (generating new Packages and Sources index files for example) have been made, a special script is called to ask all the primary mirrors to update themselves.

The archive maintenance software will also send the OpenPGP/GnuPG signed .changes file that you uploaded to the appropriate mailing lists. If a package is released with the Distribution: set to stable, the announcement is sent to debian-changes@lists.debian.org. If a package is released with Distribution: set to unstable or experimental, the announcement will be posted to debian-devel-changes@lists.debian.org instead.

Though ftp-master is restricted, a copy of the installation is available to all developers on merkel-.debian.org.

4.9 Package information

4.9.1 On the web

Each package has several dedicated web pages. http://packages.debian.org/*package-name* displays each version of the package available in the various distributions. Each version links to a page which provides information, including the package description, the dependencies, and package download links.

The bug tracking system tracks bugs for each package. You can view the bugs of a given package at the URL http://bugs.debian.org/*package-name*.

4.9.2 The dak ls utility

dak ls is part of the dak suite of tools, listing available package versions for all known distributions and architectures. The **dak** tool is available on ftp-master.debian.org , and on the mirror on m-

erkel.debian.org. It uses a single argument corresponding to a package name. An example will explain it better:

```
$ dak ls evince
evince | 0.1.5-2sarge1 |     oldstable | source, alpha, arm, hppa, i386, ia64,  ↩
    m68k, mips, mipsel, powerpc, s390, sparc
evince |    0.4.0-5 |     etch-m68k | source, m68k
evince |    0.4.0-5 |        stable | source, alpha, amd64, arm, hppa, i386, ia64 ↩
    , mips, mipsel, powerpc, s390, sparc
evince |   2.20.2-1 |       testing | source
evince | 2.20.2-1+b1 |       testing | alpha, amd64, arm, armel, hppa, i386, ia64 ↩
    , mips, mipsel, powerpc, s390, sparc
evince |   2.22.2-1 |      unstable | source, alpha, amd64, arm, armel, hppa,  ↩
    i386, ia64, m68k, mips, mipsel, powerpc, s390, sparc
```

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

4.10 The Package Tracking System

The Package Tracking System (PTS) is an email-based tool to track the activity of a source package. This really means that you can get the same emails that the package maintainer gets, simply by subscribing to the package in the PTS.

Each email sent through the PTS is classified under one of the keywords listed below. This will let you select the mails that you want to receive.

By default you will get:

bts All the bug reports and following discussions.

bts-control The email notifications from control@bugs.debian.org about bug report status changes.

upload-source The email notification from **katie** when an uploaded source package is accepted.

katie-other Other warning and error emails from **katie** (such as an override disparity for the section and/or the priority field).

default Any non-automatic email sent to the PTS by people who wanted to contact the subscribers of the package. This can be done by sending mail to *sourcepackage*@packages.qa.debian.org. In order to prevent spam, all messages sent to these addresses must contain the X-PTS-Approved header with a non-empty value.

contact Mails sent to the maintainer through the *@packages.debian.org email aliases.

summary Regular summary emails about the package's status. Currently, only progression in testing is sent.

You can also decide to receive additional information:

upload-binary The email notification from **katie** when an uploaded binary package is accepted. In other words, whenever a build daemon or a porter uploads your package for another architecture, you can get an email to track how your package gets recompiled for all architectures.

cvs VCS commit notifications, if the package has a VCS repository and the maintainer has set up forwarding of commit notifications to the PTS. The "cvs" name is historic, in most cases commit notifications will come from some other VCS like subversion or git.

ddtp Translations of descriptions or debconf templates submitted to the Debian Description Translation Project.

derivatives Information about changes made to the package in derivative distributions (for example Ubuntu).

4.10.1 The PTS email interface

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

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

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

unsubscribeall [<email>] Removes all subscriptions of the specified email address or the sender address if the second argument is left out.

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

keyword [<email>] Tells you the keywords that you are accepting. For an explanation of keywords, see above. Here's a quick summary:

- `bts`: mails coming from the Debian Bug Tracking System
- `bts-control`: reply to mails sent to control@bugs.debian.org
- `summary`: automatic summary mails about the state of a package
- `contact`: mails sent to the maintainer through the *@packages.debian.org aliases
- `cvs`: notification of VCS commits
- `ddtp`: translations of descriptions and debconf templates
- `derivatives`: changes made on the package by derivative distributions
- `upload-source`: announce of a new source upload that has been accepted
- `upload-binary`: announce of a new binary-only upload (porting)
- `katie-other`: other mails from ftpmasters (override disparity, etc.)
- `default`: all the other mails (those which aren't automatic)

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

keyword [<email>] {+|-|=} <list of keywords> Accept (+) or refuse (-) mails classified under the given keyword(s). Define the list (=) of accepted keywords. This changes the default set of keywords accepted by a user.

keywordall [<email>] {+|-|=} <list of keywords> Accept (+) or refuse (-) mails classified under the given keyword(s). Define the list (=) of accepted keywords. This changes the set of accepted keywords of all the currently active subscriptions of a user.

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

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

The **pts-subscribe** command-line utility (from the `devscripts` package) can be handy to temporarily subscribe to some packages, for example after having made an non-maintainer upload.

4.10.2 Filtering PTS mails

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

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

```
X-Loop: dpkg@packages.qa.debian.org
X-PTS-Package: dpkg
X-PTS-Keyword: upload-source
List-Unsubscribe: <mailto:pts@qa.debian.org?body=unsubscribe+dpkg>
```

4.10.3 Forwarding VCS commits in the PTS

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

Once you set up the VCS repository to generate commit notifications, you just have to make sure it sends a copy of those mails to *sourcepackage*_cvs@packages.qa.debian.org. Only the people who accept the cvs keyword will receive these notifications. Note that the mail need to be sent from a debian.org machine, otherwise you'll have to add the X-PTS-Approved: 1 header.

For Subversion repositories, the usage of svnmailer is recommended. See http://wiki.debian.org/Alioth/PackagingProject for an example on how to do it.

4.10.4 The PTS web interface

The PTS has a web interface at http://packages.qa.debian.org/ that puts together a lot of information about each source package. It features many useful links (BTS, QA stats, contact information, DDTP translation status, buildd logs) and gathers much more information from various places (30 latest changelog entries, testing status, ...). It's a very useful tool if you want to know what's going on with a specific source package. Furthermore there's a form that allows easy subscription to the PTS via email.

You can jump directly to the web page concerning a specific source package with a URL like http://packages.qa.debian.org/*sourcepackage*.

This web interface has been designed like a portal for the development of packages: you can add custom content on your packages' pages. You can add static information (news items that are meant to stay available indefinitely) and news items in the latest news section.

Static news items can be used to indicate:

- the availability of a project hosted on Alioth for co-maintaining the package
- a link to the upstream web site
- a link to the upstream bug tracker
- the existence of an IRC channel dedicated to the software
- any other available resource that could be useful in the maintenance of the package

Usual news items may be used to announce that:

- beta packages are available for testing
- final packages are expected for next week
- the packaging is about to be redone from scratch
- backports are available
- the maintainer is on vacation (if they wish to publish this information)
- a NMU is being worked on
- something important will affect the package

Both kinds of news are generated in a similar manner: you just have to send an email either to pts-static-news@qa.debian.org or to pts-news@qa.debian.org. The mail should indicate which package is concerned by having the name of the source package in a X-PTS-Package mail header or in a Package pseudo-header (like the BTS reports). If a URL is available in the X-PTS-Url mail header or in the Url pseudo-header, then the result is a link to that URL instead of a complete news item.

Here are a few examples of valid mails used to generate news items in the PTS. The first one adds a link to the cvsweb interface of debian-cd in the Static information section:

```
From: Raphael Hertzog <hertzog@debian.org>
To: pts-static-news@qa.debian.org
Subject: Browse debian-cd SVN repository

Package: debian-cd
Url: http://svn.debian.org/viewsvn/debian-cd/trunk/
```

The second one is an announcement sent to a mailing list which is also sent to the PTS so that it is published on the PTS web page of the package. Note the use of the BCC field to avoid answers sent to the PTS by mistake.

```
From: Raphael Hertzog <hertzog@debian.org>
To: debian-gtk-gnome@lists.debian.org
Bcc: pts-news@qa.debian.org
Subject: Galeon 2.0 backported for woody
X-PTS-Package: galeon

Hello gnomers!

I'm glad to announce that galeon has been backported for woody. You'll find
everything here:
...
```

Think twice before adding a news item to the PTS because you won't be able to remove it later and you won't be able to edit it either. The only thing that you can do is send a second news item that will deprecate the information contained in the previous one.

4.11 Developer's packages overview

A QA (quality assurance) web portal is available at http://qa.debian.org/developer.php which displays a table listing all the packages of a single developer (including those where the party is listed as a co-maintainer). The table gives a good summary about the developer's packages: number of bugs by severity, list of available versions in each distribution, testing status and much more including links to any other useful information.

It is a good idea to look up your own data regularly so that you don't forget any open bugs, and so that you don't forget which packages are your responsibility.

4.12 Debian's GForge installation: Alioth

Alioth is a Debian service based on a slightly modified version of the GForge software (which evolved from SourceForge). This software offers developers access to easy-to-use tools such as bug trackers, patch manager, project/task managers, file hosting services, mailing lists, CVS repositories etc. All these tools are managed via a web interface.

It is intended to provide facilities to free software projects backed or led by Debian, facilitate contributions from external developers to projects started by Debian, and help projects whose goals are the promotion of Debian or its derivatives. It's heavily used by many Debian teams and provides hosting for all sorts of VCS repositories.

All Debian developers automatically have an account on Alioth. They can activate it by using the recover password facility. External developers can request guest accounts on Alioth.

For more information please visit the following links:

- http://wiki.debian.org/Alioth
- http://wiki.debian.org/Alioth/FAQ
- http://wiki.debian.org/Alioth/PackagingProject
- http://alioth.debian.org/

4.13 Goodies for Developers

4.13.1 LWN Subscriptions

Since October of 2002, HP has sponsored a subscription to LWN for all interested Debian developers. Details on how to get access to this benefit are in http://lists.debian.org/debian-devel-announce/2002/10/msg00018.html.

4.13.2 Gandi.net Hosting Discount

As of November 2008, Gandi.net offers a discount rate on their VPS hosting for Debian Developers. See http://lists.debian.org/debian-devel-announce/2008/11/msg00004.html.

Chapitre 5

Gestion des paquets

Ce chapitre contient des informations relatives à la création, l'envoi,la maintenance et le portage des paquets.

5.1 Nouveaux paquets

If you want to create a new package for the Debian distribution, you should first check the Work-Needing and Prospective Packages (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 for more information.

Assuming no one else is already working on your prospective package, you must then submit a bug report (Section 7.1) 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`. Please send a copy to debian-devel@lists.debian.org by using the X-Debbugs-CC header (don't use CC:, because that way the message's subject won't indicate the bug number). If you are packaging so many new packages (>10) that notifying the mailing list in seperate messages is too disruptive, do send a summary after filing the bugs to the debian-devel list instead. This will inform the other developers about upcoming packages and will allow a review of your description and package name.

Please include a `Closes: bug#`*nnnnn* entry in the changelog of the new package in order for the bug report to be automatically closed once the new package is installed in the archive (see Section 5.8.4).

If you think your package needs some explanations for the administrators of the NEW package queue, include them in your changelog, send to ftpmaster@debian.org a reply to the email you receive as a maintainer after your upload, or reply to the rejection email in case you are already re-uploading.

When closing security bugs include CVE numbers as well as the Closes: #nnnnn. This is useful for the security team to track vulnerabilities. If an upload is made to fix the bug before the advisory ID is known, it is encouraged to modify the historical changelog entry with the next upload. Even in this case, please include all available pointers to background information in the original changelog entry.

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@lists.debian.org.
- 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.

Please see http://ftp-master.debian.org/REJECT-FAQ.html for common rejection reasons for a new package.

5.2 Recording changes in the package

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

The debian/changelog file conforms to a certain structure, with a number of different fields. One field of note, the distribution, is described in Section 5.5 . More information about the structure of this file can be found in the Debian Policy section titled debian/changelog.

Changelog entries can be used to automatically close Debian bugs when the package is installed into the archive. See Section 5.8.4 .

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

```
* new upstream version
```

There are tools to help you create entries and finalize the changelog for release — see Section A.6.1 and Section A.6.6 .

See also Section 6.3 .

5.3 Testing the package

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 Section A.2.1 .
- Optionally run Section A.2.2 to analyze changes from an older version, if one exists.
- Downgrade the package to the previous version (if one exists) — this tests the postrm and prerm scripts.
- Remove the package, then reinstall it.
- Copy the source package in a different directory and try unpacking it and rebuilding it. This tests if the package relies on existing files outside of it, or if it relies on permissions being preserved on the files shipped inside the .diff.gz file.

5.4 Layout of the source package

There are two types of Debian source packages:

- the so-called native packages, where there is no distinction between the original sources and the patches applied for Debian
- the (more common) packages where there's an original source tarball file accompanied by another file that contains the patches applied for Debian

For the native packages, the source package includes a Debian source control file (`.dsc`) and the source tarball (`.tar.gz`). A source package of a non-native package includes a Debian source control file, the original source tarball (`.orig.tar.gz`) and the Debian patches (`.diff.gz`).

Whether a package is native or not is determined when it is built by dpkg-buildpackage(1). The rest of this section relates only to non-native packages.

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

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

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

Please notice that, in non-native packages, permissions on files that are not present in the .orig.tar.gz will not be preserved, as diff does not store file permissions in the patch.

5.5 Picking a distribution

Each upload needs to specify which distribution the package is intended for. The package build process extracts this information from the first line of the `debian/changelog` file and places it in the `Distribution` field of the `.changes` file.

There are several possible values for this field: `stable`, `unstable`, `testing-proposed-updates` and `experimental`. Normally, packages are uploaded into `unstable`.

Actually, there are two other possible distributions: `stable-security` and `testing-security`, but read Section 5.8.5 for more information on those.

It is not possible to upload a package into several distributions at the same time.

5.5.1 Special case: uploads to the `stable` and `oldstable` distributions

Uploading to `stable` means that the package will transfered to the `proposed-updates-new` queue for review by the stable release managers, and if approved will be installed in `stable-proposed-updates` directory of the Debian archive. From there, it will be included in `stable` with the next point release.

To ensure that your upload will be accepted, you should discuss the changes with the stable release team before you upload. For that, send a mail to the debian-release@lists.debian.org mailing list, including the patch you want to apply to the package version currently in `stable`. Always be verbose and detailed in your changelog entries for uploads to the `stable` distribution.

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

- a truly critical functionality problem
- the package becomes uninstallable
- a released architecture lacks the package

In the past, uploads to `stable` were used to address security problems as well. However, this practice is deprecated, as uploads used for Debian security advisories are automatically copied to the appropriate `proposed-updates` archive when the advisory is released. See Section 5.8.5 for detailed information on handling security problems. If the security teams deems the problem to be too benign to be fixed through a `DSA`, the stable release managers are usually willing to include your fix nonetheless in a regular upload to `stable`.

Changing anything else in the package that isn't important is discouraged, because even trivial fixes can cause bugs later on.

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

Uploads to the `oldstable` distributions are possible as long as it hasn't been archived. The same rules as for `stable` apply.

5.5.2 Special case: uploads to `testing/testing-proposed-updates`

Please see the information in the testing section for details.

5.6 Uploading a package

5.6.1 Uploading to `ftp-master`

To upload a package, you should upload the files (including the signed changes and dsc-file) with anonymous ftp to `ftp-master.debian.org` in the directory `/pub/UploadQueue/`. To get the files processed there, they need to be signed with a key in the Debian Developers keyring or the Debian Maintainers keyring (see `http://wiki.debian.org/Maintainers`).

Please note that you should transfer the changes file last. Otherwise, your upload may be rejected because the archive maintenance software will parse the changes file and see that not all files have been uploaded.

You may also find the Debian packages Section A.5.1 or Section A.5.2 useful when uploading packages. These handy programs help automate the process of uploading packages into Debian.

For removing packages, please see the README file in that ftp directory, and the Debian package Section A.5.3 .

5.6.2 Delayed uploads

It is sometimes useful to upload a package immediately, but to want this package to arrive in the archive only a few days later. For example, when preparing a Non-maintainer Upload, you might want to give the maintainer a few days to react.

An upload to the delayed directory keeps the package in the deferred uploads queue". When the specified waiting time is over, the package is moved into the regular incoming directory for processing. This is done through automatic uploading to `ftp-master.debian.org` in upload-directory `DELAYED/[012345678]-day`. 0-day is uploaded multiple times per day to `ftp-master.debian.org`.

With dput, you can use the `--delayed` *DELAY* parameter to put the package into one of the queues.

5.6.3 Security uploads

Do **NOT** upload a package to the security upload queue (`oldstable-security`, `stable-security` , etc.) without prior authorization from the security team. If the package does not exactly meet the team's requirements, it will cause many problems and delays in dealing with the unwanted upload. For details, please see section Section 5.8.5 .

5.6.4 Other upload queues

The scp queues on `ftp-master.debian.org`, and `security.debian.org` are mostly unusable due to the login restrictions on those hosts.

The anonymous queues on ftp.uni-erlangen.de and ftp.uk.debian.org are currently down. Work is underway to resurrect them.

The queues on master.debian.org, samosa.debian.org, master.debian.or.jp, and ftp.chiark.greenend.org.uk are down permanently, and will not be resurrected. The queue in Japan will be replaced with a new queue on hp.debian.or.jp some day.

5.6.5 Notification that a new package has been installed

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

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

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

Note that if you upload via queues, the queue daemon software will also send you a notification by email.

5.7 Specifying the package section, subsection and priority

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

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

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

For more information about `override files`, see dpkg-scanpackages(1) and `http://www.debian.org/Bugs/Developer#maintincorrect`.

Note that the `Section` field describes both the section as well as the subsection, which are described in Section 4.6.1 . If the section is main, it should be omitted. The list of allowable subsections can be found in `http://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections`.

5.8 Handling bugs

Every developer has to be able to work with the Debian bug tracking system. This includes knowing how to file bug reports properly (see Section 7.1), how to update them and reorder them, and how to process and close them.

The bug tracking system's features are described in the BTS documentation for developers. This includes closing bugs, sending followup messages, assigning severities and tags, marking bugs as forwarded, and other issues.

Operations such as reassigning bugs to other packages, merging separate bug reports about the same issue, or reopening bugs when they are prematurely closed, are handled using the so-called control mail server. All of the commands available on this server are described in the BTS control server documentation.

