Guide du nouveau responsable Debian

Josip Rodin <joy-mg@debian.org>

Traduction par Frédéric Dumont <frederic.dumont@easynet.be>
Mohammed Adnène Trojette <adn+deb@diwi.org>
David Prévot <david@tilapin.org>
et les membres de la liste <debian-l10n-french@lists.debian.org>

version 1.2.20, 2010-06-26 11 :37 :31 UTC

Copyright

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

Making a Debian Package (le manuel de debmake), copyright © 1997 Jaldhar Vyas.

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

Table des matières

Chapitre 1

Partir du bon pied

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 : « *Longum iter est per preaecepta, breve et efficax per exempla !* » (c'est long par la règle, court et efficace par l'exemple !).

Ce document a été mis à jour pour la version `Squeeze` de Debian. [1]

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 sûre, cependant : pour créer et maintenir correctement des paquets Debian, il faut beaucoup de temps. Ne vous faites pas d'illusion, pour que votre système fonctionne, les responsables doivent à la fois être techniquement compétents et consciencieux.

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.

Si vous avez besoin d'aide sur l'empaquetage, veuillez vous reporter en 'Où demander de l'aide' page 5.

Les nouvelles versions de ce document devraient toujours être disponibles en ligne sur `http://www.debian.org/doc/maint-guide/`. La version de référence en anglais est disponible sur `http://www.debian.org/doc/maint-guide/index.en.html` et dans le paquet `maint-guide`. La traduction en français est également disponible dans le paquet `maint-guide-fr`.

1. Ce document suppose que vous utilisez le système `Squeeze`. Si vous avez l'intention de suivre ce texte avec un système `Lenny`, vous devez au moins installer les paquets `dpkg` et `debhelper` rétroportés.

1.1 Programmes nécessaires au développement

Avant de commencer quoi que ce soit, vous devriez vous assurer d'avoir correctement installé certains paquets nécessaires au développement. Notez que la liste ne contient aucun paquet marqué `essential` ou `required` (essentiel ou requis) — nous supposons que ceux-ci sont déjà installés.

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

Le paquet le plus important à installer sur un système de développement est `build-essential`. Lors de son installation, il *tirera* avec lui d'autres paquets nécessaires à un environnement de compilation de base.

Pour certaines catégories de paquets, c'est tout ce dont vous aurez besoin. Cependant d'autre paquets, bien que non essentiels à toutes les constructions de paquet, sont utile ou peuvent être nécessaire pour votre paquet :

- `file` — ce programme pratique peut déterminer la nature d'un fichier (voir `file(1)`);
- `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 modifiée (voir `patch(1)`);
- `perl` — Perl est un des langages de script les plus utilisés sur les systèmes modernes similaires à Unix, souvent qualifié de « tronçonneuse suisse d'Unix » (voir `perl(1)`);
- `python` — Python est un autre des langages de script les plus utilisés sur le système Debian qui combine une remarquable puissance et une syntaxe très claire (voir `python(1)`);
- `autoconf`, `automake` et `autotools-dev` — 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` »). `autotools-dev` conserve les versions à jour de certains de ses fichiers automatiques et fournit une documentation sur la meilleure façon d'utiliser ces fichiers ;
- `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)` et `/usr/share/doc/debhelper/README`)[2] ;
- `devscripts` — ce paquet contient de jolis scripts utiles pouvant aider les responsables, mais ils ne sont pas indispensables pour la création de paquets. Les paquets recommandés et suggérés par celui-ci valent le coup d'œil (voir `/usr/share/doc/devscripts/README.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

2. Il y a d'autres paquets similaires mais spécialisés comme `dh-make-perl`, `dh-make-php`, etc.

que vous ferez quand votre travail sera inclus dans la distribution Debian (voir `gpg(1)`);
- `gfortran` — le compilateur Fortran 95 de GNU, nécessaire si votre programme est écrit en Fortran (voir `gfortran(1)`);
- `gpc` — le compilateur Pascal de GNU, nécessaire si votre programme est écrit en Pascal. Mérite d'être mentionné ici, `fp-compiler`, le compilateur pascal libre, convient également (voir `gpc(1)`, `ppc386(1)`);
- `xutils-dev` — 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 indiquer de nombreuses erreurs courantes après la construction du paquet et expliquer les erreurs trouvées (voir `lintian(1)`, `diffstat(1)`, `/usr/share/doc/lintian/lintian.html/index.html`);
- `pbuilder` — ce paquet contient des programmes utilisés pour créer et maintenir un environnement « `chroot`é ». Construire un paquet Debian dans cet environnement permet de vérifier les dépendances correctes de construction et évite les bogues FTBFS (« Fails To Build From Source » pour les échecs de construction à partir du paquet source) (voir `pbuilder(8)` et `pdebuild(1)`);
- `patchutils` — ce paquet contient certains utilitaires pour travailler avec les correctifs comme les commandes `lsdiff`, `interdiff` et `filterdiff`;
- `quilt` — ce paquet aide à gérer un ensemble de correctifs en gardant une trace du rôle de chacun. Ils sont logiquement organisés en pile, et vous pouvez les appliquer (« push »), les enlever (« pop »), et les rafraîchir facilement en parcourant le pile (voir `quilt(1)` et `/usr/share/doc/quilt/README.Debian`);
- `git-core` — ce paquet fournit un système de gestion de version populaire conçu pour manipuler de très gros projets rapidement et efficacement ; il est utilisé pour des projets libres de grande envergure, en particulier le noyau Linux (voir `git(1)` et `/usr/share/doc/git-doc/index.html`).

Les documents suivants sont *très importants* et doivent être lus en parallèle à ce document :

- `debian-policy` — la Charte Debian (`http://www.debian.org/doc/devel-manuals#policy`) 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 (« Filesystem Hierarchy Standard » ou FHS, qui dit où chaque fichier et répertoire doit se trouver), etc. Le plus important pour vous est qu'elle décrive 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` — la Référence du développeur Debian (`http://www.debian.org/doc/devel-manuals#devref`) concerne tout ce qui n'est pas spécifique aux détails techniques de la création de paquets, comme la structure des archives, comment renommer, abandonner, choisir les paquets, faire une NMU (« Non-Maintainer Uploads » ou mise à jour indépendante), comment gérer les bogues, les meilleures pratiques d'empaquetage, où et quand faire des envois de paquets, etc. (voir `/usr/share/doc/developers-reference/`);
- Le tutoriel des Autotools (`http://www.lrde.epita.fr/~adl/autotools.html`) fournit un très bon tutoriel pour le système de construction GNU connu comme les GNU Au-

totools (http://fr.wikipedia.org/wiki/Autotools) dont les composants les plus importants sont Autoconf, Automake, Libtool, et gettext ;
– gnu-standards — ce paquet contient deux documentations issues du projet GNU : les normes GNU de codage (http://www.gnu.org/prep/standards/html_node/index.html), et les informations pour les responsables de programme GNU (http://www.gnu.org/prep/maintain/html_node/index.html). Bien que Debian n'exige pas que ces recommandations soient suivies, elles sont néanmoins utiles en tant que lignes directrices et bon sens. (voir /usr/share/doc/gnu-standards/standards.html et /usr/share/doc/gnu-standards/maintain.html).

Si ce document entre en contradiction avec ce qui est décrit dans la Charte Debian ou la référence du développeur, elles sont prioritaires. Veuillez signaler un bogue sur le paquet maint-guide.

Les courtes descriptions 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'utilisation standard. Cela peut vous sembler fastidieux maintenant, mais plus tard vous serez *très* content de l'avoir fait.

1.2 Terminologie fondamentale

Deux sortes de paquets existent :

– **paquet source** : un paquet source est un ensemble de fichiers contenant le code et les données à compiler et transformer en programmes exécutables et documents formatés. Il est généralement distribué sous forme d'une association des fichiers *.orig.tar.gz, *.debian.tar.gz (ou *.diff.gz), et *.dsc. D'autres archives et formats de compression peuvent également être utilisés ;
– **paquet binaire** : un paquet binaire contient les programmes exécutables et documents formatés. Il est généralement distribué sous la forme d'un fichier *.deb pour le système Debian normal et *.udeb pour l'installateur Debian.

Ne mélangez pas les termes comme source du programme et paquet source du programme !

Plusieurs noms de rôle sont utilisés au sein de Debian :

– **auteur amont** : la personne qui a créé le programme à l'origine ;
– **responsable amont** : la personne qui maintient actuellement le programme ;
– **responsable** : la personne qui maintient le paquet Debian du programme ;
– **parrain** : la personne qui aide les responsables à envoyer des paquets dans l'archive officielle de paquets Debian après en avoir vérifié le contenu ;
– **mentor** : la personne qui aide les responsables débutants pour l'empaquetage, etc.
– **développeur Debian** (DD) : la personne qui est membre de Debian. Elle a tout les droits d'envoi vers l'archive officielle de paquets Debian ;
– **responsable Debian** (DM) : la personne qui a des droits d'envoi limités vers l'archive officielle de paquets Debian.

Plusieurs noms de version sont utilisés dans Debian :

– **version source amont** : soit la version source amont *version* ;
– **révision Debian** : soit la révision par Debian d'un paquet *revision* ;
– **version de paquet Debian** : la version de paquet Debian est mentionnée comme suit :
  – *version* pour un paquet binaire Debian natif et pour un paquet source Debian ;
  – *version-revision* pour un paquet binaire Debian non natif.

Veuillez lire les autres manuels pour plus de détail sur la terminologie.

1.3 Développeur Debian officiel

Il n'est pas possible de devenir **développeur Debian** (DD) en une nuit car il ne suffit pas de compétences techniques. Veuillez ne pas vous décourager. Si c'est utile à d'autres, vous pouvez toujours envoyer vos paquets soit en tant que **responsable** via un **parrain** ou comme un **responsable Debian**. Voir le coin du nouveau responsable Debian (http://www.debian.org/devel/join/newmaint) pour plus d'informations.

Remarquez qu'il n'est pas nécessaire de créer de nouveau paquet pour devenir un développeur Debian officiel. Contribuer aux paquets existants peut aussi fournir une voie pour devenir un développeur Debian. Beaucoup de paquets sont en attente de bons responsables (voir 'Choix du programme' page 7).

1.4 Où demander de l'aide

Avant de vous décider à poser une question dans un lieu public, veuillez RTFM. Ceci comprend 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 man et info pour tous les programmes mentionnés dans ce document. Consultez toutes les informations dans http://nm.debian.org/.

Faire un petit paquet de test est une bonne façon d'apprendre les particularités de l'empaquetage. Examiner les paquets bien maintenus est le meilleur moyen d'apprendre comment les autres font leurs paquets.

Si vous avez encore 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 des mentors Debian en debian-mentors@lists.debian.org (http://lists.debian.org/debian-mentors/). 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 recevrez un rapport de bogue (oui, un vrai rapport de bogue !) vous saurez qu'il est temps de vous 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. La lecture de la référence du développeur, chapitre 5.8 « Manipulation des bogues » (http://www.debian.org/doc/manuals/developers-reference/pkgs.html#bug-handling) est fortement recommandée.

Si vous avez encore des questions, posez-les sur la liste de diffusion des développeurs Debian en debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/). Consultez http://lists.debian.org/debian-devel/ pour plus d'informations sur cette liste de diffusion.

Même si tout fonctionne 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… :-)

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 *vraie documentation* pour les détails). Bonne chance !

Chapitre 2

Premiers pas

Comment réaliser son premier paquet.

2.1 Choix du programme

Vous avez probablement choisi le paquet que vous voulez créer. La première chose à faire est de vérifier si le paquet ne se trouve pas déjà dans l'archive de la distribution en utilisant `aptitude`.

Vous pouvez aussi vérifier les informations de paquets avec la page de recherche de paquets (`http://www.debian.org/distrib/packages`) et le système de suivi des paquets Debian (`http://packages.qa.debian.org/common/index.html`).

Si le paquet existe déjà, et bien, installez-le ! :-) S'il est orphelin — si son responsable est « Debian QA Group » — vous devriez pouvoir le reprendre.

Vous devriez également consulter sur le site web de Debian la liste des paquets en souffrance et paquets souhaités (« Work-Needing and Prospective Packages » ou WNPP) (`http://www.debian.org/devel/wnpp/`) et l'état à jour d'adoption et d'abandon de paquets (`http://wnpp.debian.net/`).

Si vous pouvez adopter le paquet, récupérez les sources (avec quelque chose comme « `apt-get source` *`nomdepaquet`* ») et examinez-les. Malheureusement ce document n'inclut pas d'informations exhaustives sur l'adoption de paquets. Heureusement, vous ne devriez pas avoir de problèmes à 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 aimeriez le voir dans Debian, procédez comme suit :

– d'abord, assurez-vous qu'il fonctionne, et essayez-le pendant quelques temps pour confirmer son utilité ;
– 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

personne ne travaille dessus, déclarez votre intention de l'empaqueter (« Intent To Package » ou ITP) avec un bogue ITP sur le pseudo-paquet `wnpp` en utilisant `reportbug`. Si quelqu'un travaille déjà dessus, contactez-le si vous voulez. Sinon, trouvez un autre programme intéressant dont personne n'est responsable.
- le programme **doit** avoir une licence :
  - pour la section `main` (principale), elle **doit être conforme à tous les principes du logiciel libre selon Debian (« Debian Free Software Guidelines » ou DFSG)** (voir http://www.debian.org/social_contract#guidelines) et **ce programme ne doit pas dépendre de paquets hors de `main`** pour la compilation ou l'exécution, conformément à la Charte Debian. C'est le cas idéal ;
  - pour la section `contrib` (contributions), il doit être conforme à tous les DFSG mais peut dépendre de paquets hors de `main` pour la compilation ou l'exécution ;
  - pour la section `non-free` (non libre), il peut être non conforme à certaines DFSG mais **doit être distribuable**.

  En cas de doute sur la section à laquelle il devrait appartenir, envoyez la licence sur debian-legal@lists.debian.org (http://lists.debian.org/debian-legal/) et demandez conseil ;
- le programme ne devrait certainement **pas** être exécuté setuid root, ou encore mieux, il ne devrait pas être setuid ou setgid quoi que ce soit ;
- le programme ne devrait ni être un démon, ni s'installer dans les répertoires `*/sbin`, ni ouvrir un port en tant que superutilisateur ;
- le programme résultant devrait être sous forme de binaire exécutable, les bibliothèques sont plus difficiles à gérer ;
- il devrait être bien documenté, et le code doit être compréhensible (c'est-à-dire, pas volontairement obscur) ;
- vous devriez contacter les auteurs du programme pour vérifier qu'ils soient d'accord pour la création du paquet et bienveillants envers Debian. Il est important de pouvoir consulter les auteurs à propos du programme en cas de problèmes spécifiques à celui-ci, donc n'essayez pas de créer un paquet à partir de programmes non maintenus.

Bien sûr, toutes ces remarques ne sont que des mesures de sécurité, et ont pour but de vous préserver d'utilisateurs fous de rage si vous faites une erreur dans un démon setuid… Quand vous aurez plus d'expérience dans la création de paquets, vous pourrez faire de tels paquets, mais même les développeurs les plus expérimentés consultent la liste de discussion debian-mentors@lists.debian.org (http://lists.debian.org/debian-mentors/) en cas de doute. Les personnes présentes seront heureuses de vous aider.

Pour plus d'informations à ce sujet, consultez la référence du Développeur Debian (http://www.debian.org/doc/devel-manuals#devref).

2.2 Obtenir le programme, et l'essayer

La première chose à faire est de trouver et télécharger le code source d'origine. Supposons que vous ayez déjà le fichier source pris sur la page web de l'auteur. Les sources pour les logiciels Unix libres sont d'habitude distribués au format `tar+gzip` avec l'extension `.tar.gz`,

ou au format `tar+bzip2` avec l'extension `.tar.bz2`. Elles contiennent normalement un sous-répertoire nommé *`nom_du_programme-version`* avec toutes les sources dedans.

