

Guide du nouveau responsable Debian

Josip Rodin <joy-mg@debian.org>

Mohammed Adnène Trojette <adn+deb@diwi.org>
et les membres de la liste <debian-l10n-french@lists.debian.org>
Frédéric Dumont (ancien traducteur) <frederic.dumont@easynet.be>

version 1.2.3, 18 janvier 2005.

Copyright

Copyright © 1998-2002 Josip Rodin.

Ce document peut être utilisé selon les termes de la Licence publique générale de GNU version 2 ou suivante.

Ce document a été créé en se basant sur les deux suivants :

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

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

Table des matières

1 Commencer de la bonne manière	1
1.1 Programmes nécessaires au développement	1
1.2 Plus d'informations	3
2 Premiers pas	5
2.1 Choisir votre programme	5
2.2 Obtenir le programme, et l'essayer	6
2.3 Les noms et versions des paquets	7
2.4 « Debianisation » initiale	8
3 Modifier les sources	9
3.1 Installer dans un sous-répertoire	9
3.2 Bibliothèques différentes	12
4 Ce qui est requis sous debian/	13
4.1 Fichier « control »	13
4.2 fichier « copyright »	18
4.3 changelog	19
4.4 fichier « rules »	20
5 Autres fichiers dans le répertoire debian/	25
5.1 README.Debian	25
5.2 conffiles.ex	26
5.3 cron.d.ex	26
5.4 dirs	26

5.5	docs	27
5.6	emacsen-*.ex	27
5.7	init.d.ex	28
5.8	manpage.1.ex, manpage.sgml.ex	28
5.9	menu.ex	29
5.10	watch.ex	29
5.11	ex.package.doc-base	30
5.12	postinst.ex, preinst.ex, postrm.ex, prerm.ex	30
6	Construire le paquet	31
6.1	Reconstruction complète	31
6.2	Reconstruction rapide	32
6.3	La commande debuild	32
6.4	Le système dpatch	33
6.5	Inclure orig.tar.gz pour le téléchargement	34
7	Contrôler les erreurs du paquet	35
7.1	Les paquets lintian et linda	35
7.2	La commande mc	35
7.3	La commande debdiff	36
7.4	La commande interdiff	36
7.5	La commande debi	36
7.6	Le paquet pbuilder	36
8	Envoyer votre paquet	39
8.1	Envoyer vers l'archive Debian	39
8.2	Envoyer vers une archive privée	40
9	Mettre à jour le paquet	43
9.1	Nouvelle révision Debian	43
9.2	Nouvelle version amont (basique)	43
9.3	Nouvelle version amont (réel)	44
9.4	Le fichier orig.tar	46

9.5	La commande <code>cvs-buildpackage</code> et similaires	46
9.6	Vérifier les mises à jour de paquet	46
10	Où demander de l'aide	49
A	Exemples	51
A.1	Exemple d'empaquetage simple	51
A.2	Exemple d'empaquetage avec les paquets <code>dpatch</code> et <code>pbuilder</code>	51

Chapitre 1

Commencer de la bonne manière

Ce document essaie de décrire à l'utilisateur Debian moyen, et au développeur en devenir, la construction d'un paquet Debian. Il utilise un langage assez courant et est complété par des exemples, selon le vieux proverbe romain : *ongum iter est per preaecepta, breve et efficax per exempla* (c'est long par la règle, court et efficace par l'exemple).

Une des choses qui font de Debian une distribution de si haut niveau est son système de paquets. Bien qu'il existe une grande quantité de logiciels au format Debian, vous devrez parfois installer un logiciel qui ne l'est pas. Vous pouvez vous demander comment faire vos propres paquets et peut-être pensez-vous que c'est une tâche très difficile. Eh bien, si vous êtes vraiment un débutant sous Linux, c'est dur, mais si vous étiez un débutant, vous ne seriez pas en train de lire ce document. Vous devez en savoir un peu sur la programmation Unix, mais vous n'avez certainement pas besoin d'être un magicien.

Une chose est certaine, cependant : pour correctement développer et maintenir des paquets Debian, vous aurez besoin de journées/homme. Ne vous faites pas d'illusion, pour que votre système fonctionne, les responsables doivent à la fois être techniquement compétents et rapides.

Ce document va expliquer toutes les étapes les plus petites (et peut-être a priori insignifiantes), vous aider à créer ce premier paquet, et à gagner de l'expérience pour construire les versions suivantes ainsi peut-être que d'autres paquets.

Les nouvelles versions de ce document devraient toujours être disponibles en ligne sur <http://www.debian.org/doc/maint-guide/> et dans le paquet « maint-guide ». La traduction en français est également disponible dans le paquet « maint-guide-fr ».

1.1 Programmes nécessaires au développement

Avant de commencer quoi que ce soit, vous devriez vous assurer que vous avez correctement installé certains paquets supplémentaires nécessaires pour le développement. Notez que la liste ne contient aucun paquet marqué « essentiel » ou « requis » - nous supposons que vous avez déjà installé ceux-ci.

Cette version du document a été mise à jour pour les paquets de Debian 2.2 (*Potato*) et 3.0 (*Woody*).

Les paquets suivants sont fournis dans l'installation standard de Debian 2.1, de sorte que vous les avez probablement déjà (ainsi que les paquets supplémentaires dont ils dépendent). Néanmoins, vous devriez le vérifier avec « `dpkg -s <paquet>` ».

- `dpkg-dev` - ce paquet contient les outils nécessaires pour dépaqueter, construire et télécharger les paquets sources Debian (voir `dpkg-source(1)`).
- `file` - ce programme bien pratique peut déterminer la nature d'un fichier (voir `file(1)`).
- `gcc` - le compilateur C de GNU, nécessaire si votre programme, comme la plupart, est écrit en C (voir `gcc(1)`). Ce paquet va aussi « tirer » plusieurs autres paquets tels que `binutils` qui inclut les programmes utilisés pour assembler et lier des fichiers objets (voir « info binutils » dans le paquet `binutils-doc`) et `cpp`, le préprocesseur C (voir `cpp(1)`).
- `g++` - le compilateur C++ de GNU, nécessaire si votre programme est écrit en C++ (voir `g++(1)`).
- `libc6-dev` - les bibliothèques et fichiers d'en-têtes C dont `gcc` a besoin pour les lier aux fichiers objets créés (voir « info libc » dans le paquet `glibc-doc`).
- `make` - d'ordinaire, la création d'un programme se fait en plusieurs étapes. Plutôt que d'avoir à taper les mêmes commandes encore et encore, vous pouvez utiliser ce programme pour automatiser le processus, en créant des fichiers « Makefile » (voir « info make »).
- `patch` - ce programme très utile prend un fichier contenant une liste de différences (produite par le programme `diff`) et l'applique au fichier original, produisant une version mise à jour (voir `patch(1)`).
- `perl` - Perl est un des langages de script les plus utilisés sur les systèmes modernes similaires à Unix, souvent qualifié de « couteau suisse d'Unix » (voir `perl(1)`).

Vous devrez probablement aussi installer les programmes suivants :

- `autoconf` et `automake` - beaucoup de nouveaux programmes utilisent des scripts de configuration et des fichiers Makefile prétraités à l'aide de programmes comme ceux-ci (voir « info autoconf », « info automake »);
- `dh-make` et `debhelper` - `dh-make` est nécessaire pour créer le squelette de notre exemple de paquet et il utilise certains outils de `debhelper` pour créer les paquets. Ils ne sont pas indispensables pour la création des paquets, mais ils sont **fortement** recommandés pour les nouveaux responsables. Ils rendent le processus complet bien plus facile à démarrer et à contrôler par la suite (voir `dh_make(1)`, `debhelper(1)`, `/usr/share/doc/debhelper/README`);
- `devscripts` - ce paquet contient de jolis scripts utiles qui peuvent aider les responsables, mais ils ne sont pas indispensables pour la création de paquets (voir `/usr/share/doc/devscripts/README.debian.gz`);
- `fakeroot` - cet utilitaire vous laisse prétendre être le superutilisateur, ce qui est nécessaire pour certaines parties du processus de construction (voir `fakeroot(1)`);
- `gnupg` - un outil qui vous permet de *signer* numériquement les paquets. Ceci est spécialement important si vous comptez les distribuer à d'autres personnes, et c'est certainement ce que vous ferez quand votre travail sera inclus dans la distribution Debian (voir `gpg(1)`);
- `g77` - le compilateur FORTRAN de GNU, nécessaire si votre programme est écrit en FORTRAN (voir `g77(1)`);

- `gpc` - le compilateur PASCAL de GNU, nécessaire si votre programme est écrit en Pascal. Digne d'être mentionné ici, `fp-compiler`, le compilateur pascal libre, est également bon pour cette tâche (voir `gpc(1)`, `ppc386(1)`);
- `xutils` - certains programmes, d'ordinaire ceux conçus pour X11, utilisent aussi ces programmes pour générer les fichiers Makefile à partir d'un ensemble de fonctions macros (voir `imake(1)`, `xmkmf(1)`);
- `lintian` - c'est le vérificateur de paquet Debian, qui peut vous indiquer de nombreuses erreurs courantes après la construction de votre paquet et expliquer les erreurs trouvées (voir `lintian(1)`, `diffstat(1)`, `/usr/share/doc/lintian/lintian.html/index.html`);
- `linda` - ceci est le vérificateur de paquet Debian alternatif (voir `linda(1)`);
- `pbuilder` - ce paquet contient des programmes utilisés pour créer et maintenir un environnement chroot. Construire un paquet Debian dans cet environnement permet de vérifier les dépendances correctes de construction et évite les erreurs FTBFS (voir `pbuilder(8)` et `pdebuild(1)`).

Enfin, ces paquets *très importants* doivent être installés et leur contenu lu en parallèle à ce document :

- `debian-policy` - la Charte inclut des explications sur la structure et le contenu de l'archive Debian, plusieurs considérations sur l'architecture du système d'exploitation, la norme de hiérarchie des fichiers (qui dit où chaque fichier ou répertoire doit aller), etc. Le plus important pour vous est qu'il décrit les critères que chaque paquet doit vérifier pour être inclus dans la distribution (voir `/usr/share/doc/debian-policy/policy.html/index.html`);
- `developers-reference` - concerne tout ce qui n'est pas spécifique aux détails techniques de la création de paquets, comme la structure des archives, comment renommer, rendre orphelin, choisir un paquet, faire un NMU (une version du paquet non faite par le responsable), comment gérer les bogues, les meilleures pratiques d'empaquetage, où et quand faire des envois de paquets, etc. (voir `/usr/share/doc/developers-reference/index.en-us.iso-8859-1.html`).

Les courtes descriptions qui sont données ci-dessus ne servent que d'introduction à ce que fait chaque paquet. Avant de continuer, veuillez lire attentivement la documentation de chaque programme, au moins l'usage standard. Cela peut vous sembler fastidieux maintenant, mais plus tard vous serez *très* content de l'avoir fait.

Remarque : `debmake` est un paquet qui contient certains programmes qui fonctionnent d'une manière similaire à `dh-make`, mais son usage spécifique n'est **pas** couvert dans ce document, parce qu'il est obsolète. Veuillez lire le manuel Debmake (<http://www.debian.org/~jaldhar/>) pour plus d'informations.

1.2 Plus d'informations

Vous pouvez faire deux types de paquets : source et binaire. Un paquet source contient le code que vous pouvez compiler en un programme. Un paquet binaire contient juste le programme fini. Ne mélangez pas les termes comme source du programme et le paquet source du programme ! Veuillez lire les autres manuels si vous avez besoin de plus de détails sur la terminologie.

Debian utilise le terme « responsable » pour la personne qui fait des paquets, « auteur » pour la personne qui a créé le programme, et « responsable amont » pour la personne qui maintient le programme actuellement. D'ordinaire, l'auteur et le responsable amont sont une seule et même personne. Si vous avez écrit un programme, et que vous voulez qu'il soit dans Debian, vous pouvez remplir une demande pour devenir un responsable.

Après avoir construit votre paquet (ou pendant la création), vous devrez devenir responsable Debian officiel si vous souhaitez que votre programme soit dans la prochaine distribution (si le programme est utile, pourquoi pas ?) Ce processus est expliqué dans la Référence du développeur. Veuillez la lire.

