Guide du nouveau responsable Debian

Josip Rodin, Osamu Aoki, Frédéric Dumont, Mohammed Adnène Trojette,
David Prévot, et les membres de la liste debian-l10n-french

INDEXATION DU DOCUMENT

	TITRE : Guide du nouveau responsable Debian	

ACTION	*NOM*	*DATE*	*SIGNATURE*
RÉDIGÉ PAR	Josip Rodin, Osamu Aoki, Frédéric Dumont, Mohammed Adnène Trojette, David Prévot, et les membres de la liste debian-l10n-french	8 octobre 2022	
traduction française		8 octobre 2022	
traduction française		8 octobre 2022	
traduction française		8 octobre 2022	
traduction française		8 octobre 2022	

SUIVI DU DOCUMENT

INDICE	DATE	MODIFICATIONS	NOM

Table des matières

Chapitre 1

Partir du bon pied

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Ce document essaie de décrire aux utilisateurs Debian moyens, et aux développeurs en devenir, la construction d'un paquet Debian. Il utilise un langage pas trop technique et est complété par des exemples, selon le vieux proverbe latin : « *Longum iter est per praecepta, breve et efficax per exempla* » (c'est long par la règle, court et efficace par l'exemple).

This document is made available for the Debian `Buster` release since this offers many translations. This document will be dropped in the following releases since contents are getting outdated. 1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. 2

Une chose est sûre, cependant : créer et maintenir correctement des paquets Debian prend 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.

Si vous avez besoin d'aide sur l'empaquetage, veuillez consulter Section 1.4.

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`.

Puisqu'il s'agit d'un tutoriel, il a été choisi d'expliquer de façon détaillée chaque étape pour certains sujets importants. Certains d'entre eux pourraient vous sembler hors sujet. Veuillez être patient. Certains cas particuliers ont été sautés, et seuls des liens ont été fournis pour conserver la simplicité de ce document.

1.1 Dynamique sociale de Debian

Voici quelques observations sur la dynamique sociale de Debian, en espérant qu'elles puissent vous préparer à interagir avec Debian :

— Nous sommes tous bénévoles.
  — Vous ne pouvez pas forcer les autres à faire quoi que ce soit.

1. Ce document suppose que vous utilisez un système `Jessie` ou plus récent. Si vous avez l'intention de suivre ce texte avec un système plus ancien (y compris un ancien système Ubuntu par exemple), vous devez au moins installer les paquets `dpkg` et `debhelper` rétroportés.

2. Vous pouvez apprendre les bases du système Debian à partir de la Référence Debian (http://www.debian.org/doc/manuals/debian-reference/) . Elle contient aussi quelques liens pour apprendre la programmation UNIX.

  - Vous devriez être motivé à faire des choses vous-même.
- La coopération amicale est la force motrice.
  - Vos contributions ne devraient pas surmener les autres.
  - Vos contributions n'ont de sens que si les autres les apprécient.
- Debian n'est pas une école où vous attirez automatiquement l'attention des professeurs.
  - Vous devriez être capable d'apprendre la plupart des choses par vous-même.
  - L'attention des autres bénévoles est une ressource très rare.
- Debian s'améliore sans cesse.
  - On s'attend à ce que vous fassiez des paquets de haute qualité.
  - Vous devrez vous adapter vous-même aux modifications.

Plusieurs sortes de personnes ont rapport à Debian avec différent rôles :

- **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** : une personne qui aide les responsables à envoyer des paquets dans l'archive officielle de paquets Debian (après en avoir vérifié le contenu) ;
- **mentor** : une personne qui aide les responsables débutants pour l'empaquetage, etc. ;
- **développeur Debian** (DD) : un membre du projet Debian avec tous les droits d'envoi vers l'archive officielle de paquets Debian ;
- **responsable Debian** (DM) : une personne avec des droits d'envoi limités vers l'archive officielle de paquets Debian.