Si la dernière version des sources est disponible dans un dépôt de gestion de version tel que Git, Subversion ou CVS, vous devez la prendre avec « `git clone` », « `cvs co` », ou « `svn co` » et la compresser vous-même au format `tar+gzip` avec l'option « `--exclude-vcs` ».

Si les sources du programme sont disponibles dans un autre format d'archive (par exemple, le programme se termine par `.Z` ou `.zip`[1]), décompressez-le avec les outils adéquats et compressez-le de nouveau.

Comme exemple, le programme nommé `gentoo` (un gestionnaire de fichiers pour X utilisant GTK+) sera utilisé.[2]

Créez un sous-répertoire dans votre répertoire personnel nommé `debian` ou `deb` ou quoi que ce soit d'adéquat (par exemple le nom du programme, `~/gentoo`, ferait l'affaire dans ce cas). Placez l'archive téléchargée dedans, et décompressez-la avec « `tar xzf gentoo-0.9.12.tar.gz` ». Assurez-vous qu'aucune erreur ne se produit, même *sans importance*, sinon des problèmes surviendront sans doute lors de la décompression sur les systèmes d'autres personnes, dont les outils pourraient ne pas gérer ces erreurs. Sur la console, vous devriez voir ce qui suit :

```
$ mkdir ~/gentoo ; cd ~/gentoo
$ wget http://www.example.org/gentoo-0.9.12.tar.gz
$ tar xvzf gentoo-0.9.12.tar.gz
$ ls -F
gentoo-0.9.12/
gentoo-0.9.12.tar.gz
```

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 généralement de fichiers nommés `README*`, `INSTALL*`, `*.lsm` ou `*.html`. Vous devez trouver les instructions pour compiler et installer correctement le programme (elles supposent très probablement que vous voulez l'installer dans le répertoire`/usr/local/bin`; ce n'est pas le cas, mais ce point sera traité plus tard en 'Installation des fichiers à leur emplacement' page 16).

Des programmes simples sont fournis avec un fichier `Makefile` et peuvent être compilés simplement avec « `make` ». Certains d'entre eux gèrent « `make check` », qui exécute des vérifications internes. L'installation dans les répertoires de destination se fait normalement avec « `make install` ».

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

1. Vous pouvez identifier le format de l'archive en utilisant la commande `file` si l'extension du fichier ne suffit pas.
2. Ce programme est déjà empaqueté. La version actuelle 0.15.3 a changé substantiellement depuis la version 0.9.12 des exemples suivants.

Sachez aussi que vous pouvez généralement utiliser « `make clean` » (ou mieux, « `make distclean` ») pour nettoyer le répertoire de compilation. Parfois, « `make uninstall` » peut être utilisé pour retirer tous les fichiers installés.

2.3 Programmes libres portables

De nombreux programmes libres sont écrits en C (http://fr.wikipedia.org/wiki/C_(langage)) et C++ (http://fr.wikipedia.org/wiki/C++). Beaucoup d'entre-eux utilisent les Autotools ou CMake pour les rendre portables sur différentes architectures. Ces outils sont utilisés pour créer `Makefile` et d'autres fichiers sources nécessaires. Ensuite, de tels programmes sont construits avec l'habituel « `make; make install` ».

Les Autotools (http://fr.wikipedia.org/wiki/Autotools) sont les outils de construction GNU, et comprennent Autoconf (http://fr.wikipedia.org/wiki/Autoconf), Automake (http://fr.wikipedia.org/wiki/Automake), Libtool (http://fr.wikipedia.org/wiki/Libtool) et gettext (http://fr.wikipedia.org/wiki/GNU_gettext). Vous pouvez remarquer de telles sources à l'aide des fichiers `configure.ac`, `Makefile.am` et `Makefile.in`.[3]

La première étape du travail des Autotools est généralement que les auteurs amont exécutent « `autoreconf -i -f` » dans les sources avant de les distribuer avec les fichiers générés.

```
configure.ac-----+-> autoreconf -+-> configure
Makefile.am -----+               |  +-> Makefile.in
src/Makefile.am -+               |  +-> src/Makefile.in
                                 |  +-> config.h.in
                       automake
                       aclocal
                       aclocal.m4
                       autoheader
```

Modifier les fichiers `configure.ac` et `Makefile.am` nécessite un peu de connaissance de `autoconf` et `automake`. Voir « `info autoconf` » et « `info automake` ».

La deuxième étape du travail des Autotools est habituellement que les utilisateurs se procurent ces sources et exécutent « `./configure && make` » dans les sources pour compiler le programme en un *`binaire`*.

```
Makefile.in -----+                +-> Makefile -----+-> make -> binaire
src/Makefile.in -+-> ./configure -+-> src/Makefile -+
config.h.in -----+                +-> config.h -----+
                           |
              config.status -+
              config.guess --+
```

3. Voir le tutoriel des Autotools (http://www.lrde.epita.fr/~adl/autotools.html) et `/usr/share/doc/autotools-dev/README.Debian.gz`.

Vous pouvez modifier plein de choses dans le fichier `Makefile` comme le répertoire d'installation par défaut en utilisant une option de commande, par exemple « `./configure --prefix=/usr` ».

Bien que ce ne soit pas nécessaire, mettre à jour `configure` et les autres fichiers avec « `autoreconf -i -f` » en tant qu'utilisateur peut améliorer la compatibilité des sources.

CMake (http://fr.wikipedia.org/wiki/CMake) est un système de construction alternatif. De telles sources peuvent être remarquées avec le fichier `CMakeLists.txt`.

2.4 Nom et version de paquet

Vous devriez commencer la création du paquet avec un répertoire source complètement propre (originel), ou 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 *`nomdepaquet-version`*.

Si le nom du programme est constitué de plusieurs mots, réduisez-le à un mot, ou faites une abréviation. Par exemple, le paquet du programme « John's little editor for X » pourrait s'appeler `johnledx`, ou `jle4x`, ou ce que vous préférez, tant qu'il reste sous une limite raisonnable, comme vingt caractères.

Vérifiez aussi la version exacte du programme (à inclure 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, tant que les numéros de version futurs apparaîtront plus grand. Bien qu'il soit préférable d'utiliser le même numéro de version qu'en amont, si la version ressemble à `09Oct23`, vous pourriez avoir besoin de la convertir au format `AAAAMMJJ`, pour la transformer en `20091023`, afin de garantir un ordre approprié pour les mises à jour avec le programme `dpkg`.[4]

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

2.5 Paquet Debian initial

Configurez les variables d'environnement de l'interpréteur de commandes Bash `$DEBEMAIL` et `$DEBFULLNAME` de tel sorte que les nombreux outils de maintenance Debian identifient vos nom et adresse électronique comme suit :

```
$ cat >>~/.bashrc <<EOF
DEBEMAIL=votre.adresse.mail@example.org
```

4. Les versions peuvent être comparée avec « `dpkg --compare-versions` *`ver1 op ver2`* ». Voir `dpkg(1)`.

```
DEBFULLNAME="Prénom Nom"
export DEBEMAIL DEBFULLNAME
EOF
```

Préparez le paquet Debian initial en invoquant la commande `dh_make` comme suit :

```
$ . ~/.bashrc
$ cd ~/gentoo/gentoo-0.9.12
$ dh_make -f ../gentoo-0.9.12.tar.gz
```

Bien sûr, remplacez le nom de fichier par celui de votre archive source d'origine.[5] Voir `dh_make(1)` pour plus de détails.

Des informations seront affichées. Il vous sera demandé 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 la première option sera sélectionnée, avec la touche « `s` ». Une fois l'information vérifiée sur l'écran, confirmez en pressant « *Entrée* ».[6]

Après cette exécution de `dh_make`, une copie de l'archive amont est créée sous `gentoo_0.9.12.orig.tar.gz` dans le répertoire parent pour permettre ensuite la création d'un paquet source Debian non natif avec `debian.tar.gz`.

```
$ cd ~/gentoo ; ls -F
gentoo-0.9.12/
gentoo-0.9.12.tar.gz
gentoo_0.9.12.orig.tar.gz
```

Veuillez remarquer deux caractéristiques dans ce nom de fichier `gentoo_0.9.12.orig.tar.gz` :

– le nom de paquet et la version sont séparés par « _ » (tiret bas) ;
– « `orig.` » est avant le « `tar.gz` ».

Vous devriez aussi remarquer que de nombreux fichiers modèles sont créés dans les sources sous le répertoire `debian`. Ce sera expliqué en 'Fichiers nécessaires dans le répertoire `debian`' page 21 et 'Autres fichiers dans le répertoire `debian`' page 37. Vous devriez aussi comprendre que l'empaquetage n'est pas un processus automatique. Vous devez modifier les sources amont pour Debian comme en 'Modification du code source' page 15. Après tout cela, vous avez besoin de construire les paquets avec la méthode appropriée comme en 'Construction du paquet'

5. Si les sources amont fournissent le répertoire `debian` et son contenu, exécutez plutôt la commande `dh_make` avec l'option `--addmissing`. Le nouveau format source `3.0 (quilt)` est assez robuste pour ne pas casser même avec ces paquets. Vous pourriez avoir besoin de mettre à jour le contenu fourni en amont pour votre paquet Debian.

6. Il y a peu de choix à ce moment : « `s` » pour « binaire simple », « `i` » pour « binaire indépendant de l'architecture », « `m` » pour « binaires multiples », « `l` » pour « bibliothèque », « `k` » pour « module de noyau », « `n` » pour « correctif de noyau » et « `b` » pour « `cdbs` ». Ce document se concentre sur l'utilisation du paquet `debhelper` avec la commande `dh`. Ce document se concentre sur l'utilisation de la nouvelle commande `dh` pour un « binaire simple » et l'effleure pour un « binaire indépendant de l'architecture » et un « binaire multiple ». Le paquet `cdbs` propose une autre infrastructure de scripts de paquets que la commande `dh` et sort du cadre de ce document.

page 49, les vérifier comme en ‘Contrôle des erreurs du paquet’ page 57, et les envoyer comme en ‘Envoi de paquet’ page 61. Toutes ces étapes seront expliquées.

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

– des paquets à binaires multiples ;
– des paquets de bibliothèque ;
– des paquets de modules du noyau ;
– des paquets de correctifs du noyau ;
– un format de fichier source n’étant ni `tar.gz`, ni `tar.bz2` ;
– une archive source avec 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.

Si vous effacez par mégarde quelques fichiers modèles en travaillant dessus, vous pouvez les retrouver en exécutant de nouveau `dh_make` avec l’option `--addmissing` dans une arborescence de paquet Debian.

La mise à jour d’un paquet existant peut devenir compliqué puisqu’il pourrait utiliser d’anciennes techniques. Veuillez vous en tenir aux cas d’empaquetage récent pour l’instant afin d’apprendre les bases. Des explications arriveront plus loin en ‘Mise à jour de paquet’ page 63.

Chapitre 3

Modification du code source

La place manque pour entrer dans *tous* les détails de modification des sources amont, mais voici quelques étapes fondamentales et problèmes qui reviennent souvent.

3.1 Configuration de quilt

Le programme `quilt` fournit la méthode fondamentale pour enregistrer les modifications du code source pour l'empaquetage Debian. Puisque de légères modifications au paramétrage par défaut sont souhaitables pour l'empaquetage Debian, configurez `~/.quiltrc` comme suit : [1]

```
d=. ; while [ ! -d "$d/debian" -a `readlink -e $d` != / ]; do d="$d/.."; done
if [ -d "$d/debian" ] && [ -z "$QUILT_PATCHES" ]; then
    # Cas d'empaquetage Debian avec $QUILT_PATCHES non configuré
    QUILT_PATCHES=debian/patches
    QUILT_PATCH_OPTS="--unified-reject-files"
    QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
    QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
    QUILT_COLORS="diff_hdr=1;32:diff_add=1;34:diff_rem=1;31:diff_hunk=1;33:di
    if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi
```

Voir `quilt(1)` et `/usr/share/doc/quilt/quilt.html` pour apprendre à utiliser `quilt`.

3.2 Correction de bogue amont

Si vous trouvez une erreur dans le fichier `Makefile` amont comme suit, où « `install: gentoo` » aurait dû être « `install: gentoo-target` » :

1. Cette configuration peut-être désactivée en démarrant la commande `quilt` comme ceci : « `quilt --quiltrc /dev/null ...` ».

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

Corrigez l'erreur et enregistrez la modification avec la commande `quilt` sous `fix-gentoo-target.patch` :[2]

```
$ mkdir debian/patches
$ quilt new fix-gentoo-target.patch
$ quilt add Makefile
```

Modifiez le fichier `Makefile` comme suit :

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

Demandez à `quilt` de créer `debian/patches/fix-gentoo-target.patch` et ajoutez sa description :

```
$ quilt refresh
$ quilt header -e
... description du correctif
```

3.3 Installation des fichiers à leur emplacement

Normalement, les programmes s'installent d'eux-mêmes dans le sous-répertoire `/usr/local`. Puisqu'il est réservé à l'usage privé de l'administrateur système (ou de l'utilisateur), les paquets Debian ne doivent pas utiliser ce répertoire, mais devraient utiliser le sous-répertoire `/usr/bin` conformément à la norme de hiérarchie des fichiers (FHS (http://www.debian.org/doc/packaging-manuals/fhs/fhs-2.3.html), `/usr/share/doc/debian-policy/fhs/fhs-2.3.html`).

Habituellement, `make(1)` est utilisé pour automatiser la construction du programme et l'exécution de « `make install` » installe les programmes directement à l'endroit voulu par la cible `install` du fichier `Makefile`. Pour permettre à Debian de fournir des paquets binaires, le système de construction installe les programmes dans l'image de l'arborescence de fichiers créé dans un répertoire temporaire plutôt que dans la destination réelle.

2. Le répertoire `debian/patches` devrait maintenant exister si vous exécutez `dh_make` comme décrit auparavant. Cet exemple de manipulation le créé seulement si vous mettez à jour le paquet existant.

Ces deux différences entre (1) l'installation normale du programme et (2) l'empaquetage Debian peuvent être abordées de façon transparente par le paquet `debhelper` à l'aide des commandes `dh_auto_configure` et `dh_auto_install` si les conditions suivantes sont vérifiées :

– le fichier `Makefile` suit les conventions GNU pour gérer la variable `$(DESTDIR)` (`/usr/share/doc/gnu-standards/standards.html#Makefile-Conventions`);
– les sources suivent la norme de hiérarchie des fichiers (« Filesystem Hierarchy Standard » ou FHS).

Les programmes qui utilisent GNU `autoconf` suivent *automatiquement* la norme GNU et leur empaquetage est presque *automatique*. Avec ceci et d'autres paramètres, le paquet `debhelper` est estimé fonctionner pour 90 % des paquets sans modification intrusive à leur système de construction. L'empaquetage n'est donc pas aussi compliqué qu'il y parait.

Si vous devez modifier le fichier `Makefile`, vous devriez vous assurer qu'il gère la variable `$(DESTDIR)`. La variable `$(DESTDIR)` n'y est pas configurée et précède chaque chemin de fichier utilisé par le programme d'installation. Le script d'empaquetage configurera `$(DESTDIR)` en tant que répertoire temporaire.

Le répertoire temporaire utilisé par la commande `dh_auto_install` est choisi en tant que `debian/`*`paquet`* pour les paquets binaires simples.[3] Le contenu du répertoire temporaire sera copié sur le système de l'utilisateur qui installera votre paquet, la seule différence est que `dpkg` placera ces fichiers dans le répertoire racine.