5.8.1 Monitoring bugs

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

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

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

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

Replace *`address`* with your official Debian maintainer address.

5.8.2 Responding to bugs

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

If you get a bug which mentions FTBFS, this means Fails to build from source. Porters frequently use this acronym.

Once you've dealt with a bug report (e.g. fixed it), mark it as `done` (close it) by sending an explanation message to 123-done@bugs.debian.org. If you're fixing a bug by changing and uploading the package, you can automate bug closing as described in Section 5.8.4 .

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

5.8.3 Bug housekeeping

As a package maintainer, you will often find bugs in other packages or have bugs reported against your packages which are actually bugs in other packages. The bug tracking system's features are described in the BTS documentation for Debian developers. Operations such as reassigning, merging, and tagging bug reports are described in the BTS control server documentation. This section contains some guidelines for managing your own bugs, based on the collective Debian developer experience.

Filing bugs for problems that you find in other packages is one of the civic obligations of maintainership, see Section 7.1 for details. However, handling the bugs in your own packages is even more important.

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

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

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

2. If the bug is real but it's caused by another package, just reassign the bug to the right package. If you don't know which package it should be reassigned to, you should ask for help on IRC or on debian-devel@lists.debian.org. Please inform the maintainer(s) of the package you reassign the bug to, for example by Cc:ing the message that does the reassign to packagename@packages.debian.org and explaining your reasons in that mail. Please note that a simple reassignment is *not* e-mailed to the maintainers of the package being reassigned to, so they won't know about it until they look at a bug overview for their packages.

   If the bug affects the operation of your package, please consider cloning the bug and reassigning the clone to the package that really causes the behavior. Otherwise, the bug will not be shown in your package's bug list, possibly causing users to report the same bug over and over again. You should block "your" bug with the reassigned, cloned bug to document the relationship.

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

4. If the bug is real but the same problem has already been reported by someone else, then the two relevant bug reports should be merged into one using the merge command of the BTS. In this way, when the bug is fixed, all of the submitters will be informed of this. (Note, however, that emails sent to one bug report's submitter won't automatically be sent to the other report's submitter.) For

more details on the technicalities of the merge command and its relative, the unmerge command, see the BTS control server documentation.

5. The bug submitter may have forgotten to provide some information, in which case you have to ask them for the required information. You may use the `moreinfo` tag to mark the bug as such. Moreover if you can't reproduce the bug, you tag it `unreproducible`. Anyone who can reproduce the bug is then invited to provide more information on how to reproduce it. After a few months, if this information has not been sent by someone, the bug may be closed.
6. If the bug is related to the packaging, you just fix it. If you are not able to fix it yourself, then tag the bug as `help`. You can also ask for help on debian-devel@lists.debian.org or debian-qa@lists.debian.org. If it's an upstream problem, you have to forward it to the upstream author. Forwarding a bug is not enough, you have to check at each release if the bug has been fixed or not. If it has, you just close it, otherwise you have to remind the author about it. If you have the required skills you can prepare a patch that fixes the bug and send it to the author at the same time. Make sure to send the patch to the BTS and to tag the bug as `patch`.
7. If you have fixed a bug in your local copy, or if a fix has been committed to the CVS repository, you may tag the bug as `pending` to let people know that the bug is corrected and that it will be closed with the next upload (add the `closes:` in the `changelog`). This is particularly useful if you are several developers working on the same package.
8. Once a corrected package is available in the archive, the bug should be closed indicating the version in which it was fixed. This can be done automatically, read Section 5.8.4.

5.8.4 When bugs are closed by new uploads

As bugs and problems are fixed in your packages, it is your responsibility as the package maintainer to close these bugs. However, you should not close a 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. Also, the bug should be closed with the correct version.

However, it's possible to avoid having to manually close bugs after the upload — just list the fixed bugs in your `debian/changelog` file, following a certain syntax, and the archive maintenance software will close the bugs for you. For example:

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

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

Technically speaking, the following Perl regular expression describes how bug closing changelogs are identified:

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

We prefer the `closes: #`*XXX* syntax, as it is the most concise entry and the easiest to integrate with the text of the `changelog`. Unless specified different by the `-v`-switch to **dpkg-buildpackage**, only the bugs closed in the most recent changelog entry are closed (basically, exactly the bugs mentioned in the changelog-part in the `.changes` file are closed).

Historically, uploads identified as Non-maintainer upload (NMU) were tagged `fixed` instead of being closed, but that practice was ceased with the advent of version-tracking. The same applied to the tag `fixed-in-experimental`.

If you happen to mistype a bug number or forget a bug in the changelog entries, don't hesitate to undo any damage the error caused. To reopen wrongly closed bugs, send a `reopen` *XXX* command to the bug tracking system's control address, control@bugs.debian.org. To close any remaining bugs that were fixed by your upload, email the `.changes` file to XXX-done@bugs.debian.org, where *XXX* is the bug number, and put Version: YYY and an empty line as the first two lines of the body of the email, where *YYY* is the first version where the bug has been fixed.

Bear in mind that it is not obligatory to close bugs using the changelog as described above. If you simply want to close bugs that don't have anything to do with an upload you made, do it by emailing

an explanation to XXX-done@bugs.debian.org. Do **not** close bugs in the changelog entry of a version if the changes in that version of the package don't have any bearing on the bug.

For general information on how to write your changelog entries, see Section 6.3 .

5.8.5 Handling security-related bugs

Due to their sensitive nature, security-related bugs must be handled carefully. The Debian Security Team exists to coordinate this activity, keeping track of outstanding security problems, helping maintainers with security problems or fixing them themselves, sending security advisories, and maintaining `security.debian.org`.

When you become aware of a security-related bug in a Debian package, whether or not you are the maintainer, collect pertinent information about the problem, and promptly contact the security team at team@security.debian.org as soon as possible. **DO NOT UPLOAD** any packages for `stable` without contacting the team. Useful information includes, for example:

- Which versions of the package are known to be affected by the bug. Check each version that is present in a supported Debian release, as well as `testing` and `unstable`.
- The nature of the fix, if any is available (patches are especially helpful)
- Any fixed packages that you have prepared yourself (send only the `.diff.gz` and `.dsc` files and read Section 5.8.5.4 first)
- Any assistance you can provide to help with testing (exploits, regression testing, etc.)
- Any information needed for the advisory (see Section 5.8.5.3)

As the maintainer of the package, you have the responsibility to maintain it, even in the stable release. You are in the best position to evaluate patches and test updated packages, so please see the sections below on how to prepare packages for the Security Team to handle.

5.8.5.1 The Security Tracker

The security team maintains a central database, the Debian Security Tracker. This contains all public information that is known about security issues: which packages and versions are affected or fixed, and thus whether stable, testing and/or unstable are vulnerable. Information that is still confidential is not added to the tracker.

You can search it for a specific issue, but also on package name. Look for your package to see which issues are still open. If you can, please provide more information about those issues, or help to address them in your package. Instructions are on the tracker web pages.

5.8.5.2 Confidentiality

Unlike most other activities within Debian, information about security issues must sometimes be kept private for a time. This allows software distributors to coordinate their disclosure in order to minimize their users' exposure. Whether this is the case depends on the nature of the problem and corresponding fix, and whether it is already a matter of public knowledge.

There are several ways developers can learn of a security problem:

- they notice it on a public forum (mailing list, web site, etc.)
- someone files a bug report
- someone informs them via private email

In the first two cases, the information is public and it is important to have a fix as soon as possible. In the last case, however, it might not be public information. In that case there are a few possible options for dealing with the problem:

- If the security exposure is minor, there is sometimes no need to keep the problem a secret and a fix should be made and released.
- If the problem is severe, it is preferable to share the information with other vendors and coordinate a release. The security team keeps in contact with the various organizations and individuals and can take care of that.

In all cases if the person who reports the problem asks that it not be disclosed, such requests should be honored, with the obvious exception of informing the security team in order that a fix may be produced for a stable release of Debian. When sending confidential information to the security team, be sure to mention this fact.

Please note that if secrecy is needed you may not upload a fix to `unstable` (or anywhere else, such as a public CVS repository). It is not sufficient to obfuscate the details of the change, as the code itself is public, and can (and will) be examined by the general public.

There are two reasons for releasing information even though secrecy is requested: the problem has been known for a while, or the problem or exploit has become public.

The Security Team has a PGP-key to enable encrypted communication about sensitive issues. See the Security Team FAQ for details.

5.8.5.3 Security Advisories

Security advisories are only issued for the current, released stable distribution, and *not* for `testing` or `unstable`. When released, advisories are sent to the debian-security-announce@lists.debian.org mailing list and posted on the security web page. Security advisories are written and posted by the security team. However they certainly do not mind if a maintainer can supply some of the information for them, or write part of the text. Information that should be in an advisory includes:

- A description of the problem and its scope, including:
  - The type of problem (privilege escalation, denial of service, etc.)
  - What privileges may be gained, and by whom (if any)
  - How it can be exploited
  - Whether it is remotely or locally exploitable
  - How the problem was fixed

  This information allows users to assess the threat to their systems.
- Version numbers of affected packages
- Version numbers of fixed packages
- Information on where to obtain the updated packages (usually from the Debian security archive)
- References to upstream advisories, CVE identifiers, and any other information useful in cross-referencing the vulnerability

5.8.5.4 Preparing packages to address security issues

One way that you can assist the security team in their duties is to provide them with fixed packages suitable for a security advisory for the stable Debian release.

When an update is made to the stable release, care must be taken to avoid changing system behavior or introducing new bugs. In order to do this, make as few changes as possible to fix the bug. Users and administrators rely on the exact behavior of a release once it is made, so any change that is made might break someone's system. This is especially true of libraries: make sure you never change the API or ABI, no matter how small the change.

This means that moving to a new upstream version is not a good solution. Instead, the relevant changes should be back-ported to the version present in the current stable Debian release. Generally, upstream maintainers are willing to help if needed. If not, the Debian security team may be able to help.

In some cases, it is not possible to back-port a security fix, for example when large amounts of source code need to be modified or rewritten. If this happens, it may be necessary to move to a new upstream version. However, this is only done in extreme situations, and you must always coordinate that with the security team beforehand.

Related to this is another important guideline: always test your changes. If you have an exploit available, try it and see if it indeed succeeds on the unpatched package and fails on the fixed package. Test other, normal actions as well, as sometimes a security fix can break seemingly unrelated features in subtle ways.

Do **NOT** include any changes in your package which are not directly related to fixing the vulnerability. These will only need to be reverted, and this wastes time. If there are other bugs in your package that you would like to fix, make an upload to proposed-updates in the usual way, after the security advisory

is issued. The security update mechanism is not a means for introducing changes to your package which would otherwise be rejected from the stable release, so please do not attempt to do this.

Review and test your changes as much as possible. Check the differences from the previous version repeatedly (**interdiff** from the `patchutils` package and **debdiff** from `devscripts` are useful tools for this, see Section A.2.2).

Be sure to verify the following items:

- **Target the right distribution** in your `debian/changelog`. For `stable` this is `stable-security` and for testing this is `testing-security`, and for the previous stable release, this is `oldstable-security`. Do not target *`distribution`*`-proposed-updates` or `stable`!
- The upload should have **urgency=high**.
- Make descriptive, meaningful changelog entries. Others will rely on them to determine whether a particular bug was fixed. Add `closes:` statements for any **Debian bugs** filed. Always include an external reference, preferably a **CVE identifier**, so that it can be cross-referenced. However, if a CVE identifier has not yet been assigned, do not wait for it but continue the process. The identifier can be cross-referenced later.
- Make sure the **version number** is proper. It must be greater than the current package, but less than package versions in later distributions. If in doubt, test it with `dpkg --compare-versions`. Be careful not to re-use a version number that you have already used for a previous upload, or one that conflicts with a binNMU. The convention is to append `+`*`codename`*`1`, e.g. `1:2.4.3-4+etch1`, of course increasing 1 for any subsequent uploads.
- Unless the upstream source has been uploaded to `security.debian.org` before (by a previous security update), build the upload **with full upstream source** (`dpkg-buildpackage -sa`). If there has been a previous upload to `security.debian.org` with the same upstream version, you may upload without upstream source (`dpkg-buildpackage -sd`).
- Be sure to use the **exact same `*.orig.tar.gz`** as used in the normal archive, otherwise it is not possible to move the security fix into the main archives later.
- Build the package on a **clean system** which only has packages installed from the distribution you are building for. If you do not have such a system yourself, you can use a debian.org machine (see Section 4.4) or setup a chroot (see Section A.4.3 and Section A.4.2).

5.8.5.5 Uploading the fixed package

Do **NOT** upload a package to the security upload queue (`oldstable-security`, `stable-security` , etc.) without prior authorization from the security team. If the package does not exactly meet the team's requirements, it will cause many problems and delays in dealing with the unwanted upload.

Do **NOT** upload your fix to `proposed-updates` without coordinating with the security team. Packages from `security.debian.org` will be copied into the `proposed-updates` directory automatically. If a package with the same or a higher version number is already installed into the archive, the security update will be rejected by the archive system. That way, the stable distribution will end up without a security update for this package instead.

Once you have created and tested the new package and it has been approved by the security team, it needs to be uploaded so that it can be installed in the archives. For security uploads, the place to upload to is `ftp://security-master.debian.org/pub/SecurityUploadQueue/` .

Once an upload to the security queue has been accepted, the package will automatically be built for all architectures and stored for verification by the security team.

Uploads which are waiting for acceptance or verification are only accessible by the security team. This is necessary since there might be fixes for security problems that cannot be disclosed yet.

If a member of the security team accepts a package, it will be installed on `security.debian.org` as well as proposed for the proper *`distribution`*`-proposed-updates` on `ftp-master.debian.org`.

5.9 Moving, removing, renaming, adopting, and orphaning packages

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

5.9.1 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'.[1]

If you need to change the section for one of your packages, change the package control information to place the package in the desired section, and re-upload the package (see the Debian Policy Manual for details). You must ensure that you include the `.orig.tar.gz` in your upload (even if you are not uploading a new upstream version), or it will not appear in the new section together with the rest of the package. If your new section is valid, it will be moved automatically. If it does not, then contact the ftpmasters in order to understand what happened.

If, on the other hand, you need to change the `subsection` of one of your packages (e.g., ``devel'', ``admin''), the procedure is slightly different. Correct the subsection as found in the control file of the package, and re-upload that. Also, you'll need to get the override file updated, as described in Section 5.7 .

5.9.2 Removing packages

If for some reason you want to completely remove a package (say, if it is an old compatibility library which is no longer required), you need to file a bug against `ftp.debian.org` asking that the package be removed; as all bugs, this bug should normally have normal severity. The bug title should be in the form `RM: package [architecture list] -- reason`, where *package* is the package to be removed and *reason* is a short summary of the reason for the removal request. *[architecture list]* is optional and only needed if the removal request only applies to some architectures, not all. Note that the **reportbug** will create a title conforming to these rules when you use it to report a bug against the `ftp.debian.org` pseudo-package.

If you want to remove a package you maintain, you should note this in the bug title by prepending `ROM` (Request Of Maintainer). There are several other standard acronyms used in the reasoning for a package removal, see `http://ftp-master.debian.org/removals.html` for a complete list. That page also provides a convenient overview of pending removal requests.

Note that removals can only be done for the `unstable` , `experimental` and `stable` distribution. Packages are not removed from `testing` directly. Rather, they will be removed automatically after the package has been removed from `unstable` and no package in `testing` depends on it.

There is one exception when an explicit removal request is not necessary: If a (source or binary) package is an orphan, it will be removed semi-automatically. For a binary-package, this means if there is no longer any source package producing this binary package; if the binary package is just no longer produced on some architectures, a removal request is still necessary. For a source-package, this means that all binary packages it refers to have been taken over by another source package.

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

Usually you only ask for the removal of a package maintained by yourself. If you want to remove another package, you have to get the approval of its maintainer. Should the package be orphaned and thus have no maintainer, you should first discuss the removal request on debian-qa@lists.debian.org. If there is a consensus that the package should be removed, you should reassign and retitle the `O:` bug filed against the `wnpp` package instead of filing a new bug as removal request.

Further information relating to these and other package removal related topics may be found at `http://wiki.debian.org/ftpmaster_Removals` and `http://qa.debian.org/howto-remove.html`.

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. Other useful programs include `apt-cache rdepends`, **apt-rdepends**, **build-rdeps** (in the `devscripts` package) and **grep-dctrl**. Removal of orphaned packages is discussed on debian-qa@lists.debian.org.

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

[1] See the Debian Policy Manual for guidelines on what section a package belongs in.

in previous Debian releases, they should be marked as fixed in the version `<most-recent-version--ever-in-Debian>+rm`.

5.9.2.1 Removing packages from `Incoming`

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

5.9.3 Replacing or renaming packages

When the upstream maintainers for one of your packages chose to rename their software (or you made a mistake naming your package), you should follow a two-step process to rename it. In the first step, change the `debian/control` file to reflect the new name and to replace, provide and conflict with the obsolete package name (see the Debian Policy Manual for details). Please note that you should only add a `Provides` relation if all packages depending on the obsolete package name continue to work after the renaming. Once you've uploaded the 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 (see Section 5.9.2). Do not forget to properly reassign the package's bugs at the same time.

At other times, you may make a mistake in constructing your package and wish to replace it. The only way to do this is to increase the version number and upload a new version. The old version will be expired in the usual manner. Note that this applies to each part of your package, including the sources: if you wish to replace the upstream source tarball of your package, you will need to upload it with a different version. An easy possibility is to replace `foo_1.00.orig.tar.gz` with `foo_1.00+0.orig.tar.gz`. This restriction gives each file on the ftp site a unique name, which helps to ensure consistency across the mirror network.

5.9.4 Orphaning a package

If you can no longer maintain a package, you need to inform others, 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 the package has a priority of standard or higher, it should be set to important. 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 you just intend to give the package away, but you can keep maintainership for the moment, then you should instead submit a bug against `wnpp` and title it `RFA: package -- short description`. `RFA` stands for `Request For Adoption`.

More information is on the WNPP web pages.

5.9.5 Adopting a package

A list of packages in need of a new maintainer is available in the Work-Needing and Prospective Packages list (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. If you have reason to believe a maintainer has gone AWOL (absent without leave), see Section 7.4 .

Generally, you may not take over the package without the assent of the current maintainer. Even if they ignore you, that is still not grounds to take over a package. Complaints about maintainers should be brought up on the developers' mailing list. If the discussion doesn't end with a positive conclusion, and the issue is of a technical nature, consider bringing it to the attention of the technical committee (see the technical committee web page for more information).