Chapitre 2

Premiers pas

2.1 Choisir votre programme

Vous avez probablement choisi le paquet que vous voulez créer. La première chose à faire est de vérifier si le paquet se trouve déjà dans la distribution. Si vous utilisez la distribution « stable », le mieux est d'aller sur la page de recherche des paquets (<http://www.debian.org/distrib/packages>). Si vous utilisez une distribution « unstable » **courante**, vérifiez-le avec ces commandes :

```
dpkg -s programme
dpkg -l '*programme*'
```

Si le paquet existe déjà, et bien, installez-le :-). S'il se trouve qu'il est orphelin — si son responsable est « Debian QA Group », vous devriez pouvoir le reprendre. Consultez la liste des paquets orphelins (<http://www.debian.org/devel/wnpp/orphaned>) et la liste des paquets disponibles pour adoption (http://www.debian.org/devel/wnpp/rfa_bypackage) pour vérifier que le paquet est bien disponible.

Si vous pouvez adopter le paquet, récupérez les sources (avec quelque chose comme `apt-get source nom_du_paquet`) et examinez-les. Malheureusement ce document n'inclut pas d'informations exhaustives sur l'adoption de paquets. Cependant vous ne devriez pas avoir de problèmes pour comprendre comment le paquet fonctionne puisque quelqu'un a déjà fait la configuration pour vous. Continuez quand même à lire ce document, une bonne partie des conseils qui suivent seront applicables dans votre cas.

Si le paquet est nouveau, et que vous décidez que vous voulez le voir dans Debian, procédez comme suit :

- vérifiez que personne d'autre ne travaille déjà sur ce paquet en consultant la liste des paquets en cours de création (http://www.debian.org/devel/wnpp/being_packaged). Si quelqu'un travaille déjà dessus, contactez-le si vous pensez que vous le devez. Sinon, trouvez un autre programme intéressant dont personne n'est responsable.

- le programme **doit** avoir une licence, si possible libre conformément aux principes du logiciel libre selon Debian (http://www.debian.org/social_contract.html#guidelines). S'il n'est pas conforme à certaines de ces règles mais peut quand même être distribué, il peut malgré tout être inclus dans les sections « contrib » ou « non-free » de Debian. Si vous ne savez pas trop où il doit aller, postez la licence sur <debian-legal@lists.debian.org> et demandez conseil.
- le programme ne devrait certainement **pas** être setuid root, ou encore mieux, il ne devrait pas être setuid ou setgid quoi que ce soit.
- le programme ne devrait pas être un démon, ou quelque chose qui va dans les répertoires */sbin, ou ouvre un port comme root.
- le programme devrait être sous forme de binaire exécutable, les bibliothèques sont plus dures à gérer.
- il devrait être bien documenté, et le code doit être compréhensible (c'est-à-dire, pas volontairement obscur).
- vous devriez contacter le(s) auteur(s) du programme pour vérifier qu'il(s) est(sont) d'accord pour la création du paquet. Il est important d'être à même de consulter le(s) auteur(s) à propos du programme en cas de problèmes spécifiques à celui-ci, aussi n'essayez pas de créer un paquet à partir de programmes non maintenus.
- enfin, vous devez être sûr qu'il fonctionne, et l'avoir testé pendant quelques temps.

Bien sûr, toutes ces remarques ne sont que des mesures de sécurité, et ont pour but de vous sauver d'utilisateurs fous de rage si vous faites une erreur dans un démon setuid. Quand vous aurez plus d'expérience dans la création des paquets, vous serez capable de faire de tels paquets, mais même les développeurs les plus expérimentés consultent la liste de discussion debian-mentors en cas de doute. Et là les gens seront heureux de vous aider.

Pour plus d'informations à ce sujet, consultez la Référence du Développeur.

2.2 Obtenir le programme, et l'essayer

La première chose à faire est de trouver et de télécharger le paquet original. Je suppose que vous avez déjà le fichier source que vous avez pris sur la page web de l'auteur. Les sources pour les logiciels Unix libres sont d'habitude au format tar/gzip avec l'extension .tar.gz, et contiennent normalement un sous-répertoire nommé programme-version avec toutes les sources dedans. Si le source de votre programme est disponible dans une autre sorte d'archive (par exemple, le programme se termine par '.Z' ou '.zip'), décompressez-le avec les outils adéquats ou demandez sur la liste de discussion debian-mentors si vous n'êtes pas sûr quant à la façon de le décompresser correctement (indice : utilisez « file archive.extension »).

Comme exemple, je vais utiliser un programme nommé « gentoo », un gestionnaire de fichiers pour X utilisant GTK+. Sachez qu'il y a déjà un paquet pour ce programme, et qu'il a changé substantiellement depuis que ce texte a été écrit la première fois.

Créez un sous-répertoire sous votre répertoire racine nommé « debian » ou « deb » ou quoi que ce soit d'adéquat (ou le nom du programme, ~/gentoo, ferait l'affaire dans notre cas). Placez l'archive téléchargée dedans, et décompressez-la avec « tar -xzf gentoo-0.9.12.tar.gz ».

Assurez-vous qu'il n'y a pas d'erreurs, même « sans importance », parce qu'alors il y aura des problèmes pour décompresser sur les systèmes d'autres personnes, dont les outils de décompression pourraient ne pas gérer ces erreurs.

Maintenant vous avez un autre sous-répertoire, nommé « gentoo-0.9.12 ». Allez dans ce répertoire et lisez **attentivement** la documentation fournie. Il s'agit d'habitude de fichiers nommés README*, INSTALL*, *.lsm ou *.html. Dedans, vous devez trouver les instructions pour compiler et installer correctement le programme (elles supposent très probablement que vous voulez installer dans le répertoire /usr/local/bin ; ce n'est pas le cas, mais on reviendra sur ce point plus tard dans 'Installer dans un sous-répertoire' page 9).

La méthode varie d'un programme à l'autre, mais de nombreux programmes modernes viennent avec un script « configure » qui configure les sources selon votre système et s'assure que votre système est à même de les compiler. Après la configuration avec « ./configure », les programmes sont compilés avec « make ». Certains d'entre eux supportent « make check », pour se tester eux-mêmes. L'installation dans les répertoires de destination est généralement obtenue avec « make install ».

Maintenant, essayez de compiler et d'exécuter votre programme, pour vous assurer qu'il fonctionne correctement et ne casse rien d'autre quand il est installé ou qu'il tourne.

Sachez aussi que vous pouvez généralement entrer « make clean » (ou mieux, « make distclean ») pour nettoyer le répertoire de compilation. Parfois, il y a même un « make uninstall » qui peut être utilisé pour retirer tous les fichiers installés.

2.3 Les noms et versions des paquets

Vous devriez commencer la création du paquet avec un répertoire source complètement propre (originel), ou plus simplement avec les sources fraîchement décompressées.

Pour que le paquet soit correctement construit, vous devez changer le nom du programme en minuscule (si ce n'est déjà fait), et vous devriez changer le répertoire source en <nompaquet>-<version>.

Si le nom du programme consiste en plus d'un mot, réduisez-le à un mot, ou faites une abréviation. Par exemple, le paquet du programme « John's little editor for X » serait nommé johnledx, ou jle4x, ou quoi que vous vouliez, aussi longtemps qu'il reste sous une limite raisonnable, en général 20 caractères.

Vérifiez aussi la version exacte du programme (qui sera inclus dans la version du paquet). Si ce logiciel n'est pas numéroté avec un numéro de version comme X.Y.Z, mais avec une date de distribution, vous pouvez utiliser cette date comme numéro de version, avec comme préfixe « 0.0. » (juste au cas où les responsables amont décident de distribuer une jolie version comme 1.0). Donc, si la date est le 19 décembre 1998, vous pouvez utiliser 0.0.19981219 comme chaîne pour la version.

Certains ne seront pas numérotés du tout, auquel cas vous devriez contacter le responsable amont pour voir s'il a une autre méthode de gestion des révisions.

2.4 « Debianisation » initiale

Vérifiez que vous êtes dans le répertoire du code source du programme, et lancez ceci :

```
dh_make -e votre.adresse@de.responsable -f ../gentoo-0.9.12.tar.gz
```

Bien sûr, remplacez la chaîne « votre.adresse@de.responsable » avec votre adresse électronique pour l'inclure dans l'entrée changelog et dans d'autres fichiers, et le nom du fichier par le nom de la source d'archive originale. Voyez `dh_make(1)` pour plus de détails.

Des informations sont affichées. Il vous demande quelle sorte de paquet vous voulez créer. Gentoo est un paquet binaire simple — il ne crée qu'un exécutable, et donc un seul fichier `.deb` — donc nous sélectionnons la première option, avec la touche « s », vérifions l'information sur l'écran et confirmons en pressant <Entrée>.

Après cette exécution de `dh_make`, une copie de l'archive TAR est créée sous `gentoo_0.9.12.orig.tar.gz` dans le répertoire parent pour permettre la création d'un paquet source Debian non natif avec le `diff.gz`. Veuillez noter deux aspects clés dans ce nom de fichier :

- le nom de paquet et la version sont séparés par « _ »
- il y a « orig. » avant le « tar.gz ».

Rappelons-le, en tant que nouveau responsable, vous ne devriez pas créer des paquets compliqués, par exemple :

- des paquets à exécutables multiples ;
- des paquets de bibliothèque ;
- des formats de fichier source n'étant ni en `tar.gz` ni `tar.bz2` ;
- une archive TAR source ayant des contenus non distribuables.

Ce n'est pas si difficile, mais cela requiert un peu plus de connaissances, et nous n'entrerons pas dans les détails ici.

Notez que vous ne pouvez exécuter `dh_make` **qu'une fois**, et qu'il ne se comportera pas correctement si vous l'exécutez à nouveau dans le même répertoire déjà debianisé. Cela signifie aussi que vous devrez utiliser une autre méthode pour distribuer une nouvelle révision ou une nouvelle version de votre paquet dans le futur. Pour plus d'informations à ce sujet, lisez 'Mettre à jour le paquet' page 43.

Chapitre 3

Modifier les sources

Normalement, les programmes s'installent d'eux-mêmes dans les sous-répertoires `/usr/local`. Mais les paquets Debian ne doivent pas utiliser ce répertoire, car il est réservé à l'usage privé de l'administrateur système (ou de l'utilisateur). Cela signifie que vous devez examiner le système de création de votre programme, en général en commençant par le Makefile. C'est le script que `make` (1) utilisera pour automatiser la création du programme. Pour plus de détails sur les Makefile, regardez 'fichier « rules »' page 20.

Notez que si votre programme utilise GNU `automake` (1) et/ou `autoconf` (1), ce qui signifie que les sources incluent des fichiers Makefile.am et/ou Makefile.in, respectivement, vous devrez modifier ces fichiers, parce que chaque appel d'`automake` va écraser les Makefile.in avec des informations générées à partir des Makefile.am, et que chaque appel de `./configure` fera de même avec les fichiers Makefile, avec des données des Makefile.in. Éditer les fichiers Makefile.am requiert des connaissances sur `automake`, que vous pouvez apprendre dans la section info d'`automake`, alors qu'éditer les fichiers Makefile.in est globalement identique à l'édition de fichiers Makefile, si ce n'est qu'il faut faire attention à toutes les variables, à savoir toute chaîne entourée de « @ », comme par exemple `@CFLAGS@` ou `@LN_S@`, qui sont remplacées par des valeurs réelles à chaque appel de `./configure`. Veuillez lire `/usr/share/doc/autotools-dev/README.Debian.gz` avant de commencer.

Notez qu'il n'y a pas la place ici pour entrer dans *tous* les détails sur les modifications, mais voici quelques-uns des problèmes qui reviennent souvent.

3.1 Installer dans un sous-répertoire

La plupart des programmes ont une certaine manière de s'installer dans la structure de répertoires de votre système, de sorte que leurs exécutables sont inclus dans votre `$PATH`, et que vous trouvez leurs documentation et pages de manuel aux places habituelles. Cependant, si vous faisiez cela, le programme serait installé au milieu de tout ce qui est déjà sur votre système. Les outils de création de paquet auraient plus de difficultés pour décider ce qui appartient ou non à votre paquet.

Dès lors, vous devez faire autre chose : installer le programme dans un sous-répertoire temporaire à partir duquel les outils du responsable vont construire un paquet `.deb` fonctionnel. Tout ce qui est contenu dans ce répertoire sera installé sur le système de l'utilisateur quand il installe votre paquet, la seule différence est que `dpkg` installera les fichiers dans le répertoire racine.