Gardez à l'esprit que même si le programme s'installe dans `debian/`*`paquet`*, il doit continuer à s'exécuter correctement quand il est installé dans le répertoire racine, c'est-à-dire quand il est installé à partir du paquet `.deb`. Vous ne devez donc pas laisser le système de construction coder en dur des chaînes de caractères comme `/home/moi/deb/`*`paquet-version`*`/usr/share/`*`paquet`* dans les fichiers du paquet.

Voici les parties concernées du `Makefile` de `gentoo`[4] :

```
# Emplacement des binaires lors du 'make install'
BIN     = /usr/local/bin

# Emplacement des icônes lors du 'make install'
ICONS   = /usr/local/share/gentoo
```

Les fichiers sont configurés pour s'installer sous `/usr/local/`. Changez ces chemins en :

3. Pour les paquets binaires multiples, la commande `dh_auto_install` utilise `debian/tmp` comme répertoire temporaire alors que la commande `dh_install`, à l'aide des fichiers `debian/`*`paquet-1`*`.install` et `debian/`*`paquet-2`*`.install`, séparera le contenu de `debian/tmp` dans les répertoires temporaires `debian/`*`paquet-1`* et `debian/`*`paquet-2`* pour créer les paquets binaires multiples `*.deb`.

4. Il s'agit simplement d'un exemple pour montrer à quoi le fichier `Makefile` devrait ressembler. Si le fichier `Makefile` est créé par la commande `./configure`, la bonne façon de modifier ce genre de `Makefile` est d'exécuter la commande `./configure` à partir de la commande `dh_auto_configure` avec les options par défaut y compris `--prefix=/usr`.

```
# Emplacement des binaires lors du 'make install'
BIN     = $(DESTDIR)/usr/bin

# Emplacement des icônes lors du '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 emplacements exacts des exécutables, icônes, documentation, etc., sont spécifiées dans la norme de hiérarchie des fichiers (voir `/usr/share/doc/debian-policy/fhs/`). Vous devriez la consulter et lire les sections relatives à votre paquet.

Dès lors, l'exécutable devrait être installé sous `/usr/bin` plutôt que sous `/usr/local/bin`, la page de manuel sous `/usr/share/man/man1` plutôt que sous `/usr/local/man/man1`, etc. Remarquez qu'il n'y a pas de page de manuel mentionnée dans le fichier `Makefile` de `gentoo`, mais comme la Charte Debian demande que chaque programme en ait une, il faudra en créer une plus tard et l'installer dans `/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 risquez de devoir modifier de vrais fichiers sources C pour qu'ils utilisent les emplacements corrects. Mais où, et que chercher exactement ? Vous pouvez le découvrir avec :

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

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

Modifiez ces fichiers et à ces lignes, remplacez `usr/local/lib` par `usr/lib` :

```
$ vim '+argdo %s/usr\/local\/lib/usr\/lib/gce|update' +q \
      $(find . -type f -name '*.[c|h]')
```

Soyez attentif à ne pas casser le reste du code ! :-)

Ensuite, vous devriez trouver la cible d'installation (cherchez une ligne qui commence par `install:`, d'ordinaire cela fonctionne) et renommez toutes les références aux répertoires autres que ceux définis au début du `Makefile`.

Après la correction du bogue amont, la cible « install » de `gentoo` est sous la forme :

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

Corrigez l'erreur et enregistrez la modification avec la commande `quilt` sous `debian/patches/install.patch` :

```
$ quilt new install.patch
$ quilt add Makefile
```

Pour le paquet Debian, il faut modifier comme suit avec l'éditeur :

```
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 dans la règle. Elle n'existait pas dans Le fichier `Makefile` originel parce qu'habituellement `/usr/local/bin` et les autres répertoires existent déjà sur le système dans lequel « `make install` » est exécuté. Cependant, puisque dans notre cas les répertoires d'installation sont vides (ou même inexistants), chacun de ces répertoires doit être créé.

D'autres choses peuvent être ajoutés 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
```

Après avoir soigneusement vérifié que tout est bon, demandez à `quilt` de rafraîchir le correctif pour créer `debian/patches/install.patch` et ajoutez sa description.

```
$ quilt refresh
$ quilt header -e
... description du correctif
```

Vous avez maintenant un ensemble de correctifs.

1 correction d'un bogue amont : `debian/patches/fix-gentoo-target.patch`;
2 modification spécifique à l'empaquetage Debian : `debian/patches/install.patch`;

Chaque fois que vous faites des modifications qui ne sont pas spécifiquement liées à Debian comme `debian/patches/fix-gentoo-target.patch`, envoyez-les au responsable amont pour qu'elles puissent être incluses dans la version suivante du programme et que tout le monde puisse en profiter. 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.

3.4 Bibliothèques différentes

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

Ainsi, si une ligne de `Makefile` (ou `Makefile.in`) du programme indique quelque chose comme ceci (et que le programme ne compile pas) [5] :

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

Il suffit de le corriger dans `debian/patches/ncurse.patch` en modifiant `curses` en `ncurses` :

```
$ quilt new ncurse.patch
$ quilt add Makefile
$ sed -i -e "s/-lcurses/-lncurses/g" Makefile
$ quilt refresh
$ quilt header -e
... description du correctif
```

5. NdA : ceci n'est pas le meilleur exemple dans la mesure où le paquet `libncurses` est maintenant livré avec un lien symbolique `libcurses.so`. Les suggestions sont les bienvenues :-)

Chapitre 4

Fichiers nécessaires dans le répertoire `debian`

Un nouveau sous-répertoire se trouve dans le répertoire des sources du programme, nommé `debian`. Certain fichiers de ce répertoire sont à modifier pour personnaliser le comportement du paquet. Les plus importants sont `control`, `changelog`, `copyright` et `rules`, ils sont nécessaires pour tous les paquets.

4.1 Fichier `control`

Ce fichier contient plusieurs valeurs utilisées par `dpkg`, `dselect`, `apt-get`, `apt-cache`, `aptitude` et d'autres outils de gestions de paquets pour gérer le paquet. Il est défini par la Charte Debian, 5 « Fichiers de contrôle et leurs champs » (http://www.debian.org/doc/debian-policy/ch-controlfields.html).

Le fichier `control` créé par `dh_make` ressemble à :

```
 1 Source: gentoo
 2 Section: unknown
 3 Priority: extra
 4 Maintainer: Prénom Nom <votre.adresse.mail@example.org>
 5 Build-Depends: debhelper (>= 7.0.50~)
 6 Standards-Version: 3.8.4
 7 Homepage: <URL amont, si pertinente>
 8
 9 Package: gentoo
10 Architecture: any
11 Depends: ${shlibs:Depends}, ${misc:Depends}
12 Description: <description courte de 60 caractères au maximum>
13  <description longue, indentée par des espaces>
```

(Les numéros de ligne ont été ajoutés.)

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 à laquelle ce paquet appartient.

Comme vous avez pu le constater, l'archive Debian est divisée en sections : main (logiciels libres), non-free (logiciels non libres), et contrib (logiciels libres qui dépendent de logiciels non libres). Des sous-sections logiques décrivent de manière concise le type de paquets qui s'y trouvent, par exemple admin pour les programmes destinés aux administrateurs, base pour les outils fondamentaux, devel pour les outils de programmation, doc pour la documentation, libs pour les bibliothèques, mail pour les lecteurs et les démons de courrier électronique, net pour les applications et démons de réseau, x11 pour les programmes X11 qui ne conviennent pas mieux ailleurs, et bien d'autres. Lisez la Charte Debian, 2.4 « Sections » (http://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections) et la liste des sections dans <tt>Sid</tt> (http://packages.debian.org/unstable/) pour des conseils.

Changez la section en x11 (le préfixe main/ est implicite, et peut donc être omis).

La ligne 3 décrit l'importance pour l'utilisateur d'installer ce paquet. Lisez la Charte Debian, 2.5 « Priorités » (http://www.debian.org/doc/debian-policy/ch-archive.html#s-priorities) pour des informations sur ces valeurs :

– la priorité optional fonctionne habituellement pour les nouveaux paquets qui ne sont pas en conflit avec d'autres de priorité required, important ou standard;
– la priorité extra fonctionne habituellement pour les nouveaux paquets qui sont en conflit avec d'autres de priorité non extra.

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

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

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

La cinquième ligne contient la liste des paquets nécessaires pour construire le paquet dans le champ Build-Depends. Le champ Build-Depends-Indep peut-être ajouté dans une ligne supplémentaire (voir la Charte Debian, 7.7 « Relations entre paquets sources et binaires — Build-Depends, Build-Depends-Indep, Build-Conflicts, Build-Conflicts-Indep » (http://www.debian.org/doc/debian-policy/ch-relationships.html#s-sourcebinarydeps)). Certains paquets comme gcc ou make sont implicites, puisqu'ils dépendent de build-essential. Si d'autres outils sont nécessaires pour construire le paquet, vous devez les ajouter à ces champs. Les multiples éléments sont séparées par des

virgules ; lisez ci-après les explications sur les dépendances entre binaires pour mieux comprendre la syntaxe de ces lignes :

- pour tous les paquets empaquetés avec la commande dh dans le fichier debian/rules, « debhelper (>=7.0.50~) » doit faire partie du champ Build-Depends pour être conforme à la Charte Debian au sujet de la cible clean ;
- les paquets sources de certains paquets binaires avec « Architecture: any » sont reconstruits par les empaqueteurs automatiques (« autobuilder »). Puisque la procédure des serveurs d'empaquetage automatique exécute « debian/rules build » en installant seulement les paquets indiqués dans le champ Build-Depends (voir 'Serveurs d'empaquetage automatique (« autobuilder »)' page 50), ce champ Build-Depends doit indiquer pratiquement tous les paquets nécessaires et Build-Depends-indep est rarement utilisé ;
- pour les paquets sources des seuls paquets binaires avec « Architecture: all », le champ Build-Depends-Indep peut indiquer tous les paquets nécessaires à moins qu'ils soient déjà indiqués dans le champ Build-Depends pour être conforme à la Charte Debian au sujet de la cible clean.

En cas de doute, utilisez le champ Build-Depends pour être sûr. [1]

Pour déterminer les paquets nécessaires à la construction, exécutez la commande :

```
$ dpkg-depcheck -d ./configure
```

Pour déterminer manuellement les dépendances de construction exactes pour */usr/bin/toto*, exécutez :

```
$ objdump -p /usr/bin/toto | grep NEEDED
```

et pour chaque bibliothèque affichée, par exemple libtoto.so.6, exécutez

```
$ dpkg -S libtoto.so.6
```

Ajoutez ensuite simplement la version -dev de chaque paquet dans le champ Build-Depends. Si vous utilisez ldd à cet effet, des dépendances de bibliothèque indirectes seront indiquées, introduisant un problème de dépendances de construction excessives.

gentoo a aussi besoin de xlibs-dev, libgtk1.2-dev et libgl1.2-dev pour être construit, ils doivent donc être ajoutés à côté de debhelper.

La ligne 6 est la version de la Charte Debian (http://www.debian.org/doc/devel-manuals#policy) que ce paquet respecte, celle que vous lisez quand vous créez le paquet.

1. Cette situation quelque peu étrange est une fonctionnalité bien expliquée dans la Charte Debian, note de bas de page 43 (http://www.debian.org/doc/debian-policy/footnotes.html#f43). Ce n'est pas lié à l'utilisation de la commande dh dans le fichier debian/rules mais au fonctionnement de dpkg-buildpackage. La même situation s'applique au système de construction automatique pour Ubuntu (https://bugs.launchpad.net/launchpad-buildd/+bug/238141).

En ligne 7 vous pouvez indiquer l'URL de la page d'accueil du programme amont.

La ligne 9 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 10 décrit l'architecture du CPU pour laquelle le paquet binaire peut être compilé. « any » est laissé car dpkg-gencontrol(1) trouvera la valeur appropriée pour chaque machine sur laquelle ce paquet sera compilé.

Si le paquet est indépendant d'une architecture (par exemple, un script shell ou Perl, ou un document), changez ce paramètre en « all », et lisez plus loin en 'Fichier rules' page 29 comment utiliser la règle binary-indep au lieu de binary-arch pour construire le paquet.

La ligne 11 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 précisé (voir dpkg(8), dselect(8), apt(8), aptitude(1), etc.)

Voici la signification des dépendances :

- Depends
  le paquet ne sera pas installé sans que les paquets dont il dépend ne soient installés. Utilisez-le si le programme ne s'exécute absolument pas (ou cause des dégâts sérieux) si un paquet particulier n'est pas présent ;
- Recommends
  à utiliser pour les paquets qui ne sont pas vraiment indispensables mais qui sont typiquement utilisés avec le programme. Lorsqu'un utilisateur installe le paquet, toutes les interfaces devraient proposer d'installer les paquets recommandés. aptitude et apt-get installent les paquets recommandés avec le paquet (mais l'utilisateur peut désactiver ce comportement par défaut). dpkg ignorent ce champ ;
- Suggests
  à utiliser pour les paquets qui fonctionnent bien avec le programme mais qui ne sont pas du tout indispensables. Lorsqu'un utilisateur installe le programme, toutes les interfaces devraient proposer d'installer les paquets suggérés. aptitude peut être configuré pour installer les paquets suggérés avec le paquet mais ce n'est pas le comportement par défaut. dpkg et apt-get ignore ce champ ;
- Pre-Depends
  ceci est plus fort que Depends. Le paquet ne sera pas installé tant que les paquets dont il prédépend ne soient installés et *correctement configurés*. Utilisez-le **très** rarement et seulement après en avoir discuté sur la liste de discussion debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/). Comprendre : ne l'utilisez pas du tout ; :-)
- Conflicts
  le paquet ne sera pas installé tant que les paquets avec lesquels il est en conflit n'aient été retirés. À utiliser si le 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. La liste complète est disponible dans le fichier /usr/share/doc/debian-policy/virtual-package-name-list.text.gz. À utiliser si le programme fournit une fonction d'un paquet virtuel existant ;
– Replaces
  à utiliser quand le 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 barres verticales « | » (symbole tube ou « pipe »).

Le domaine d'application des champs peut être restreint à des versions particulières de chaque paquet nommé. Ces versions sont précisées 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 >> (respectivement : strictement plus petit, plus petit ou égal, exactement égal, plus grand ou égal et strictement plus grand). Par exemple :

```
Depends: toto (>= 1.2), libbidule1 (= 1.3.4)
Conflicts: machin
Recommends: libmachin4 (>> 4.0.7)
Suggests: truc
Replaces: truc (<< 5), truc-toto (<= 7.6)
```

La dernière fonctionnalité à connaître est ${shlibs:Depends}, ${perl:Depends}, ${misc:Depends}, etc. Ces paramètres seront remplacés par une liste générée par les autres composants de debhelper quand la commande dh_gencontrol(1) est exécutée.

dh_shlibdeps(1) parcourra le paquet après sa construction et son installation dans le répertoire temporaire, afin de rechercher les exécutables et bibliothèques, déterminer leurs dépendances en bibliothèques partagées, et détecter dans quels paquets elles se trouvent (libc6 ou xlib6g par exemple). La liste des bibliothèques partagées dépendantes est utilisée pour remplacer ${shlibs:Depends}.

La liste de paquets créée par dh_perl(1) est utilisée pour remplacer ${perl:Depends}.

Certaines commandes de debhelper peuvent rendre la création de paquet dépendante d'autres paquets. Cette liste de paquets nécessaires est utilisée pour remplacer ${misc:Depends}.

Ceci dit, le champ Depends peut rester exactement comme il est maintenant, et une autre ligne avec « Suggests: file » peut être ajoutée, car gentoo peut utiliser certaines fonctionnalités fournies par le paquet file.

La ligne 12 est la description courte. La plupart des écrans sont larges de 80 colonnes, aussi cela ne devrait pas dépasser les 60 caractères. « fully GUI-configurable, two-pane X file manager » convient ici.

À la ligne 13 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.

Le champ `Vcs-*` documenté dans la référence du développeur, 6.2.5 « Emplacement du système de gestion de versions » (http://www.debian.org/doc/manuals/developers-reference/best-pkging-practices.html#bpp-vcs) peut être inséré entre les lignes 6 et 7. Considérons que le paquet `gentoo` est localisé sur le service Git d'Alioth en `git://git.debian.org/git/collab-maint/gentoo.git`.

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