If you take over an old package, you probably want to be listed as the package's official maintainer in the bug system. This will happen automatically once you upload a new version with an updated `Maintainer:` field, although it can take a few hours after the upload is done. If you do not expect to upload a new version for a while, you can use Section 4.10 to get the bug reports. However, make sure that the old maintainer has no problem with the fact that they will continue to receive the bugs during that time.

5.10 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 are 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, when a maintainer uploads a (portable) source packages with binaries for the `i386` architecture, it will be built for each of the other architectures, amounting to 12 more builds.

5.10.1 Being kind to porters

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

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

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

1. Make sure that your `Build-Depends` and `Build-Depends-Indep` settings in `debian/control` are set properly. The best way to validate this is to use the `debootstrap` package to create an `unstable` chroot environment (see Section A.4.2). Within that chrooted environment, install the `build-essential` package and any package dependencies mentioned in `Build-Depends` and/or `Build-Depends-Indep`. Finally, try building your package within that chrooted environment. These steps can be automated by the use of the **pbuilder** program which is provided by the package of the same name (see Section A.4.3).

   If you can't set up a proper chroot, **dpkg-depcheck** may be of assistance (see Section A.6.7).

   See the Debian Policy Manual for instructions on setting build dependencies.

2. Don't set architecture to a value other than `all` or `any` unless you really mean it. In too many cases, maintainers don't follow the instructions in the Debian Policy Manual. Setting your architecture to only one architecture (such as `i386` or `amd64`) is usually incorrect.

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

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

5. Make sure you don't rely on locally installed or hacked configurations or programs. For instance, you should never be calling programs in `/usr/local/bin` or the like. Try not to rely on programs being setup in a special way. Try building your package on another machine, even if it's the same architecture.

6. Don't depend on the package you're building being installed already (a sub-case of the above issue). There are, of course, exceptions to this rule, but be aware that any case like this needs manual bootstrapping and cannot be done by automated package builders.

7. Don't rely on the compiler being a certain version, if possible. If not, then make sure your build dependencies reflect the restrictions, although you are probably asking for trouble, since different architectures sometimes standardize on different compilers.
8. Make sure your debian/rules contains separate `binary-arch` and `binary-indep` targets, as the Debian Policy Manual requires. Make sure that both targets work independently, that is, that you can call the target without having called the other before. To test this, try to run **dpkg-buildpackage -B**.

5.10.2 Guidelines for porter uploads

If the package builds out of the box for the architecture to be ported to, you are in luck and your job is easy. This section applies to that case; it describes how to build and upload your binary package so that it is properly installed into the archive. If you do have to patch the package in order to get it to compile for the other architecture, you are actually doing a source NMU, so consult Section 5.11.1 instead.

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

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

If you are working on a Debian machine for your porting efforts and you need to sign your upload locally for its acceptance in the archive, you can run **debsign** on your `.changes` file to have it signed conveniently, or use the remote signing mode of **dpkg-sig**.

5.10.2.1 Recompilation or binary-only NMU

Sometimes the initial porter upload is problematic because the environment in which the package was built was not good enough (outdated or obsolete library, bad compiler, ...). Then you may just need to recompile it in an updated environment. However, you have to bump the version number in this case, so that the old bad package can be replaced in the Debian archive (**dak** refuses to install new packages if they don't have a version number greater than the currently available one).

You have to make sure that your binary-only NMU doesn't render the package uninstallable. This could happen when a source package generates arch-dependent and arch-independent packages that have inter-dependencies generated using dpkg's substitution variable `$(Source-Version)` .

Despite the required modification of the changelog, these are called binary-only NMUs — there is no need in this case to trigger all other architectures to consider themselves out of date or requiring recompilation.

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

The ``magic'' for a recompilation-only NMU is triggered by using a suffix appended to the package version number, following the form `b`*`number`*. For instance, if the latest version you are recompiling against was version `2.9-3`, your binary-only NMU should carry a version of `2.9-3+b1`. If the latest version was `3.4+b1` (i.e, a native package with a previous recompilation NMU), your binary-only NMU should have a version number of `3.4+b2`. [2]

Similar to initial porter uploads, the correct way of invoking **dpkg-buildpackage** is `dpkg-buildpackage -B` to only build the architecture-dependent parts of the package.

5.10.2.2 When to do a source NMU if you are a porter

Porters doing a source NMU generally follow the guidelines found in Section 5.11 , 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. It also varies whether the architecture is a candidate

[2] In the past, such NMUs used the third-level number on the Debian part of the revision to denote their recompilation-only status; however, this syntax was ambiguous with native packages and did not allow proper ordering of recompile-only NMUs, source NMUs, and security NMUs on the same package, and has therefore been abandoned in favor of this new syntax.

for inclusion into the next stable release; the release managers decide and announce which architectures are candidates.

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.) For uploads to `stable` or `testing` , please coordinate with the appropriate release team first.

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 architectures 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 compiler bugs and the like, make sure you `#ifdef` your work properly; also, document your kludge so that people know to remove it once the external problems have been fixed.

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

5.10.3 Porting infrastructure and automation

There is infrastructure and several tools to help automate package porting. This section contains a brief overview of this automation and porting to these tools; see the package documentation or references for full information.

5.10.3.1 Mailing lists and web pages

Web pages containing the status of each port can be found at `http://www.debian.org/ports/`.

Each port of Debian has a mailing list. The list of porting mailing lists can be found at `http://lists.debian.org/ports.html`. These lists are used to coordinate porters, and to connect the users of a given port with the porters.

5.10.3.2 Porter tools

Descriptions of several porting tools can be found in Section A.7 .

5.10.3.3 `wanna-build`

The `wanna-build` system is used as a distributed, client-server build distribution system. It is usually used in conjunction with build daemons running the `buildd` program. `Build daemons` are ``slave'' hosts which contact the central `wanna-build` system to receive a list of packages that need to be built.

`wanna-build` is not yet available as a package; however, all Debian porting efforts are using it for automated package building. The tool used to do the actual package builds, `sbuild` is available as a package, see its description in Section A.4.4 . Please note that the packaged version is not the same as the one used on build daemons, but it is close enough to reproduce problems.

Most of the data produced by `wanna-build` which is generally useful to porters is available on the web at `http://buildd.debian.org/`. This data includes nightly updated statistics, queueing information and logs for build attempts.

We are quite proud of this system, since it has so many possible 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.

The wanna-build team, in charge of the buildds, can be reached at `debian-wb-team@lists.debian.org`. To determine who (wanna-build team, release team) and how (mail, BTS) to contact, refer to `http://lists.debian.org/debian-project/2009/03/msg00096.html`.

When requesting binNMUs or give-backs (retries after a failed build), please use the format described at http://release.debian.org/wanna-build.txt.

5.10.4 When your package is *not* portable

Some packages still have issues with building and/or working on some of the architectures supported by Debian, and cannot be ported at all, or not within a reasonable amount of time. An example is a package that is SVGA-specific (only available for i386 and amd64), or uses other hardware-specific features not supported on all architectures.

In order to prevent broken packages from being uploaded to the archive, and wasting buildd time, you need to do a few things:

- First, make sure your package *does* fail to build on architectures that it cannot support. There are a few ways to achieve this. The preferred way is to have a small testsuite during build time that will test the functionality, and fail if it doesn't work. This is a good idea anyway, as this will prevent (some) broken uploads on all architectures, and also will allow the package to build as soon as the required functionality is available.
  Additionally, if you believe the list of supported architectures is pretty constant, you should change any to a list of supported architectures in debian/control. This way, the build will fail also, and indicate this to a human reader without actually trying.
- In order to prevent autobuilders from needlessly trying to build your package, it must be included in packages-arch-specific, a list used by the **wanna-build** script. The current version is available as http://cvs.debian.org/srcdep/Packages-arch-specific?cvsroot=dak; please see the top of the file for whom to contact for changes.

Please note that it is insufficient to only add your package to Packages-arch-specific without making it fail to build on unsupported architectures: A porter or any other person trying to build your package might accidently upload it without noticing it doesn't work. If in the past some binary packages were uploaded on unsupported architectures, request their removal by filing a bug against ftp.debian.org

5.11 Non-Maintainer Uploads (NMUs)

Every package has one or more maintainers. Normally, these are the people who work on and upload new versions of the package. In some situations, it is useful that other developers can upload a new version as well, for example if they want to fix a bug in a package they don't maintain, when the maintainer needs help to respond to issues. Such uploads are called *Non-Maintainer Uploads (NMU)*.

5.11.1 When and how to do an NMU

Before doing an NMU, consider the following questions:

- Does your NMU really fix bugs? Fixing cosmetic issues or changing the packaging style in NMUs is discouraged.
- Did you give enough time to the maintainer? When was the bug reported to the BTS? Being busy for a week or two isn't unusual. Is the bug so severe that it needs to be fixed right now, or can it wait a few more days?
- How confident are you about your changes? Please remember the Hippocratic Oath: "Above all, do no harm." It is better to leave a package with an open grave bug than applying a non-functional patch, or one that hides the bug instead of resolving it. If you are not 100% sure of what you did, it might be a good idea to seek advice from others. Remember that if you break something in your NMU, many people will be very unhappy about it.
- Have you clearly expressed your intention to NMU, at least in the BTS? It is also a good idea to try to contact the maintainer by other means (private email, IRC).
- If the maintainer is usually active and responsive, have you tried to contact him? In general it should be considered preferable that a maintainer takes care of an issue himself and that he is given the chance to review and correct your patch, because he can be expected to be more aware of potential issues which an NMUer might miss. It is often a better use of everyone's time if the maintainer is given an opportunity to upload a fix on their own.

When doing an NMU, you must first make sure that your intention to NMU is clear. Then, you must send a patch with the differences between the current package and your proposed NMU to the BTS. The `nmudiff` script in the `devscripts` package might be helpful.

While preparing the patch, you should better be aware of any package-specific practices that the maintainer might be using. Taking them into account reduces the burden of getting your changes integrated back in the normal package workflow and thus increases the possibilities that that will happen. A good place where to look for for possible package-specific practices is `debian/README.source`.

Unless you have an excellent reason not to do so, you must then give some time to the maintainer to react (for example, by uploading to the `DELAYED` queue). Here are some recommended values to use for delays:

- Upload fixing only release-critical bugs older than 7 days: 2 days
- Upload fixing only release-critical and important bugs: 5 days
- Other NMUs: 10 days

Those delays are only examples. In some cases, such as uploads fixing security issues, or fixes for trivial bugs that blocking a transition, it is desirable that the fixed package reaches `unstable` sooner.

Sometimes, release managers decide to allow NMUs with shorter delays for a subset of bugs (e.g release-critical bugs older than 7 days). Also, some maintainers list themselves in the Low Threshold NMU list, and accept that NMUs are uploaded without delay. But even in those cases, it's still a good idea to give the maintainer a few days to react before you upload, especially if the patch wasn't available in the BTS before, or if you know that the maintainer is generally active.

After you upload an NMU, you are responsible for the possible problems that you might have introduced. You must keep an eye on the package (subscribing to the package on the PTS is a good way to achieve this).

This is not a license to perform NMUs thoughtlessly. If you NMU when it is clear that the maintainers are active and would have acknowledged a patch in a timely manner, or if you ignore the recommendations of this document, your upload might be a cause of conflict with the maintainer. You should always be prepared to defend the wisdom of any NMU you perform on its own merits.

5.11.2 NMUs and debian/changelog

Just like any other (source) upload, NMUs must add an entry to `debian/changelog`, telling what has changed with this upload. The first line of this entry must explicitely mention that this upload is an NMU, e.g.:

```
  * Non-maintainer upload.
```

The way to version NMUs differs for native and non-native packages.

If the package is a native package (without a debian revision in the version number), the version must be the version of the last maintainer upload, plus `+nmu`*X*, where *X* is a counter starting at `1`. If the last upload was also an NMU, the counter should be increased. For example, if the current version is `1.5`, then an NMU would get version `1.5+nmu1`.

If the package is a not a native package, you should add a minor version number to the debian revision part of the version number (the portion after the last hyphen). This extra number must start at 1. For example, if the current version is `1.5-2`, then an NMU would get version `1.5-2.1`. If a new upstream version is packaged in the NMU, the debian revision is set to `0`, for example `1.6-0.1`.

In both cases, if the last upload was also an NMU, the counter should be increased. For example, if the current version is `1.5+nmu3` (a native package which has already been NMUed), the NMU would get version `1.5+nmu4.`.

A special versioning scheme is needed to avoid disrupting the maintainer's work, since using an integer for the Debian revision will potentially conflict with a maintainer upload already in preparation at the time of an NMU, or even one sitting in the ftp NEW queue. It also has the benefit of making it visually clear that a package in the archive was not made by the official maintainer.

If you upload a package to testing or stable, you sometimes need to "fork" the version number tree. This is the case for security uploads, for example. For this, a version of the form `+deb`*X*`u`*Z* should be used, where *X* and *Y* are the major and minor release numbers, and *Z* is a counter starting at `1`. When the release number is not yet known (often the case for `testing`, at the beginning of release cycles), the lowest release number higher than the last stable release number must be used. For example, while Etch (Debian 4.0) is stable, a security NMU to stable for a package at version `1.5-3` would have version

`1.5-3+deb40u1`, whereas a security NMU to Lenny would get version `1.5-3+deb50u1`. After the release of Lenny, security uploads to the `testing` distribution will be versioned `+deb51uZ`, until it is known whether that release will be Debian 5.1 or Debian 6.0 (if that becomes the case, uploads will be versioned as `+deb60uZ`.

5.11.3 Using the `DELAYED/` queue

Having to wait for a response after you request permission to NMU is inefficient, because it costs the NMUer a context switch to come back to the issue. The `DELAYED` queue (see Section 5.6.2) allows the developer doing the NMU to perform all the necessary tasks at the same time. For instance, instead of telling the maintainer that you will upload the updated package in 7 days, you should upload the package to `DELAYED/7` and tell the maintainer that he has 7 days to react. During this time, the maintainer can ask you to delay the upload some more, or cancel your upload.

The `DELAYED` queue should not be used to put additional pressure on the maintainer. In particular, it's important that you are available to cancel or delay the upload before the delay expires since the maintainer cannot cancel the upload himself.

If you make an NMU to `DELAYED` and the maintainer updates his package before the delay expires, your upload will be rejected because a newer version is already available in the archive. Ideally, the maintainer will take care to include your proposed changes (or at least a solution for the problems they address) in that upload.

5.11.4 NMUs from the maintainer's point of view

When someone NMUs your package, this means they want to help you to keep it in good shape. This gives users fixed packages faster. You can consider asking the NMUer to become a co-maintainer of the package. Receiving an NMU on a package is not a bad thing; it just means that the package is interesting enough for other people to work on it.

To acknowledge an NMU, include its changes and changelog entry in your next maintainer upload. If you do not acknowledge the NMU by including the NMU changelog entry in your changelog, the bugs will remain closed in the BTS but will be listed as affecting your maintainer version of the package.

5.11.5 Source NMUs vs Binary-only NMUs (binNMUs)

The full name of an NMU is *source NMU*. There is also another type, namely the *binary-only NMU*, or *binNMU*. A binNMU is also a package upload by someone other than the package's maintainer. However, it is a binary-only upload.

When a library (or other dependency) is updated, the packages using it may need to be rebuilt. Since no changes to the source are needed, the same source package is used.

BinNMUs are usually triggered on the buildds by wanna-build. An entry is added to debian/changelog, explaining why the upload was needed and increasing the version number as described in Section 5.10.2.1. This entry should not be included in the next upload.

Buildds upload packages for their architecture to the archive as binary-only uploads. Strictly speaking, these are binNMUs. However, they are not normally called NMU, and they don't add an entry to debian/changelog.

5.11.6 NMUs vs QA uploads

NMUs are uploads of packages by somebody else than their assigned maintainer. There is another type of upload where the uploaded package is not yours: QA uploads. QA uploads are uploads of orphaned packages.

QA uploads are very much like normal maintainer uploads: they may fix anything, even minor issues; the version numbering is normal, and there is no need to use a delayed upload. The difference is that you are not listed as the Maintainer or Uploader for the package. Also, the changelog entry of a QA upload has a special first line:

```
* QA upload.
```

If you want to do an NMU, and it seems that the maintainer is not active, it is wise to check if the package is orphaned (this information is displayed on the package's Package Tracking System page).

When doing the first QA upload to an orphaned package, the maintainer should be set to `Debian QA Group <packages@qa.debian.org>`. Orphaned packages which did not yet have a QA upload still have their old maintainer set. There is a list of them at `http://qa.debian.org/orphaned.html`.

Instead of doing a QA upload, you can also consider adopting the package by making yourself the maintainer. You don't need permission from anybody to adopt an orphaned package, you can just set yourself as maintainer and upload the new version (see Section 5.9.5).

5.12 Collaborative maintenance

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

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

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

– Setup the co-maintainer with access to the sources you build the package from. Generally this implies you are using a network-capable version control system, such as **CVS** or **Subversion**. Alioth (see Section 4.12) provides such tools, amongst others.

– Add the co-maintainer's correct maintainer name and address to the `Uploaders` field in the first paragraph of the `debian/control` file.

```
Uploaders: John Buzz <jbuzz@debian.org>, Adam Rex <arex@debian.org>
```

– Using the PTS (Section 4.10), the co-maintainers should subscribe themselves to the appropriate source package.

Another form of collaborative maintenance is team maintenance, which is recommended if you maintain several packages with the same group of developers. In that case, the Maintainer and Uploaders field of each package must be managed with care. It is recommended to choose between one of the two following schemes:

1. Put the team member mainly responsible for the package in the Maintainer field. In the Uploaders, put the mailing list address, and the team members who care for the package.
2. Put the mailing list address in the Maintainer field. In the Uploaders field, put the team members who care for the package. In this case, you must make sure the mailing list accept bug reports without any human interaction (like moderation for non-subscribers).

In any case, it is a bad idea to automatically put all team members in the Uploaders field. It clutters the Developer's Package Overview listing (see Section 4.11) with packages one doesn't really care for, and creates a false sense of good maintenance.

5.13 The testing distribution

5.13.1 Basics

Packages are usually installed into the `testing` distribution after they have undergone some degree of `testing` in `unstable`.

They must be in sync on all architectures and mustn't have dependencies that make them uninstallable; they also have to have generally no known release-critical bugs at the time they're installed into `testing` . This way, `testing` should always be close to being a release candidate. Please see below for details.