Ce répertoire temporaire est d'ordinaire créé sous votre répertoire `debian/` dans l'arbre des sources décompressées. Il est normalement nommé `debian/nom_du_paquet`.

Gardez à l'esprit que bien que vous deviez faire en sorte que le programme s'installe sous `debian/nom_du_paquet`, il doit continuer à s'exécuter correctement quand il est installé sous le répertoire racine, c'est-à-dire quand il est installé à partir du paquet `.deb`. Aussi vous ne devez pas laisser le système de création coder en dur dans les fichiers du paquet des chaînes de caractères comme `/home/me/deb/gentoo-0.9.12/usr/share/gentoo`.

Avec des programmes utilisant GNU autoconf, cela est relativement facile. La plupart de ces programmes ont des fichiers `makefile` et sont par défaut configurés de manière à permettre l'installation dans un répertoire quelconque, en gardant à l'esprit que `/usr` (par exemple) est le préfixe standard. Quand il détecte que votre programme utilise autoconf, `dh_make` va mettre en place des commandes pour le faire automatiquement, et vous pouvez tout aussi bien passer à la section suivante. Mais avec d'autres programmes, vous devrez plus que probablement examiner et éditer les fichiers `Makefile`.

Voici les parties concernées du `Makefile` de `gentoo` :

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

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

Nous voyons que les fichiers sont configurés pour s'installer sous `/usr/local/`. Changez ces chemins en :

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

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

Mais pourquoi dans ce répertoire, et pas dans un autre ? Parce que Debian n'installe jamais de fichiers sous `/usr/local` — cet arbre est réservé à l'usage de l'administrateur système. Sur un système Debian, de tels fichiers doivent plutôt aller sous `/usr`.

Les positions exactes des exécutables, icônes, documentation, etc., sont spécifiées dans le standard de la hiérarchie de fichiers (voir `/usr/share/doc/debian/debian-policy/fhs/`). Je vous recommande de le consulter et de lire les sections relatives à votre paquet.

Dès lors, nous devrions installer l'exécutable sous `/usr/bin` plutôt que `/usr/local/bin`, la page de manuel sous `/usr/share/man/man1` plutôt que sous `/usr/local/man/man1`, etc. Notez qu'il n'y a pas de page de manuel mentionnée dans le fichier `Makefile`, mais comme la Charte Debian requiert que chaque programme en ait une, nous en créerons une plus tard et l'installerons sous `/usr/share/man/man1`.

Certains programmes n'utilisent pas les variables des fichiers `Makefile` pour définir des chemins comme ceux-ci. Cela signifie que vous pouvez avoir à éditer des fichiers sources C réels pour les faire utiliser les positions correctes. Mais où et que chercher ? Vous pouvez trouver où en lançant :

```
grep -rn usr/local/lib *.[ch]
```

`Grep` va récursivement parcourir l'arbre des sources et vous dire le nom des fichiers et le numéro des lignes où il trouve une occurrence.

Éditez ces fichiers et à ces lignes, remplacez `/usr/local/*` par `/usr/*` — et c'est à peu près tout. Soyez attentif à ne pas casser le reste du code !

Après quoi, vous devriez trouver la cible d'installation (cherchez une ligne qui commence avec « `install :` », d'ordinaire cela fonctionne) et renommez toutes les références aux répertoires autres que ceux définis au début du `Makefile`. Auparavant, la cible d'installation de `gentoo` disait :

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

Après votre modification elle dit :

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

Vous aurez sûrement remarqué qu'il y a maintenant une commande `install -d` avant les autres commandes dans la règle. Le fichier `Makefile` originel ne l'avait pas parce qu'habituellement `/usr/local/bin` et les autres répertoires existent déjà sur le système dans lequel « `make install` » est exécuté. Cependant, comme nous installerons dans nos propres répertoires vides (ou même non existants), nous aurons à créer chacun de ces répertoires.

Nous pouvons ajouter d'autres choses à la fin de la règle, comme l'installation de la documentation additionnelle que l'auteur amont oublie parfois :

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

Le lecteur attentif notera que j'ai changé « gentoo » en « gentoo-target » dans la ligne « install : ». C'est ce qu'on appelle une correction de bogue sans rapport :-).

Chaque fois que vous faites des modifications qui ne sont pas spécifiquement liées à Debian, envoyez-les au responsable amont pour qu'elles puissent être incluses dans la version suivante du programme et que tout le monde puisse en bénéficier. Souvenez-vous aussi de ne pas rendre vos corrections spécifiques à Debian ou Linux (ou même Unix !) avant de les envoyer — faites-les portables. Cela rendra vos corrections beaucoup plus faciles à appliquer.

Remarquez que vous ne devez pas envoyer les fichiers `debian/*` en amont.

3.2 Bibliothèques différentes

Il y a souvent un problème courant : des bibliothèques sont souvent différentes d'une plateforme à l'autre. Par exemple, `Makefile` peut contenir une référence à une bibliothèque qui n'existe pas sur les systèmes Debian. Dans ce cas, nous devons la changer en une bibliothèque qui existe dans Debian, et qui sert à la même chose.

Ainsi, s'il y a une ligne dans le `Makefile` (ou `Makefile.in`) de votre programme qui dit quelque chose comme ceci (et votre programme ne compile pas) :

```
LIBS = -lcurses -lquelquechose -lautrechose
```

Changez-la en ceci, et cela marchera probablement :

```
LIBS = -lncurses -lquelquechose -lautrechose
```

(L'auteur réalise que ceci n'est pas le meilleur exemple dans la mesure où notre paquet `libncurses` est maintenant livré avec un lien symbolique `libcurses.o`, mais il n'en a pas trouvé de meilleur. Les suggestions sont les bienvenues).

Chapitre 4

Ce qui est requis sous debian/

Il y a un nouveau sous-répertoire sous le répertoire des sources du programme (« gentoo-0.9.12 »), nommé « debian ». Il y a un certain nombre de fichiers dans ce répertoire que vous devriez éditer pour configurer le comportement du paquet. Les plus importants d'entre eux sont « control », « changelog », « copyright » et « rules », qui sont requis pour tous les paquets.

4.1 Fichier « control »

Ce fichier contient plusieurs valeurs que `dpkg`, `dselect` et d'autres outils de gestions de paquets vont utiliser pour gérer le paquet.

Voici le fichier « control » que `dh_make` crée pour nous.

```
1 Source: gentoo
2 Section: unknown
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (> 3.0.0)
6 Standards-Version: 3.6.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Description: <insert up to 60 chars description>
12 <insert long description, indented with spaces>
```

(J'ai ajouté les numéros de ligne.)

Les lignes 1 à 6 sont les informations de contrôle pour le paquet source.

La ligne 1 est le nom du paquet source.

La ligne 2 est la section de la distribution dans laquelle ce paquet va.

Comme vous l'avez constaté, Debian est divisée en sections : main (logiciels libres), non-free (logiciels non libres), et contrib (logiciels libres qui dépendent de logiciels non libres). Sous celles-ci, il y a des sous-sections logiques qui décrivent de manière concise les paquets qui s'y trouvent. Ainsi nous avons « admin » pour les programmes réservés à l'administrateur, « base » pour les outils de base, « devel » pour les outils de programmation, « doc » pour la documentation, « libs » pour les bibliothèques « mail » pour les lecteurs et les démons de courriel, « net » pour les applications et démons réseaux, « x11 » pour les programmes X11 qui ne sont pas plus appropriés ailleurs, et bien d'autres.

Changeons donc la section en x11 (le préfixe « main/ » est implicite, donc nous pouvons l'omettre).

La ligne 3 décrit l'importance pour l'utilisateur d'installer ce paquet. Lisez la Charte Debian pour des informations sur ces valeurs. La priorité « optional » marche habituellement pour les nouveaux paquets.

Les sections et les priorités sont utilisées par des interfaces comme `dselect` quand elles trient les paquets et sélectionnent les valeurs par défaut. Quand vous enverrez votre paquet dans Debian, les valeurs de ces deux champs peuvent être modifiées par les responsables des archives, auquel cas vous serez notifié par courriel.

Comme c'est un paquet de priorité normale et qu'il n'entre pas en conflit avec quoi que ce soit, nous le laissons à « optional ».

La ligne 4 est le nom et l'adresse électronique du responsable. Assurez-vous que ce champ contient un en-tête « To : » valide pour un courrier électronique, car après le téléchargement, le système de suivi des bogues l'utilisera pour vous délivrer les courriels de bogues. Évitez d'utiliser des virgules, esperluètes (« & ») ou parenthèses.

La ligne 5 contient la liste des paquets nécessaires pour construire le paquet. Certains paquets comme `gcc` ou `make` sont implicites, voyez le paquet `build-essential` pour les détails. Si un compilateur non standard ou un autre outil est nécessaire pour construire le paquet, vous devez l'ajouter dans la ligne « Build-Depends ». Les différentes entrées sont séparées par des virgules ; lisez ci-dessous les explications sur les dépendances entre binaires pour mieux comprendre la syntaxe de ce champ.

Vous pouvez avoir aussi ici des champs `Build-Depends-Indep`, `Build-Conflicts` et d'autres champs. Ces données seront utilisées par le logiciel de construction de paquets automatique Debian pour créer les paquets binaires pour d'autres plates-formes d'ordinateurs. Lisez la Charte Debian pour plus d'informations sur les dépendances de construction et la Référence du Développeur pour plus d'informations sur ces autres plates-formes (architectures) et comment porter des logiciels vers celles-ci.

Voici une bidouille que vous pouvez utiliser pour découvrir les paquets dont le vôtre a besoin pour être construit :

```
strace -f -o /tmp/log ./configure
# ou make à la place de ./configure, si votre paquet n'utilise pas autoconf
```

```

for x in `dpkg -S $(grep open /tmp/log|\
                perl -pe 's!.* open\(\\"([^\"]*)\.*!$1!' |\
                grep "^/" | sort | uniq|\
                grep -v "^\( /tmp\| /dev\| /proc\)\" ) 2>/dev/null|\
                cut -f1 -d":" | sort | uniq`; \
do \
    echo -n "$x (>=" `dpkg -s $x|grep ^Version|cut -f2 -d":"` " ), "; \
done

```

Pour déterminer manuellement la dépendance de construction exacte pour `/usr/bin/foo`, vous exécuterez

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

et pour chaque bibliothèque listée, par exemple `libfoo.so.6`, exécutez

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

Ensuite vous prenez simplement la version `-dev` de chaque paquet comme entrée « Build-deps ». Si vous utilisez `ldd` à cet effet, il va rapporter les dépendances de bibliothèque indirectes, ayant pour résultat un problème de dépendances de constructions excessives.

Il se trouve que Gentoo a aussi besoin de `xlibs-dev`, `libgtk1.2-dev` et `libgl1.2-dev` pour être construit, aussi nous les ajouterons ici à côté de `debhelper`.

La ligne 6 est la version de la Charte Debian que ce paquet respecte, la version de la Charte Debian que vous lisez quand vous créez votre paquet.

La ligne 8 est le nom du paquet binaire. C'est d'ordinaire le même que le nom du paquet source, mais ce n'est pas nécessairement le cas.

La ligne 9 décrit l'architecture CPU pour laquelle le paquet binaire peut être compilé. Nous le laissons à « any » car `dpkg-gencontrol(1)` trouvera la valeur appropriée pour toute machine sur laquelle ce paquet sera compilé.

Si votre paquet est indépendant d'une architecture (par exemple, un script shell ou Perl, ou un document), changez cette entrée en « all », et lisez plus loin dans 'fichier « rules »' page 20 comment utiliser la règle « binary-indep » au lieu de « binary-arch » pour construire le paquet.

La ligne 10 montre une des caractéristiques les plus puissantes du système de paquet Debian. Les paquets peuvent être liés entre eux de plusieurs façons. Hormis `Depends :`, les autres champs décrivant ces relations sont `Recommends :`, `Suggests :`, `Pre-Depends :`, `Conflicts :`, `Provides :`, et `Replaces :`.

Les outils de gestion de paquets se comportent d'ordinaire de la même manière quand ils gèrent ces relations; sinon, ce sera expliqué (voir `dpkg(8)`, `dselect(8)`, `apt(8)`, `aptitude(1)`, etc.).