```
 1 Source: gentoo
 2 Section: x11
 3 Priority: optional
 4 Maintainer: Prénom Nom <votre.adresse.mail@example.org>
 5 Build-Depends: debhelper (>= 7.0.5), xlibs-dev, libgtk1.2-dev, libglib1.2-
 6 Standards-Version: 3.8.4
 7 Vcs-Git: git://git.debian.org/git/collab-maint/gentoo.git
 8 Vcs-browser: http://git.debian.org/?p=collab-maint/gentoo.git
 9 Homepage: http://www.obsession.se/gentoo/
10
11 Package: gentoo
12 Architecture: any
13 Depends: ${shlibs:Depends}, ${misc:Depends}
14 Suggests: file
15 Description: fully GUI-configurable, two-pane X file manager
16  gentoo is a two-pane file manager for the X Window System. gentoo lets th
17  user do (almost) all of the configuration and customizing from within the
18  program itself. If you still prefer to hand-edit configuration files,
19  they're fairly easy to work with since they are written in an XML format.
20  .
21  gentoo features a fairly complex and powerful file identification system,
22  coupled to a object-oriented style system, which together give you a lot
23  of control over how files of different types are displayed and acted upon
24  Additionally, over a hundred pixmap images are available for use in file
25  type descriptions.
26  .
29  gentoo was written from scratch in ANSI C, and it utilises the GTK+ toolk
30  for its interface.
```

(Les numéros de ligne ont été ajoutés.)

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 (Charte Debian, 12.5 « Informations sur le copyright » (http://www.debian.org/doc/debian-policy/ch-docs.html#s-copyrightfile)). Vous pouvez aussi consulter la DEP-5 : debian/copyright analysable automatiquement (http://dep.debian.net/deps/dep5/).

dh_make peut proposer un modèle de fichier copyright. L'option « --copyright gpl2 » peut être utilisée ici pour obtenir un modèle de fichier pour le paquet gentoo package publié sous GPL-2.

Les choses à ajouter obligatoirement à ce fichier sont l'endroit où vous avez trouvé ce paquet, ainsi que le copyright et la licence d'exploitation réelle. Si la licence est une des licences de logiciel libre populaires comme GNU GPL-2, GNU GPL-3, LGPL-2, LGPL-3, Apache ou Artistic, vous pouvez juste faire référence au fichier approprié dans le répertoire /usr/share/common-licenses/, qui existe sur chaque système Debian. Sinon, vous devez inclure la licence en entier.

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

```
 1 Format-Specification: http://svn.debian.org/wsvn/dep/web/deps/dep5.mdwn?op
 2 Name: gentoo
 3 Maintainer: Prénom Nom <votre.adresse.mail@example.org>
 4 Source: http://sourceforge.net/projects/gentoo/files/
 5
 6 Copyright: 1998-2010 Emil Brink <emil@obsession.se>
 7 License: GPL-2+
 8
 9 Files: icons/*
10 Copyright: 1998 Johan Hanson <johan@tiq.com>
11 License: GPL-2+
12
13 Files: debian/*
14 Copyright: 2010 Prénom Nom <votre.adresse.mail@example.org>
15 License: GPL-2+
16
17 License: GPL-2+
18  This program is free software; you can redistribute it and/or modify
19  it under the terms of the GNU General Public License as published by
20  the Free Software Foundation; either version 2 of the License, or
21  (at your option) any later version.
22  .
23  This program is distributed in the hope that it will be useful,
24  but WITHOUT ANY WARRANTY; without even the implied warranty of
25  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
26  GNU General Public License for more details.
27 .
28  You should have received a copy of the GNU General Public License along
29  with this program; if not, write to the Free Software Foundation, Inc.,
30  51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
31  .
32  On Debian systems, the full text of the GNU General Public
33  License version 2 can be found in the file
34  `/usr/share/common-licenses/GPL-2'.
```

(Les numéros de ligne ont été ajoutés.)

Veuillez suivre les indications fournies par les responsables de l'archive et envoyées sur la liste debian-devel-announce : http://lists.debian.org/debian-devel-announce/2006/03/msg00023.html.

4.3 Fichier `changelog`

C'est un fichier obligatoire, avec un format spécifique décrit dans la Charte Debian, 4.4 « debian/changelog » (http://www.debian.org/doc/debian-policy/ch-source.html#s-dpkgchangelog). 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 également important, puisqu'il est bon de documenter toutes les modifications effectuées. Cela permettra aux personnes qui téléchargent le paquet de voir s'il y a des problèmes à connaître. Il sera conservé en `/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 (Closes: #nnnn)  <nnnn est le numéro de bogue de votre
4
5  -- Prénom Nom <votre.adresse.mail@example.org>  Mon, 22 Mar 2010 00:37:31
6
```

(Les numéros de ligne ont été ajoutés.)

La ligne 1 contient 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`)[2], et l'urgence ne devrait pas être changée en quoique ce soit de supérieur à « low ». :-)

2. Certaines personnes utilisent des valeurs de distribution non valables comme `UNRELEASED` pour éviter d'envoyer un paquet accidentellement lors de la mise à jour d'un paquet dans un système de gestion de version partagé.

Les lignes 3 à 5 composent l'entrée de journal, où vous documentez les modifications effectuées dans la révision du paquet (pas les modifications amont — il y a un fichier spécial pour cela, créé par les auteurs amont, que vous installerez comme /usr/share/doc/gentoo/changelog.gz). Supposons que votre rapport de bogue ITP (« Intent To Package » ou intention d'empaqueter) avait pour numéro « 12345 ». Les nouvelles lignes doivent être insérées au niveau de la première ligne qui commence avec une astérisque (« * »). Vous pouvez utiliser dch(1) pour le modifier.

Vous obtiendrez quelque chose comme :

```
1  gentoo (0.9.12-1) unstable; urgency=low
2
3   * Initial Release. Closes: #12345
4   * This is my first Debian package.
5   * Adjusted the Makefile to fix $(DESTDIR) problems.
6
7  -- Prénom Nom <votre.adresse.mail@example.org>  Mon, 22 Mar 2010 00:37:31
8
```

(Les numéros de ligne ont été ajoutés.)

Vous pouvez en apprendre plus sur la mise à jour du fichier changelog plus loin en 'Mise à jour de paquet' page 63.

4.4 Fichier rules

Il faut maintenant examiner les règles utilisées par dpkg-buildpackage(1) pour créer vraiment le paquet. Ce fichier est en fait un autre Makefile, mais différent de ceux des sources amont. Contrairement aux autres fichiers de debian, celui-ci est marqué comme exécutable.

4.4.1 Cibles du fichier rules

Tout fichier rules, comme tout autre Makefile, consiste en plusieurs cibles et leurs règles précisant comment gérer le code source. La Charte Debian, 4.9 « Script de construction principal : debian/rules » (http://www.debian.org/doc/debian-policy/ch-source.html#s-debianrules) l'explique en détail.

Voici une explication simplifiée des cibles :

- clean (obligatoire) : pour nettoyer tout les fichiers compilés, créés, et inutiles de l'arborescence de construction ;
- build (obligatoire) : pour construire les programmes compilés et les documents formatés à partir des sources dans l'arborescence de construction ;
- install (optionnelle) : pour installer les fichiers dans l'arborescence de chaque paquet binaire dans le répertoire debian. Si elles existent, les cibles binary* dépendent en réalité de cette cible.

– `binary` (obligatoire) : pour créer tous les paquets binaires (en réalité, combinaison des cibles `binary-arch` et `binary-indep`) ;[3]
– `binary-arch` (obligatoire) : pour créer tous les paquets binaires dépendants de l'architecture (`Architecture: any`) dans le répertoire parent ;[4]
– `binary-indep` (obligatoire) : pour créer tous les paquets binaires indépendants de l'architecture (`Architecture: all`) dans le répertoire parent ;[5]
– `get-orig-source` (optionnelle) : pour obtenir la dernière version du paquet source d'origine à partir du site de l'archive amont.

Les règles à exécuter sont en paramètres de ligne de commande (par exemple, « `./debian/rules build` » ou « `fakeroot make -f debian/rules binary` »). Après le nom de la cible, vous pouvez nommer la dépendance, programme ou fichier dont la cible dépend. Ensuite, il peut y avoir un certain nombre de commandes, indentées par *tab*. Une nouvelle règle commence par la déclaration de cible dès 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` donne par défaut. Vous pouvez aussi lire « `info make` » pour plus d'informations.

4.4.2 Fichier `rules` par défaut

Les versions récentes de `dh_make` créent un fichier `rules` par défaut très simple mais aussi très puissant en utilisant la commande `dh` :

```
 1 #!/usr/bin/make -f
 2 # -*- makefile -*-
 3 # Exemple de debian/rules utilisant debhelper.
 4 # Ce fichier a initialement été écrit par Joey Hess et Craig Small.
 5 # En tant qu'exception particulière, lorsque ce fichier est copié
 6 # par dh-make dans un fichier de sortie dh-make, vous pouvez
 7 # utiliser ce fichier sans restriction. Cette exception particulière
 8 # a été ajoutée par Craig Small dans la version 0.37 de dh-make.
 9
10 # Décommentez la ligne suivante pour le mode volubile.
11 #export DH_VERBOSE=1
12
13 %:
14         dh $@
```

(Les numéros de ligne ont été ajoutés. Dans le vrai fichier `rules`, le grand espace est une tabulation.)

3. Cette cible est utilisée par « `dpkg-buildpackage` » comme en 'Reconstruction complète' page 49.
4. Cette cible est utilisée par « `dpkg-buildpackage -B` » comme en 'Serveurs d'empaquetage automatique (« autobuilder »)' page 50.
5. Cette cible est utilisée par « `dpkg-buildpackage -A` ».

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 ligne 11 peut être décommentée pour configurer la variable DH_VERBOSE à 1. Dans ce cas, la commande dh indiquera quelles commandes dh_* elle exécute. Vous pouvez également ajouter ici une ligne« export DH_OPTIONS=-v ». Dans ce cas, chaque commande dh_* indiquera quelles commandes elle exécute. Ceci permet de comprendre ce qui se passe exactement derrière ce simple fichier rules, et de déboguer ses problèmes. Ce nouveau dh est un élément essentiel des outils debhelper et ne vous cache rien.

Tout le travail est réalisé aux lignes 13 et 14. Le caractère pourcent (« % ») représente toutes les cibles qui appellent alors un seul programme, dh avec le nom de la cible. [6] La commande dh est un script d'emballage qui exécute les séquences opportunes de programmes dh_* dépendant de ses paramètres : [7]

– « debian/rules clean » exécute « dh clean », qui exécute à son tour :
```
dh_testdir
dh_auto_clean
dh_clean
```
– « debian/rules build » exécute « dh build », qui exécute à son tour :
```
dh_testdir
dh_auto_configure
dh_auto_build
dh_auto_test
```
– « fakeroot debian/rules binary » exécute « fakeroot dh binary », qui exécute à son tour : [8]
```
dh_testroot
dh_prep
dh_installdirs
dh_auto_install
dh_install
dh_installdocs
dh_installchangelogs
dh_installexamples
dh_installman
dh_installcatalogs
dh_installcron
dh_installdebconf
```

6. Ceci utilise les fonctionnalités du nouveau debhelper V7. Ses concepts fondateurs sont expliqués dans la présentation « Pas le debhelper de papy » (http://joey.kitenet.net/talks/debhelper/debhelper-slides.pdf) réalisé lors de la dixième conférence Debian par l'auteur de debhelper. Sous Lenny, dh_make crée un fichier rules beaucoup plus compliqué avec de nombreux scripts dh_* énumérés pour chaque cible obligatoire explicitement et les gèle dans l'état de l'empaquetage initial. Cette nouvelle commande dh plus simple libère le responsable de cette contrainte. Vous gardez les pleins pouvoirs de personnalisation avec les cibles override_dh_*. Voir 'Personnalisation du fichier rules' page 33. Il se base uniquement sur le paquet debhelper et ne rend pas obscur le processus de construction comme le paquet cdbs.

7. Vous pouvez vérifier les réelles séquences de programmes dh_* pour une *cible* donnée avec « dh --no-act *cible* » ou « debian/rules -- '--no-act *cible*' » sans vraiment les exécuter.

8. En assumant que le paquet python-support soit installé sur le système.

```
        dh_installemacsen
        dh_installifupdown
        dh_installinfo
        dh_pysupport
        dh_installinit
        dh_installmenu
        dh_installmime
        dh_installmodules
        dh_installlogcheck
        dh_installlogrotate
        dh_installpam
        dh_installppp
        dh_installudev
        dh_installwm
        dh_installxfonts
        dh_bugfiles
        dh_lintian
        dh_gconf
        dh_icons
        dh_perl
        dh_usrlocal
        dh_link
        dh_compress
        dh_fixperms
        dh_strip
        dh_makeshlibs
        dh_shlibdeps
        dh_installdeb
        dh_gencontrol
        dh_md5sums
        dh_builddeb
```

– « `fakeroot debian/rules binary-arch` » exécute « `fakeroot dh binary-arch` », qui exécute à son tour la même séquence que « `fakeroot dh binary` » mais avec l'option « `-a` » ajoutée à chaque commande ;
– « `fakeroot debian/rules binary-indep` » exécute « `fakeroot dh binary-indep` », qui exécute à son tour à peu près la même séquence que « `fakeroot dh binary` » à l'exception de `dh_strip`, `dh_makeshlibs` et `dh_shlibdeps`, et avec l'option « `-i` » ajoutée à chaque commande restante.

Le rôle des commandes `dh_*` est presque évident d'après leur nom.[9] Les plus remarquables d'entre-elles valent la peine d'en faire ici une présentation (très) simplifiée, dans l'hypothèse d'un environnement de construction basé sur `Makefile` :[10]

9. Pour obtenir des informations complètes sur le rôle exact de tout ces scripts `dh_*` et sur leurs options, veuillez lire leur page de manuel respective et la documentation de `debhelper`.
10. Ces commandes gèrent d'autres environnements, comme `setup.py`, qui peuvent être énumérés en exécutant « `dh_auto_build --list` » dans le répertoire d'un paquet source.

– dh_auto_clean exécute normalement ce qui suit si Makefile fournit la cible distclean :[11]
```
make distclean
```
– dh_auto_configure exécute normalement ce qui suit si ./configure existe (les paramètres sont abrégés pour la lisibilité) :
```
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var ...
```
– dh_auto_build exécute normalement ce qui suit pour exécuter la première cible de Makefile si elle existe :
```
make
```
– dh_auto_test exécute normalement ce qui suit si Makefile fournit la cible test :[12]
```
make test
```
– dh_auto_install exécute normalement ce qui suit si Makefile fournit la cible install (la ligne est coupée pour la lisibilité) :
```
make install \
  DESTDIR=/chemin/vers/paquet_version-revision/debian/paquet
```

Les cibles ayant besoin de la commande fakeroot contiennent dh_testroot. Si vous ne vous faites pas passer pour le superutilisateur lors de l'utilisation de cette commande, elle se terminera en erreur.

Le plus important à propos du fichier rules créé par dh_make, c'est qu'il s'agit d'une simple suggestion. Il fonctionnera pour la plupart des paquets, mais pour les plus compliqués, vous ne devez pas hésiter à le modifier pour le faire correspondre à vos besoins. Les seules choses à ne pas pas changer sont les noms des règles, car tous les outils utilisent ces noms, conformément à la Charte.