5.13.2 Updates from unstable

The scripts that update the `testing` distribution are run twice each day, right after the installation of the updated packages; these scripts are called `britney`. They generate the `Packages` files for the

testing distribution, but they do so in an intelligent manner; they try to avoid any inconsistency and to use only non-buggy packages.

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

- The package must have been available in unstable for 2, 5 or 10 days, depending on the urgency (high, medium or low). Please note that the urgency is sticky, meaning that the highest urgency uploaded since the previous testing transition is taken into account. Those delays may be doubled during a freeze, or testing transitions may be switched off altogether;
- It must not have new release-critical bugs (RC bugs affecting the version available in unstable, but not affecting the version in testing);
- It must be available on all architectures on which it has previously been built in unstable. Section 4.9.2 may be of interest to check that information;
- It must not break any dependency of a package which is already available in testing;
- The packages on which it depends must either be available in testing or they must be accepted into testing at the same time (and they will be if they fulfill all the necessary criteria);

To find out whether a package is progressing into testing or not, see the testing script output on the web page of the testing distribution, or use the program **grep-excuses** which is in the devscripts package. This utility can easily be used in a crontab(5) to keep yourself informed of the progression of your packages into testing.

The update_excuses file does not always give the precise reason why the package is refused; you may have to find it on your own by looking for what would break with the inclusion of the package. The testing web page gives some more information about the usual problems which may be causing such troubles.

Sometimes, some packages never enter testing because the set of inter-relationship is too complicated and cannot be sorted out by the scripts. See below for details.

Some further dependency analysis is shown on http://release.debian.org/migration/ — but be warned, this page also shows build dependencies which are not considered by britney.

5.13.2.1 out-of-date

For the testing migration script, outdated means: There are different versions in unstable for the release architectures (except for the architectures in fuckedarches; fuckedarches is a list of architectures that don't keep up (in update_out.py), but currently, it's empty). outdated has nothing whatsoever to do with the architectures this package has in testing.

Consider this example:

	alpha	**arm**
testing	1	-
unstable	1	2

The package is out of date on alpha in unstable, and will not go to testing. Removing the package would not help at all, the package is still out of date on alpha, and will not propagate to testing.

However, if ftp-master removes a package in unstable (here on arm):

	alpha	**arm**	**hurd-i386**
testing	1	1	-
unstable	2	-	1

In this case, the package is up to date on all release architectures in unstable (and the extra hurd-i386 doesn't matter, as it's not a release architecture).

Sometimes, the question is raised if it is possible to allow packages in that are not yet built on all architectures: No. Just plainly no. (Except if you maintain glibc or so.)

5.13.2.2 Removals from testing

Sometimes, a package is removed to allow another package in: This happens only to allow *another* package to go in if it's ready in every other sense. Suppose e.g. that a cannot be installed with the new

version of b; then a may be removed to allow b in.

Of course, there is another reason to remove a package from testing : It's just too buggy (and having a single RC-bug is enough to be in this state).

Furthermore, if a package has been removed from unstable, and no package in testing depends on it any more, then it will automatically be removed.

5.13.2.3 circular dependencies

A situation which is not handled very well by britney is if package a depends on the new version of package b, and vice versa.

An example of this is:

	testing	**unstable**
a	1; depends: b=1	2; depends: b=2
b	1; depends: a=1	2; depends: a=2

Neither package a nor package b is considered for update.

Currently, this requires some manual hinting from the release team. Please contact them by sending mail to debian-release@lists.debian.org if this happens to one of your packages.

5.13.2.4 influence of package in testing

Generally, there is nothing that the status of a package in testing means for transition of the next version from unstable to testing, with two exceptions: If the RC-bugginess of the package goes down, it may go in even if it is still RC-buggy. The second exception is if the version of the package in testing is out of sync on the different arches: Then any arch might just upgrade to the version of the source package; however, this can happen only if the package was previously forced through, the arch is in fuckedarches, or there was no binary package of that arch present in unstable at all during the testing migration.

In summary this means: The only influence that a package being in testing has on a new version of the same package is that the new version might go in easier.

5.13.2.5 details

If you are interested in details, this is how britney works:

The packages are looked at to determine whether they are valid candidates. This gives the update excuses. The most common reasons why a package is not considered are too young, RC-bugginess, and out of date on some arches. For this part of britney, the release managers have hammers of various sizes to force britney to consider a package. (Also, the base freeze is coded in that part of britney.) (There is a similar thing for binary-only updates, but this is not described here. If you're interested in that, please peruse the code.)

Now, the more complex part happens: Britney tries to update testing with the valid candidates. For that, britney tries to add each valid candidate to the testing distribution. If the number of uninstallable packages in testing doesn't increase, the package is accepted. From that point on, the accepted package is considered to be part of testing, such that all subsequent installability tests include this package. Hints from the release team are processed before or after this main run, depending on the exact type.

If you want to see more details, you can look it up on merkel:/org/ftp.debian.org/testing/update_out/ (or in merkel:~aba/testing/update_out to see a setup with a smaller packages file). Via web, it's at http://ftp-master.debian.org/testing/update_out_code/

The hints are available via http://ftp-master.debian.org/testing/hints/.

5.13.3 Direct updates to testing

The testing distribution is fed with packages from unstable according to the rules explained above. However, in some cases, it is necessary to upload packages built only for testing. For that, you may want to upload to testing-proposed-updates.

Keep in mind that packages uploaded there are not automatically processed, they have to go through the hands of the release manager. So you'd better have a good reason to upload there. In order to know

what a good reason is in the release managers' eyes, you should read the instructions that they regularly give on debian-devel-announce@lists.debian.org.

You should not upload to `testing-proposed-updates` when you can update your packages through `unstable`. If you can't (for example because you have a newer development version in `unstable`), you may use this facility, but it is recommended that you ask for authorization from the release manager first. Even if a package is frozen, updates through `unstable` are possible, if the upload via `unstable` does not pull in any new dependencies.

Version numbers are usually selected by adding the codename of the `testing` distribution and a running number, like `1.2sarge1` for the first upload through `testing-proposed-updates` of package version `1.2`.

Please make sure you didn't miss any of these items in your upload:

- Make sure that your package really needs to go through `testing-proposed-updates`, and can't go through `unstable`;
- Make sure that you included only the minimal amount of changes;
- Make sure that you included an appropriate explanation in the changelog;
- Make sure that you've written `testing` or `testing-proposed-updates` into your target distribution;
- Make sure that you've built and tested your package in `testing`, not in `unstable`;
- Make sure that your version number is higher than the version in `testing` and `testing-proposed-updates`, and lower than in `unstable`;
- After uploading and successful build on all platforms, contact the release team at debian-release@lists.debian.org and ask them to approve your upload.

5.13.4 Frequently asked questions

5.13.4.1 What are release-critical bugs, and how do they get counted?

All bugs of some higher severities are by default considered release-critical; currently, these are `critical`, `grave` and `serious` bugs.

Such bugs are presumed to have an impact on the chances that the package will be released with the `stable` release of Debian: in general, if a package has open release-critical bugs filed on it, it won't get into `testing`, and consequently won't be released in `stable`.

The `unstable` bug count are all release-critical bugs which are marked to apply to *`package`*`/`*`version`* combinations that are available in unstable for a release architecture. The `testing` bug count is defined analogously.

5.13.4.2 How could installing a package into `testing` possibly break other packages?

The structure of the distribution archives is such that they can only contain one version of a package; a package is defined by its name. So when the source package `acmefoo` is installed into `testing`, along with its binary packages `acme-foo-bin`, `acme-bar-bin`, `libacme-foo1` and `libacme-foo-dev`, the old version is removed.

However, the old version may have provided a binary package with an old soname of a library, such as `libacme-foo0`. Removing the old `acmefoo` will remove `libacme-foo0`, which will break any packages which depend on it.

Evidently, this mainly affects packages which provide changing sets of binary packages in different versions (in turn, mainly libraries). However, it will also affect packages upon which versioned dependencies have been declared of the ==, <=, or << varieties.

When the set of binary packages provided by a source package change in this way, all the packages that depended on the old binaries will have to be updated to depend on the new binaries instead. Because installing such a source package into `testing` breaks all the packages that depended on it in `testing`, some care has to be taken now: all the depending packages must be updated and ready to be installed themselves so that they won't be broken, and, once everything is ready, manual intervention by the release manager or an assistant is normally required.

If you are having problems with complicated groups of packages like this, contact debian-devel@lists.debian.org or debian-release@lists.debian.org for help.

Chapitre 6

Les meilleurs pratiques pour la construction des paquets

La qualité de Debian est largement due à la Charte Debian qui définit les prérequis explicites de base que tous les paquets Debian doivent satisfaire. Cependant, il existe également une expérience générale partagée qui va bien au delà de la Charte Debian et constitue une somme d'années d'expérience dans la construction de paquets. De nombreux contributeurs talentueux ont créé d'excellents outils qui peuvent vous aider, en tant que mainteneur Debian, à créer et maintenir des paquets d'excellente qualité.

Ce chapitre rassemble les meilleures pratiques pou rles mainteneurs Debian. La majorité de son contenu est constitué de recommandations plus que d'obligations. Il s'agit essentiellement d'informations subjectives, d'avis et de pointeurs, rassemblés par les développeurs Debian. Il est conseillé d'y choisir ce qui vous convient le mieux.

6.1 Les meilleures pratiques pour le fichier `debian/rules`

Les recommandations qui suivent s'appliquent au fichier `debian/rules`. Comme ce fichier contrôle le processus de construction des paquets et fait le choix des fichiers qui entreront dans ce paquet, directement ou indirectement, il s'agit du fichier dont les mainteneurs s'occupent généralement le plus.

6.1.1 Scripts d'assistance

La justification à l'utilisation de scripts d'assistance dans `debian/rules` est de permettre aux mainteneurs de définir puis utiliser une logique commune pour de nombreux paquets. Si on prend par exemple l'installaiton d'entrées de menu, il est nécessaire de placer le fichier dans `/usr/lib/menu` (ou `/usr/lib/menu` des fichiers de menu exécutables, s'il en existe), puis d'ajouter des commandes aux scripts des mainteneurs pour enregistrer ou désenregistrer les entrées de menu. Comme cette action est commune à de très nombreux paquets, pourquoi faudrait-il que chaque mainteneur ait à réécrire ses propres méthodes pour cela, bogues compris ? De plus, si jamais le répertoire des menus venait à changer, chaque paquet devrait être modifié.

Les scripts d'assistance s'occupent de ce type de tâche. À condition d'être compatible avec les conventions utilisées par le script d'assistance, celui-ci s'occupe de tous les détails. Les modifications dans la charte peuvent alors être implémentées dans le script d'assistance et les paquets n'ont plus qu'à être reconstruits sans autre modification.

Annexe A contient un certain nombre d'assistants variés. Le système le plus répandu et (selon nous) le plus adapté est `debhelper`. Des systèmes antérieurs, tels que `debmake`, étaient monolithiques ; ils ne permettaient pas de choisir quelle partie de l'assistant serait utile, en obligeant à se servir de l'ensemble de l'assistant. A contrario, `debhelper` est constitué d'un grand nombre de petits programmes **dh_*** différents. Par exemple, **dh_installman** installe et compresse les pages de manuel, **dh_installmenu** istalle les fichiers de menu, et ainsi de suite. En conséquence, il offre la possibilité d'utiliser certains des scripts d'assistance tout en conservant des commandes manuelles dans `debian/rules`.

Pour démarrer avec `debhelper`, il est conseillé de lire debhelper(1) et de consulter les exemples fournis avec le paquet. **dh_make**, fourni avec le paquet `dh-make` (voir Section A.3.3) peut être utilisé pour convertir un paquet source originel en paquet géré par `debhelper`. Cette méthode rapide ne doit

cependant pas se substituer à une compréhension individuelle des commandes **dh_***. Si vous utilisez un assistant, vous devez prendre le temps de l'apprendre, pour comprendre ses besoins et son comportement.

Certains mainteneurs pensent que l'utilisation de fichiers `debian/rules` sans assistants est préférable car elle évite d'avoir à apprendre les subtilité de ces systèmes d'assistance. Utiliser l'une ou l'autre méthode est un entièrement à la discrétion du mainteneur d'un paquet qui devrait utiliser la méthode qui lui convient le mieux. De nombreux exemples de fichiers `debian/rules` qui n'utilisent pas d'assistants sont disponibles à l'adresse `http://arch.debian.org/arch/private/srivasta/`.

6.1.2 Séparation des modifications (« patches ») en plusieurs fichiers

Les paquets complexes ont souvent de nombreux bogues qui doivent être gérés par le mainteneur. Si certains de ces bogues sont corrigés par des modifications effectuées directement dans le code source, sans discernement, il peut devenir difficile de retrouver l'origine et la motivation de ces modifications. Cela peut également rendre bien plus complexe l'intégration d'une nouvelle version amont qui pourrait inclure certaines de ces modifications (mais pas toutes). Il est en effet alors quasiment impossible de reprendre le jeu initial de changements (p. ex. dans le fichier `.diff.gz`) et supprimer ceux qui correspondent à des correctifs appliqués par le mainteneur amont.

Unfortunately, the packaging system as such currently doesn't provide for separating the patches into several files. Nevertheless, there are ways to separate patches: the patch files are shipped within the Debian patch file (`.diff.gz`), usually within the `debian/` directory. The only difference is that they aren't applied immediately by dpkg-source, but by the `build` rule of `debian/rules`, through a dependency on the `patch` rule. Conversely, they are reverted in the `clean` rule, through a dependency on the `unpatch` rule.

quilt is the recommended tool for this. It does all of the above, and also allows to manage patch series. See the `quilt` package for more information.

There are other tools to manage patches, like **dpatch**, and the patch system integrated with `cdbs`.

6.1.3 Paquets binaires multiples

A seul paquet source créera souvent plusieurs paquets binaires, soit pour fournir plusieurs variantes du même logiciels (p. ex. le paquet source `vim`) ou pour répartir les fichiers en plusieurs paquets plus petits au lieu d'un seul paquet monolithique (ce qui peut permettre à un utilisateur de d'installer que les éléments nécessaires et donc préserver l'espace disque).

Le second cas est simple à gérer dans le fichier `debian/rules`. Il suffit de déplacer les fichiers nécessaires depuis le répertoire de construction vers l'arborescence temporaire du paquet. Cela peut se faire avec les commandes **install** ou **dh_install** du paquet `debhelper`. Veillez alors à contrôler les différentes permutations des paquets, afin de pouvoir indiquer les dépendances inter-paquets appropriées dans `debian/control`.

Le premier cas est plus délicat à gérer car il implique des recompilations multiples du même logiciel avec différentes options de configuration. Le paquet source `vim` en est un exemple, avec la gestion manuelle de l'ensemble des actions dans le fichier `debian/rules` géré manuellement.

6.2 Meilleures pratiques pour `debian/control`

Les conseils qui suivent sont destinés au fichier `debian/control`. Ils complètent la Charte Debian pour ce qui concerne les descriptions de paquets.

La description d'un paquet telle que définie par le champ correspondant du fichier `control`, comprend à la fois le résumé et la description longue du paquet. Section 6.2.1 donne des indications communes à ces deux parties, Section 6.2.2 donne des indications spécifiques pour le résumé et Section 6.2.3 donne des indications pour la description.

6.2.1 Conseils généraux pour les descriptions de paquets

La description d'un paquet doit être écrite pour son utilisateur moyen, c'est à dire la personne qui utilisera et tirera profit du paquet. Par exemple, les paquets de développement sont destinés aux développeurs et leur description peut comporter des détails techniques alors que les applications d'usage plus général, tels que les éditeurs, doivent avoir une description accessible à tout utilisateur.

Un examen général des description de paquets tend à montrer que la plupart d'entre elles ont une orientation fortement technique et ne sont donc pas destinées à l'utilisateur moyen. Sauf dans le cas de paquets destinés à des spécialistes, cela doit être considéré comme un problème.

Une des recommendations pour rester accessibles à tout utilisateur est d'éviter l'utilisation de jargon. Il est déconseillé de faire référence à des applications ou environnements qui pourraient être inconnus de l'utilisateur : parler de GNOME ou KDE est correct, car la plupart des utilisateurs sont familiers avec ces termes mais parler de GTK+ ne l'est pas. Il est préférable de supposer que le lecteur n'aura pas de connaissance du sujet et, si des termes techniques doivent être utilisés, ils doivent être expliqués.

Il est conseillé de rester objectif. Les descriptions de paquets ne sont pas une plaquette publicitaire, quelles que soient vos opinions personnelles. Le lecteur peut très bien ne pas avoir les mêmes centres d'intérêt que vous.

Les références aux noms d'autres logiciels, de protocoles, normes ou spécifications doivent utiliser leur forme canonique si elle existe. Par exemple, utilisez « X Window System », « X11 » ou « X » mais pas « X Windows », « X-Windows », ou « X Window ». Utilisez « GTK+ » et non « GTK » ou « gtk », « GNOME » et non « Gnome », « PostScript » et non « Postscript » ou « postscript ».

Si vous rencontrez des difficultés pour écrire la description d'un paquet, vous pouvez demander de l'aide ou une relecture sur debian-l10n-english@lists.debian.org.

6.2.2 Le résumé, ou description courte, d'un paquet

Policy says the synopsis line (the short description) must be concise, not repeating the package name, but also informative.

The synopsis functions as a phrase describing the package, not a complete sentence, so sentential punctuation is inappropriate: it does not need extra capital letters or a final period (full stop). It should also omit any initial indefinite or definite article - "a", "an", or "the". Thus for instance:

```
Package: libeg0
Description: exemplification support library
```

Technically this is a noun phrase minus articles, as opposed to a verb phrase. A good heuristic is that it should be possible to substitute the package name and synopsis into this formula:

The package *name* provides {a,an,the,some} *synopsis*.

Sets of related packages may use an alternative scheme that divides the synopsis into two parts, the first a description of the whole suite and the second a summary of the package's role within it:

```
Package: eg-tools
Description: simple exemplification system (utilities)

Package: eg-doc
Description: simple exemplification system - documentation
```

These synopses follow a modified formula. Where a package "*name*" has a synopsis "*suite* (*role*)" or "*suite* - *role*", the elements should be phrased so that they fit into the formula:

The package *name* provides {a,an,the} *role* for the *suite*.

6.2.3 La description longue

La description longue est l'information principale disponible pour les utilisateurs avant qu'ils ne décident d'installer un paquet. Elle doit fournir toute l'information nécessaire pour détermienr si le paquet doit être installé. Elle complète le résumé qui est donc supposé avoir été lu précédemment.

La description longue est constituée de phrases complètes.