Voici ce que les dépendances veulent dire :

- Depends :
Le paquet ne sera pas installé à moins que les paquets dont il dépend ne soient installés. Utilisez-le si votre programme ne s'exécute absolument pas (ou cause des dégâts sérieux) tant qu'un paquet particulier n'est pas présent ;
- Recommends :
Des interfaces comme dselect ou aptitude vous demanderont d'installer les paquets recommandés en même temps que votre paquet ; dselect insistera même. dpkg et apt-get ignorent ce champ, cependant. Utilisez-le pour les paquets qui ne sont pas vraiment indispensables mais qui sont typiquement utilisés avec votre programme ;
- Suggests :
Quand un utilisateur installe votre programme, toute interface lui demandera probablement s'il faut installer les programmes qu'il suggère. dpkg et apt-get ne s'en soucient pas. Utilisez-le pour les paquets qui marchent bien avec votre programme mais qui ne sont pas nécessaires ;
- Pre-Depends :
Ceci est plus fort que Depends. Le paquet ne sera pas installé à moins que les paquets dont il pré-dépend ne soient installés *et correctement configurés*. Utilisez-le **très** rarement et seulement après en avoir discuté sur la liste de discussion debian-devel. Traduction : ne l'utilisez pas du tout ; :-)
- Conflicts :
Le paquet ne sera pas installé avant que les paquets avec lesquels il est en conflit n'aient été retirés. Utilisez ceci si votre programme ne peut absolument pas fonctionner ou s'il cause d'énormes problèmes quand un paquet particulier est présent ;
- Provides :
Quand il y a plusieurs alternatives pour certains types de paquets, des noms virtuels ont été définis. Vous pouvez trouver la liste complète dans /usr/share/doc/debian-policy/virtual-package-name-list.text.gz. Utilisez ceci si votre programme fournit une fonction d'un paquet virtuel existant ;
- Replaces :
Utilisez ceci quand votre programme remplace des fichiers d'un autre paquet, ou remplace complètement un autre paquet (utilisé en conjonction avec Conflicts :). Les fichiers du paquet nommé seront écrasés par les fichiers de votre paquet.

Tous ces champs ont une syntaxe uniforme. Il s'agit d'une liste de paquets séparés par des virgules. Ces noms de paquets peuvent aussi être une liste d'alternatives, séparés par des symboles barre verticale | (symbole tube).

Le domaine d'application des champs peut être restreint à des versions particulières de chaque paquet nommé. Ces versions sont listées entre parenthèses après chaque nom de paquet individuel, et doivent contenir une relation de la liste suivante suivie par un numéro de version. Les relations autorisées sont <<, <=, =, >= et >> pour strictement plus petit, plus petit ou égal, exactement égal, plus grand ou égal et strictement plus grand, respectivement. Par exemple,

```
Depends: foo (>= 1.2), libbar1 (= 1.3.4)
Conflicts: baz
Recommends: libbaz4 (>> 4.0.7)
Suggests: quux
```

```
Replaces: quux (<< 5), quux-foo (<= 7.6)
```

La dernière caractéristique que vous devez connaître est `${shlibs :Depends}`. Lorsque votre paquet aura été construit et installé dans le répertoire temporaire, `dh_shlibdeps(1)` le scannera pour les exécutables et bibliothèques, déterminera leurs dépendances en bibliothèques partagées et détectera dans quels paquets elles se trouvent. Il passera la liste à `dh_gencontrol(1)` qui l'insérera à la bonne place, et vous ne devrez pas vous en soucier.

Ceci étant dit, nous pouvons laisser la ligne `Depends` : exactement comme elle est maintenant, et insérer après une ligne disant `Suggests: file`, car `gentoo` peut utiliser certaines fonctionnalités fournies par ce programme/paquet.

La ligne 11 est la description courte. L'écran de la plupart des gens est large de 80 colonnes, aussi cela ne devrait pas dépasser les 60 caractères. Je le change en « fully GUI configurable X file manager using GTK+ ».

À la ligne 12 commence la description longue. Celle-ci devrait être un paragraphe qui donne plus de détails sur le paquet. La colonne 1 de chaque ligne doit être vide. Il ne peut y avoir de ligne vide, mais vous pouvez mettre un seul « . » (point) dans la colonne 2 pour simuler une ligne vide. De plus, il ne peut pas y avoir plus d'une ligne vide après la description longue.

Finalement, voici le fichier `control` mis à jour :

```
1 Source: gentoo
2 Section: x11
3 Priority: optional
4 Maintainer: Josip Rodin <joy-mg@debian.org>
5 Build-Depends: debhelper (>> 3.0.0), xlibs-dev, libgtk1.2-dev, libglib1.
6 Standards-Version: 3.5.2
7
8 Package: gentoo
9 Architecture: any
10 Depends: ${shlibs:Depends}
11 Suggests: file
12 Description: fully GUI configurable GTK+ file manager
13 gentoo is a file manager for Linux written from scratch in pure C. It
14 uses the GTK+ toolkit for all of its interface needs. gentoo provides
15 100% GUI configurability; no need to edit config files by hand and re-
16 start the program. gentoo supports identifying the type of various
17 files (using extension, regular expressions, or the 'file' command),
18 and can display files of different types with different colors and icon
19 .
20 gentoo borrows some of its look and feel from the classic Amiga file
21 manager "Directory OPUS" (written by Jonathan Potter).
```

(J'ai ajouté les numéros de ligne.)

4.2 fichier « copyright »

Ce fichier contient les informations sur les ressources en amont, le copyright et la licence du paquet. Le format n'est pas dicté par la Charte Debian, mais son contenu l'est (voir section 13.6 « Copyright Information »).

dh_make en crée un par défaut, qui ressemble à ceci :

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

(J'ai ajouté les numéros de ligne.)

Les choses importantes à ajouter à ce fichier sont l'endroit où vous avez trouvé ce paquet, ainsi que le copyright et la licence d'exploitation réelle (incluez-la en entier). Si la licence est une des licences de logiciel libre populaires comme GNU GPL ou LGPL, BSD ou Artistic, vous pouvez juste faire référence au fichier approprié dans le répertoire /usr/share/common-licenses/, qui existe sur chaque système Debian.

En bref, voici ce à quoi le fichier copyright de gentoo devrait ressembler :

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

(J'ai ajouté les numéros de ligne.)

4.3 changelog

C'est un fichier requis, qui a un format spécial décrit dans la Charte section 4.4 « debian/changelog ». Ce format est utilisé par dpkg et d'autres programmes pour obtenir le numéro de version, de révision, de distribution et l'urgence de votre paquet.

Pour vous, il est aussi important, puisqu'il est bon de documenter toutes les modifications que vous avez faites. Cela aidera les gens qui téléchargent votre paquet à voir s'il y a des problèmes non résolus à propos desquels ils doivent être immédiatement mis au courant. Il sera sauvé sous « /usr/share/doc/gentoo/changelog.Debian.gz » dans le paquet binaire.

dh_make en crée un par défaut, et il ressemble à ceci :

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

(J'ai ajouté les numéros de ligne.)

La ligne 1 est le nom du paquet, la version, la distribution et l'urgence. Le nom doit correspondre au nom du paquet source, la distribution devrait être « unstable » (ou même « experimental », et l'urgence ne devrait pas être changée en quoique ce soit de plus haut que « low ». :-)

Les lignes 3 à 5 sont l'entrée d'audit, où vous documentez les modifications faites dans la révision du paquet (pas les modifications amont – il y a un fichier spécial pour cela, créé par les auteurs en amont, que vous installerez comme /usr/share/doc/gentoo/changelog.gz). Les nouvelles lignes doivent être ajoutées juste avant la première ligne qui commence avec une astérisque (« * »). Vous pouvez le faire avec dch(1), emacs(1), ou manuellement avec un éditeur de texte.

Vous obtiendrez quelque chose comme :

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

(J'ai ajouté les numéros de ligne.)

Vous pouvez en apprendre plus sur la mise à jour du fichier changelog plus loin dans 'Mettre à jour le paquet' page [43](#).

4.4 fichier « rules »

Maintenant nous devons examiner les règles que `dpkg-buildpackage(1)` va utiliser pour créer vraiment le paquet. Ce fichier est en fait un autre Makefile, mais différent de celui/ceux des sources amont. Contrairement aux autres fichiers sous `debian/`, celui-ci est marqué comme exécutable.

Chaque fichier « rules », comme tout autre Makefile, consiste en plusieurs règles indiquant comment manipuler les sources. Les règles sont des cibles, noms de fichiers ou d'actions à exécuter (par exemple, « build : » ou « install : »). Les règles que vous voulez exécuter doivent être données comme paramètre à la ligne de commande (par exemple, « rules build » ou « rules install »). Après le nom de la cible, vous pouvez nommer les dépendances, le programme ou le fichier dont la cible dépend. Après cela il peut y avoir un nombre quelconque de commandes indentées par <tab>, jusqu'à ce qu'une ligne vide soit trouvée. Une nouvelle règle commence avec une déclaration de cible dans la première colonne. Les lignes vides ainsi que celles qui commencent par un « # » (dièse) sont considérées comme des commentaires et sont ignorées.

Tout ceci vous semble probablement confus pour l'instant, mais cela va devenir clair à l'examen du fichier « rules » que `dh_make` nous donne par défaut. Vous devriez avoir lu l'entrée « make » dans `info` pour plus d'information.

Ce qu'il faut savoir à propos du fichier `rules` créé par `dh_make`, est qu'il s'agit juste d'une suggestion. Il fonctionnera pour des paquets simples, mais pour ceux qui sont plus compliqués, vous ne devez pas craindre de le modifier pour le faire correspondre à vos besoins. Les seules choses que vous ne pouvez pas changer sont les noms des règles, car tous les outils utilisent ces noms comme requis par la Charte.

Voici (approximativement) ce à quoi ressemble le fichier par défaut `debian/rules` généré pour nous par `dh_make` :

```
1  #!/usr/bin/make -f
2  # Sample debian/rules that uses debhelper.
3  # GNU copyright 1997 to 1999 by Joey Hess.
4
5  # Uncomment this to turn on verbose mode.
6  #export DH_VERBOSE=1
7
8  # This is the debhelper compatibility version to use.
9  export DH_COMPAT=4
10
11 CFLAGS = -g
12 ifneq (, $(findstring noopt, $(DEB_BUILD_OPTIONS)))
13 CFLAGS += -O0
14 else
15 CFLAGS += -O2
16 endif
17
```

```
18 build: build-stamp
19 build-stamp:
20     dh_testdir
21
22     # Add here commands to compile the package.
23     $(MAKE)
24     #docbook-to-man debian/gentoo.sgml > gentoo.1
25
26     touch build-stamp
27
28 clean:
29     dh_testdir
30     dh_testroot
31     rm -f build-stamp
32
33     # Add here commands to clean up after the build process.
34     -$(MAKE) clean
35
36     dh_clean
37
38 install: build
39     dh_testdir
40     dh_testroot
41     dh_clean -k
42     dh_installdirs
43
44     # Add here commands to install the package into debian/gentoo.
45     $(MAKE) install DESTDIR=$(CURDIR)/debian/gentoo
46
47 # Build architecture-independent files here.
48 binary-indep: build install
49 # We have nothing to do by default.
50
51 # Build architecture-dependent files here.
52 binary-arch: build install
53     dh_testdir
54     dh_testroot
55 #     dh_installdebconf
56     dh_installdocs
57     dh_installexamples
58     dh_installmenu
59 #     dh_installogrotate
60 #     dh_installemacsen
61 #     dh_installpam
62 #     dh_installmime
63 #     dh_installinit
```

```
64         dh_installcron
65         dh_installman
66         dh_installinfo
67 #        dh_undocumented
68         dh_installchangelogs ChangeLog
69         dh_link
70         dh_strip
71         dh_compress
72         dh_fixperms
73 #        dh_makeshlibs
74         dh_installdeb
75 #        dh_perl
76         dh_shlibdeps
77         dh_gencontrol
78         dh_md5sums
79         dh_builddeb
80
81 binary: binary-indep binary-arch
82 .PHONY: build clean binary-indep binary-arch binary install
```

(J'ai ajouté les numéros de ligne. Dans le `debian/rules` réel, les espaces en début de ligne sont des codes TAB).

Vous avez probablement l'habitude de la ligne 1 avec les scripts shell et perl. Cela signifie que ce fichier doit être exécuté par `/usr/bin/make`.