Bien que la cible « install » ne soit pas obligatoire, elle est gérée. « fakeroot dh install » se comporte comme « fakeroot dh binary » mais s'arrête après dh_fixperms.

4.4.3 Personnalisation du fichier `rules`

Il existe plusieurs façons de personnaliser le fichier rules créé avec la nouvelle commande dh.

La commande « dh $@ » peut être personnalisée comme suit :[13]

– ajout de l'assistance de la commande dh_pysupport (la meilleure solution pour Python) :[14]
  – ajout de python-support à « Build-Depends » ;
  – utilisation normale de « dh $@ » (activé par défaut) ;

11. En fait il recherche la première cible disponible parmi distclean, realclean et clean dans Makefile et l'exécute.
12. En fait il recherche la première cible disponible parmi test et check dans Makefile et l'exécute.
13. Si un paquet installe le fichier /usr/share/perl5/Debian/Debhelper/Sequence/*nom_personnalise*.pm, vous devriez déclencher sa fonction de personnalisation avec « dh --with *nom-personnalise* $@ ».
14. L'utilisation de la commande dh_pysupport est préférable à dh_pycentral. N'utilisez pas la commande dh_python.

  - conséquence : gestion des modules Python avec la structure `python-support` ;
- ajout de l'assistance de la commande `dh_pycentral` :
  - ajout de `python-central` à « `Build-Depends` » ;
  - utilisation de « `dh --with python-central $@` » à la place ;
  - conséquence : désactivation de la commande `dh_pysupport` ;
  - conséquence : gestion des modules Python avec la structure `python-central` ;
- ajout de l'assistance de la commande `dh_installtex` :
  - ajout de `tex-common` à « `Build-Depends` » ;
  - utilisation de « `dh --with tex $@` » à la place ;
  - conséquence : enregistrement des fontes de Type 1, des modèles de césure, ou des formats avec Tex ;
- ajout de l'assistance des commandes `dh_quilt_patch` et `dh_quilt_unpatch` :
  - ajout de `quilt` à « `Build-Depends` » ;
  - utilisation de « `dh --with quilt $@` » à la place ;
  - conséquence : application et retrait des correctifs au code source amont à partir des fichiers du répertoire `debian/patches` pour les paquets source au format `1.0` ;
  - inutile pour les paquets source au format `3.0 (quilt)` ;
- ajout de l'assistance de la commande `dh_dkms` :
  - ajout de `dkms` à « `Build-Depends` » ;
  - utilisation de « `dh --with dkms $@` » à la place ;
  - conséquence : gestion correcte de l'utilisation de DKMS par le paquet de modules du noyau ;
- ajout de l'assistance des commandes `dh_autotools-dev_updateconfig` et `dh_autotools-dev_restoreconfig` :
  - ajout de `autotools-dev` à « `Build-Depends` » ;
  - utilisation de « `dh --with autotools-dev $@` » à la place ;
  - conséquence : mise à jour et restauration de `config.sub` et `config.guess` ;
- ajout de l'assistance des commandes `dh_autoreconf` et `dh_autoreconf_clean` :
  - ajout de `dh-autoreconf` à « `Build-Depends` » ;
  - utilisation de « `dh --with autoreconf $@` » à la place ;
  - conséquence : mise à jour des fichiers du système de construction GNU et restauration de ceux-ci après la construction ;
- ajout de l'assistance de la fonctionnalité de complétion de `bash` :
  - ajout de `bash-completion` à « `Build-Depends` » ;
  - utilisation de « `dh --with bash-completion $@` » à la place ;
  - conséquence : installation des complétions de `bash` en utilisant le fichier de configuration `debian/`*`paquet`*`.bash-completion`.

Pour les sources utilisant Autotools, la combinaison « `dh --with autotools-dev --with autoreconf $@` » est la plus courante avec le système de construction GNU.

De nombreuses commandes `dh_`⋆ invoquées par la nouvelle commande `dh` peuvent être personnalisées par les fichiers de configuration correspondants dans le répertoire `debian`. Voir 'Autres fichiers dans le répertoire `debian`' page 37 et la page de manuel de chaque commande pour la personnalisation de telles fonctionnalités.

Certaines commandes `dh_`⋆ invoquées par la nouvelle commande `dh` doivent parfois utili-

ser des paramètres, exécuter des commandes supplémentaires, ou être ignorées. Pour ce faire, créez une cible override_dh_*toto* avec sa règle dans le fichier rules uniquement pour la commande dh_*toto* à remplacer. Son rôle est fondamentalement de dire « *exécute-moi à la place* ».[15]

Les commandes dh_auto_* ont tendance à en faire plus que ce ce qui a été présenté (très) simplement, pour prendre en compte toutes les situations délicates. L'utilisation d'une commande équivalente simplifiée à la place de celles des cibles override_dh_* (à part pour la cible override_dh_auto_clean) est une mauvaise idée qui risque de détruire les fonctionnalités intelligentes de debhelper.

Pour conserver les données de configuration du paquet gentoo dans le répertoire /etc/gentoo plutôt que dans le répertoire consacré /etc, il suffit de remplacer l'option --sysconfig=/etc donnée par la commande dh_auto_configure à la commande ./configure avec :[16]

```
override_dh_auto_configure:
        dh_auto_configure -- --sysconfig=/etc/gentoo
```

Les options données après -- sont ajoutées aux options par défaut du programme exécuté automatiquement dans le but de les remplacer. L'utilisation de la commande dh_auto_configure est ici préférable à la commande ./configure puisqu'elle remplacera seulement l'option --sysconfig et conservera les autre options pertinentes de la commande ./configure.

Si le Makefile des sources de gentoo nécessite de préciser build comme cible pour la construction[17], vous pouvez créer une cible override_dh_auto_build pour ce faire.

```
override_dh_auto_build:
        dh_auto_build -- build
```

Ceci garanti d'exécuter $(MAKE) avec toutes les options par défaut données à la commande dh_auto_build avec en plus le paramètre build.

Si le Makefile des sources de gentoo oblige à préciser la cible packageclean pour nettoyer le paquet Debian à la place des cibles distclean ou clean dans le fichier Makefile, vous pouvez créer une cible override_dh_auto_clean pour ce faire.

```
override_dh_auto_clean:
        $(MAKE) packageclean
```

15. Avec Lenny, pour modifier le comportement d'un script dh_* il fallait trouver et adapter la ligne pertinente du fichier rules.
16. Le paquet gentoo utilise le système de construction GNU, ou Autotools. Voir http://fr.wikipedia.org/wiki/Autotools.
17. dh_auto_build sans autre option exécutera la première cible du fichier Makefile.

Si le Makefile des sources pour gentoo contient une cible test dont vous ne voulez pas pour exécuter le processus de construction du paquet Debian, vous pouvez utiliser une cible override_dh_auto_test vide pour l'omettre.

```
override_dh_auto_test:
```

Si gentoo possède un fichier journal de modification inhabituel appelé FIXES, dh_installchangelogs ne l'installera pas par défaut. La commande dh_installchangelogs a besoin de FIXES en option pour l'installer. [18]

```
override_dh_installchangelogs:
        dh_installchangelogs FIXES
```

Lors de l'utilisation de la nouvelle commande dh, la compréhension des effets exacts des cibles explicites comme celles énumérées en 'Cibles du fichier rules' page 29 (à part get-orig-source) peut être rendue difficile. Veuillez limiter les cibles explicites aux cibles override_dh_*, et à celle complètement indépendantes, si possible.

18. Les fichiers debian/changelog et debian/NEWS sont toujours installés automatiquement. Le journal de modification amont est cherché en convertissant les noms de fichiers en minuscule puis en vérifiant l'existence de changelog, changes, changelog.txt, et changes.txt.

Chapitre 5

Autres fichiers dans le répertoire `debian`

Pour contrôler la plupart des actions de `debhelper` lors de la construction du paquet, placez des fichiers de configuration optionnels dans le répertoire `debian`. Ce chapitre présente globalement le rôle de ces fichiers et leur format. Veuillez vous reporter à la Charte Debian (`http://www.debian.org/doc/devel-manuals#policy`) et à la référence du développeur Debian (`http://www.debian.org/doc/devel-manuals#devref`) pour les principes d'empaquetage.

La commande `dh_make` crée quelques modèles de fichiers de configuration dans le répertoire `debian`. La plupart d'entre-eux ont une extension « `.ex` ». Certains d'entre-eux ont le nom du paquet binaire comme préfixe : *`paquet`*. Regardez les tous.

Pour activer certains d'entre-eux, veuillez suivre les instructions suivantes :

– renommer les fichiers modèles en enlevant l'extension `.ex` ou `.EX` si elle existe ;
– modifiez le contenu des fichiers modèles selon vos besoins ;
– enlevez les fichiers modèles dont vous n'avez pas besoin ;
– si nécessaire, modifiez le fichier `control` (cf. 'Fichier `control`' page 21) ;
– si nécessaire, modifiez le fichier `rules` (cf. 'Fichier `rules`' page 29).

Les fichiers de configuration de `debhelper` sans préfixe *`paquet`* comme `install` sont relatifs au premier paquet binaire. Lorsqu'il y a plusieurs paquets binaires, leur configuration respective peut être précisée en ajoutant leur nom en préfixe du nom de fichier de configuration : *`paquet-1`*`.install`, *`paquet-2`*`.install`, etc.

5.1 Fichier `README.Debian`

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

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

```
gentoo for Debian
-----------------

<considérations éventuelles à propos de ce paquet - sinon, effacer ce fichier

 -- Prénom Nom <votre.adresse.mail@example.org>  Mon, 22 Mar 2010 00:37:31 +0
```

Si vous n'avez rien à documenter, effacez ce fichier, cf. `dh_installdocs(1)`.

5.2 Fichier `compat`

Le fichier `compat` définit le niveau de compatibilité de `debhelper`. Actuellement, vous devriez le définir à `debhelper` V7 comme suit :

```
$ echo 7 > debian/compat
```

5.3 Fichier `conffiles`

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 ses 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 gardez votre ancienne configuration ou pas.

Depuis `debhelper` V3, `dh_installdeb(1)` va *automatiquement* considérer tous les fichiers du répertoire `/etc` comme des fichiers de configuration, donc si tous les fichiers de configuration de votre programme sont dans ce répertoire, il n'est pas nécessaire de les indiquer dans ce fichier. Pour la plupart des paquets, le seul endroit (et ce devrait toujours être le cas) contenant des fichiers de configuration est `/etc`, donc ce fichier ne devrait pas exister.

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

Si le programme que vous empaquetez oblige chaque utilisateur à modifier les fichiers de configuration du répertoire `/etc`, il y a deux façons usuelles de ne pas les marquer comme fichiers de configuration pour que `dpkg` reste silencieux :

- faire un lien symbolique dans le répertoire `/etc` vers un fichier du répertoire `/var` créé par les *scripts du responsable* ;
- créer un fichier dans le répertoire `/etc` par les *scripts du responsable* ;

Pour plus d'informations sur les *scripts du responsable*, veuillez consulter 'Fichier `{post|pre}{inst|rm}` page 45.

5.4 Fichiers *paquet*.cron.*

Si votre paquet a besoin d'exécuter des tâches régulièrement pour fonctionner correctement, vous pouvez utiliser ce fichier pour les configurer. Vous pouvez soit programmer des tâches régulières horaires, quotidiennes ou mensuelles, soit des tâches qui se produiront au moment que vous préférez. Les noms de fichiers sont :

- cron.hourly — installé comme /etc/cron.hourly/*paquet* : exécuté une fois par heure, toutes les heures ;
- cron.daily — installé comme /etc/cron.daily/*paquet* : exécuté une fois par jour, normalement tôt le matin ;
- cron.weekly — installé comme /etc/cron.weekly/*paquet* : exécuté une fois par semaine, normalement tôt le dimanche matin ;
- cron.monthly — installé comme /etc/cron.monthly/*paquet* : exécuté une fois par mois, normalement tôt le matin du premier du mois ;
- cron.d — installé comme /etc/cron.d/*paquet* : exécuté à tout autre moment ;

Ces fichiers sont des scripts shell, à l'exception de *paquet*.cron.d dont le format est celui de crontab(5).

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

5.5 Fichier dirs

Ce fichier indique les répertoires nécessaires mais que la procédure d'installation normale (« make install DESTDIR=... » invoquée par « dh_auto_install ») ne crée pas. C'est souvent dû à un problème avec le Makefile.

Les fichiers énumérés dans le fichier install n'ont pas besoin que les répertoires soient créés avant, cf. 'Fichier install' page 41.

Il est préférable d'essayer d'exécuter d'abord l'installation, et d'utiliser seulement ce recours si vous avez des problèmes. Il n'y a pas de barre oblique (« / ») au début des noms de répertoire.

5.6 Fichier *paquet*.doc-base

Si votre paquet est fournit avec une autre documentation 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 la trouver avec par exemple dhelp(1), dwww(1) ou doccentral(1).

Cela inclut normalement les fichiers HTML, PS et PDF, inclus dans /usr/share/doc/nomdepaquet/.

Voici ce à quoi le fichier gentoo.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: File Management

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`, en `/usr/share/doc/doc-base/doc-base.html/index.html`.

Pour plus d'informations sur l'installation de documentation supplémentaire, voir 'Installation des fichiers à leur emplacement' page 16.

5.7 Fichier `docs`

Ce fichier spécifie les noms des fichiers de documentation que `dh_installdocs(1)` peut installer pour vous 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`, sont aussi inclus :

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

5.8 Fichier `emacsen-*`

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)`.

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

5.9 Fichier *paquet*.examples

La commande dh_installexamples(1) installe les fichiers et répertoires énumérés ici comme des fichiers d'exemple.

5.10 Fichiers *paquet*.init et *paquet*.default

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

Le fichier *paquet*.init est installé comme un script en /etc/init.d/*paquet*. Un squelette de modèle générique est fournit par la commande dh_make en init.d.ex. Vous devrez sans doute le renommer et le modifier, beaucoup, en vous assurant que les en-têtes respectent la norme de hiérarchie de fichiers (cf. /usr/share/doc/debian-policy/fhs/). Il est installé dans le répertoire temporaire par dh_installinit(1).

Le fichier *paquet*.default est installé en /etc/default/*paquet*. Ce fichier définit les valeurs par défaut des variables utilisées par le script .init. La plupart du temps, ce fichier est utilisé pour désactiver le démarrage d'un démon, paramétrer quelques drapeaux (« flags ») ou temps limites. Si le script init contient certaines fonctionnalités *paramétrables*, elles ont leur place dans ce fichier .default, pas dans le script init.

Si le programme amont possède un fichier init, vous avez le choix de l'utiliser ou non. Si vous n'utilisez pas leur script init.d, créez-en un nouveau en debian/*paquet*.init. Cependant, si le script init amont semble fonctionnel et s'installe au bon endroit, il vous faudra tout de même configurer le lien symbolique rc*. Pour ce faire il faudra remplacer dh_installinit dans le fichier rules avec les lignes suivantes :

```
override_dh_installinit:
        dh_installinit --onlyscripts
```

Si vous n'en avez pas besoin, effacez ces fichiers.

5.11 Fichier install

Si des fichiers doivent être installés dans le paquet mais que « make install » normal ne le fait pas, il faut ajouter les noms de fichiers et leur destination dans ce fichier install. Il sont installés par dh_install(1).[1] Vous devriez d'abord vérifier s'il n'y a pas un outil plus approprié à utiliser. Par exemple, les documents devraient être dans le fichier docs et non dans celui-ci.

Ce fichier install possède une ligne par fichier installé, avec le nom du fichier (par rapport au répertoire de construction principal) suivi d'un espace puis du répertoire d'installation (par

1. Cela remplace la commande obsolète dh_movefiles(1) configurée par le fichier files.

rapport au répertoire d'installation). Par exemple, si un binaire est oublié lors de l'installation, le fichier `install` ressemblera à :

```
src/toto/monbin usr/bin
```

Cela signifie que lors de l'installation du paquet, il y aura un fichier binaire `/usr/bin/monbin`.