Le premier paragraphe de cette description devrait tenter de répondre aux questions suivantes : que fait ce paquet ? Dans quelle tâche aidera-t-il l'utilisateur ? Il est important que cette description se fasse de la manière la moins technique possible, sauf si le public auquel est destiné le paquet est par définition technique.

Les paragraphes suivants devraient répondre aux questions : pourquoi, en tant qu'utilisateur, ai-je besoin de ce paquet ? Quelles autres fonctionnalités ce paquet apporte-t-il ? Quelles fonctionnalités et défauts comporte-t-il par rapport à d'autres paquets (p. ex., « si vous avez besoin de X, utilisez plutôt Y ») ? Ce paquet est-il lié à d'autres paquets d'une manière non gérée par le système de gestion des paquets (p. ex., « ceci est le client destiné au serveur toto ») ?

Veillez à éviter les erreurs d'orthographe et de grammaire. Vérifier l'orthographe avec un outil adapté. Les deux programmes **ispell** et **aspell** comportent un mode spécial permettant de contrôler un fichier `debian/control` files :

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

```
aspell -d en -D -c debian/control
```

Les utilisateurs attente en général des descriptions de paquets les réponses aux questions suivantes :

- Que fait ce paquet ? S'il s'agit d'un additif à un autre paquet, la description de cet autre paquet doit y être reprise.
- Pourquoi ai-je besoin de ce paquet ? Cela est lié à la remarque précédente, de manière différente : ceci est un agent utilisateur pour le courrier électronique, avec une interface répide et pratique vers PGP, LDAP et IMAP et les fonctionnalités X, Y ou Z.
- Si ce paquet ne doit pas être installé seul, mais est installé avec un autre paquet, cela devrait être mentionné.
- Si le paquet est `expérimental` ou ne doit pas être utilisé pour toute autre raison et que d'autres paquets doivent être utilisés à la place, cela doit également être mentionné.
- En quoi ce paquet diffère-t-il de ses concurrents ? Est-il une meilleure implémentation ? A-t-il plus de focntionnalités ? Des fonctionnalités différentes ? Pourquoi devrais-je choisir ce paquet ?

6.2.4 Page d'accueil amont

Il est recommandé d'ajouter l'URL d'accès à la page d'accueil du paquet dans le champ `Homepage` de la section `Source`du fichier `debian/control`. L'ajout de cette information à la description même du paquet est une pratique considérée comme obsolète.

6.2.5 Emplacement du système de gestion de versions

Des champs supplémentaires permettent d'indiquer l'emplacement du système de gestion de versions dans `debian/control`.

6.2.5.1 Vcs-Browser

La valeur de ce champ doit être une URL `http://` pointant sur la copie navigable par le web du dépôt de gestion de versions utilisé pour la maintenance du paquet, s'il est disponible.

Cette information est destinée à l'utilisateur final qui voudrait parcourir le travail en cours sur le paquet (p. ex. à la recherche d'un correctif qui corrige un bogue marqué `pending` dans le système de suivi des bogues.

6.2.5.2 Vcs-*

La valeur de ce champ doit être une chaîne identifiant sans équivoque l'emplacement du dépôt de gestion de versions utilisé pour la maintenance de ce paquet, s'il est disponible. `*` doit être rempalcé par le système de gestion de versions. Les systèmes suivants sont actuellement gérés par le système de suivi des paquets : `arch`, `bzr` (Bazaar), `cvs`, `darcs`, `git`, `hg` (Mercurial), `mtn` (Monotone), `svn` (Subversion). Il est possible d'indiquer plusieurs champs VCS pour le même paquet : ils seront alors tous mentionnés dans l'interface web du système de suivi des paquets.

Cette information est destiné aux utilisateurs qui ont une connaissance suffisante du système de gestion de versions et qui veulent construire une version à jour du paquet depuis les sources du système de suivi. Une autre utilisation possible de cette information pourrait être la construction automatique de la dernière version, dans le système de suivi, d'un paquet donné. À cet effet, l'emplacement pointé devrait éviter d'être lié à une version spécifique et pointer vers la branche principale de développemnt (pour les systèmes qui ont un tel concept). De plus, l'emplacement indiqué doit être accessible à l'utilisateur final, par exemple en indiquant une adresse d'accès snonyme au dépôt, plutôt qu'une version accessible par SSH.

L'exemple qui suit montre une instance de ce champ pour un dépôt Subversion du paquet `vim`. Veuillez noter que l'URL a la forme `svn://` (au lieu de `svn+ssh://`) et pointe sur la branche `trunk/`. Une utilisation des champs `Vcs-Browser` et `Homepage`, décrits précédemments, est aussi indiquée.

```
Source: vim
Section: editors
Priority: optional
<snip>
Vcs-Svn: svn://svn.debian.org/svn/pkg-vim/trunk/packages/vim
Vcs-Browser: http://svn.debian.org/wsvn/pkg-vim/trunk/packages/vim
Homepage: http://www.vim.org
```

6.3 Meilleures pratiques pour `debian/changelog`

Les indications de cette partie complètent la Charte Debian pour ce qui concerne les fichiers de journaux de changements (« changelog »).

6.3.1 Écrire des entrées de journalisation utiles

La document de suivi des modifications (« changelog ») documente uniquement les changement intervenus dans la version courante. Il est suggéré de mettre l'accent sur les modifications visibles ou affectant potentiellement les utilisateurs, réalisées depuis la version précédente.

Il est conseillé de mettre l'accent sur *ce* qui a été modifié, plutôt que comment, par qui et quand elle a été réalisée.Cela dit, il est conseillé, par courtoisie, d'indiquer les auteurs qui ont apporté une aide significative à la maintenance du paquet (p. ex. lorsque ces personnes ont envoyé des correctifs).

Il n'est pas indispensable d'indiquer les détails des modifications triviales. Il est également possible de grouper plusieurs modifications sur une même entrée. Cependant, évitez une documentation trop concise pour les modifications majeures. Il est particulièrement conseillé d'être très clair sur les modifications qui affectent le comportement du programme. Pour des explications plus détaillées, vous pouvez aussi utiliser le fichier `README.Debian`.

Utilisez un anglais simple que la majorité des lecteurs puissent comprendre. Évitez les abréviations et le jargon technique lorsque des modifications permettent la clôture de bogues. Cel est vrai notamment quand vous pensez que les utilisateurs qui les ont envoyés n'ont pas de connaissances techniques importantes. Une formulation polie est à préférer et la vulgarité à prohiber.

Il est parfois souhaitable de faire précéder les entrées du journal des modifications par les noms des fichiers modifiés. Cependant, rien n'oblige à mentionner le moindre fichier modifié, notamment si la modification est simple ou répétitive. L'utilisation de caractères joker est possible.

Ne faites pas de suppositions lorsque vous faites référence à un bogue. Indiquez quel était le problème, comment il a été corrigé et ajoutez la chaîne closes: #nnnnn. Veuillez consulter Section 5.8.4 pour plus d'informations.

6.3.2 Erreur communes pour les entrées de journaux de modifications

Les entrée de journal des modifications ne devraient **pas** documenter les points spécifiques de la réalisation du paquet (« si vous cherchez le fichief toto.conf, il est situé dans /etc/titi ») car les administateurs et les utilisateurs sont censés avoir l'habitude de la façon dont ces aspects sont traités sur un système Debian. Pensez, par contre, à documenter la modification de l'emplacement d'un fichier de configuration.

Les seuls bogues fermés par une entrée de journal de modifications devraient être ceux qui sont corrigés par la version correspondante du paquet. Fermer de cette manière des bogues qui n'ont aucun rapport avec la nouvelle version est considéré comme une mavaise habitude. Veuillez consulter Section 5.8.4 .

Les entrées du journal des modifications ne devraient **pas** être utilisées pour des discussions variées avec les émetteurs des rapports de bogues (p. ex. : « je n'ai pas d'erreur de segmentation quand je lance toto avec l'option titi, merci d'envoyer plus d'informations »). De même, les considérations générales sur la vie, l'univers et le reste (« désolé, cet envoi m'a pris plus longtemps que prévu, mais j'avais un rhume ») ou encore des demandes d'aide (« la liste de bogues de ce paquet est très longue, merci de me donner un coup de main ») sont à éviter. Ces mentions ne seront généralement pas remarquées par leur public potentiel et peuvent ennuyer les personnes qui cherchent à lire les modifications encours du paquet. Veuillez vous reporter à Section 5.8.2 pour plus d'informations sur l'utilisation du système de gestion des bogues.

Une tradition assez ancienne veut que les bogues fixés dans les NMU soient acquittés dans la première entrée du journal des modification d'une nouvelle version construite par le mainteneur. Depuis l'existence du suivi de version pour le système de gestion de bogues, cette pratique est obsolète à condition de conserver les entrées du journal des modification des NMUs. Il est éventuellement possible de simplement mentionner les NMUs dans votre propre entrée de journal des modifications.

6.3.3 Erreurs usuelles dans les entrées du journal des modifications

Les exemples suivants sont des erreurs usuelles ou des exemples de mauvaises pratiques dans le style des entrées de journaux de modifications (NdT : le texte est volontairement laissé non traduit).

```
* Fixed all outstanding bugs.
```

Cela ne donne évidemment aucune indication au lecteur.

```
* Applied patch from Jane Random.
```

Que faisait ce correctif ?

```
* Late night install target overhaul.
```

Qu'est-ce que cela a amené ? Est-ce que la mention du fait que cela ait été fait tard la nuit doit nous alerter sur la probable mauvaise qualité du code ?

```
* Fix vsync FU w/ ancient CRTs.
```

Trop d'acronymes qui rendent difficile de savoir ce qu'était le « merdoyage » (NdT : FU signifie « fsckup », donc cet exemple ajoute la vulgarité à l'incompréhensibilité) ou comment il a été corrigé.

```
* This is not a bug, closes: #nnnnnn.
```

Il est inutile de faire un nouvel envoi de paquet pour envoyer cette information. Il suffit de simplement utiliser le système de suivi des bogues. De plus, aucune explication n'est donnée sur les raisons qui font que le problème n'est pas un bogue.

```
* Has been fixed for ages, but I forgot to close; closes: #54321.
```

Si, pour une raison donnée, vous avez omis de mentionner un numéro de bogue dans une entrée précédente, il n'y a pas de problèmes : il suffit de clôturer le bogue normalement dans le système de suivi des bogues. Il est inutile de changer le journal des modifications si on suppose que les explications sur la correction du bogue sont dans le bogue lui-même (cela s'applique également au suivi des bogues des auteurs amont : il est inutile de suivre, dans le journal des modifications, les bogues qu'ils ont corrigés depuis longtemps).

```
* Closes: #12345, #12346, #15432
```

Où est la description ?Si vous ne trouvez pas de message suffisamment explicite, vous pouvez au moins utiliser le titre du rapport de bogue.

6.3.4 Compléter les journaux de modifications avec des fichiers NEWS.Debian

Les nouvelles importantes sur les modifications survenues dans un paquet peuvent êre palcées dans des fichiers `NEWS.Debian`. Ces nouvelles seront arrichées par des outils tels que apt-listchanges, avant tout le reste des modifications. Cette méthode est à privilégier pour diffuser aux utilisateurs d'un paquet les modifications importantes qu'il subit. Il est préférable de l'utiliser plutôt que des notes debconf car ce système permet de revenir lire les fichiers `NEWS.Debian` après l'installation alors qu'un utilisateur peut assez facilement ne pas remarquer l'affichage d'une note debconf (Ndt : a contrario, les fichiers `NEWS.Debian` ne peuvent être traduits).

Le format de ce fichier est analogue à un journal de modifications Debian, mais n'utilise pas d'astérisques et chaque entrée utilise un paragraphe complet plutôt que les mentions succinctes qui prendraient pas dans le journal des modifications. Il est conseillé de traiter le fichier avec `dpkg-parsechangelog`, ce qui permet d'en vérifier la mise en forme, car il ne sera pas automatiquement modifié pendant la construction du paquet, au contraire du journal des modifications. Voici un exemple d'un fichier `NEWS.Debian` réel :

```
cron (3.0pl1-74) unstable; urgency=low

    The checksecurity script is no longer included with the cron package:
    it now has its own package, checksecurity. If you liked the
    functionality provided with that script, please install the new
    package.

 -- Steve Greenland <stevegr@debian.org>  Sat,  6 Sep 2003 17:15:03 -0500
```

Le fichier `NEWS.Debian` est installé sous le nom `/usr/share/doc/paquet/NEWS.Debian.gz`. Il est compressé et porte toujours ce nom même pou rles paquets Debian natifs. Si vous utilisez `debhelper`, `dh_installchangelogs` installera les fichiers `debian/NEWS` automatiquement.

À la différence des journaux de modifications, vous n'avez pas besoin de mettre `NEWS.Debian` à jour à chaque nouvelle version. Il est suffisant de le mettre à jour quand une information importante doit être diffusée aux utilisateurs. Si vous n'avez pas d'information importante à diffuser, il n'est pas nécessaire d'utiliser un fichier `NEWS.Debian` avec le paquet. Pas de nouvelles, bonnes nouvelles !

6.4 Meilleures pratiques pour les scripts du responsable

Les scripts du responsable (« maintainer scripts ») sont les fichiers `debian/postinst`, `debian/preinst`, `debian/prerm` and `debian/postrm`. Ces scripts peuvent prendre en charge les phases d'installation ou de désinstallation non autonatiquement gérées dans la phasesla création ou la suppression de fichiers ou de répertoires. Les instruction qui suivent complètent celles de la charte Debian.

Les scripts du responsable doivent être robustes. Cela signifie que vous devez vous assurer que rien de grave ne se produit si un script est lancé deux fois au lieu d'une.

L'entrée et la sortie standard peuvent être redirigées (p. ex. dans des tuyaux) pour des besoins de journalisation. Il est donc recommandé qu'il ne soient pas dépendants d'un terminal.

Toute interaction avec l'utilisateur doit être limitée au maximum. Lorsqu'elle est nécessaire, vous devriez utiliser le paquet `debconf` comme interface. Veuillez noter que l'interaction doit impérativement se faire à l'étape `configure` du script `postinst`.

Vous devriez garder les scripts du responsable aussi simples que possible et utiliser de préférence des scripts shell POSIX stricts. Veuillez noter que si vous avez besoin de spécificités de bash, vous devez utiliser une ligne « shebang » pour bash. Les scripts POSIX ou Bash sont encouragés par rapport aux scripts Perl, car `debhelper` peut alors y ajouter des fonctions.

Si vous modifiez les scripts du reponsable, veillez à vérifier la suppression du paquet, la double installation et la purge. Vérifiez qu'un paquet purgé est entièrement éliminé, c'est à dire qu'il a supprimé tous les fichiers qu'il a créé, directement ou indirectement dans ses scripts du responsable.

Si vous avez besoin de vérifier l'existence d'une commande, vous devriez utiliser quelque chose comme

```
if [ -x /usr/sbin/install-docs ]; then ...
```

Si vous ne voulez pas coder en dur le chemin d'une commande dans vos scripts de responsable, la fonction shell conforme à POSIX suivante peut vous aider :

```
pathfind() {
    OLDIFS="$IFS"
    IFS=:
    for p in $PATH; do
        if [ -x "$p/$*" ]; then
            IFS="$OLDIFS"
            return 0
        fi
    done
    IFS="$OLDIFS"
    return 1
}
```

Vous pouvez utiliser cette fonction pour rechercher dans `$PATH` une commande donnée, passée en paramètre. Elle renvoie « true » (zéro) si la commande est trouvée et « false » dans le cas contraire. Il s'agit de la méthode la plus portable car `command -v`, **type**, et **which** ne sont pas conformes à POSIX.

Bien que **which** soit acceptable car fourni dans le paquet requis `debianutils`, il n'est pas disponible sur la partition racine mais est situé dans le répertoire `/usr/bin` au lieu de `/bin`, ce qui rend son utilisation impossible si `/usr` n'est pas encore monté. De nombreux scripts ne seront toutefois pas affectés par cela, cependant.

6.5 Gestion de la configuration avec `debconf`

`Debconf` est un système de gestion de configuration qui peut être utilisé par les divers scripts des paquets (`postinst` notamment) pour interagir avec l'utilisateur sur des choix à opérer pour la configuration du paquet. Les interactions directes avec l'utilisateurs doivent être prohibées en faveur de `debconf`, notamment pour permettre des installations non interactives.

`Debconf` est un outil très partique mais souvent mal utilisé. De nombreuses erreurs classiques sotn mentionnées dans la page de manuel debconf-devel(7). Il est indispensable de lire cette page de manuel avant de décider d'utiliser debconf. Quelques bonnes pratiques sont également indiquées dans le présent document.

Les présents conseils comportent des indications sur le style d'écriture et la typographie, des considérations générales sur l'utilisation de `debconf` ainsi que des recommandations plus spécifiques relatives à certaines parties de la distribution (le système d'installation notamment).

6.5.1 N'abusez pas de `debconf`

Depuis que `debconf` est apparu dans Debian, il a été largement trop utilisé et de nombreuses critiques ont été émises à l'encontre de la distribution Debian pour abus d'utilisation de `debconf`, avec la nécessité de répondre à un nombre très important de questions avant d'avoir un quelconque outil installé.

Les notes d'utilisation doivent être réservées à leur emplacement naturel : le fichier `NEWS.Debian` ou `README.Debian`. N'utilisez les notes que pour des points importants qui peuvent directement concerner l'utilisabilité du paquet. Les notes interrompent l'installation tant qu'elles ne sont pas confirmées et elles peuvent conduire à des envois de courriers électroniques aux utilisateurs.

Choisissez soigneusement les questions posées dans les scripts du responsable. Veuillez consulter la page de manuel debconf-devel(7) pour plus de détails sur les priorités. La plupart des questions devraient utiliser les priorités « intermédiaire » ou « basse ».

6.5.2 Recommandations générales pour les auteurs et les traducteurs

6.5.2.1 Écrivez en anglais correct

La plupart des responsables de paquets Debian ne sont pas anglophones. Il n'est donc pas nécessairement facile pour eux d'écrire des écrans correctement.

Penez à utiliser, voire abuser, la liste debian-l10n-english@lists.debian.org. Faites relire vos écrans.

Des écrans mal écrits fournissent une image négative de votre paquet, de votre travail ou même de Debian en général.

Évitez le plus possible le jargon technique.Si certains termes vous sont familiers, ils peuvent être impossible à comprendre pour d'autres. Si vous ne pouvez les éviter, tentez de les expliquer (avec la description étendue). Dans ce cas, tentez de faire la part des choses entre simplicité et verbiage.