La signification des variables `DH_*` mentionnées des lignes 6 à 9 devrait être évidente à partir du commentaire. Pour plus d'informations sur `DH_COMPAT`, lisez la section « Debhelper compatibility levels » de la page de manuel `debhelper(1)`.

Les lignes 11 à 16 sont un squelette de support pour les paramètres `DEB_BUILD_OPTIONS`, décrits dans la Charte section 11.1 « Binaries ». Fondamentalement, ces choses déterminent si l'exécutable doit être construit avec les symboles de débogage, et s'ils doivent être retirés à l'installation. Une fois encore, il s'agit juste d'un squelette, une indication que vous devriez le faire. Vous devriez vérifier comment le système de construction amont gère l'inclusion des symboles de débogage, et comment il les retire à l'installation, et implémenter cela vous-même.

D'habitude, vous pouvez dire à `gcc` de compiler avec « `-g` » en utilisant la variable `CFLAGS` — si c'est le cas pour votre paquet, propagez la variable en *ajoutant* `CFLAGS="$ (CFLAGS) "` à l'invocation de `$(MAKE)` dans la règle de construction (voir plus bas). Alternativement, si votre paquet utilise un script de configuration `autoconf`, vous pouvez la lui passer en *préfixant* la chaîne ci-dessus à l'appel de `./configure` dans la règle de construction.

Pour ce qui est de retirer les symboles, les programmes sont configurés couramment pour s'installer avec, et souvent sans option pour changer cela. Heureusement, vous avez toujours `dh_strip(1)` qui détecte quand le drapeau `DEB_BUILD_OPTIONS=nostrip` est mis, et qui quitte silencieusement.

Les lignes 18 à 26 décrivent la règle « `build` » (et son enfant « `build-stamp` »), qui exécute

make avec le fichier Makefile de l'application pour compiler le programme. Si votre paquet utilise les utilitaires GNU configure pour construire les exécutable, soyez absolument certain d'avoir lu `/usr/share/doc/autotools-dev/README.Debian.gz`. Nous en dirons plus sur l'exemple commenté `docbook-to-man` plus loin dans 'manpage.1.ex, manpage.sgml.ex' page 28.

La règle « clean », spécifiée aux lignes 28 à 36, efface tous les binaires inutiles et les trucs générés automatiquement, laissés là par une construction du paquet. Cette règle doit être opérationnelle tout le temps (même si les répertoires sources *sont* nettoyés!), donc vous devriez utiliser les options pour forcer (par exemple pour `rm`, c'est « -f ») ou pour ignorer la valeur de retour (les échecs), avec un « - » devant le nom de la commande.

Le processus d'installation, la règle « install », commence à la ligne 38. Fondamentalement, elle exécute la règle `install` du fichier Makefile du programme, mais installe dans le répertoire `$(CURDIR)/debian/gentoo` – c'est pour cette raison que nous avons spécifié `$(DESTDIR)` comme racine de l'installation dans le Makefile de `gentoo`.

Comme le commentaire le laisse penser, la règle « binary-indep », sur la ligne 48, est utilisée pour construire des paquets indépendants de l'architecture. Comme il n'y en a pas dans cet exemple, rien n'est fait.

Ensuite on trouve la règle « binary-arch », des lignes 52 à 79, pour laquelle nous exécutons plusieurs petits utilitaires du paquet `debhelper` qui font quelques opérations sur votre paquet pour le rendre conforme à la Charte Debian.

Si votre paquet est un « Architecture : all », vous devez inclure toutes les commandes pour construire le paquet sous la règle « binary-indep », et laisser la règle « binary-arch » vide.

Les noms des programmes `debhelper` commencent par `dh_` et la suite indique ce que chaque petit utilitaire fait. Tout cela est plutôt explicite, mais voici quelques explications supplémentaires :

- `dh_testdir(1)` vérifie que vous êtes dans le bon répertoire (i.e. le répertoire racine des sources),
- `dh_testroot(1)` vérifie que vous avez les permissions root, nécessaire pour les cibles « binary-arch », « binary-indep » et « clean »,
- `dh_installmanpages(1)` copie les pages de manuel à la bonne place dans le répertoire de destination, vous devez juste lui dire où elles sont, relativement au répertoire racine des sources,
- `dh_strip(1)` retire les en-têtes de débogage des fichiers exécutable pour les rendre plus petits,
- `dh_compress(1)` compresse les pages de manuel et la documentation plus large que 4 kb, avec `gzip(1)`,
- `dh_installdeb(1)` copie les fichiers relatifs au paquet (par exemple les scripts du responsable) sous le répertoire `debian/gentoo/DEBIAN`,
- `dh_shlibdeps(1)` calcule les dépendances des bibliothèques et des exécutable,
- `dh_gencontrol(1)` génère et installe une version soigneusement ajustée du fichier `control` dans `debian/gentoo/DEBIAN`
- `dh_md5sums(1)` génère les sommes de contrôle MD5 pour tous les fichiers dans le paquet.

Pour une information plus complète sur ce que font tous ces scripts `dh_*`, et ce que sont leurs options, lisez les pages de manuel respectives. Il y en a d'autres, potentiellement très utiles, qui ne sont pas mentionnés ici. Si vous en avez besoin, lisez la documentation de `debhelper`.

La section `binary-arch` est celle où vous devriez vraiment commenter ou retirer toutes les lignes qui appellent des fonctionnalités dont vous n'avez pas besoin. Pour `gentoo`, je commente les lignes concernant `exemples`, `cron`, `init`, `man` et `info`, simplement parce que `gentoo` n'en a pas besoin. De plus, à la ligne 68, je remplace « `ChangeLog` » par « `FIXES` », parce que c'est le nom du fichier des modifications amont.

Les deux dernières lignes (avec toutes celles qui ne sont pas expliquées ici) sont juste des choses plus ou moins nécessaires, à propos desquelles vous pouvez lire le manuel de `make`, et la Charte Debian. Pour l'instant il n'est pas important d'en savoir plus.

Chapitre 5

Autres fichiers dans le répertoire debian/

Vous verrez qu'il y a plusieurs autres fichiers dans le sous-répertoire debian, la plupart d'entre eux avec le suffixe « .ex », ce qui signifie qu'ils sont des exemples. Jetez un coup d'œil à chacun d'entre eux. Si vous souhaitez ou devez utiliser une de ces options :

- lisez la documentation relative (astuce : la Charte Debian),
- si nécessaire, modifiez les fichiers selon vos besoins,
- renommez-les pour retirer le suffixe « .ex » s'ils en ont,
- renommez-les pour retirer le préfixe « ex. » s'ils en ont,
- modifiez le fichier « rules » si nécessaire.

Certains de ces fichiers, les plus utilisés, sont décrits dans les sections suivantes.

5.1 README.Debian

Tous les détails ou différences entre le paquet original et votre version debianisée devraient être documentés ici.

dh_make en crée un par défaut, qui ressemble à ceci :

```
gentoo for Debian
-----

<possible notes regarding this package - if none, delete this file>

-- Josip Rodin <joy-mg@debian.org>, Wed, 11 Nov 1998 21:02:14 +0100
```

Puisque nous n'avons rien en particulier à mettre ici, nous effaçons le fichier.

5.2 `conffiles.ex`

L'une des choses les plus irritantes à propos des logiciels est de consacrer beaucoup de temps et d'efforts pour configurer un programme, et de voir une seule mise à jour détruire tous vos changements. Debian résout ce problème en marquant les fichiers de configuration de sorte que quand vous mettez à jour un paquet, il vous sera demandé si vous voulez garder votre ancienne configuration ou pas.

La façon de procéder pour un paquet est d'entrer le chemin complet de chaque fichier de configuration (en général sous `/etc`), un par ligne dans un fichier nommé `conffiles`. Gentoo a un fichier de configuration, `/etc/gentoorc`, et nous le mettons dans `conffiles`.

Si votre programme utilise des fichiers de configuration, mais les réécrit aussi de son côté, il vaut mieux ne pas les marquer comme `conffiles`, parce que `dpkg` va alors interroger les utilisateurs pour vérifier les modifications tout le temps.

Si le programme que vous empaquetez requiert que chaque utilisateur modifie le fichier de configuration pour fonctionner tout court, envisagez de ne pas le marquer non plus comme `conffile`.

Vous pouvez gérer des fichiers d'exemple de configuration à partir des « scripts du responsable ». Lisez '`postinst.ex`, `preinst.ex`, `postrm.ex`, `prerm.ex`' page 30 pour plus d'information.

Si votre programme n'a pas de `conffiles`, vous pouvez tranquillement effacer le fichier `conffiles` du répertoire `debian`.

5.3 `cron.d.ex`

Si votre paquet requiert des tâches programmées régulièrement pour fonctionner correctement, vous pouvez utiliser ce fichier pour le configurer.

Notez que ceci n'inclut pas la rotation des journaux; pour cela, voyez `dh_installlogrotate(1)` et `logrotate(8)`.

Sinon, supprimez-le.

5.4 `dirs`

Ce fichier spécifie les répertoires dont nous avons besoin mais que la procédure d'installation normale (`make install`) ne crée pas.

Par défaut, il ressemble à ceci :

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

Remarquez que le préfixe slash n'est pas inclus. Nous devrions normalement le changer comme ceci :

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

mais ces répertoires sont déjà créés dans Makefile, donc nous n'avons pas besoin de ce fichier, et allons plutôt l'effacer.

5.5 docs

Ce fichier spécifie les noms des fichiers de documentation que `dh_installdocs` peut installer pour nous dans le répertoire temporaire.

Par défaut, il inclut tous les fichiers, existant dans le répertoire racine des sources, qui sont nommés « `BUGS` », « `README*` », « `TODO` », etc.

Pour gentoo, j'ai aussi inclus d'autres choses :

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

Nous pouvons aussi retirer ce fichier et à la place lister ces fichiers directement dans la ligne de commande `dh_installdocs` dans le fichier `rules`, comme ceci :

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

Aussi incroyable que cela puisse être, vous pouvez ne pas avoir de tels fichiers dans les sources de votre paquet. Dans ce cas, vous pouvez tranquillement effacer ce fichier. Mais ne retirez pas l'appel `dh_installdocs` du fichier `rules` parce qu'il est utilisé pour installer le fichier `copyright` et d'autres choses.

5.6 emacsen-*.ex

Si votre paquet fournit des fichiers Emacs qui peuvent être byte-compilés au moment de l'installation, vous pouvez utiliser ces fichiers pour les configurer.

Ils sont installés dans le répertoire temporaire par `dh_installemacsen(1)`, donc n'oubliez pas de décommenter cette ligne dans le fichier `rules` si vous l'utilisez.

Si vous n'en avez pas besoin, effacez-les.

5.7 `init.d.ex`

Si votre paquet est un démon qui doit être lancé au démarrage du système, vous avez de toute évidence ignoré mes recommandations initiales, n'est-ce pas ? :-)

Ceci est un squelette de fichier générique pour un fichier script `/etc/init.d/`, donc vous aurez à l'éditer souvent. Il est installé dans le répertoire temporaire par `dh_installinit(1)`.

Si vous n'en avez pas besoin, effacez-le.

5.8 `manpage.1.ex`, `manpage.sgml.ex`

Votre programme devrait avoir une page de manuel. S'il n'en a pas, chacun de ces fichiers est un squelette que vous pouvez remplir.

Les pages de manuel sont normalement écrites en `nroff(1)`. L'exemple `manpage.1.ex` est aussi écrit en `nroff`. Lisez la page de manuel `man(7)` pour une description brève de la façon d'éditer ce genre de fichier.

D'un autre côté, si vous préférez écrire du SGML à la place de `nroff`, vous pouvez utiliser le patron `manpage.sgml.ex`. Si vous le faites, vous devez :

- installer le paquet `docbook-to-man`
- ajouter `docbook-to-man` à la ligne `Build-Depends` dans le fichier `control`
- retirer le commentaire de l'appel `docbook-to-man` dans la règle « `build` » de votre fichier `rules`.

Et souvenez-vous de renommer le fichier en quelque chose comme `gentoo.sgml` !