Ce fichier `install` peut renseigner seulement le nom du fichier sans le répertoire d'installation quand le chemin relatif n'est pas modifié. Ce format est normalement utilisé pour un gros paquet qui découpe le résultat de la construction en multiples paquets binaires à l'aide de *`paquet-1`*`.install`, *`paquet-2`*`.install`, etc.

La commande `dh_install` finira par chercher les fichiers dans `debian/tmp`, s'il ne les trouve pas dans le répertoire courant (ou quelque soit l'endroit où il lui a été demandé de chercher avec `--sourcedir`).

5.12 Fichier *`paquet`*`.info`

Si le paquet possède des pages d'information, vous devriez les installer avec `dh_installinfo(1)` en les énumérant dans les fichiers *`paquet`*`.info`.

5.13 Fichiers {*`paquet`*`.|source/}lintian-overrides`

Si `lintian` signale des erreurs dans son diagnostique alors que la Charte accepte des exceptions à certaines règles, vous pouvez utiliser *`paquet`*`.lintian-overrides` ou `source/lintian-overrides` pour le rendre silencieux. Veuillez lire `/usr/share/doc/lintian/lintian.html/index.html` et vous abstenir d'en abuser.

`paquet``.lintian-overrides` concerne le paquet binaire appelé *`paquet`*, il est installé en `usr/share/lintian/overrides/`*`paquet`* par la commande `dh_lintian`.

`source/lintian-overrides` concerne le paquet source. Il n'est pas installé.

5.14 Fichiers `manpage.*`

Le programme devrait être livré avec une page de manuel. Sinon, vous devriez en créer au moins une. La commande `dh_make` crée quelques modèles de fichiers pour page de manuel. Ils doivent être copiés et modifiés pour chaque commande sans page de manuel. Assurez-vous d'avoir effacé les modèles non utilisés.

5.14.1 Fichier manpage.1.ex

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.

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

Section	Description	Notes
1	Commandes utilisateur	Commandes ou scripts exécutables.
2	Appels système	Fonctions fournies par le noyau.
3	Appels bibliothèque	Fonctions des bibliothèques système.
4	Fichiers spéciaux	D'ordinaire trouvés dans /dev.
5	Formats de fichiers	Par ex. le format /etc/password.
6	Jeux	Ou d'autres programmes frivoles.
7	Paquets de macros	Comme les macros de man.
8	Administration système	Programmes normalement exécutés par root.
9	Routines noyau	Appels non standards et routines internes.

Ainsi, la page de manuel de gentoo devrait être appelée gentoo.1. S'il n'y avait pas de page de manuel gentoo.1 dans les sources originales, il aurait fallu renommer le modèle manpage.1.ex en gentoo.1 puis le modifier à partir des informations de l'exemple et de la documentation amont.

La commande help2man peut aussi être utilisée pour créer une page de manuel à partir de la sortie de « --help » et « --version » pour chaque programme. [2]

5.14.2 Fichier manpage.sgml.ex

D'un autre côté, si vous préférez écrire du SGML plutôt que du nroff, vous pouvez utiliser le modèle manpage.sgml.ex. Si vous le faites, vous devez :

– renommer le fichier en quelque chose comme gentoo.sgml ;
– installer le paquet docbook-to-man ;
– ajouter docbook-to-man à la ligne Build-Depends dans le fichier control ;
– ajouter une cible override_dh_auto_build au fichier rules :

```
override_dh_auto_build:
        docbook-to-man debian/gentoo.sgml > debian/gentoo.1
        dh_auto_build
```

2. Si la commande n'a pas non plus de page info, mais possède des fichiers de documentation dans le répertoire /usr/share/*paquet*, vous devriez modifier manuellement la page créée par la commande help2man.

5.14.3 Fichier manpage.xml.ex

Si vous préférez XML à SGML, vous pouvez utiliser le modèle manpage.xml.ex. Si vous le faites, vous devez :

- renommer le fichier en quelque chose comme gentoo.xml ;
- installer le paquet docbook-xsl et un processeur XSLT comme xsltproc (recommandé) ;
- ajouter docbook-xsl, docbook-xml et xsltproc à la ligne Build-Depends dans le fichier control ;
- ajouter une cible override_dh_auto_build au fichier rules :

```
override_dh_auto_build:
        xsltproc --nonet \
         --param make.year.ranges 1 \
         --param make.single.year.ranges 1 \
         --param man.charmap.use.subset 0 \
         -o debian/ \
 http://docbook.sourceforge.net/release/xsl/current/manpages/docbook.xsl\
        debian/gentoo.1.xml
        dh_auto_build
```

5.15 Fichier *paquet*.manpages

Si le paquet possède des pages de manuel, vous devriez les installer avec dh_installman(1) en les énumérant dans les fichiers *paquet*.manpages.

5.16 Fichier menu

Les utilisateurs de X Window ont un gestionnaire de fenêtres avec un menu qui peut être configuré pour lancer des programmes. 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="Applications/see-menu-manual"\
        title="gentoo" command="/usr/bin/gentoo"
```

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

Le suivant est section, avec le menu et le sous-menu dans lesquels l'entrée devrait apparaître. La liste actuelle des sections[3] est dans : /usr/share/doc/debian-policy/menu-policy.html/ch2.html#s2.1.

3. Il y a eu une réorganisation majeure de la structure du menu pour Squeeze.

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 exécute le programme.

Une fois le fichier renommé en menu, l'entrée de menu peut être modifiée comme ceci :

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

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

5.17 Fichier NEWS

La commande dh_installchangelogs(1) l'installe.

5.18 Fichier {post|pre}{inst|rm}

Ces fichiers postinst, preinst, postrm, et prerm[4] sont nommés *scripts du responsable*. Ces scripts sont placés dans la zone de contrôle du paquet et sont exécutés par dpkg lorsque le paquet est installé, mis à jour ou supprimé.

En tant que responsable débutant, vous devriez éviter de modifier manuellement les *scripts du responsable* parce qu'ils ont tendance à être complexes. Pour plus d'informations, regardez dans la Charte Debian, chapitre 6 « scripts du responsable et procédure d'installation » (http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html), et examinez les fichiers d'exemple fournis par dh_make.

Si vous préférez n'en faire qu'à votre tête en personnalisant vos *scripts du responsable* pour un paquet, vous devriez les essayer non seulement lors de l'**installation** et la **mise à jour**, mais aussi pour la **désinstallation** et la **purge**.

Les mises à jour devraient être silencieuses et non intrusives (les utilisateurs existants ne devraient pas remarquer la mise à jour à part en constatant la résolution de vieux bogues et peut-être l'arrivée de nouvelles fonctionnalités).

Lorsque la mise à jour est forcément intrusive (par exemple des fichiers de configuration dispersés dans plusieurs répertoires personnels avec des structures complètement différentes), vous pouvez envisager de remettre une configuration par défaut sûre du paquet (par exemple

4. Même s'ils ont été présentés par un raccourci Bash « {post|pre}{inst|rm} », il est préférable d'utiliser un shell pur POSIX (non Bash) pour ces *scripts du responsable* pour des raisons de compatibilité.

avec service désactivé), et de fournir les documentations adéquates requises par la Charte Debian (README.Debian et NEWS.Debian) en dernier ressort. N'embêtez pas l'utilisateur avec une note debconf déclenchée par ces *scripts du responsable* lors des mises à jour.

Le paquet ucf fournit un dispositif *à la conffile* pour conserver les modifications des utilisateurs pour les fichiers qui n'ont pas été marqués *conffiles* comme ceux gérés par les *scripts du responsable*. Cela permet de réduire les problèmes associés à ces fichiers.

Ces *scripts du responsable* sont des améliorations Debian qui expliquent **pourquoi les gens choisissent Debian**. Prenez garde de ne pas les ennuyer avec ces scripts.

5.19 Fichier TODO

La commande dh_installdocs(1) l'installe.

5.20 Fichier watch

Le format du fichier watch est documenté dans la page de manuel uscan(1). Le fichier watch configure le programme uscan (dans le paquet devscripts) pour surveiller le site sur lequel vous avez obtenu les sources. C'est également utilisé par l'état de santé exterieur à Debian ou « Debian External Health Status » (DEHS) (http://wiki.debian.org/DEHS).

Voici ce qu'il contient ici :

```
# watch control file for uscan
version=3
http://sf.net/gentoo/gentoo-(.*)\.tar\.gz debian uupdate
```

Normalement, avec ce fichier watch, la page servie à l'URL « http://sf.net/gentoo » est téléchargée pour chercher des liens sous la forme « <a href=...> ». Ces liens sont comparés au nom de base (juste après la dernière « / ») « gentoo-(.*)\.tar\.gz » comme motif d'expression rationnelle Perl (cf. perlre(1)). Parmi les fichiers concordants, celui avec le plus grand numéro de version est téléchargé puis le programme uupdate est exécuté pour créer une arborescence source mise à jour à partir de ce fichier.

Bien que ce soit vrai pour les autres sites, le service de téléchargement de SourceForge en http://sf.net est une exception. Quand le fichier watch comporte une URL concordant avec le motif d'expression rationnelle « ^http://sf\.net/ », le programme uscan le remplace par « http://qa.debian.org/watch/sf.php/ » et applique ensuite cette règle. Le service de redirection en http://qa.debian.org/ est conçu pour offrir un service stable vers le fichier voulu pour un fichier watch avec « http://sf.net/*projet/nom-d-archive*-(.+)\.tar\.gz ». Le but est de résoudre le problème lié au fréquent changement d'URL pour ce domaine.

5.21 Fichier source/format

Dans le fichier debian/source/format, il devrait y avoir une seule ligne indiquant le format voulu du paquet source (vérifier en dpkg-source(1) pour une liste exhaustive). Après Squeeze, il devrait contenir selon les cas :

– 3.0 (native) pour les paquets Debian natifs ;
– 3.0 (quilt) pour tout le reste.

Le récent format source 3.0 (quilt) enregistre les modifications dans un ensemble de correctifs quilt dans debian/patches. Ces changements sont alors appliqués automatiquement lors de l'extraction du paquet source. [5] Les modifications Debian sont simplement conservées dans une archive debian.tar.gz contenant tous les fichiers du répertoire debian. Ce nouveau format permet d'inclure des fichiers binaires comme des icônes PNG sans bidouillage. [6]

Quand dpkg-source extrait un paquet source au format 3.0 (quilt), il applique automatiquement tous les correctifs énumérés dans debian/patches/series. Vous pouvez éviter que les correctifs soient appliqués à la fin de l'extraction avec l'option --skip-patches.

5.22 Fichiers patches/*

L'ancien format source 1.0 créait un unique et gros fichier diff.gz avec les fichiers de maintenance de paquet dans debian/ et les correctifs. 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 conseillé.

Le nouveau format source 3.0 (quilt) conserve les correctifs dans les fichiers debian/patches/* en utilisant la commande quilt. Ces correctifs et les autres données de paquet qui sont toutes contenues dans le répertoire debian sont empaquetées dans le fichier debian.tar.gz. Puisque la commande dpkg-source peut gérer les correctifs des sources 3.0 (quilt) sans le paquet quilt, il n'est pas nécessaire d'ajouter quilt en Build-Depends. [7]

La commande quilt est expliquée en quilt(1). Elle enregistre les modifications des sources comme une pile de fichiers correctifs -p1 dans le répertoire debian/patches et l'arborescence source n'est pas modifiée en dehors du répertoire debian. L'ordre d'application de ces correctifs est enregistré dans le fichier debian/patches/series. Vous pouvez appliquer (« push »), enlever (« pop »), et rafraîchir (« refresh ») les correctifs facilement. [8]

5. Consulter DebSrc3.0 (http://wiki.debian.org/Projects/DebSrc3.0) pour un condensé d'informations sur la conversion des sources au formats source 3.0 (quilt) et 3.0 (native).

6. En fait, ce nouveau format permet de gérer des archives amont multiples et plusieurs méthodes de compression. Ces considérations sortent du cadre de ce document.

7. Plusieurs méthodes de maintenance des ensembles de correctifs ont été proposés et sont utilisés dans les paquets Debian. Le système quilt est le système de maintenance conseillé. Les autres sont dpatch, dbs, cdbs, etc. La plupart d'entre-eux conservent aussi les correctifs dans des fichiers debian/patches/*.

8. Si vous demandez à un parrain d'envoyer le paquet, cette séparation plutôt claire et cette documentation de vos changements sont très importantes pour accélérer l'examen du paquet.

En ‘Modification du code source’ page 15, trois correctifs ont été créés dans `debian/patches`.

Puisque les correctifs Debian sont localisés en `debian/patches`, veuillez vous assurer d’avoir configuré correctement la commande `quilt` conformément à la description en ‘Configuration de quilt’ page 15.

Quand quelqu’un (éventuellement vous-même) fournit un correctif *`toto`*`.patch` par la suite, la modification d’un paquet source `3.0 (quilt)` est assez simple :

```
$ dpkg-source -x gentoo_0.9.12.dsc
$ cd gentoo-0.9.12
$ quilt import ../toto.patch
$ quilt push
$ quilt refresh
$ quilt header -e
... description du correctif
```

Les correctifs conservés au nouveau format source `3.0 (quilt)` doivent être sans approximation (« *fuzz* »). Vous devriez vous en assurer avec « `quilt pop -a; while quilt push; do quilt refresh; done` ».

Chapitre 6

Construction du paquet

Tout devrait maintenant être prêt pour construire le paquet.

6.1 Reconstruction complète

Pour faire correctement la reconstruction complète d'un paquet, doivent être installés :

- le paquet `build-essential` ;
- les paquets énumérés dans le champ `Build-Depends`, cf. 'Fichier `control`' page 21 ;
- les paquets énumérés dans le champ `Build-Depends-indep`, cf. 'Fichier `control`' page 21.

Ensuite, exécutez la commande suivante dans le répertoire des sources :