6.5.2.2 Pensez aux traducteurs

Les écrans de `debconf` peuvent être traduits. `Debconf` et son paquet jumeau `po-debconf` fournissent un ensemble simple permettant la traduction des écrans par des équipes de traductions ou des traducteurs isolés.

Utilisez des écrans permettant l'utilisation de gettext. Installez le paquet `po-debconf` sur votre machine de développement et lisez sa documentation (`man po-debconf` est un bon début).

Avoid changing templates too often. Changing templates text induces more work to translators which will get their translation fuzzied. A fuzzy translation is a string for which the original changed since it was translated, therefore requiring some update by a translator to be usable. When changes are small enough, the original translation is kept in PO files but marked as `fuzzy`.

If you plan to do changes to your original templates, please use the notification system provided with the `po-debconf` package, namely the **podebconf-report-po**, to contact translators. Most active translators are very responsive and getting their work included along with your modified templates will save you additional uploads. If you use gettext-based templates, the translator's name and e-mail addresses are mentioned in the PO files headers and will be used by **podebconf-report-po**.

A recommended use of that utility is:

```
cd debian/po && podebconf-report-po --languageteam --withtranslators --call -- ↩
    deadline="+10 days"
```

This command will first synchronize the PO and POT files in debian/po with the templates files listed in `debian/po/POTFILES.in`. Then, it will send a call for translation updates to the language team (mentioned in the `Language-Team` field of each PO file) as well as the last translator (mentioned in `Last-translator`). Finally, it will also send a call for new translations, in the debian-i18n@lists.debian.org mailing list.

Giving a deadline to translators is always appreciated, so that they can organize their work. Please remember that some translation teams have a formalized translate/review process and a delay lower than 10 days is considered as unreasonable. A shorter delay puts too much pressure on translation teams and should be kept for very minor changes.

Dans le doute, vous pouvez également contacter l'équipe de traduction d'une langue donnée (debian-l10n-xxxxx@lists.debian.org) ou la liste de diffusion debian-i18n@lists.debian.org.

6.5.2.3 Correction (« unfuzzy ») des traductions complètes lors des corrections de fautes de frappe ou d'orthographe

When the text of a debconf template is corrected and you are **sure** that the change does **not** affect translations, please be kind to translators and *unfuzzy* their translations.

Si cela n'est pas fait, l'ensemble de l'écran debconf ne sera plus traduit tant qu'un traducteur n'aura pas réenvoyé un fichier corrigé.

To *unfuzzy* translations, you can use two methods. The first method does *preventive* search and replace actions in the PO files. The latter uses **gettext** utilities to *unfuzzy* strings.

Preventive unfuzzy method:

1. Try finding a complete translation file **before** the change:

   ```
   for i in debian/po/*po; do echo -n $i:~; msgfmt -o /dev/null
   --statistics $i; done
   ```

   The file only showing *translated* items will be used as the reference file. If there is none (which should not happen if you take care to properly interact with translators), you should use the file with the most translated strings.

2. Identify the needed change. In this example, let's assume the change is about fixing a typo in the word `typo` which was inadvertently written as `tpyo`. Therefore, the change is **s/tpyo/typo**.

3. Check that this change is only applied to the place where you really intend to make it and **not** in any other place where the original string is appropriate. This specifically applies to change in punctuation, for instance.

4. Modify all PO files by using **sed**. The use of that command is recommended over any text editor to guarantee that the files encoding will not be broken by the edit action.

   ```
   cd debian/po
   for i in *.po; do sed -i 's/tpyo/typo/g' $i; done
   ```

5. Change the debconf template file to fix the typo.

6. Run **debconf-updatepo**

7. Check the `foo.po` reference file. Its statistics should not be changed:

   ```
   msgfmt -o /dev/null --statistics debian/po/foo.po
   ```

8. If the file's statistics changed, you did something wrong. Try again or ask for help on the debian-i18n@lists.debian.org mailing list.

Gettext utilities method:

1. Déplacez les fichiers PO incomplets à un endroit temporaire. Les fichiers peuvent être contrôlés avec la commande suivante (qui nécessite l'installation du paquet `gettext`) :

```
for i in debian/po/*po; do echo -n $i:~; msgfmt -o /dev/null
--statistics $i; done
```

2. déplacez tous les fichiers qui indiquent comporter des chaînes floues (fuzzy) dans un répertoire temporaire. Les fichiers qui ne comportent pas de chaînes floues (seulement des chaînes traduites ou non traduites) peuvent être laissés en place.
3. maintenant **et pas avant**, vous pouvez corriger le fichier « templates » en vous assurant à nouveau que les traductions ne risquent pas d'être affectées (des fautes de frappes, des erreurs grammaticales et parfois des erreurs typographiques ne posent en général pas de problèmes).
4. exécutez la commande **debconf-updatepo**. Cette commande va rendre floues les traductions des chaînes que vous avez modifiées. Cela peut être contrôlé avec la commande ci-dessus à nouveau.
5. utilisez la commande suivante :

```
for i in debian/po/*po; do msgattrib --output-file=$i --clear-fuzzy $i; done
```

6. replacez à nouveau dans `debian/po` les fichiers précédemment déplacés.
7. exécutez la commande **debconf-updatepo** à nouveau

6.5.2.4 Ne faites pas de suppositions sur les interfaces utilisateurs

Templates text should not make reference to widgets belonging to some debconf interfaces. Sentences like *If you answer Yes...* have no meaning for users of graphical interfaces which use checkboxes for boolean questions.

Les écrans de type « string » ne devraient pas faire référence aux valeurs par défaut dans leur description. Cela est tout d'abord redondant avec les valeurs visibles par les utilisateurs. Mais également, les valeurs présentées par défaut peuvent être différentes du choix du mainteneur (par exemple, lorsque la base de données de debconf a été pré-renseignée).

De manière plus générale, évitez de faire référence à des actions particulières des utilisateurs et donnez simplement des faits.

6.5.2.5 N'utilisez pas la première personne

You should avoid the use of first person (*I will do this...* or *We recommend...*). The computer is not a person and the Debconf templates do not speak for the Debian developers. You should use neutral construction. Those of you who already wrote scientific publications, just write your templates like you would write a scientific paper. However, try using active voice if still possible, like *Enable this if ...* instead of *This can be enabled if....*

6.5.2.6 Restez neutre en genre

Le monde est fait d'hommes et de femmes. Veuillez utiliser des constructions neutres en genre dans vos écrits.

6.5.3 Définition de champs des modèles (« templates »)

Les informations présentées dans cette partie proviennent pour l'essentiel de la page de manuel debconf-devel(7)

6.5.3.1 Type

6.5.3.1.1 string: offre un champ de saisie où l'utilisateur peut entrer n'importe quelle chaîne de caractère.

6.5.3.1.2 password: demande un mot de passe. Ce champ est à utiliser avec précaution car le mot de passe saisi sera conservé dans la base de données de debconf. Il est conseillé d'effacer cette valeur de la base de donnes dès que possible.

6.5.3.1.3 boolean: offre un choix vrai/faux. Veuillez noter que c'est bien d'un choix vrai/faux, **pas oui/non**...

6.5.3.1.4 select: A choice between one of a number of values. The choices must be specified in a field named 'Choices'. Separate the possible values with commas and spaces, like this: `Choices: yes, no, maybe`.

If choices are translatable strings, the 'Choices' field may be marked as translatable by using `__Choices`. The double underscore will split out each choice in a separate string.

The **po-debconf** system also offers interesting possibilities to only mark **some** choices as translatable. Example:

```
Template: foo/bar
Type: Select
#flag:translate:3
__Choices: PAL, SECAM, Other
_Description: TV standard:
 Please choose the TV standard used in your country.
```

In that example, only the 'Other' string is translatable while others are acronyms that should not be translated. The above allows only 'Other' to be included in PO and POT files.

The debconf templates flag system offers many such possibilities. The po-debconf(7) manual page lists all these possibilities.

6.5.3.1.5 multiselect: similaire au type « select », mais permet de choisir plusieurs (ou aucune) valeurs parmi la liste de choix.

6.5.3.1.6 note: plus qu'une vraie question, ce type indique une note affichée aux utilisateurs. Elle doit être réservée à des informations importantes que l'utilisateur doit absolument voir, car debconf fera tout pour s'assurer qu'elle est visible et notamment interrompra l'installation en attente qu'une touche soit appuyée, voire envoyer la note par courrier électronique dans certains cas.

6.5.3.1.7 text: ce type de données est obsolète. Il ne faut pas l'utiliser.

6.5.3.1.8 error: ce type permet de gérer des messages d'erreur. Il est analogue au type « note ». Les interfaces utilisateur peuvent le présenter différemment (ainsi l'interface « dialog » dessine un écran à fond rouge au lieu de l'écran bleu habituel).

Il est recommandé d'utiliser ce type pour tout message qui requiert l'attention de l'utilisateur pour procéder à une correction, quelle qu'elle soit.

6.5.3.2 Description: description courte et étendue

Les descriptions de modèles comportent deux parties : la partie courte et la partie étendue. La partie courte est celle qui est placée sur la ligne «Description: » du modèle.

La partie courte doit rester courte (environ 50 caractères) afin de pouvoir être gérée par la majorité des interfaces de debconf. La garder courte facilite également le travail des traducteurs car les traductions sont en général plus longues que les textes originaux.

La description courte doit être autonome. Certaines interfaces ne montrent pas la description longue par défaut ou ne la montrent que si l'utilisateur le demande explicitement. Il est ainsi déconseillé d'utiliser des phrases comme « Que voulez-vous faire maintenant ? »

La description courte ne doit pas nécessairement être une phrase entière. Cela est une façon de la garder courte et efficace.

La partie longue ne doit pas répéter la partie courte. Si vous ne trouvez pas de partie longue appropriée, réfléchissez un peu plus. Demandez dans debian-devel. Demandez de l'aide. Prenez un cours d'écriture ! La description longue est importante..et si, malgré tout cela, vous ne trouvez rien d'intéressant à ajouter, laissez-la vide.

La partie longue doit utiliser des phrases complètes. Les paragraphes doivent rester courts pour améliorer la lisibilité. Ne placez pas deux idées différentes dans le même paragraphe mais séparez-les en deux paragraphes.

Ne soyez pas trop verbeux. Les utilisateurs ont tendance à ne pas lire les écrans trop longs. Un maximum de 20 lignes est une limite que vous ne devriez pas dépasser car, avec l'interface standard « dialog », les utilisateurs devront monter et descendre avec des ascenseurs, ce qui sera omis par la plupart des utilisateurs.

La partie longue de la description ne devrait **jamais** comporter de question.

Les parties qui suivent donnent des recommandations spécifiques pour certains types de modèles (string, boolean, etc.).

6.5.3.3 Choices

Ce champ doit être utilisé pour les types Select et Multiselect. Il contient les choix proposés aux utilisateurs. Ces choix doivent être séparés par des virgules.

6.5.3.4 Default

Ce champ optionnel contient la réponse par défaut pour les modèles string, select et multiselect. Dans ce dernier cas, il peut comporter une liste de choix multiples, séparés par des virgules.

6.5.4 Guide de style spécifique à certains modèles

6.5.4.1 Champ type

Pas d'indication particulière si ce n'est choisir le type adapté en se référant à la section précédente.

6.5.4.2 Champ Description

Vous trouverez ici des instruction particulières pour l'écriture du champ Description (parties courtes et longue) selon le type de modèle.

6.5.4.2.1 Modèles string et password

- La description courte est une invite et **pas** un titre. Il faut éviter la forme interrogative (« Adresse IP ? ») et préférer une invite ouverte (« Adresse IP : »). L'utilisation d'un « deux-points » final est recommandée.
- La partie longue complète la partie courte. Il est conseillé d'y expliquer ce qui est demandé, plutôt que répéter la même demande. Utilisez des phrases complètes. Un style d'écriture abrégé est déconseillé.

6.5.4.2.2 Modèles « boolean »

- La partie courte doit utiliser la forme interrogative et se terminer par un point d'interrogation. Un style abrégé est toléré et même encouragé si la question est complexe (les traductions vont être plus longues que la version originale).
- Il est important d'éviter de faire référence aux objets de certains interfaces spécifiques. Une erreur classique est d'utiliser une construction comme « If you answer Yes... » (« Si vous répondez Oui... »).

6.5.4.2.3 Select/Multiselect

- La description courte est une invite et **pas** un titre. N'utilisez **pas** de constructions comme « Please choose... » (« Veuillez choisir... »). Les utilisateurs sont suffisamment intelligents pour comprendre qu'il est nécessaire de choisir quelque chose.
- La description longue complète la partie courte. Elle peut faire référence aux choix disponibles. Elle peut également indiquer que l'utilisateur peut choisir plus d'un parmi les choix disponibles, dans le cas de modèles « multiselect » (bien que l'interface rende en général cela tout à fait clair).

6.5.4.2.4 Notes

- La description courte doit être considérée comme un *titre*.
- La partie longue est ce qui sera affiché comme description plus détaillée de la note. Il est conseillé d'éviter d'y utiliser un style abrégé.
- **N'abusez pas de debconf.**. Les notes sont un des abus les plus fréquents de debconf. Comm eindiqué dans la page de manuel de debconf, elle devraient être réservée pour avertir les utilisateurs de problème très importants. Les fichiers NEWS.Debian ou README.Debian sont les endroits appropriés pour l'information qu'affichent la majorité des notes. Si, à la lecture de ces conseils, vous envisagez de convertir vos modèles de type note en entrée dans NEWS.Debian ou README.Debian, pensez à conserver d'éventuelles traductions existantes.

6.5.4.3 Champ Choices

Si les choix changent souvent, il est suggéré d'utiliser l'astuce « __Choices ». Avec ce format, chaque choix sera une chaîne différente proposée à la traduction, ce qui facilite grandement le travail des traducteurs.

6.5.4.4 Champ Default

Si la valeur par défaut d'un modèle « select » peut être dépendante de la langue utilisée (par exemple s'il s'agit du choix d'une langue par défaut), pensez à utiliser l'astuce « DefaultChoice ».

This special field allow translators to put the most appropriate choice according to their own language. It will become the default choice when their language is used while your own mentioned Default Choice will be used when using English.

Exemple, pris dans le paquet geneweb :

```
Template: geneweb/lang
Type: select
__Choices: Afrikaans (af), Bulgarian (bg), Catalan (ca), Chinese (zh), Czech (cs) ↩
    , Danish (da), Dutch (nl), English (en), Esperanto (eo), Estonian (et),  ↩
    Finnish (fi), French (fr), German (de), Hebrew (he), Icelandic (is), Italian  ↩
    (it), Latvian (lv), Norwegian (no), Polish (pl), Portuguese (pt), Romanian ( ↩
    ro), Russian (ru), Spanish (es), Swedish (sv)
# This is the default choice. Translators may put their own language here
# instead of the default.
# WARNING : you MUST use the ENGLISH FORM of your language
# For instance, the french translator will need to put French (fr) here.
_DefaultChoice: English (en)[ translators, please see comment in PO files]
_Description: Geneweb default language:
```

Veuillez noter l'utilisation de crochets pour autoriser des commentaires internes dans les champs de debconf. Notez également l'utilisation de commentaires qui apparaîtront dans les fichiers de travail des traducteurs.

Les commentaires sont très utiles car l'astuce DefaultChoice est parfois déroutante pour les traducteurs qui doivent y mettre leur propre choix et non une simple traduction.

6.5.4.5 Champ Default

N'utilisez PAS de champ Default vide. Si vous ne souhaitez pas avoir de valeur par défaut, n'utilisez pas du tout ce champ.

Quand vous utilisez po-debconf, pensez à rendre ce champ traduisible si vous pensez qu'il peut l'être.

Si la valeur par défaut peut dépendre de la lanue ou du pays (par exemple une langue par défaut dans un programme), pensez à utiliser le type « _DefaultChoice » documenté dans la page de manuel po-debconf(7).

6.6 Internationalisation

6.6.1 Gestion des traductions debconf

Comme les porteurs, les traducteurs ont une tâche difficile. Ils travaillent sur de nombreux paquets et doivent collaborer avec de nombreux responsables. De plus, ils n'ont généralement pas la langue anglaise comme langue maternelle et vous devez donc faire preuve d'une patience particulière avec eux.

L'objectif de `debconf` est de rendre la configuration des paquets plus facile pour les responsables de paquets et pour les utilisateurs. Initialement, la traduction des écrans de debconf était gérée avec **debconf-mergetemplate**. Cependant, cette technique est désormais obsolète et la meilleure façon d'internationaliser `debconf` est d'utiliser le paquet `po-debconf`. Cette méthode rend plus simple le travail des traducteurs et des responsables et des scripts de transition sont fournis.

Avec `po-debconf`, les traductions sont gérées dans des fichiers `po` (hérités des techniques de traduction utilisées avec **gettext**). Des fichiers modèles contiennent les messages d'origine et les champs à traduire y sont marqués spécifiquement. Lorsque le contenu d'un champ traduisible est modifié, l'emploi de la commande **debconf-updatepo** permet d'indiquer que la traduction a besoin d'une mise à jour par les traducteurs. Ensuite, au moment de la construction du paquet, le programme **dh_installdebconf** s'occupe des opérations nécessaires pour ajouter le modèle avec les traductions à jour dans les paquets binaires. Vous pouvez consulter la page de manuel de po-debconf(7) pour plus d'informations.

6.6.2 Documentation internatonalisée

L'internationalisation de la documentation est primordiale pour les utilisateurs mais représente un travail très important. Même s'il n'est pas possible de supprimer toute le travail nécessaire, il est possible de faciliter le travail des traducteurs.

Si vous maintenez une documentation de quelque taille que ce soit, il sera plus pratique pour les traducteurs d'avoir accès au système de suivi des versions source. Cela leur permet de voir les différences entre deux versions de la documentation et, par conséquent, de mieux voir où les traductions doivent être modifiées. Il est recommandé que la documentation traduite contienne l'indication du système de suivi des versions source qui est utilisé. Un système pratique est fourni par doc-check du paquet `boot-floppies`, qui permet un survol de l'état de la traduction pour toute langue, par l'utilisation de commentaires structurés dans la version du fichier à traduire et, pour le fichier traduit, la version du fichier sur laquelle est basée la traduction. Il est possible d'adapter ce système dans votre propre dépôt CVS.

Si vous maintenez de la documentation en format XML ou SGML, il est conseillé d'isoler l'information indépendant de la langue et de la définir sous forme d'entités dans un fichier à part qui sera inclus par toutes les traductions. Cela rend par exemple plus simple la maintenance d'URL dans de nombreux fichiers.