Le nom du fichier de la page de manuel finale devrait inclure le nom du programme qu'elle documente, donc nous le renommons de « `manpage` » en « `gentoo` ». Le nom du fichier inclut aussi « `.1` » comme premier suffixe, ce qui signifie que c'est une page de manuel pour une commande utilisateur. Assurez-vous de vérifier que cette section est bien la bonne. Voici une courte liste des sections de pages de manuel :

Section	Description	Notes
1	Commandes utilisateur	Commandes ou scripts exécutables.
2	Appel système	Fonctions fournies par le noyau.
3	Appel bibliothèque	Fonctions des bibliothèques système.
4	Fichiers spéciaux	D'ordinaire trouvés dans <code>/dev</code> .
5	Formats de fichiers	Par ex. le format <code>/etc/password</code> .
6	Jeux	Ou d'autres programmes frivoles.

7	Paquets de macros	Comme les macros de man.
8	Administration système	Des programmes d'habitude exécutés par root
9	Routines noyau	Appels non standards et routines internes.

Donc, la page de manuel de gentoo devrait être appelée `gentoo.1`, ou `gentoo.1x` parce que c'est un programme X11. Il n'y avait pas de page de manuel `gentoo.1` dans les sources originales, donc j'en ai écrit une à partir de l'exemple et de la documentation amont.

5.9 menu.ex

Les utilisateurs de X Window ont un gestionnaire de fenêtres avec un menu qui peut être configuré. S'ils ont installé le paquet `menu` de Debian, un ensemble de menus pour chaque programme sur le système sera créé pour eux.

Voici le fichier `menu.ex` créé par défaut par `dh_make` :

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

Le premier champ après les deux points est « `needs` », et il indique le genre d'interface dont a besoin le programme. Changez ceci en une des alternatives listées, par exemple « `text` » ou « `X11` ».

Le suivant est « `section` », avec le menu et le sous-menu dans lesquels l'entrée devrait apparaître. La liste courante des sections est dans : `/usr/share/doc/debian-policy/menu-policy.html/ch2.html#s2.1`.

le champ « `title` » est le nom du programme. Vous pouvez le commencer par une majuscule si vous le souhaitez. Mais gardez-le court.

Enfin, le champ « `command` » est la commande qui lance le programme.

Maintenant nous changeons l'entrée `menu` en ceci :

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

Vous pouvez aussi ajouter d'autres champs comme « `longtitle` », « `icon` », « `hint` », etc. Voir `menufile(5)`, `update-menus(1)` et `/usr/share/doc/debian-policy/menu-policy.html/` pour plus d'informations.

5.10 watch.ex

Ce fichier est utilisé pour configurer les programmes `uscan(1)` et `uupdate(1)` (dans le paquet `devscripts`). Ils sont utilisés pour surveiller le site sur lequel vous avez obtenu les sources.

Voici ce que j'y ai mis :

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

Astuce : connectez-vous à Internet, et essayez d'exécuter « uscan » dans le répertoire du programme une fois que vous avez créé ce fichier. Et lisez les manuels. :-)

5.11 ex.package.doc-base

Si votre paquet a de la documentation autre que des pages de manuel et des documents info, vous devriez utiliser le fichier `doc-base` pour l'enregistrer, de sorte que l'utilisateur puisse le trouver avec par exemple `dhelp(1)`, `dwww(1)` ou `doccentral(1)`.

Cela inclut normalement les fichiers HTML, PS et PDF, délivrés dans `/usr/share/doc/nom_du_paquet/`.

Voici ce à quoi le fichier `doc-base` de `gentoo` ressemble :

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

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

Pour plus d'informations sur le format de ce fichier, voir `install-docs(8)` et le manuel de `doc-base`, dans `/usr/doc/doc-base/doc-base.html/index.html`.

Pour plus d'informations sur l'installation de documentation supplémentaire, voir 'Installer dans un sous-répertoire' page 9.

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

Ces fichiers sont nommés scripts de responsable. Ce sont des scripts placés dans la zone de contrôle du paquet et sont exécutés par `dpkg` lorsque votre paquet est installé, mis à jour ou supprimé.

Pour l'instant, vous devriez éviter les scripts de responsable si vous le pouvez parce qu'ils ont tendance à être complexes. Pour plus d'informations regardez dans le chapitre 6 de la Charte Debian, et examinez les fichiers d'exemples fournis par `dh_make`.

Chapitre 6

Construire le paquet

Nous devrions maintenant être prêts à construire le paquet.

6.1 Reconstruction complète

Allez dans le répertoire principal du programme et lancez ceci :

```
dpkg-buildpackage -rfakeroot
```

Ceci fera tout pour vous. Il va :

- nettoyer l'arbre des sources (debian/rules clean), en utilisant `fakeroot`
- construire le paquet source (`dpkg-source -b`)
- construire le programme (debian/rules build)
- construire le paquet binaire (debian/rules binary), en utilisant `fakeroot`
- signer le fichier source `.dsc`, en utilisant `gnupg`
- créer et signer le fichier de téléchargement `.changes`, en utilisant `dpkg-genchanges` et `gnupg`.

La seule entrée qui vous sera demandée est votre phrase de passe secrète GPG, deux fois.

Une fois que c'est fait, vous verrez les fichiers suivants dans le répertoire au-dessus (`~/gentoo` /):

- `gentoo_0.9.12.orig.tar.gz`
Ceci est le code source original, simplement renommé pour être conforme aux standards Debian. Notez qu'il a été créé en utilisant l'option « `-f` » du programme `dh_make` quand nous l'avons initialement appelé.
- `gentoo_0.9.12-1.dsc`
Ceci est un résumé du contenu du code source. Ce fichier est généré à partir du fichier « `control` », et est utilisé pour décompresser les sources avec `dpkg-source(1)`. Ceci est un fichier signé en GPG, de sorte que les gens peuvent être sûrs qu'il s'agit bien du vôtre.

– *gentoo_0.9.12-1.diff.gz*

Ce fichier compressé contient chacune des additions que vous avez faites au code source original, sous une forme connue comme « différence unifiée ». Il est créé et utilisé par `dpkg-source(1)`. Attention : si vous ne nommez pas le paquet source original `nomdupaquet_version.orig.tar.gz`, `dpkg-source` échouera à générer le fichier `.diff.gz` correctement. Si quelqu'un d'autre veut recréer votre paquet depuis le début, il peut facilement le faire en utilisant ces trois fichiers. La procédure d'extraction est facile : copier simplement ces trois fichiers quelque part et lancer `dpkg-source -x gentoo_0.9.12-1.dsc`.

– *gentoo_0.9.12-1_i386.deb*

Ceci est le paquet binaire finalisé. Vous pouvez utiliser `dpkg` pour l'installer ou le retirer comme tout autre paquet.

– *gentoo_0.9.12-1_i386.changes*

Ce fichier contient toutes les modifications faites dans la révision courante du paquet, et est utilisé par les programmes de maintenance des archives FTP Debian pour y installer les paquets binaires et sources. Il est partiellement généré à partir du fichier « changelog » et du fichier `.dsc`. Ce fichier est signé en GPG, de sorte que les gens peuvent être sûrs qu'il s'agit bien du vôtre.

Au fur et à mesure que vous travaillez sur le paquet, son comportement va changer et de nouvelles capacités seront ajoutées. Les gens qui téléchargent votre paquet peuvent lire ce fichier et voir rapidement ce qui a changé. Les programmes de maintenance des archives Debian vont aussi poster le contenu de ce fichier sur la liste de distribution `debian-devel-change`.

Les longues chaînes de chiffres dans les fichiers `.dsc` et `.changes` sont des sommes MD5 pour les fichiers mentionnés. Les personnes téléchargeant vos fichiers peuvent les tester avec `md5sum(1)` et si les fichiers ne correspondent pas, elles sauront que le fichier a été corrompu ou qu'il a été falsifié.

6.2 Reconstruction rapide

Avec un paquet imposant, vous ne voudrez sans doute pas reconstruire depuis le début chaque fois que vous faites une petite modification à `debian/rules`. Pour tester, vous pouvez faire un fichier `.deb` sans reconstruire les sources amont comme ceci :

```
fakeroot debian/rules binary
```

Une fois que vous avez fini vos ajustements, souvenez-vous de reconstruire en suivant la procédure correcte ci-dessus. Vous pouvez ne pas être capable de télécharger correctement si vous essayez de télécharger des fichiers `.deb` construits comme ceci.

6.3 La commande `debuild`

Vous pouvez automatiser encore plus le processus de construction avec la commande `debuild`. Voir `debuild(1)`.

La configuration de la commande `debuild` peut être faite via `/etc/devscripts.conf` ou `~/.devscripts`. Je suggère au moins les entrées suivantes :

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

Avec ceux-ci, vous pouvez construire des paquets en utilisant toujours votre clé GPG et éviter d'inclure des composants non désirés (c'est bon aussi pour le parrainage). Par exemple, nettoyer les sources et reconstruire le paquet depuis un compte utilisateur est simple comme :

```
debuild clean  
debuild
```

6.4 Le système `dpatch`

Le simple usage des commandes `dh_make` et `dpkg-buildpackage` va créer un unique et large fichier `diff.gz` qui contient les fichiers de maintenance de paquet dans `debian/` et les fichiers de différences par rapport aux sources. Un tel paquet est un peu délicat à inspecter et à comprendre pour chaque modification de l'arbre source par la suite. Ce n'est pas bien.¹

Plusieurs méthodes pour la maintenance de l'ensemble de différences ont été proposées et sont utilisées avec les paquets Debian. Le système `dpatch` est l'un des plus simples parmi ce genre de systèmes. D'autres sont `dbfs`, `cdbs`, etc.

Un paquet qui est empaqueté correctement avec le système `dpatch` a les modifications des sources clairement documentées comme des ensembles de fichiers de différences dans `debian/patches` et l'arbre source est intact à l'extérieur du répertoire `debian`. Si vous demandez à votre parrain de télécharger votre paquet, cette séparation plutôt claire et cette documentation de vos changements sont très importantes pour accélérer l'examen du paquet par votre parrain. La méthode d'utilisation de `dpatch` est expliquée dans `dpatch(1)`.

Quand quelqu'un (éventuellement vous-même) fournit par la suite une différence sur les sources, la modification du paquet avec `dpatch` est assez simple :

- éditez la différence pour en faire une différence `-p1` sur l'arbre source ;
- ajoutez l'en-tête en utilisant `dpatch patch-template` ;
- déposez-le dans `debian/patches` ;
- ajoutez les noms de fichiers `dpatch` à `debian/patches/00list`.

De plus, `dpatch` a la capacité de créer des différences liées à l'architecture en utilisant des macros CPP.

¹Si vous n'êtes pas encore développeur Debian et que vous demandez à votre parrain de télécharger votre paquet après son examen, vous devriez rendre ce paquet aussi facile que possible à examiner.

6.5 Inclure `orig.tar.gz` pour le téléchargement

Lorsque vous téléchargez le paquet vers l'archive pour la première fois, vous devez inclure les sources `orig.tar.gz` originales. Si la version du paquet n'est pas à une révision Debian `-0` ou `-t`, vous devez passer l'option « `-sa` » à la commande `dpkg-buildpackage`. D'un autre côté, l'option « `-sd` » forcera l'exclusion des sources `orig.tar.gz` originales.

Chapitre 7

Contrôler les erreurs du paquet

7.1 Les paquets `lintian` et `linda`

Lancez `lintian(1)` et `linda(1)` sur votre fichier `.changes` ; ces programmes vont rechercher un grand nombre d'erreurs d'emballage courantes. Les commandes sont :

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

Bien sûr, remplacez le nom de fichier par celui du fichier `.changes` généré pour votre paquet. S'il s'avère qu'il y a des erreurs (les lignes commençant avec `E :`), lisez l'explication (les lignes `N :`), corrigez les erreurs, et reconstruisez comme décrit dans 'Reconstruction complète' page 31. S'il y a des lignes qui commencent avec `W :`, il s'agit de mises en garde, donc vous pouvez ajuster votre paquet ou vous assurer que les mises en garde sont inutiles (et faire en sorte que `lintian` les ignore ; voir la documentation pour les détails).

Remarquez que vous pouvez reconstruire le paquet avec `dpkg-buildpackage`, lancer `lintian` et `linda` en une seule commande avec `debuild(1)`.

7.2 La commande `mc`

Vous pouvez décompresser le contenu d'un paquet `*.deb` avec la commande `dpkg-deb(1)`. Vous pouvez lister le contenu d'un paquet Debian généré avec `debc(1)`.

Tout ceci peut être transformé en un processus intuitif en utilisant un gestionnaire de fichiers comme `mc(1)` qui vous permet de consulter non seulement le contenu des fichiers paquet `*.deb`, mais aussi les fichiers `*.diff.gz` et `*.tar.gz`.