```
$ dpkg-buildpackage
```

Ceci fera tout le nécessaire pour créer entièrement les paquets binaires et source à votre place :

- nettoyage de l'arbre des sources (« `debian/rules clean` ») ;
- construction du paquet source (« `dpkg-source -b` ») ;
- construction du programme (« `debian/rules build` ») ;
- construction des paquets binaires (« `fakeroot debian/rules binary` ») ;
- signature du fichier source `.dsc`, en utilisant `gpg` ;
- création et signature du fichier de téléchargement `.changes`, en utilisant `dpkg-genchanges` et `gpg`.

Seul votre mot de passe GPG vous sera demandé, deux fois.

Une fois terminé, les fichiers suivants seront dans le répertoire parent (`~/gentoo/`) :

- `gentoo_0.9.12.orig.tar.gz`
  C'est le code source d'origine, simplement renommé pour être conforme aux normes Debian. Il a été créé au début avec « `dh_make -f ../gentoo-0.9.12.tar.gz` ».

– `gentoo_0.9.12-1.dsc`
  C'est un résumé du contenu du code source. Il est créé à partir du fichier `control`, et est utilisé pour décompresser les sources avec `dpkg-source(1)`. C'est un fichier signé avec GPG, de sorte que les gens peuvent être sûrs qu'il s'agit bien du vôtre.
– `gentoo_0.9.12-1.debian.tar.gz`
  Cette archive compressée contient le répertoire `debian`. Tous les ajouts et modifications au code source d'origine sont conservés en tant que correctifs `quilt` dans `debian/patches`. Si quelqu'un d'autre veut recréer le 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 exécuter « `dpkg-source -x gentoo_0.9.12-1.dsc` ».[1]
– `gentoo_0.9.12-1_i386.deb`
  C'est le paquet binaire terminé. Vous pouvez utiliser `dpkg` pour l'installer ou le retirer comme n'importe quel autre paquet.
– `gentoo_0.9.12-1_i386.changes`
  Ce fichier contient toutes les modifications effectuées dans la révision actuelle 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 créé à partir des fichiers `changelog` et `.dsc`. Ce fichier est signé avec 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 évoluer et de nouvelles fonctionnalités 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 diffusion debian-devel-announce@lists.debian.org (`http://lists.debian.org/debian-devel-announce/`).

Les longues chaînes de chiffres dans les fichiers `.dsc` et `.changes` sont les sommes MD5, SHA1 et SHA256 des fichiers mentionnés. Les personnes téléchargeant vos fichiers peuvent les tester avec `md5sum(1)`, `sha1sum(1)` ou `sha256sum(1)`, et si les fichiers ne correspondent pas, elles sauront que le fichier a été corrompu ou falsifié.

6.2 Serveurs d'empaquetage automatique (« autobuilder »)

Debian gère de nombreux portages (`http://www.debian.org/ports/`) avec le réseau de serveurs d'empaquetage automatique (`http://www.debian.org/devel/buildd/`) faisant fonctionner des démons `buildd` sur de nombreux ordinateurs d'architectures différentes. Même si vous n'avez pas besoin de le faire vous-même, vous devriez être au courant de ce qui va arriver à vos paquets. La suite présente brièvement comment les paquets sont reconstruits sur ces différentes architectures.[2]

1. Il est possible de ne pas appliquer les correctifs `quilt` du format source `3.0 (quilt)` à la fin de l'extraction avec l'option `--skip-patches`. Sinon, il est aussi possible d'exécuter « `quilt pop -a` » après l'extraction normale.
2. Le véritable réseau de serveurs d'empaquetage automatique a un fonctionnement autrement plus compliqué que celui présenté ici. De tels détails sortent du cadre de ce document.

Les paquets « `Architecture: any` » sont reconstruits par les serveurs d'empaquetage automatique. Seront installés :

– le paquet `build-essential` ;
– les paquets énumérés dans le champ `Build-Depends`, cf. 'Fichier `control`' page 21.

Ensuite, la commande suivante est exécutée dans le répertoire des sources :

```
$ dpkg-buildpackage -B
```

Ceci fera tout le nécessaire pour créer entièrement les paquets binaires dépendants de l'architecture sur l'architecture concernée :

– nettoyage de l'arbre des sources (« `debian/rules clean` ») ;
– construction du programme (« `debian/rules build` ») ;
– construction des paquets binaires dépendants de l'architecture (« `fakeroot debian/rules binary-arch` ») ;
– signature du fichier source `.dsc`, en utilisant `gpg` ;
– création et signature du fichier de téléchargement `.changes`, en utilisant `dpkg-genchanges` et `gpg`.

C'est pourquoi votre paquet est disponible sur plusieurs architectures.

Bien qu'il soit nécessaire d'installer les paquets énumérés dans le champ `Build-Depends-indep` pour l'empaquetage normal (cf. 'Reconstruction complète' page 49), il n'est pas nécessaire de les installer sur le serveur d'empaquetage automatique puisqu'il ne construit que les paquets binaires dépendants de l'architecture. [3] Cette différence entre l'empaquetage normal et celui des serveurs d'empaquetage automatique permet de déterminer si les paquets doivent être énumérés dans le champ `Build-Depends` ou `Build-Depends-indep` du fichier `debian/control`, cf. 'Fichier `control`' page 21.

6.3 Inclusion de `orig.tar.gz` pour l'envoi

Lors du premier envoi d'un paquet vers l'archive, il faut inclure les sources `orig.tar.gz` d'origine. Si la dernière entrée du journal de modification n'est pas la première pour cette version amont, 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.

6.4 Commande debuild

Vous pouvez automatiser encore plus le processus de construction de la commande `dpkg-buildpackage` avec la commande `debuild`, cf. `debuild(1)`.

3. contrairement à celui du paquet `pbuilder`, l'environnement `chroot` du paquet `sbuild` utilisé par les serveurs d'empaquetage automatique n'est pas forcément minimal et plusieurs paquets peuvent rester installés.

La configuration de la commande `debuild` peut être faite via `/etc/devscript.conf` ou `~/.devscript`. Les entrées suivantes sont suggérées :

```
DEBSIGN_KEYID="Votre_ID_cle_GPG"
DEBUILD_LINTIAN=yes
DEBUILD_LINTIAN_OPTS="-i -I --show-overrides"
```

Ainsi, les paquets seront signés avec votre identifiant de clé GPG (bon pour les paquets parrainés) et vérifiés avec la commande `lintian` en détail.

Le nettoyage des sources et la reconstruction du paquet avec un compte utilisateur est aussi simple que :

```
$ debuild
```

Remarquez que le paramètre « `-sa` » de `dpkg-buildpackage` pour intégrer les sources d'origine `orig.tar.gz` peut être précisé :

```
$ debuild -sa
```

Le nettoyage de l'arborescence des sources est aussi simple que :

```
$ debuild clean
```

6.5 Paquet pbuilder

Pour un environnement de construction propre (`chroot`) permettant de vérifier les dépendances de construction, `pbuilder` est très utile.[4] Cela garantit une construction propre des sources en construction automatique sous `sid` pour différentes architectures et évite l'erreur de gravité sérieuse FTBFS (« Fails To Build From Source » pour les échecs de construction à partir du paquet source), qui est toujours en catégorie RC (« Release Critical », bloquant la publication). Voir `http://buildd.debian.org/` pour plus d'informations sur le constructeur automatique de paquet Debian.

Le paquet `pbuilder` est personnalisable de la manière suivante :

- configuration du répertoire `/var/cache/pbuilder/result` accessible en écriture pour l'utilisateur ;
- création d'un répertoire, par exemple *`/var/cache/pbuilder/hooks`*, accessible en écriture pour l'utilisateur pour y placer ses scripts hook ;
- configuration de `~/.pbuilderrc` or `/etc/pbuilderrc` pour intégrer :

4. Comme le paquet `pbuilder` est en constante évolution, vous devez vérifier les possibilités réelles de la configuration en consultant la dernière documentation officielle.

```
AUTO_DEBSIGN=yes
HOOKDIR="/var/cache/pbuilder/hooks"
```

Les paquets créés seront alors signés avec votre clé GPG du répertoire `~/.gnupg/`.

Le système `chroot` local de `pbuilder` doit d'abord être initialisé :

```
$ sudo pbuilder create
```

Si vous possédez déjà les paquets sources terminés, exécutez les commande suivantes dans le répertoire contenant les fichiers *toto*`.orig.tar.gz`, *toto*`.debian.tar.gz`, et *toto*`.dsc` pour mettre à jour le système `chroot` de `pbuilder` et y construire les paquets binaires :

```
$ sudo pbuilder --update
$ sudo pbuilder --build toto.dsc
```

Remarquez que le paramètre « `-sa` » de `dpkg-buildpackage` pour intégrer les sources d'origine `orig.tar.gz` peut être précisé :

```
$ sudo pbuilder --build --debbuildopts "-sa" toto.dsc
```

Les paquets nouvellement créés sont accessibles en `/var/cache/pbuilder/result/` et n'appartiennent pas au superutilisateur.

Si vous possédez déjà l'arborescence des sources à jour, sans avoir créé les paquets sources correspondants, exécutez plutôt les commande suivantes dans le répertoire des sources avec le répertoire `debian` :

```
$ sudo pbuilder --update
$ pdebuild
```

Remarquez que le paramètre « `-sa` » de `dpkg-buildpackage` pour intégrer les sources d'origine `orig.tar.gz` peut être précisé :

```
$ pdebuild --debbuildopts "-sa"
```

Vous pouvez vous connecter à l'environnement `chroot` avec la commande « `pbuilder --login --save-after-login` » et le configurer à votre convenance. Cet environnement peut être sauvegardé en quittant l'invite de commande avec `^D` (Contrôle-D).

La dernière version de la commande `lintian` peut être exécutée dans l'environnement `chroot` en utilisant le script hook */var/cache/pbuilder/hooks/*`B90lintian` configuré comme suit : [5]

5. Il est supposé que `HOOKDIR="`*`/var/cache/pbuilder/hooks`*`"` est déjà configuré. De nombreux exemples de scripts hook sont disponibles dans le répertoire `/usr/share/doc/pbuilder/examples`.

```
#!/bin/sh
set -e
install_packages() {
    apt-get -y --force-yes install "$@"
    }
install_packages lintian
echo "+++ début de lintian +++"
su -c "lintian -i -I --show-overrides /tmp/buildd/*.changes" - pbuilder
# utilisez ceci si vous ne voulez pas que lintian arrête la construction
#su -c "lintian -i -I --show-overrides /tmp/buildd/*.changes; :" - pbuilder
echo "+++ fin de lintian +++"
```

Un environnement `sid` à jour est nécessaire pour construire correctement les paquet destinés à `sid`. En fait, `sid` peut parfois être victime de problèmes qui peuvent rendre non souhaitable la migration de votre système complet. Le paquet `pbuilder` peut vous aider à faire face à ce genre de situation.

Vous pouvez avoir besoin de mettre à jour vos paquets `stable` après sa publication à destination de `stable-proposed-updates`, `stable/updates`, etc.[6] Dans ces cas-là, « j'utilise `sid` » est une mauvaise excuse pour ne pas les mettre à jour au plus tôt. Le paquet `pbuilder` vous permet d'accéder à la plupart des distributions dérivées de Debian pour la même architecture de microprocesseur.

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

6.6 Commande git-buildpackage et similaires

Si les développeurs amont utilisent un système de gestion de version (VCS (http://www.debian.org/doc/manuals/debian-reference/ch10.fr.html#_version_control_systems)) pour maintenir leur code source, vous devriez envisager de l'utiliser. Cela rend la fusion et la sélection (« cherry-picking ») des correctifs amont plus facile. De nombreux paquets de scripts d'enrobage sont disponibles pour la construction de paquets Debian pour chaque système de gestion de version :

- `git-buildpackage` : assistants pour les paquets Debian en dépôts Git ;
- `topgit` : gestionnaire de file de correctifs Git ;
- `svn-buildpackage` : assistants pour la maintenance de paquets Debian en dépôt Subversion ;
- `cvs-buildpackage` : scripts pour paquets Debian en dépôt CVS.

Ces paquets permettent la mise en place d'un environnement de développement plus agréable que l'utilisation manuelle des commandes `quilt` pour les utilisateurs avancés afin d'*automatiser* la construction de paquets. Ils ne seront pas détaillés dans ce tutoriel.[7]

6. Il existe des restrictions pour de telles mises à jour de paquet `stable`.
7. Voici quelques ressources disponibles sur la toile pour les utilisateurs avancés :

6.7 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[8] :

```
$ fakeroot debian/rules binary
```

Ou, simplement comme ceci pour voir s'il y a construction ou non :

```
$ fakeroot debian/rules build
```

Une fois terminés vos ajustements, souvenez-vous de reconstruire en suivant la procédure correcte ci-dessus. Vous pouvez être incapable d'envoyer proprement si vous essayez d'envoyer des fichiers `.deb` construits de cette façon.

– « Construction de paquets Debian avec git-buildpackage » en `/usr/share/doc/git-buildpackage/manual-html/gbp.html` ;
– « paquets Debian avec Git (https://honk.sigxcpu.org/piki/development/debian_packages_in_git/) » ;
– « Utilisation de Git pour l'empaquetage Debian (http://www.eyrie.org/~eagle/notes/debian/git.html) » ;
– « Utilisation de TopGit pour créer un ensemble quilt pour l'empaquetage Debian (http://git.debian.org/?p=collab-maint/topgit.git;a=blob_plain;f=debian/HOWTO-tg2quilt;hb=HEAD) » ;
– « git-dpm : paquets Debian en gestionnaire Git (http://git-dpm.alioth.debian.org/) »

8. Les variables d'environnement qui devraient normalement être configurées correctement à cette étape ne sont pas configurées par cette méthode. Ne créez jamais de vrai paquets pour l'envoi avec cette méthode **rapide**.

Chapitre 7

Contrôle des erreurs du paquet

Quelques routines sont à connaître pour chercher vous-même des erreurs sur le paquet avant de l'envoyer sur des archives publiques.

Essayer sur une machine différente de la vôtre est aussi une bonne idée. Vous devez observer de près chaque alerte ou erreur pour tous les tests décrits ici.

7.1 Vérification de l'installation du paquet

Vous devez essayer votre paquet pour vérifier qu'il s'installe sans problème. La commande `debi(1)` vous permet d'essayer l'installation de tout paquet binaire créé.

```
$ sudo debi gentoo_0.9.12-1_i386.changes
```

Vous devez vérifier qu'il n'y a pas de fichier en conflit avec ceux existants dans d'autres paquets à l'aide du fichier `Contents-`*`i386`* téléchargé depuis l'archive Debian pour éviter les problèmes d'installation sur différents systèmes. La commande `apt-file` peut être pratique pour cela. S'il existe des fichiers en conflit, veuillez prendre les mesures nécessaires pour éviter les vrais problèmes avec le mécanisme d'alternatives (cf. `update-alternatives(1)`) en vous coordonnant avec les responsables de l'autre paquet concerné ou en configurant le champs « `Conflicts` » du fichier `debian/control`.

7.2 Vérification des *scripts du responsable* du paquet

Tous les *scripts du responsable*, c'est-à-dire les fichiers `preinst`, `prerm`, `postinst` et `postrm` files, ne sont pas triviaux, à moins qu'ils n'aient été générés automatiquement par les programmes `debhelper`. Ne les utilisez donc pas si vous êtes un responsable débutant, cf. 'Fichier `{post|pre}{inst|rm}` page 45.

Si le paquet utilise des *scripts du responsable* non triviaux, veuillez les essayer non seulement pour l'installation, mais aussi la suppression, la purge et la mise à niveau. De nombreux bogues dans les *scripts du responsable* surviennent lors de la suppression et de la purge. Utilisez la commande `dpkg` comme ceci pour les essayer :

```
$ sudo dpkg -r gentoo
$ sudo dpkg -P gentoo
$ sudo dpkg -i gentoo_version-revision_i386.deb
```

Les séquences suivantes devraient être essayées :

- installation de la version précédente (si elle existe) ;
- mise à niveau depuis la version précédente ;
- dégradation (« downgrade ») à la version précédente (optionnel) ;
- purge ;
- installation du nouveau paquet ;
- suppression (« remove ») du paquet.
- installation du paquet, encore.
- purge ;

Pour votre premier paquet, vous devriez créer des paquets factices avec différentes versions pour essayer votre paquet à l'avance et éviter des problèmes par la suite.

Gardez à l'esprit que si votre paquet a déjà été publié avec Debian, des gens vont mettre à jour ce paquet à partir de la version qui était dans la publication Debian précédente. Souvenez-vous d'essayer aussi les mises à jour à partir de cette version.

Même si la dégradation n'est pas officiellement gérée, il est préférable de la permettre.

7.3 Paquet lintian

Exécutez `lintian(1)` sur le fichier `.changes`. La commande `lintian` exécute de nombreux scripts de tests pour vérifier la plupart des erreurs habituelles d'empaquetage. [1]

```
$ lintian -i -I --show-overrides gentoo_0.9.12-1_i386.changes
```

Bien sûr, remplacez le nom de fichier par celui du fichier `.changes` créé pour votre paquet. La sortie de la commande `lintian` est marquée ainsi :

- `E` : pour erreur ; une violation certaine de la Charte ou erreur d'empaquetage ;
- `W` : pour avertissement ; une violation possible de la Charte ou erreur d'empaquetage ;
- `I` : pour information ; une information sur certains aspects d'empaquetage ;
- `N` : pour note ; un message détaillé pour vous aider à déboguer ;

1. Il n'est pas nécessaire d'ajouter le paramètre « `-i -I --show-overrides` » à `lintian` si vous avez personnalisé `/etc/devscripts.conf` ou `~/.devscripts` comme décrit en 'Commande debuild' page 51.

– `O` : pour ignoré ; un message ignoré par le fichier `lintian-overrides` mais affiché avec le paramètre `--show-overrides`.

Pour les avertissements, mettez au point le paquet pour les éviter ou vérifier qu'ils sont infondés. S'ils sont infondés, configurez les fichiers `lintian-overrides` comme décrit en 'Fichiers `{`*`paquet`*`.|source/}lintian-overrides` page 42.

Vous pouvez reconstruire le paquet avec `dpkg-buildpackage` et lancer `lintian` en une seule commande avec `debuild(1)` ou `pdebuild(1)`.

7.4 Commande debc

La commande `debc(1)` peut énumérer les fichiers du paquet Debian binaire.