6.7 Situation usuelle de gestion de paquets

6.7.1 Paquets qui utilisent autoconf/automake

Pouvoir disposer de fichiers `config.sub` et `config.guess` à jour est un point critique pour les porteurs, particulièrement pour les architectures assez volatiles. de très bonnes pratiques applicable à tout paquet qui utilise **autoconf** et/ou **automake** ont été résumées dans `/usr/share/doc/autotools-dev/README.Debian.gz` du paquet `autotools-dev`. Il est fortement recommandé de lire ce fichier et d'en suivre les recommandations.

6.7.2 Bibliothèques

Les paquets fournissant des bibliothèques sont plus difficiles à maintenir pour plusieurs raisons. La charte impose de nombreuses contraintes pour en faciliter la maintenance et garantir que les mises à niveau sont aussi simple que possible quand une nouvelle version amont est disponible. Des erreurs dans une bibliothèque sont susceptibles de rendre inutilisables de très nombreux paquets.

Les bonnes pratiques pour la maintenance de paquets fournissant des bibliothèques ont été rassemblées dans le guide de gestion des paquets de bibliothèques.

6.7.3 Documentation

Veuillez vous assurer que vous suivez la charte de documentation.

Si votre paquet contient de la documentation construite à partir de fichiers XML ou SGML, il est recommandé de ne pas fournir ces fichiers source dans les paquets binaires. Les utilisateurs qui souhaiteraient disposer des sources de la documentation peuvent alors récupérer le paquet source.

La charte indique que la documentation devrait être fournie en format HTML. Il est recommandé de la fournir également dans les formats PDF et texte si cela est pratique et si un affichage de qualité raisonnable est possible. Cependant, il est le plus souvent inapproprié de fournir en format texte simple des versions de documentations dont le format source est HTML.

Les manuels les plus importants qui sont fournis devraient être enregistrés avec `doc-base` lors de leur installation. Veuillez consulter la documentation du paquet `doc-base` pour plus d'informations.

6.7.4 Type particuliers de paquets

Plusieurs types particuliers de paquets utilisent des chartes spécifiques avec les règles et de bonnes pratiques particulières.

- Les paquets liés à Perl utilisent une charte Perl. Des exemples de tels paquets qui appliquent cette charte spécifique sont `libdbd-pg-perl` (module Perl binaire) ou `libmldbm-perl` (module Perl indépendant de l'architecture).
- Les paquets liés à Python utilisent une charte Python. Veuillez consulter le fichier `/usr/share/doc/python/python-policy.txt.gz` du paquet `python` pour plus d'informations.
- Les paquets liés à Emacs utilisent une charte Emacs.
- Les paquets liés à Java utilisent une charte Java.
- Les paquets liés à Ocaml utilisent leur propre charte, que l'on peut trouver dans le fichier `/usr/share/doc/ocaml/ocaml_packaging_policy.gz` du paquet `ocaml`. Un bon exemple est fourni par le paquet source `camlzip`.
- Les paquets fournissant des DTD XML ou SGML devraient suivre les recommandations données dans le paquet `sgml-base-doc`.
- Les paquets Lisp doivent s'enregistrer avec `common-lisp-controller`, pour lequel plus d'information est disponible dans `/usr/share/doc/common-lisp-controller/README.packaging`.

6.7.5 Données indépendantes de l'architecture

Il est fréquent qu'un grand nombre de données indépendantes de l'architecture soient fournies avec un programme. Cela peut être par exemple des fichiers audio, un ensemble d'icônes, des motifs de papier-peint ou d'autres fichiers graphiques. Si la taille de ces données est négligeable par rapport à la taille du reste du paquet, il est probablement préférable de laisser l'ensemble dans un seul paquet.

Cependant, si cette taille est importante, vous devriez considérer de les fournir dans un paquet séparé, indépendant de l'architecture (_all.deb). Cela permet ainsi d'aviter la duplication des mêmes données dans de nombreux paquets binaires, un par architecture. Bien que cela ajoute des entrées dans les fichiers `Packages`, cela permet d'économiser une place importante sur les miroirs de Debian. La séparation des données indépendantes de l'architecture réduit également le temps de traitement de **lintian** (voir Section A.2) lorsqu'il est utilisé sur l'archive Debian en entier.

6.7.6 Nécessitant des paramètres régionaux spécifiques lors de la construction

Si des paramètres régionaux (« locale ») sont nécessaires pour la construction d'un paquet, vous pouvez créer un fichier temporaire avec cette astuce :

Si la variable `LOCPATH` est placée sur l'équivalent de `/usr/lib/locale` et `LC_ALL` sur le nom des paramètres régionaux à créer, vous devriez pouvoir obtenir le résultat escompté dans avoir les privilèges du superutilisateur. La séquence ressemblera alors à :

```
LOCALE_PATH=debian/tmpdir/usr/lib/locale
LOCALE_NAME=en_IN
LOCALE_CHARSET=UTF-8

mkdir -p $LOCALE_PATH
```

```
localedef -i $LOCALE_NAME.$LOCALE_CHARSET -f $LOCALE_CHARSET $LOCALE_PATH/ ↩
    $LOCALE_NAME.$LOCALE_CHARSET

# Using the locale
LOCPATH=$LOCALE_PATH LC_ALL=$LOCALE_NAME.$LOCALE_CHARSET date
```

6.7.7 Rendre les paquets de transition conformes à deborphan

Deborphan est un programme qui permet aux utilisateurs d'identifier les paquets qui peuvent être supprimés sans crainte du système, c'est à dire ceux dont aucun autre paquet ne dépend. Par défaut, l'utilitaire n'effectue sa recherche que parmi les paquets de bibliothèques et les sections « oldlibs », afin de traquer les bibliothèques inutilisées. Cependant, avec le paramètre approprié, il peut rechercher d'autres paquets inutiles.

Par exemple, le paramètre `--guess-dummy` de la commande **deborphan** permet de rechercher les paquets de transition qui étaient nécessaires lors de mises à niveau mais peuvent être supprimés sans problème. Pour cela, il recherche la chaîne « dummy » ou « transitional » dans leur description courte.

Ainsi, lorsque vous avez besoin de créer un tel paquet, veuillez prendre soin d'ajouter ce texte à sa description courte. Il est facile de trouver des exemples avec les commandes **apt-cache search . | grep dummy** ou **apt-cache search . | grep transitional**.

6.7.8 Les meilleures pratiques pour les fichiers `orig.tar.gz`

Il existe deux sortes différentes d'archives source d'origine. Les sources originelles (« pristine ») et les sources reconstruites (« repackaged »).

6.7.8.1 Sources originelles (« pristine »)

The defining characteristic of a pristine source tarball is that the `.orig.tar.gz` file is byte-for-byte identical to a tarball officially distributed by the upstream author. [1] This makes it possible to use checksums to easily verify that all changes between Debian's version and upstream's are contained in the Debian diff. Also, if the original source is huge, upstream authors and others who already have the upstream tarball can save download time if they want to inspect your packaging in detail.

Il n'existe pas de convention universellement acceptée pour la structure de répertoires que devraient adopter les auteurs amont dans les archives qu'ils publient, mais **dpkg-source** peut de toute manière traiter le plupart des archives amont comme des sources originelles. La stratégie de cette commande est la suivante :

1. Elle extrait l'archive dans un répertoire temporaire :

   ```
   zcat path/to/<packagename>_<upstream-version>.orig.tar.gz | tar xf -
   ```

2. Si, après cela, le répertoire temporaire ne contient qu'un seul répertoire sans fichiers, **dpkg-source** renomme ce répertoire en `<nomdupaquet>-<version-amont>(.orig)`. Le nom du répertoire parent de l'archive tar n'a pas d'importance et est oublié;
3. Si ce n'est pas le cas, l'archive amont a été créée sans répertoire parent (honte à l'auteur amont !). Dans ce cas, **dpkg-source** renomme le répertoire temporaire lui-même en `<nomdupaquet>-<version-amont>(.orig)`.

6.7.8.2 Source amont reconstruite

Vous **devriez** envoyer les paquets avec une archive source inchangée, dans la mesure du possible. Il existe cependant plusieurs raisons qui peuvent rendre cela impossible. C'est notamment le cas si les auteurs amont ne distribuent pas d'archive tar compressée du tout ou si l'archive amont contient des

[1] We cannot prevent upstream authors from changing the tarball they distribute without also incrementing the version number, so there can be no guarantee that a pristine tarball is identical to what upstream *currently* distributing at any point in time. All that can be expected is that it is identical to something that upstream once *did* distribute. If a difference arises later (say, if upstream notices that he wasn't using maximal compression in his original distribution and then re-**gzip**s it), that's just too bad. Since there is no good way to upload a new `.orig.tar.gz` for the same version, there is not even any point in treating this situation as a bug.

parties non conformes aux principes du logiciel libre selon Debian, qui doivent être supprimées avant l'envoi.

Dans ces cas, le responsable doit construire menuellement une archive `.orig.tar.gz` . Cette archive sera appelée une archive amont reconstruite. Il est important de noter qu'elle reste différente d'un paquet natif. Une archive reconstruite est toujours fournie avec les changements propres à Debian dans un fichier `.diff.gz` séparé et son numéro de version est toujours composé de `<version-amont>` et `<révision-debian>`.

Il peut exister des cas où il est souhaitable de reconstruire une archive source alors que les auteurs amont fournissent bien une archive `.tar.gz` qui pourrait être utilisée directement. Le plus évident est la recherche d'un gain de place significatif par recompression ou par suppression de scories inutiles de l'archive source d'origine. Il est important que le responsable exerce avec discernement son propre jugement et soit prêt à le justifier si l'archive source est reconstruite alors qu'elle aurait pu être fournie telle quelle.

Un fichier `.orig.tar.gz` reconstruit

1. should be documented in the resulting source package. Detailed information on how the repackaged source was obtained, and on how this can be reproduced should be provided in `debian/copyright`. It is also a good idea to provide a `get-orig-source` target in your `debian/rules` file that repeats the process, as described in the Policy Manual, Main building script: debian/rules.
2. **should not** contain any file that does not come from the upstream author(s), or whose contents has been changed by you. [2]
3. **should**, except where impossible for legal reasons, preserve the entire building and portablility infrastructure provided by the upstream author. For example, it is not a sufficient reason for omitting a file that it is used only when building on MS-DOS. Similarly, a Makefile provided by upstream should not be omitted even if the first thing your `debian/rules` does is to overwrite it by running a configure script.

   (*Rationale:* It is common for Debian users who need to build software for non-Debian platforms to fetch the source from a Debian mirror rather than trying to locate a canonical upstream distribution point).
4. **should** use `<packagename>-<upstream-version>.orig` as the name of the top-level directory in its tarball. This makes it possible to distinguish pristine tarballs from repackaged ones.
5. **should** be gzipped with maximal compression.

The canonical way to meet the latter two points is to let `dpkg-source -b` construct the repackaged tarball from an unpacked directory.

6.7.8.3 Changing binary files in `diff.gz`

Sometimes it is necessary to change binary files contained in the original tarball, or to add binary files that are not in it. If this is done by simply copying the files into the debianized source tree, **dpkg-source** will not be able to handle this. On the other hand, according to the guidelines given above, you cannot include such a changed binary file in a repackaged `orig.tar.gz`. Instead, include the file in the `debian` directory in **uuencode**d (or similar) form [3]. The file would then be decoded and copied to its place during the build process. Thus the change will be visible quite easy.

Some packages use **dbs** to manage patches to their upstream source, and always create a new `orig.tar.gz` file that contains the real `orig.tar.gz` in its toplevel directory. This is questionable with respect to the preference for pristine source. On the other hand, it is easy to modify or add binary files in

[2] As a special exception, if the omission of non-free files would lead to the source failing to build without assistance from the Debian diff, it might be appropriate to instead edit the files, omitting only the non-free parts of them, and/or explain the situation in a README.source file in the root of the source tree. But in that case please also urge the upstream author to make the non-free components easier seperable from the rest of the source.

[3] The file should have a name that makes it clear which binary file it encodes. Usually, some postfix indicating the encoding should be appended to the original filename. Note that you don't need to depend on `sharutils` to get the **uudecode** program if you use **perl**'s `pack` function. The code could look like

```
uuencode-file:
    perl -ne 'print(pack "u", $$_);' $(file) > $(file).uuencoded

uudecode-file:
    perl -ne 'print(unpack "u", $$_);' $(file).uuencoded > $(file)
```

this case: Just put them into the newly created `orig.tar.gz` file, besides the real one, and copy them to the right place during the build process.

6.7.9 Best practices for debug packages

A debug package is a package with a name ending in -dbg, that contains additional information that gdb can use. Since Debian binaries are stripped by default, debugging information, including function names and line numbers, is otherwise not available when running gdb on Debian binaries. Debug packages allow users who need this additional debugging information to install it, without bloating a regular system with the information.

It is up to a package's maintainer whether to create a debug package or not. Maintainers are encouraged to create debug packages for library packages, since this can aid in debugging many programs linked to a library. In general, debug packages do not need to be added for all programs; doing so would bloat the archive. But if a maintainer finds that users often need a debugging version of a program, it can be worthwhile to make a debug package for it. Programs that are core infrastructure, such as apache and the X server are also good candidates for debug packages.

Some debug packages may contain an entire special debugging build of a library or other binary, but most of them can save space and build time by instead containing separated debugging symbols that gdb can find and load on the fly when debugging a program or library. The convention in Debian is to keep these symbols in `/usr/lib/debug/path`, where *`path`* is the path to the executable or library. For example, debugging symbols for `/usr/bin/foo` go in `/usr/lib/debug/usr/bin/foo`, and debugging symbols for `/usr/lib/libfoo.so.1` go in `/usr/lib/debug/usr/lib/libfoo.so.1`.

The debugging symbols can be extracted from an object file using **objcopy --only-keep-debug**. Then the object file can be stripped, and **objcopy --add-gnu-debuglink** used to specify the path to the debugging symbol file. objcopy(1) explains in detail how this works.

The **dh_strip** command in debhelper supports creating debug packages, and can take care of using **objcopy** to separate out the debugging symbols for you. If your package uses debhelper, all you need to do is call **dh_strip --dbg-package=libfoo-dbg**, and add an entry to `debian/control` for the debug package.

Note that the debug package should depend on the package that it provides debugging symbols for, and this dependency should be versioned. For example:

```
Depends: libfoo (= ${binary:Version})
```

Chapitre 7

Au-delà de l'empaquetage

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

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

7.1 Rapporter des bogues

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

Read the instructions for reporting bugs in the Debian bug tracking system.

Try to submit the bug from a normal user account at which you are likely to receive mail, so that people can reach you if they need further information about the bug. Do not submit bugs as root.

You can use a tool like reportbug(1) to submit bugs. It can automate and generally ease the process.

Make sure the bug is not already filed against a package. Each package has a bug list easily reachable at `http://bugs.debian.org/`*`packagename`* Utilities like querybts(1) can also provide you with this information (and **reportbug** will usually invoke **querybts** before sending, too).

Try to direct your bugs to the proper location. When for example your bug is about a package which overwrites files from another package, check the bug lists for *both* of those packages in order to avoid filing duplicate bug reports.

For extra credit, you can go through other packages, merging bugs which are reported more than once, or tagging bugs `fixed' when they have already been fixed. Note that when you are neither the bug submitter nor the package maintainer, you should not actually close the bug (unless you secure permission from the maintainer).

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

7.1.1 Reporting lots of bugs at once (mass bug filing)

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, and mentioning the fact in the subject of your mail. 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.

Please use the programms **dd-list** and if appropriate **whodepends** (from the package devscripts) to generate a list of all affected packages, and include the output in your mail to debian-devel@lists.debian.org.

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

7.1.1.1 Usertags

You may wish to use BTS usertags when submitting bugs across a number of packages. Usertags are similar to normal tags such as 'patch' and 'wishlist' but differ in that they are user-defined and occupy a namespace that is unique to a particular user. This allows multiple sets of developers to 'usertag' the same bug in different ways without conflicting.

To add usertags when filing bugs, specify the `User` and `Usertags` pseudo-headers:

```
To: submit@bugs.debian.org
Subject: <title-of-bug>

Package: <pkgname>
[ ... ]
User: <email-addr>
Usertags: <tag-name> [ <tag-name> ... ]

<description-of-bug ...>
```

Note that tags are seperated by spaces and cannot contain underscores. If you are filing bugs for for a particular group or team it is recommended that you set the `User` to an appropriate mailing list after describing your intention there.

To view bugs tagged with a specific usertag, visit `http://bugs.debian.org/cgi-bin/pkgreport.cgi?users=<email-addr>&tag=<tag-name>`.

7.2 Quality Assurance effort

7.2.1 Daily work

Even though there is a dedicated group of people for Quality Assurance, QA duties are not reserved solely for them. You can participate in this effort by keeping your packages as bug-free as possible, and as lintian-clean (see Section A.2.1) as possible. If you do not find that possible, then you should consider orphaning some of your packages (see Section 5.9.4). Alternatively, you may ask the help of other people in order to catch up with the backlog of bugs that you have (you can ask for help on debian-qa@lists.debian.org or debian-devel@lists.debian.org). At the same time, you can look for co-maintainers (see Section 5.12).

7.2.2 Bug squashing parties

From time to time the QA group organizes bug squashing parties to get rid of as many problems as possible. They are announced on debian-devel-announce@lists.debian.org and the announcement explains which area will be the focus of the party: usually they focus on release critical bugs but it may happen that they decide to help finish a major upgrade (like a new **perl** version which requires recompilation of all the binary modules).

The rules for non-maintainer uploads differ during the parties because the announcement of the party is considered prior notice for NMU. If you have packages that may be affected by the party (because they have release critical bugs for example), you should send an update to each of the corresponding bug to explain their current status and what you expect from the party. If you don't want an NMU, or if you're only interested in a patch, or if you will deal yourself with the bug, please explain that in the BTS.

People participating in the party have special rules for NMU, they can NMU without prior notice if they upload their NMU to DELAYED/3-day at least. All other NMU rules apply as usually; they should send the patch of the NMU to the BTS (to one of the open bugs fixed by the NMU, or to a new bug, tagged fixed). They should also respect any particular wishes of the maintainer.

If you don't feel confident about doing an NMU, just send a patch to the BTS. It's far better than a broken NMU.

7.3 Contacting other maintainers

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

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

You may also be interested in contacting the persons who are subscribed to a given source package via Section 4.10 . You can do so by using the `<package>@packages.qa.debian.org` email address.