Soyez attentif aux fichiers inutiles ou de taille nulle, à la fois dans les paquets binaires et source. Souvent les fichiers inutiles ne sont pas nettoyés correctement ; ajustez votre fichier `rules` pour compenser.

Astuce : « `zgrep ^+++ ../gentoo_0.9.12-1.diff.gz` » vous donnera la liste de changements/additions faites aux fichiers source, et « `dpkg-deb -c gentoo_0.9.12-1_i386.deb` » ou « `debc gentoo_0.9.12-1_i386.changes` » donnera la liste des fichiers dans le paquet binaire.

7.3 La commande `debdiff`

Vous pouvez comparer la liste des fichiers de deux paquets Debian binaires avec `debdiff(1)`. Ceci est utile pour vérifier qu'aucun fichier n'a été déplacé ou supprimé malencontreusement, et qu'aucune autre modification inattendue n'a été faite en mettant à jour les paquets. Vous pouvez vérifier un groupe de fichiers `*.deb` simplement avec « `debdiff old-package.change new-package.change` ».

7.4 La commande `interdiff`

Vous pouvez comparer deux fichiers `diff.gz` avec la commande `interdiff(1)`. Ceci est utile pour vérifier qu'aucune modification inattendue n'a été effectuée sur les sources par le gestionnaire en mettant à jour les paquets. Exécutez « `interdiff -z old-package.diff.gz new-package.diff.gz` ».

7.5 La commande `debi`

Installez le paquet pour le tester vous-même, par exemple en utilisant `debi(1)` en tant que `root`. Essayez de l'installer sur d'autres machines que la vôtre et vérifiez attentivement chaque avertissement ou erreur tant à l'installation qu'en exécutant le programme.

7.6 Le paquet `pbuilder`

Pour un environnement de construction propre (`chroot`) permettant de vérifier les dépendances de construction, `pbuilder` est très utile. Cela garantit une construction propre des sources en construction automatique pour différentes architectures et évite l'erreur de gravité sérieuse FTBFS (Fails To Build From Source, ne se construit pas à partir des sources), qui est toujours en catégorie RC (Release Critical, bloquant l'intégration). Voir <http://buildd.debian.org/> pour plus d'informations sur le constructeur automatique de paquet Debian.

L'usage le plus basique du paquet `pbuilder` est l'invocation directe de la commande `pbuilder` en tant que `root`. Par exemple, exécutez les commandes qui suivent dans le répertoire où `.orig.tar.gz`, `.diff.gz` et `.dsc` se trouvent pour construire un paquet.

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

Les paquets nouvellement créés seront placés dans `/var/cache/pbuilder/result/` et appartiendront au superutilisateur.

La commande `pdebuild` vous aide à utiliser les fonctions du paquet `pbuilder` depuis un compte utilisateur normal. Depuis la racine de l'arbre source, en ayant le fichier `orig.tar.gz` dans son répertoire parent, vous exécutez les commandes suivantes :

```
$ sudo pbuilder create # si deuxième fois, sudo pbuilder update
$ pdebuild
```

Les paquets nouvellement créés seront placés dans `/var/cache/pbuilder/result/` et n'appartiendront pas au superutilisateur.¹

Si vous voulez ajouter des sources supplémentaires apt pour être utilisées par le paquet `pbuilder`, mettez `OTHERMIRROR` dans `~/pbuilder.rc` ou `/etc/pbuilder.rc` et exécutez (pour *Sarge*) :

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

L'utilisation de `--override-config` est nécessaire pour mettre à jour les sources apt dans l'environnement chroot.

Voir <http://www.netfort.gr.jp/~dancer/software/pbuilder.html>, `pdebuild(1)`, `pbuilder.rc(5)`, et `pbuilder(8)`.

¹Pour l'instant, je vous recommande de configurer votre système en donnant accès en écriture au répertoire `/var/cache/pbuilder/result/` à l'utilisateur, et en ajoutant aux fichiers `~/pbuilder.rc` au `/etc/pbuilder.rc` la ligne

```
AUTO_DEBSIGN=yes
```

Ceci vous permettra de signer les paquets générés avec votre clé secrète GPG sous `~/gnupg/`. Dans la mesure où le paquet `pbuilder` évolue toujours, vous aurez à vérifier la configuration réelle en consultant la dernière documentation officielle.

Chapitre 8

Envoyer votre paquet

Maintenant que vous avez testé votre nouveau paquet en détail, vous êtes prêt à commencer le processus d'application de nouveau responsable Debian, comme décrit dans <http://www.debian.org/devel/join/newmaint>.

8.1 Envoyer vers l'archive Debian

Une fois que vous êtes devenu un responsable Debian officiel, vous devrez télécharger le paquet sur les archives Debian. Vous pouvez le faire manuellement, mais c'est plus facile d'utiliser les outils automatiques fournis, comme `dupload(1)` ou `dput(1)`. Nous décrirons la façon de faire avec `dupload`.

D'abord vous devez écrire le fichier de configuration de `dupload`. Vous pouvez soit éditer le fichier global `/etc/dupload.conf`, ou avoir votre propre fichier `~/dupload` pour remplacer les quelques détails que vous voulez changer. Mettez quelque chose comme ceci dans le fichier :

```
package config;

$default_host = "anonymous-ftp-master";

$cfg{'anonymous-ftp-master'} = {
    fqdn => "ftp-master.debian.org",
    method => "ftp",
    incoming => "/pub/UploadQueue/",
    # les fichiers passent par dinstall sur ftp-master qui envoie
    # les courriers électroniques lui-même
    dinstall_runs => 1,
};

1;
```

Bien sûr, remplacez mes informations personnelles par les vôtres, et lisez la page de manuel `dupload.conf(5)` pour comprendre ce que chacune de ces options signifie.

L'option `$default_host` est la plus compliquée — elle détermine la queue de téléchargement qui sera utilisée par défaut. « `anonymous-ftp-master` » est la principale, mais il est possible que vous souhaitiez en utiliser une autre, plus rapide. Pour plus d'informations sur les queues de téléchargement, lisez la Référence du Développeur, section « La mise à jour d'un paquet », dans `/usr/share/doc/developers-reference/ch-pkgs.en-us.iso-8859-1.html#s-upload`.

Puis connectez-vous à votre fournisseur d'accès Internet et lancez cette commande :

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

`dupload` vérifie que les sommes md5 des fichiers sont identiques à celles du fichier `.changes`, pour qu'il puisse vous avertir de reconstruire comme décrit dans 'Reconstruction complète' page 31 et qu'il charge le fichier correctement.

Si vous rencontrez un problème d'envoi à `ftp://ftp-master.debian.org/pub/UploadQueue/`, vous pouvez le résoudre manuellement en envoyant un fichier `*.commands` signé gpg à `ftp://ftp-master.debian.org/pub/UploadQueue/` avec `ftp`.¹ Par exemple, utilisez `hello.commands` :

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

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

```
Commands:
```

```
rm hello_1.0-1_i386.deb
```

```
mv hello_1.0-1.dsc hello_1.0-1.dsc
```

```
-----BEGIN PGP SIGNATURE-----
```

```
Version: 2.6.3ia
```

```
iQCVAwUBNFiQSXVhJ0HiWnvJAQG58AP+IDJVeSWmDvzMUpHScg1EK0mvChgnuD7h  
BRiVQubXkB2DphLJW5UUSRnjwliuFcYwH/lFpNp17XP95LkLX3iFza9qItw4k2/q  
tvylZkmIA9jxCyv/YB6zZCbHmbvUnL473eLRoxlnYZd3JFaCZMJ86B0Ph4GFNPaf  
Z4jxNrgH7Bc=
```

```
=pH94
```

```
-----END PGP SIGNATURE-----
```

8.2 Envoyer vers une archive privée

Si vous souhaitez créer une archive personnelle de paquets à `URL="http://people.debian.org/~nom_compte"` en tant que développeur avec

¹Voyez `ftp://ftp-master.debian.org/pub/UploadQueue/README`. Alternativement, vous pouvez utiliser la commande `dcut` du paquet `dput`.