```
$ debc paquet.changes
```

7.5 Commande debdiff

La commande `debdiff(1)` peut comparer les contenus de fichiers entre deux paquets Debian sources.

```
$ debdiff ancien-paquet.dsc nouveau-paquet.dsc
```

La commande `debdiff(1)` permet aussi de comparer les listes de fichiers entre deux ensembles de paquets Debian binaires.

```
$ debdiff ancien-paquet.changes nouveau-paquet.changes
```

Ces commandes sont utiles pour identifier ce qui a été modifié dans les paquets sources, si aucun fichier n'a été déplacé ou enlevé involontairement dans les paquets binaires, et qu'aucune autre modification n'a été faite par inadvertance lors de la mise à jour des paquets binaires.

7.6 Commande interdiff

Vous pouvez comparer deux fichiers `diff.gz` avec la commande `interdiff(1)`. C'est utile pour vérifier qu'aucune modification involontaire n'a été effectuée sur les sources par le responsable en mettant à jour les paquets à l'ancien format source `1.0`.

```
$ interdiff -z ancien-paquet.diff.gz nouveau-paquet.diff.gz
```

7.7 Commande mc

Toutes ces opérations d'inspection de fichier peuvent être transformées en un processus intuitif avec un gestionnaire de fichiers comme `mc(1)` qui vous permet de consulter non seulement le contenu des fichiers paquet `*.deb`, mais aussi les fichiers `*.udeb`, `*.debian.tar.gz`, `*.diff.gz` et `*.orig.tar.gz`.

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

Chapitre 8

Envoi de paquet

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

8.1 Envoi vers l'archive Debian

Une fois devenu un développeur Debian officiel, vous devrez envoyer le paquet sur les archives Debian. Vous pouvez le faire vous-même, mais il est plus facile d'utiliser les outils automatiques disponibles, comme `dupload(1)` ou `dput(1)`. Nous décrirons la façon de faire avec `dupload`.[1]

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

Vous pouvez lire la page de manuel `dupload.conf(5)` pour comprendre ce que chacune de ces options signifie.

L'option `$default_host` 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.

Une fois connecté à Internet, vous pouvez envoyer le paquet :

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

`dupload` vérifie que les sommes MD5, SHA1 et SHA256 des fichiers sont identiques à celles du fichier `.changes`, pour pouvoir vous avertir de reconstruire comme décrit en 'Reconstruction complète' page 49 et charger le fichier correctement.

1. Le paquet `dput` semble fournir plus de fonctionnalités, et devient plus populaire que le paquet `dupload`. Il utilise le fichier `/etc/dput` pour la configuration globale et le fichier `~/.dput.cf` pour la configuration par utilisateur. Il gère aussi les services relatifs à Ubuntu d'origine.

Si vous rencontrez un problème d'envoi à ftp://ftp.upload.debian.org/pub/UploadQueue/, vous pouvez le résoudre manuellement en envoyant un fichier `*.commands` signé avec GPG à ftp://ftp.upload.debian.org/pub/UploadQueue/ avec `ftp`.[2] Par exemple, utilisez `hello.commands` :

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

Uploader: Prénom Nom <votre.adresse.mail@example.org>
Commands:
 rm hello_1.0-1_i386.deb
 mv hello_1.0-1.dsx hello_1.0-1.dsc

-----BEGIN PGP SIGNATURE-----
Version: 2.6.3ia

iQCVAwUBNFiQSXVhJ0HiWnvJAQG58AP+IDJVeSWmDvzMUphScg1EK0mvChgnuD7h
BRiVQubXkB2DphLJW5UUSRnjw1iuFcYwH/lFpNpl7XP95LkLX3iFza9qItw4k2/q
tvylZkmIA9jxCyv/YB6zZCbHmbvUnL473eLRoxlnYZd3JFaCZMJ86B0Ph4GFNPAf
Z4jxNrgh7Bc=
=pH94
-----END PGP SIGNATURE-----
```

Voyez mentors.debian.net (http://mentors.debian.net/cgi-bin/welcome) pour un espace d'envoi accessible publiquement aux non-DD (développeurs Debian).

Voyez la référence Debian, 2.7.12 « Petite archive publique de paquets » (http://www.debian.org/doc/manuals/debian-reference/ch02.fr.html#_small_public_package_archive) pour un exemple de création de petite archive publique de paquets compatible avec le système APT moderne et sécurisé.

2. Voyez ftp://ftp.upload.debian.org/pub/UploadQueue/README. Vous pouvez aussi utiliser la commande `dcut` du paquet `dput`.

Chapitre 9

Mise à jour de paquet

Une fois un paquet publié, il sera nécessaire de le mettre à jour assez vite.

9.1 Nouvelle révision Debian

Soit un rapport de bogue numéroté `#54321`, concernant votre paquet et décrivant un problème que vous pouvez résoudre. Pour créer une nouvelle révision du paquet, vous devez :

- pour un nouveau correctif :
  - configurer le nom du correctif : « `quilt new` *`nomdubogue.patch`* » ;
  - déclarer le fichier à modifier : « `quilt add` *`fichier-bogué`* » ;
  - corriger le problème dans le paquet source pour le bogue amont ;
  - l'enregistrer en *`nomdubogue.patch`* : « `quilt refresh` » ;
  - ajouter sa description : « `quilt header -e` » ;
- pour la mise à jour d'un correctif :
  - rappeler le correctif *`toto.patch`* existant : « `quilt pop` *`toto.patch`* » ;
  - corriger le problème dans l'ancien *`toto.patch`* ;
  - mettre à jour *`toto.patch`* : « `quilt refresh` » ;
  - mettre à jour sa description : « `quilt header -e` » ;
  - appliquer tous les correctifs en enlevant les approximations (« *fuzz* ») : « `while quilt push; do quilt refresh; done` » ;
- ajouter une nouvelle révision au début 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 ;[1]
- ajoutez une courte description du bogue et de la solution dans l'entrée du changelog, suivie par « `Closes: #54321` ». De cette manière, le rapport de bogue sera automatiquement fermé par le logiciel de maintenance des archives une fois le paquet accepté dans l'archive Debian ;
- répéter les opérations précédentes pour corriger plus de bogues tout en mettant à jour le fichier `changelog` avec « `dch` » selon votre besoin ;

1. Pour obtenir la date au format voulu, utilisez « `LANG=C date -R` ».

– recommencer ce que vous aviez fait dans 'Reconstruction complète' page 49, 'Contrôle des erreurs du paquet' page 57, et 'Envoi de paquet' page 61. 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 Examen d'une nouvelle version amont

Lors de la préparation de paquets d'une nouvelle version amont pour l'archive Debian, vous devez commencer par vérifier la nouvelle version amont.

Commencez par lire les `changelog` et `NEWS` amonts, ainsi que toute autre documentation distribuée avec la nouvelle version.

Examinez ensuite les modifications entre les anciennes et nouvelles sources amont, pour guetter tout changement suspect :

```
$ diff -urN toto-anciennеversion toto-nouvelleversion
```

Les modifications de certains fichiers automatiquement créés par Autotools comme `missing`, `aclocal.m4`, `config.guess`, `config.h.in`, `config.sub`, `configure`, `depcomp`, `install-sh`, `ltmain.sh` et `Makefile.in` peuvent être ignorées. Vous pouvez les effacer avant d'exécuter `diff` pour examiner les sources.

9.3 Nouvelle version amont

Si un paquet *toto* est correctement empaqueté au nouveau format `3.0 (native)` ou `3.0 (quilt)`, empaqueter une nouvelle version amont consiste essentiellement à déplacer l'ancien répertoire `debian` dans les nouvelles sources. Ce peut être réalisé en exécutant « `tar xvzf /chemin/vers/toto_anciennеversion.debian.tar.gz` » depuis la nouvelle arborescence source décompressée. [2] Bien sûr, vous devez vous occuper de quelques routines évidentes :

– création d'une copie des sources amont dans un fichier `toto_nouvelleversion.tar.gz` ;
– mise à jour du ficher `changelog` Debian avec « `dch -v nouvelleversion-1` » :
  – ajout d'une entrée avec « New upstream release. » (nouvelle version amont) ;
  – description succincte des modifications *dans la nouvelle version amont* qui règlent des bogues et ferment les rapports associés ;
  – description succincte des modifications *à la nouvelle version amont* par le responsable qui règlent des bogues et ferment les rapports associés ;

2. Si un paquet *toto* est empaqueté avec l'ancien format `1.0`, ce peut plutôt être réalisé en exécutant « `zcat /chemin/vers/toto_anciennеversion.diff.gz|patch -p1` » depuis la nouvelle arborescence source décompressée.

– application de tous les correctifs en enlevant les approximations (« *fuzz* ») : « `while quilt push; do quilt refresh; done` ».

Si la fusion des correctifs ne s'applique pas proprement, examinez la situation (des indices sont laissés dans les fichiers `.rej`) :

– si un correctif appliqué aux sources a été intégré aux sources amont :
  – « `quilt delete` » pour l'enlever ;
– si un correctif appliqué aux sources entre en confit avec les nouvelles modifications des sources amont :
  – « quilt push -f » pour appliquer les anciens correctifs tout en forçant les rejets comme *`truc`*`.rej` ;
  – édition manuelle du fichier *`truc`* pour obtenir le résultat attendu de *`truc`*`.rej` ;
  – « `quilt refresh` » pour mettre à jour le correctif ;
– continuer jusqu'à « `while quilt push; do quilt refresh; done` ».

Cette méthode peut être automatisé avec `uupdate(1)` :

```
$ apt-get source foo
...
dpkg-source: info: extraction de toto dans toto-ancienneversion
dpkg-source: info: extraction de toto_ancienneversion.orig.tar.gz
dpkg-source: info: extraction de toto_ancienneversion-1.debian.tar.gz
$ ls -F
toto-ancienneversion/
toto_ancienneversion-1.debian.tar.gz
toto_ancienneversion-1.dsc
toto_ancienneversion.orig.tar.gz
$ wget http://example.org/toto/toto-nouvelleversion.tar.gz
$ cd toto-ancienneversion
$ uupdate -v nouvelleversion ../toto-nouvelleversion.tar.gz
$ cd ../toto-nouvelleversion
$ while quilt push; do quilt refresh; done
$ dch
... documentation des modifications réalisées
```

Si le fichier `debian/watch` est configuré comme décrit en 'Fichier `watch`' page 46, la commande `wget` est inutile. Exécutez simplement `uscan(1)` dans le répertoire *`toto-ancienneversion`* à la place de la commande `uupdate` suffit. Les sources mises à jour seront automatiquement recherchées, téléchargées, et la commande `uupdate` sera exécutée. [3]

Vous pouvez publier ces sources mises à jour en recommençant ce qui a été fait en 'Reconstruction complète' page 49, 'Contrôle des erreurs du paquet' page 57 et 'Envoi de paquet' page 61.

3. Si la commande `uscan` télécharge les sources mises à jour mais n'exécute pas la commande `uupdate`, vous devriez corriger le fichier `debian/watch` pour avoir « `debian uupdate` » après l'URL.

9.4 Mise à jour du style d'empaquetage

La mise à jour du style d'empaquetage n'est pas nécessaire lors de la mise à jour d'un paquet. Néanmoins, vous pouvez profiter de tout le potentiel du système `debhelper` moderne et du format source `3.0` en même temps : [4]

- si vous devez, pour quelque raison que ce soit, ajouter des fichiers modèles qui avaient été effacés, pour pouvez exécuter `dh_make` à nouveau depuis le répertoire des sources Debian, avec l'option `--addmissing`. Puis modifier les correctement ;
- si le paquet n'a pas été mis à jour pour utiliser la syntaxe `dh` de `debhelper` V7 dans le fichier `debian/rules`, mettez-le à jour pour utiliser `dh`. Mettez à jour le fichier `debian/control` en conséquence ;
- si vous voulez mettre à jour le fichier `rules` créé avec le mécanisme d'héritage `Makefile` du système de compilation usuel Debian (`cdbs`) vers la syntaxe `dh`, voyez `/usr/share/doc/cdbs/cdbs-doc.html` et comprenez ses variables de configuration `DEB_*` ; [5]
- si vous avez un paquet source `1.0` sans fichier *`toto`*`.diff.gz`, vous pouvez le mettre à jour au récent format source `3.0 (native)` en créant `debian/source/format` contenant « `3.0 (native)` ». Le reste des fichiers `debian/*` peut être simplement copié ;
- si vous avez un paquet source `1.0` avec fichier *`toto`*`.diff.gz`, vous pouvez le mettre à jour au récent format source `3.0 (quilt)` en créant `debian/source/format` contenant « `3.0 (quilt)` ». Le reste des fichiers `debian/*` peut être simplement copié. Importez le fichier `gros.diff` généré par la commande « `filterdiff -z -x '*/debian/*' `*`toto`*`.diff.gz > gros.diff` » dans votre système `quilt`, au besoin ; [6]
- si l'empaquetage a été créé avec un autre système de correctif comme `dpatch`, `dbs` ou `cdbs` avec `-p0`, `-p1` ou `-p2`, convertissez-le à la commande `quilt` avec `deb3` disponible en `http://bugs.debian.org/581186` ;
- si l'empaquetage a été créé avec la commande `dh` et le paramètre « `--with quilt` » ou les commandes `dh_quilt_patch` et `dh_quilt_unpatch`, enlevez ces choses et utilisez le nouveau format source `3.0 (native)`.

D'autres tâches décrites en 'Nouvelle version amont' page 64 sont aussi à effectuer.

9.5 Rappels pour la mise à jour de paquets

Voici quelques rappels pour la mise à jour de paquets :

- préservez les anciennes entrées `changelog` (cela va de soit, mais il y a déjà eu des incidents du genre « `dch` » alors qu'il aurait fallu taper « `dch -i` ») ;

4. Si votre parrain ou d'autres responsables s'opposent à la mise à jour du style d'empaquetage existant, ça ne vaut pas la peine de s'embêter à argumenter. Il y a des choses plus importantes à faire.

5. Dans le paquet `cdbs` (0.4.74), vous trouverez certaines descriptions négatives du fichier `rules` créé par la commande `dh_make` pour les choix non `cdbs`. Ne vous en inquiétez pas. C'est uniquement relatif à la version de `Lenny` qui créait des cibles explicites et une longue liste de commandes `dh_*`.

6. vous pouvez découper `gros.diff` en plusieurs petits correctifs incrémentaux avec la commande `splitdiff`.

- 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 modifié (avec un peu de chance, vous êtes au courant depuis l'inspection des modifications amont), mettez à jour les dépendances de construction `debian/rules` et `debian/control`, si besoin est ;
- vérifiez dans le système de gestion de bogues (BTS) (`http://www.debian.org/Bugs/`) que personne n'a fourni de correctifs aux bogues ouverts ;
- vérifiez le contenu du fichier `.changes` pour vous assurer que vous envoyez 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 avec GPG, etc.