7.4 Dealing with inactive and/or unreachable maintainers

If you notice that a package is lacking maintenance, you should make sure that the maintainer is active and will continue to work on their packages. It is possible that they are not active any more, but haven't registered out of the system, so to speak. On the other hand, it is also possible that they just need a reminder.

There is a simple system (the MIA database) in which information about maintainers who are deemed Missing In Action is recorded. When a member of the QA group contacts an inactive maintainer or finds more information about one, this is recorded in the MIA database. This system is available in `/org/qa.debian.org/mia` on the host `qa.debian.org` , and can be queried with the **mia-query** tool. Use **mia-query --help** to see how to query the database. If you find that no information has been recorded about an inactive maintainer yet, or that you can add more information, you should generally proceed as follows.

The first step is to politely contact the maintainer, and wait a reasonable time for a response. It is quite hard to define reasonable time, but it is important to take into account that real life is sometimes very hectic. One way to handle this would be to send a reminder after two weeks.

If the maintainer doesn't reply within four weeks (a month), one can assume that a response will probably not happen. If that happens, you should investigate further, and try to gather as much useful information about the maintainer in question as possible. This includes:

- The `echelon` information available through the developers' LDAP database, which indicates when the developer last posted to a Debian mailing list. (This includes mails about uploads distributed via the debian-devel-changes@lists.debian.org list.) Also, remember to check whether the maintainer is marked as on vacation in the database.
- The number of packages this maintainer is responsible for, and the condition of those packages. In particular, are there any RC bugs that have been open for ages? Furthermore, how many bugs are there in general? Another important piece of information is whether the packages have been NMUed, and if so, by whom.
- Is there any activity of the maintainer outside of Debian? For example, they might have posted something recently to non-Debian mailing lists or news groups.

A bit of a problem are packages which were sponsored — the maintainer is not an official Debian developer. The `echelon` information is not available for sponsored people, for example, so you need to find and contact the Debian developer who has actually uploaded the package. Given that they signed the package, they're responsible for the upload anyhow, and are likely to know what happened to the person they sponsored.

It is also allowed to post a query to debian-devel@lists.debian.org, asking if anyone is aware of the whereabouts of the missing maintainer. Please Cc: the person in question.

Once you have gathered all of this, you can contact mia@qa.debian.org. People on this alias will use the information you provide in order to decide how to proceed. For example, they might orphan one or all of the packages of the maintainer. If a package has been NMUed, they might prefer to contact the NMUer before orphaning the package — perhaps the person who has done the NMU is interested in the package.

One last word: please remember to be polite. We are all volunteers and cannot dedicate all of our time to Debian. Also, you are not aware of the circumstances of the person who is involved. Perhaps they might be seriously ill or might even have died — you do not know who may be on the receiving

side. Imagine how a relative will feel if they read the e-mail of the deceased and find a very impolite, angry and accusing message!

On the other hand, although we are volunteers, we do have a responsibility. So you can stress the importance of the greater good — if a maintainer does not have the time or interest anymore, they should let go and give the package to someone with more time.

If you are interested in working in the MIA team, please have a look at the README file in `/org/qa.debian.org/mia` on `qa.debian.org` where the technical details and the MIA procedures are documented and contact mia@qa.debian.org.

7.5 Interacting with prospective Debian developers

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

7.5.1 Sponsoring packages

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

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

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

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

7.5.2 Managing sponsored packages

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

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

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

Once the package meets Debian standards, build and sign it with

```
dpkg-buildpackage -k KEY-ID
```

before uploading it to the incoming directory. Of course, you can also use any part of your *KEY-ID*, as long as it's unique in your secret keyring.

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

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

You are encouraged to keep tabs on the package you sponsor using Section 4.10 .

7.5.3 Advocating new developers

See the page about advocating a prospective developer at the Debian web site.

7.5.4 Handling new maintainer applications

Please see Checklist for Application Managers at the Debian web site.

Chapitre 8

Internationalization and Translations

Debian supports an ever-increasing number of natural languages. Even if you are a native English speaker and do not speak any other language, it is part of your duty as a maintainer to be aware of issues of internationalization (abbreviated i18n because there are 18 letters between the 'i' and the 'n' in internationalization). Therefore, even if you are ok with English-only programs, you should read most of this chapter.

According to Introduction to i18n from Tomohiro KUBOTA, I18N (internationalization) means modification of a software or related technologies so that a software can potentially handle multiple languages, customs, and so on in the world, while L10N (localization) means implementation of a specific language for an already internationalized software.

l10n and i18n are interconnected, but the difficulties related to each of them are very different. It's not really difficult to allow a program to change the language in which texts are displayed based on user settings, but it is very time consuming to actually translate these messages. On the other hand, setting the character encoding is trivial, but adapting the code to use several character encodings is a really hard problem.

Setting aside the i18n problems, where no general guideline can be given, there is actually no central infrastructure for l10n within Debian which could be compared to the buildd mechanism for porting. So most of the work has to be done manually.

8.1 How translations are handled within Debian

Handling translation of the texts contained in a package is still a manual task, and the process depends on the kind of text you want to see translated.

For program messages, the gettext infrastructure is used most of the time. Most of the time, the translation is handled upstream within projects like the Free Translation Project, the Gnome translation Project or the KDE one. The only centralized resource within Debian is the Central Debian translation statistics, where you can find some statistics about the translation files found in the actual packages, but no real infrastructure to ease the translation process.

An effort to translate the package descriptions started long ago, even if very little support is offered by the tools to actually use them (i.e., only APT can use them, when configured correctly). Maintainers don't need to do anything special to support translated package descriptions; translators should use the DDTP.

For debconf templates, maintainers should use the po-debconf package to ease the work of translators, who could use the DDTP to do their work (but the French and Brazilian teams don't). Some statistics can be found both on the DDTP site (about what is actually translated), and on the Central Debian translation statistics site (about what is integrated in the packages).

For web pages, each l10n team has access to the relevant CVS, and the statistics are available from the Central Debian translation statistics site.

For general documentation about Debian, the process is more or less the same as for the web pages (the translators have access to the CVS), but there are no statistics pages.

For package-specific documentation (man pages, info documents, other formats), almost everything remains to be done.

Most notably, the KDE project handles translation of its documentation in the same way as its program messages.

There is an effort to handle Debian-specific man pages within a specific CVS repository.

8.2 I18N & L10N FAQ for maintainers

This is a list of problems that maintainers may face concerning i18n and l10n. While reading this, keep in mind that there is no real consensus on these points within Debian, and that this is only advice. If you have a better idea for a given problem, or if you disagree on some points, feel free to provide your feedback, so that this document can be enhanced.

8.2.1 How to get a given text translated

To translate package descriptions or debconf templates, you have nothing to do; the DDTP infrastructure will dispatch the material to translate to volunteers with no need for interaction from your part.

For all other material (gettext files, man pages, or other documentation), the best solution is to put your text somewhere on the Internet, and ask on debian-i18n for a translation in different languages. Some translation team members are subscribed to this list, and they will take care of the translation and of the reviewing process. Once they are done, you will get your translated document from them in your mailbox.

8.2.2 How to get a given translation reviewed

From time to time, individuals translate some texts in your package and will ask you for inclusion of the translation in the package. This can become problematic if you are not fluent in the given language. It is a good idea to send the document to the corresponding l10n mailing list, asking for a review. Once it has been done, you should feel more confident in the quality of the translation, and feel safe to include it in your package.

8.2.3 How to get a given translation updated

If you have some translations of a given text lying around, each time you update the original, you should ask the previous translator to update the translation with your new changes. Keep in mind that this task takes time; at least one week to get the update reviewed and all.

If the translator is unresponsive, you may ask for help on the corresponding l10n mailing list. If everything fails, don't forget to put a warning in the translated document, stating that the translation is somehow outdated, and that the reader should refer to the original document if possible.

Avoid removing a translation completely because it is outdated. Old documentation is often better than no documentation at all for non-English speakers.

8.2.4 How to handle a bug report concerning a translation

The best solution may be to mark the bug as forwarded to upstream, and forward it to both the previous translator and his/her team (using the corresponding debian-l10n-XXX mailing list).

8.3 I18N & L10N FAQ for translators

While reading this, please keep in mind that there is no general procedure within Debian concerning these points, and that in any case, you should collaborate with your team and the package maintainer.

8.3.1 How to help the translation effort

Choose what you want to translate, make sure that nobody is already working on it (using your debian-l10n-XXX mailing list), translate it, get it reviewed by other native speakers on your l10n mailing list, and provide it to the maintainer of the package (see next point).

8.3.2 How to provide a translation for inclusion in a package

Make sure your translation is correct (asking for review on your l10n mailing list) before providing it for inclusion. It will save time for everyone, and avoid the chaos resulting in having several versions of the same document in bug reports.

The best solution is to file a regular bug containing the translation against the package. Make sure to use the 'PATCH' tag, and to not use a severity higher than 'wishlist', since the lack of translation never prevented a program from running.

8.4 Best current practice concerning l10n

- As a maintainer, never edit the translations in any way (even to reformat the layout) without asking on the corresponding l10n mailing list. You risk for example breaksing the encoding of the file by doing so. Moreover, what you consider an error can be right (or even needed) in the given language.
- As a translator, if you find an error in the original text, make sure to report it. Translators are often the most attentive readers of a given text, and if they don't report the errors they find, nobody will.
- In any case, remember that the major issue with l10n is that it requires several people to cooperate, and that it is very easy to start a flamewar about small problems because of misunderstandings. So if you have problems with your interlocutor, ask for help on the corresponding l10n mailing list, on debian-i18n, or even on debian-devel (but beware, l10n discussions very often become flamewars on that list :)
- In any case, cooperation can only be achieved with **mutual respect**.

Annexe A

Aperçu des outils du responsable Debian

Cette section contient un aperçu rapide des outils dont dispose leresponsable. Cette liste n'est ni complète, ni définitive, il s'agitjuste d'un guide des outils les plus utilisés.

Les outils du responsable Debian sont destinés à aider les responsableset libérer leur temps pour des tâches plus cruciales. Comme le dit LarryWall, il y a plus d'une manière de le faire.

Certaines personnes préfèrent utiliser des outils de haut niveau,d'autres pas. Debian n'a pas de position officielle sur laquestion ; tout outil conviendra du moment qu'il fait leboulot. C'est pourquoi cette section n'a pas été conçue pour indiquer àchacun quel outil il doit utiliser ou comment il devrait faire pourgérer sa charge de responsable Debian. Elle n'est pas non plus destinéeà favoriser l'utilisation d'un outil aux dépens d'un autre.

La plupart des descriptions de ces outils proviennent des descriptionsde leurs paquets. Vous trouverez plus d'informations dans lesdocumentations de ces paquets. Vous pouvez aussi obtenir plusd'informations avec la commande `apt-cache show<nom_paquet>`.

A.1 Outils de base

Les outils suivants sont pratiquement nécessaires à tout responsable.

A.1.1 `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 manipulate packages; as such, they are essential for any Debian maintainer.

A.1.2 `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 as modules.

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

Many feel that this system should be used for all packages which require interactive configuration; see Section 6.5 . `debconf` is not currently required by Debian Policy, but that may change in the future.

A.1.3 `fakeroot`

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

A.2 Package lint tools

According to the Free On-line Dictionary of Computing (FOLDOC), `lint' is a Unix C language processor which carries out more thorough checks on the code than is usual with C compilers. Package lint tools help package maintainers by automatically finding common problems and policy violations in their packages.

A.2.1 `lintian`

`lintian` dissects Debian packages and emits information about bugs and policy violations. It contains automated checks for many aspects of Debian policy as well as some checks for common errors.

You should periodically get the newest `lintian` from `unstable` and check over all your packages. Notice that the `-i` option provides detailed explanations of what each error or warning means, what its basis in Policy is, and commonly how you can fix the problem.

Refer to Section 5.3 for more information on how and when to use Lintian.

You can also see a summary of all problems reported by Lintian on your packages at `http://lintian.debian.org/`. These reports contain the latest **lintian** output for the whole development distribution (`unstable`).

A.2.2 `debdiff`

debdiff (from the `devscripts` package, Section A.6.1) compares file lists and control files of two packages. It is a simple regression test, as it will help you notice if the number of binary packages has changed since the last upload, or if something has changed in the control file. Of course, some of the changes it reports will be all right, but it can help you prevent various accidents.

You can run it over a pair of binary packages:

```
debdiff package_1-1_arch.deb package_2-1_arch.deb
```

Or even a pair of changes files:

```
debdiff package_1-1_arch.changes package_2-1_arch.changes
```

For more information please see debdiff(1).

A.3 Helpers for `debian/rules`

Package building tools make the process of writing `debian/rules` files easier. See Section 6.1.1 for more information about why these might or might not be desired.

A.3.1 `debhelper`

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

Unlike some approaches, `debhelper` is broken into several small, simple commands which act in a consistent manner. As such, it allows more fine-grained control than some of the other debian/rules tools.

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

A.3.2 `debmake`

`debmake`, a precursor to `debhelper`, is a more coarse-grained `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`. It is a bug to use `debmake` in new packages. New packages using `debmake` will be rejected from the archive.

A.3.3 dh-make

The dh-make package contains **dh_make**, a program that creates a skeleton of files necessary to build a Debian package out of a source tree. As the name suggests, **dh_make** is a rewrite of debmake and its template files use dh_* programs from debhelper.

While the rules files generated by **dh_make** are in general a sufficient basis for a working package, they are still just the groundwork: the burden still lies on the maintainer to finely tune the generated files and make the package entirely functional and Policy-compliant.

A.3.4 yada

yada is another packaging helper tool. It uses a debian/packages file to auto-generate debian/rules and other necessary files in the debian/ subdirectory. The debian/packages file contains instruction to build packages and there is no need to create any Makefile files. There is possibility to use macro engine similar to the one used in SPECS files from RPM source packages.

For more informations see YADA site.

A.3.5 equivs

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

A.4 Package builders

The following packages help with the package building process, general driving **dpkg-buildpackage** as well as handling supporting tasks.

A.4.1 cvs-buildpackage

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

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

A.4.2 debootstrap

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

Having a system like this can be useful in many ways. For instance, you can **chroot** into it if you want to test your build dependencies. Or you can test how your package behaves when installed into a bare base system. Chroot builders use this package; see below.

A.4.3 pbuilder

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

A related package is pbuilder-uml, which goes even further by doing the build within a User Mode Linux environment.

A.4.4 sbuild

sbuild is another automated builder. It can use chrooted environments as well. It can be used stand-alone, or as part of a networked, distributed build environment. As the latter, it is part of the system used by porters to build binary packages for all the available architectures. See Section 5.10.3.3 for more information, and http://buildd.debian.org/ to see the system in action.

A.5 Package uploaders

The following packages help automate or simplify the process of uploading packages into the official archive.

A.5.1 dupload

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

A.5.2 dput

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

A.5.3 dcut

The dcut script (part of the package Section A.5.2) helps in removing files from the ftp upload directory.

A.6 Maintenance automation

The following tools help automate different maintenance tasks, from adding changelog entries or signature lines and looking up bugs in Emacs to making use of the newest and official config.sub.

A.6.1 devscripts

devscripts is a package containing wrappers and tools which are very helpful for maintaining your Debian packages. Example scripts include **debchange** and **dch**, which manipulate your debian/changelog file from the command-line, and **debuild**, which is a wrapper around **dpkg-buildpackage**. The **bts** utility is also very helpful to update the state of bug reports on the command line. **uscan** can be used to watch for new upstream versions of your packages. **debrsign** can be used to remotely sign a package prior to upload, which is nice when the machine you build the package on is different from where your GPG keys are.

See the devscripts(1) manual page for a complete list of available scripts.

A.6.2 autotools-dev

autotools-dev contains best practices for people who maintain packages which use **autoconf** and/or **automake**. Also contains canonical config.sub and config.guess files which are known to work on all Debian ports.

A.6.3 dpkg-repack

dpkg-repack creates Debian package file out of a package that has already been installed. If any changes have been made to the package while it was unpacked (e.g., files in /etc were modified), the new package will inherit the changes.

This utility can make it easy to copy packages from one computer to another, or to recreate packages which are installed on your system but no longer available elsewhere, or to save the current state of a package before you upgrade it.

A.6.4 alien

alien converts binary packages between various packaging formats, including Debian, RPM (RedHat), LSB (Linux Standard Base), Solaris, and Slackware packages.

A.6.5 debsums

debsums checks installed packages against their MD5 sums. Note that not all packages have MD5 sums, since they aren't required by Policy.

A.6.6 dpkg-dev-el

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

A.6.7 dpkg-depcheck

dpkg-depcheck (from the `devscripts` package, Section A.6.1) runs a command under **strace** to determine all the packages that were used by the said command.

For Debian packages, this is useful when you have to compose a `Build-Depends` line for your new package: running the build process through **dpkg-depcheck** will provide you with a good first approximation of the build-dependencies. For example:

```
dpkg-depcheck -b debian/rules build
```

dpkg-depcheck can also be used to check for run-time dependencies, especially if your package uses exec(2) to run other programs.

For more information please see dpkg-depcheck(1).

A.7 Porting tools

The following tools are helpful for porters and for cross-compilation.

A.7.1 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`.

A.7.2 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.

A.8 Documentation and information

The following packages provide information for maintainers or help with building documentation.

A.8.1 docbook-xml

docbook-xml provides the DocBook XML DTDs, which are commonly used for Debian documentation (as is the older debiandoc SGML DTD). This manual, for instance, is written in DocBook XML.

The docbook-xsl package provides the XSL files for building and styling the source to various output formats. You will need an XSLT processor, such as xsltproc, to use the XSL stylesheets. Documentation for the stylesheets can be found in the various docbook-xsl-doc-* packages.

To produce PDF from FO, you need an FO processor, such as xmlroff or fop. Another tool to generate PDF from DocBook XML is dblatex.

A.8.2 debiandoc-sgml

debiandoc-sgml provides the DebianDoc SGML DTD, which is commonly used for Debian documentation, but is now deprecated (docbook-xml should be used instead). It also provides scripts for building and styling the source to various output formats.

Documentation for the DTD can be found in the debiandoc-sgml-doc package.

A.8.3 debian-keyring

Contains the public GPG and PGP keys of Debian developers. See Section 3.2 and the package documentation for more information.

A.8.4 debian-maintainers

Contains the public GPG keys of Debian Maintainers. See http://wiki.debian.org/Maintainers for more information.

A.8.5 debview

debview provides an Emacs mode for viewing Debian binary packages. This lets you examine a package without unpacking it.