une simple invocation de `dupload -t nom_cible`, vous devriez ajouter les lignes suivantes au fichier `/etc/dupload.conf` :

```
# Compte developpeur
$cfg{'nom_cible'} = {
    fqdn => "people.debian.org",
    method => "scpb",
    incoming => "/home/nom_compte/public_html/package/",
    # Je n'ai pas besoin d'annonce
    dinstall_runs => 1,
};
$cfg{'nom_cible'}{preupload}{'changes'} = "
    echo 'mkdir -p public_html/package' | ssh people.debian.org 2>/dev/n
    echo 'Répertoire paquet créé !'";

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

Ici, l'archive APT est créée avec une exécution rapide et sale shell à distance sur SSH. Les fichiers de remplacement requis par `dpkg-scanpackages` et `dpkg-scansources` sont spécifiés comme `/dev/null`. Cette technique peut être utilisée par d'autres que les Développeur Debian pour placer leurs paquets dans leur site web personnel. Vous pouvez aussi utiliser `apt-ftpparchive` ou d'autres scripts pour créer une archive APT.

Chapitre 9

Mettre à jour le paquet

9.1 Nouvelle révision Debian

Disons qu'un rapport de bogue a été rempli pour votre paquet, #54321, et qu'il décrit un problème que vous pouvez résoudre. Pour créer une nouvelle révision du paquet, vous devez :

- Corriger le problème dans le paquet source, bien sûr.
- Ajouter une nouvelle révision au sommet du fichier changelog Debian, par exemple avec « `dch -i` », ou explicitement avec « `dch -v <version>-<revision>` », et ajoutez ensuite les commentaires en utilisant votre éditeur favori.
Astuce : comment obtenir facilement la date au format requis ? Utilisez « `822-date` » ou « `date -R` ».
- Ajoutez une courte description du bogue et de la solution dans l'entrée du changelog, suivie par ceci : « Closes : #54321 ». De cette manière, le rapport de bogue sera automatiquement fermé par le logiciel de maintenance des archives au moment où votre paquet sera accepté dans l'archive Debian.
- Recommencez ce que vous aviez fait dans 'Reconstruction complète' page 31, 'Contrôler les erreurs du paquet' page 35, et 'Envoyer votre paquet' page 39. La différence est que cette fois, l'archive des sources originales ne sera pas incluse, car elle n'a pas été changée et est déjà dans l'archive Debian.

9.2 Nouvelle version amont (basique)

Considérons maintenant une autre situation, légèrement plus compliquée – une nouvelle version amont est disponible, et bien sûr vous voulez en faire un paquet. Vous devez donc :

- télécharger les sources et mettre l'archive source (par exemple nommée « `gentoo-0.9.13.tar.gz` ») dans le répertoire au dessus des anciennes sources (par exemple `~/gentoo/`);
- entrer dans l'ancien répertoire source, et lancer :

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

Bien sûr, remplacez le nom de fichier par celui de l'archive source de votre programme. `uupdate(1)` va correctement renommer cette archive, essayer d'appliquer les modifications de votre précédent fichier `.diff.gz`, et mettre à jour le nouveau fichier `debian/changelog`;

- aller dans le répertoire « `../gentoo-0.9.13` », l'arbre des sources du nouveau paquet, et recommencer ce que vous aviez fait dans 'Reconstruction complète' page 31, 'Contrôler les erreurs du paquet' page 35 et 'Envoyer votre paquet' page 39.

Remarquez que si vous configurez « `debian/watch` » comme indiqué dans 'watch.ex' page 29, vous pouvez lancer `uscan(1)` pour automatiquement chercher les nouvelles sources, les télécharger et exécuter `uupdate` dessus.

9.3 Nouvelle version amont (réel)

Lorsque vous préparez des paquets pour l'archive Debian, vous devez vérifier les paquets en résultant en détail. Voici un exemple plus réaliste de cette procédure :

- 1 vérifiez les modification dans la source amont ;
 - lisez les `changelog` et `NEWS` amonts, ainsi que toute autre documentation distribuée avec la nouvelle version ;
 - faites un « `diff -urN` » entre l'ancienne et la nouvelle source amont pour avoir une idée de l'étendue des modifications, où le travail est en cours (et donc où les nouveaux bogues sont susceptibles d'apparaître), et pour avoir à l'œil quoi que ce soit de suspect ;
- 2 portez l'ancien empaquetage Debian dans la nouvelle version ;
 - décompressez l'archive source et renommez la racine de l'arborescence source en `<nompaquet>-<version_amont>/` et « `cd` » dans ce répertoire ;
 - copiez l'archive source et renommez-la en `<nompaquet>-<version_amont>.orig.tar.gz` ;
 - appliquez les mêmes modifications au nouvel arbre source qu'à l'ancien. Les méthodes possibles sont ; :
 - la commande « `zcat /path/to/<nompaquet>_<ancienne-version>.diff.gz | patch -p1` » ;
 - la commande « `uupdate` » ;
 - la commande « `svn merge` » si vous gérez les sources dans un répertoire Subversion ;
 - simplement copier le répertoire `debian/` de l'ancien arbre source s'il a été empaqueté avec `dpatch`.
 - préservez les anciennes entrées `changelog` (cela va de soit, mais il y a déjà eu des incidents...);
 - le nouveau numéro de version et le numéro de version amont ajouté au numéro de révision Debian `-t`, par exemple « `'0.9.13-1` » ;
 - ajoutez une entrée `changelog` avec "New upstream release" pour cette nouvelle version au sommet de `debian/changelog`. Par exemple, « `dch -v 0.9.13-1` » ;
 - décrivez succinctement les modifications *dans* la nouvelle version amont qui fixent des bogues et qui ferment les rapports associés dans le `changelog` ;

- décrivez succinctement les modifications à la nouvelle version amont par le responsable qui fixent des bogues rapportés et qui ferment les rapports dans le changelog ;
 - si les modifications/fusions¹ ne s’appliquent pas proprement, inspectez la situation pour déterminer ce qui a échoué (les indices sont dans les fichiers `.rej`). La plupart du temps, le problème est qu’une modification que vous aviez appliquée aux sources a été intégrée en amont, et donc n’est plus pertinente ;
 - les mises à jour vers la nouvelle version devraient être silencieuses et non intrusives (les utilisateurs existants ne devraient pas remarquer la mise à jour à part en découvrant que de vieux bogues ont été résolus, et peut-être que de nouvelles fonctionnalités sont présentes) ;²
 - si vous devez, pour quelque raison que ce soit, ajouter des fichiers modèles qui avaient été effacés, pour pouvez exécuter `dh_make` à nouveau dans le même répertoire, déjà « debianisé », avec l’option `-o`. Puis éditez les correctement ;
 - les modifications Debian existantes doivent être réévaluées ; jetez tout ce qui a été incorporé en amont (sous une forme ou une autre), et souvenez-vous de garder ce qui ne l’a pas été, à moins qu’il n’y ait une bonne raison de ne pas le faire ;
 - si le système de construction a été changé (avec un peu de chance, vous êtes au courant depuis l’étape 1), mettez à jour les dépendances de construction `debian/rules` et `debian/control`, si besoin est.
- 3 construisez le nouveau paquet comme expliqué dans ‘La commande `debuild`’ page 32 ou ‘Le paquet `pbuilder`’ page 36. L’utilisation de `pbuilder` est souhaitable ;
 - 4 vérifiez que le nouveau paquet est construit correctement ;
 - effectuez ‘Contrôler les erreurs du paquet’ page 35 ;
 - effectuez ‘Vérifier les mises à jour de paquet’ page suivante ;
 - vérifiez à nouveau si des bogues rapportés dans le système de suivi des bogues Debian (Bug Tracking System BTS) (<http://www.debian.org/Bugs/>) ont été résolus ;
 - vérifiez le contenu du fichier `.changes` pour vous assurer que vous téléchargez vers la bonne distribution, que les rapports de bogues refermés sont correctement listés dans les champs `Closes :`, que les champs `Maintainer :` et `Changed-By :` correspondent, que le fichier est signé GPG, etc. ;
 - 5 si des modifications devaient être faites pour corriger quoi que ce soit dans l’empaquetage, recommencez à l’étape 2 jusqu’à obtenir satisfaction ;
 - 6 si votre téléchargement doit être parrainé, souvenez-vous de noter toute option spéciale requise pour construire le paquet (comme « `dpkg-buildpackage -sa -v ...` ») et d’en informer votre parrain/marraine pour qu’il/elle construise le paquet correctement ;
 - 7 si vous téléchargez vous-même, effectuez ‘Envoyer votre paquet’ page 39.

¹NdT : patch/merge

²Faites en sorte que votre paquet mette à jour correctement les fichiers de configuration lors des mises à jour en utilisant des scripts de responsable (`postinst`, ...) bien conçus, de sorte qu’il ne fasse pas ce que l’utilisateur ne veut pas ! Ce sont ces améliorations qui expliquent **pourquoi** les gens choisissent Debian. Lorsque la mise à jour est nécessairement intrusive (par exemple des fichiers de configuration éparpillés à travers plusieurs répertoires home avec des structures complètement différentes), vous pouvez envisager de remettre une configuration par défaut sûre du paquet (par exemple, le service est désactivé), et de fournir les documentations adéquates requises par la Charte Debian (`README.Debian` et `NEWS.Debian`) en dernier ressort. Mais ne vous souciez pas d’une note `debconf`.

9.4 Le fichier `orig.tar`

Si vous essayez de construire des paquets seulement à partir de la nouvelle arborescence source avec le répertoire `debian` sans le fichier `orig.tar.gz` dans le répertoire parent, vous aurez créé sans le vouloir un paquet source natif, qui vient sans fichier `diff.gz`. Ce genre d'empaquetage n'est approprié que pour les paquets spécifiques à Debian, qui ne seront jamais utiles dans une autre distribution.³

Pour obtenir un paquet source non natif qui contient à la fois le fichier `orig.tar.gz` et le fichier `diff.gz`, vous devez copier manuellement l'archive amont dans le répertoire parent avec son nom changé en `<nompaquet>_<version_amont>.orig.tar.gz`, comme l'avait fait la commande `dh_make` dans « Debianisation » initiale³ page 8.

9.5 La commande `cvs-buildpackage` et similaires

Vous devriez considérer l'usage d'un système de gestion de code source pour gérer l'activité d'empaquetage. Il y a plusieurs scripts qui sont personnalisés pour être utilisés avec les plus populaires.

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

Ces commandes automatisent aussi l'empaquetage de nouvelles versions amonts.

9.6 Vérifier les mises à jour de paquet

Quand vous construisez une nouvelle version du paquet, vous devriez toujours suivre cette procédure pour vérifier le paquet, pour être tranquillement mis à jour :

- mettez à jour depuis la version précédente ;
- revenez à la version précédente, puis retirez-la ;
- installez le nouveau paquet ;
- retirez-le et réinstallez-le à nouveau ;
- purgez-le.

Si le paquet utilise des scripts `pre/post/inst/rm` non triviaux, veillez à tester leurs chemins de mises à jour.

³Certains affirment que, même pour des paquets spécifiques à Debian, il est préférable de conserver le contenu du répertoire `debian/` dans le fichier `diff.gz`, plutôt que dans le fichier `orig.tar.gz`.

Gardez à l'esprit que si votre paquet a été livré avec Debian, les gens vont souvent mettre à jour votre paquet à partir de la révision qui était dans la dernière version Debian. Souvenez-vous de tester aussi les mises à jour à partir de cette révision.

Chapitre 10

Où demander de l'aide

Avant de vous décider à poser une question dans un lieu public, s.v.p. RTFM. Ceci inclut la documentation sous `/usr/share/doc/dpkg`, `/usr/share/doc/debian`, `/usr/share/doc/autotools-dev/README.Debian.gz`, `/usr/share/doc/paquet/*` et les pages de manuel/d'info pour tous les programmes mentionnés dans ce document. Consultez toutes les informations dans <http://nm.debian.org/> et http://people.debian.org/~mpalmer/debian-mentors_FAQ.html.

Si vous avez des questions sur la création de paquets pour lesquelles vous n'avez pas pu trouver de réponse dans la documentation, vous pouvez les poser sur la liste de diffusion Debian Mentors, sur `<debian-mentors@lists.debian.org>`. Les responsables Debian les plus expérimentés seront heureux de vous aider, mais au moins lisez une partie de la documentation avant de poser une question !

Consultez <http://lists.debian.org/debian-mentors/> pour plus d'informations sur cette liste de diffusion.

Quand vous recevez un rapport de bogue (oui, un rapport de bogue réel!) vous saurez qu'il est temps pour vous de plonger dans le système de suivi de bogues Debian (<http://www.debian.org/Bugs/>) et de lire la documentation, pour être à même de gérer les rapports efficacement. Je recommande chaudement la lecture de la Référence du Développeur, chapitre « Gérer les bogues », dans `/usr/share/doc/developers-reference/ch-pkgs.en-us.iso-8859-1.html#s-bug-handling`.

Si vous avez encore des questions, posez-les sur la liste de diffusion Debian Developers à `<debian-devel@lists.debian.org>`. Consultez <http://lists.debian.org/debian-devel/> pour plus d'informations sur cette liste de diffusion.

Même si tout marche bien, il est temps de commencer à prier. Pourquoi? Parce que dans quelques heures (ou jours) les utilisateurs du monde entier vont commencer à utiliser votre paquet, et si vous avez fait des erreurs critiques, vous serez bombardé par les courriers électroniques d'utilisateurs Debian furieux... Je plaisante. :-)

Relaxez-vous et soyez prêt pour les rapports de bogues, parce qu'il y aura beaucoup plus de travail à faire avant que votre paquet soit parfaitement conforme aux règles Debian (une fois encore, lisez la *documentation réelle* pour les détails). Bonne chance !

Annexe A

Exemples

Ici nous empaquetons l'archive amont *gentoo-1.0.2.tar.gz* et envoyons tous les paquets vers la *cible_nm*.

A.1 Exemple d'empaquetage simple

```
$ mkdir -p /chemin/vers # nouveau répertoire vide
$ cd /chemin/vers
$ tar -xvzf /chemin/depuis/gentoo-1.0.2.tar.gz # prendre les sources
$ cd gentoo-1.0.2
$ dh_make -e nom@domaine.dom -f /chemin/depuis/gentoo-1.0.2.tar.gz
... Répondre aux questions.
... Corriger l'arbre source
... Si c'est un paquet script, mettre debian/control à "Architecture: all"
... Ne pas effacer ../gentoo_1.0.2.orig.tar.gz
$ debuild
... S'assurer qu'il n'y a pas d'alerte
$ cd ..
$ dupload -t cible_nm gentoo_1.0.2-1_i386.changes
```

A.2 Exemple d'empaquetage avec les paquets *dpatch* et *pbuilder*

```
$ mkdir -p /chemin/vers # nouveau répertoire vide
$ cd /chemin/vers
$ tar -xvzf /chemin/depuis/gentoo-1.0.2.tar.gz
$ cp -a gentoo-1.0.2 gentoo-1.0.2-orig
$ cd gentoo-1.0.2
$ dh_make -e nom@domaine.dom -f /chemin/depuis/gentoo-1.0.2.tar.gz
... Répondre aux questions.
```

```
... Corriger l'arbre source
... Essayer de construire le paquet avec "dpkg-buildpackage -rfakeroot -us
... Éditer les sources pour permettre la construction du paquet
... Ne pas effacer ../gentoo-1.0.2.orig.tar.gz
$ cd ..
$ cp -a gentoo-1.0.2 gentoo-1.0.2-keep # copie de sauvegarde
$ mv gentoo-1.0.2/debian debian
$ diff -Nru gentoo-1.0.2-orig gentoo-1.0.2 > fichier-diff
... Vous pouvez écraser le répertoire gentoo-1.0.2 en faisant ceci.
... Assurez-vous de garder gentoo-1.0.2-keep pour votre sécurité
$ mkdir -p debian/patches
$ dpatch patch-template fichier-diff \
    -p "01_patchname" "description fichier-diff" \
    < fichier-diff > debian/patches/01_patchname.dpatch
$ cd debian/patches
$ echo 01_patchname.dpatch >00list
$ cd ../../ # retour vers /chemin/vers
$ rm -rf gentoo-1.0.2
$ editor debian/rules
```

Voici à quoi ressemble le `debian/rules` originel :

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

Éditez `debian/rules` de la manière suivante pour utiliser `dpatch` :

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

```
patch-stamp:
    dpatch apply-all
    dpatch call-all -a=pkg-info >patch-stamp

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

Vous êtes maintenant prêt pour réempaqueter l'arbre source avec le système `dpatch`.

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