Veuillez remarquer qu'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** à l'aide d'un **parrain**, soit comme un **responsable Debian**.

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 (consultez Section 2.2).

Puisque nous nous concentrons dans ce document exclusivement sur les aspects techniques de l'empaquetage, veuillez consulter les documents suivants pour apprendre comment Debian fonctionne et comment vous investir :

- Debian : 17 ans de logiciel libre, « do-ocracy » et démocratie (http://upsilon.cc/~zack/talks/2011/20110321-taipei.pdf) (diapositives introductives en anglais)
- Comment pouvez-vous aider Debian ? (http://www.debian.org/intro/help) (officiel)
- La FAQ Debian GNU/Linux, Chapitre 13 — Participer au projet Debian (https://www.debian.org/doc/manuals/debian-faq/-contributing.en.html) (semi-officiel)
- Wiki Debian, HelpDebian (http://wiki.debian.org/fr/HelpDebian) (supplémentaire)
- Site du nouveau membre Debian (https://nm.debian.org/) (officiel)
- FAQ de Debian Mentors (http://wiki.debian.org/DebianMentorsFaq) (supplémentaire)

1.2 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`* ou avec `dpkg -s` *`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'autres paquets, bien que non essentiels à toutes les constructions de paquet, sont utiles ou peuvent être nécessaires pour votre paquet :

— `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 (consultez `info autoconf` et `info automake`). `autotools-dev` conserve les versions à jour de certains de ces fichiers automatiques et fournit une documentation sur la meilleure façon d'utiliser ces fichiers ;

— `debhelper` et `dh-make` — `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 cela, mais sont *fortement* recommandés pour les nouveaux responsables. Ils rendent le processus complet bien plus facile à démarrer et à contrôler par la suite (consultez dh_make(8) et debhelper(1)) [3] ;

Le nouveau paquet `debmake` peut être utilisé comme alternative au paquet standard `dh-make`. Il est plus puissant et fourni avec une documentation HTML incluant des exemples complets d'empaquetage dans `debmake-doc`.

— `devscripts` — ce paquet contient des 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 (consultez `/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 (consultez fakeroot(1)) ;

— `file` — ce programme pratique peut déterminer la nature d'un fichier (consultez file(1)) ;

— `gfortran` — le compilateur Fortran 95 de GNU, nécessaire si votre programme est écrit en Fortran (consultez gfortran(1)) ;

— `git` — ce paquet fournit un système populaire de gestion de versions 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 (consultez git(1) et le manuel de Git, `/usr/share/doc/git-doc/index.html`).

— `gnupg` — un outil qui vous permet de *signer* numériquement les paquets. Cela est spécialement important si vous comptez les distribuer à d'autres personnes, et c'est certainement ce que vous ferez quand votre travail sera inclus dans la distribution Debian (consultez gpg(1)) ;

— `gpc` — le compilateur Pascal de GNU, nécessaire si votre programme est écrit en Pascal. Méritant d'être mentionné ici, `fp-compiler`, le compilateur Pascal libre, convient également (consultez gpc(1), ppc386(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 (consultez lintian(1) et le manuel utilisateur de Lintian (https://lintian.debian.org/-manual/index.html)) ;

— `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 (consultez patch(1)) ;

— `patchutils` — ce paquet contient certains utilitaires pour travailler avec les correctifs comme les commandes **lsdiff**, **interdiff** et **filterdiff** ;

— `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) (consultez pbuilder(8) et pdebuild(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 » (consultez perl(1)) ;

— `python` — Python fait aussi partie des langages de script les plus utilisés sur le système Debian, combinant une remarquable puissance et une syntaxe très claire (consultez python(1)) ;

— `quilt` — ce paquet aide à gérer un grand nombre de correctifs en gardant une trace du rôle de chacun. Les correctifs peuvent être appliqués, enlevés, rafraîchis, etc. (consultez quilt(1) et `/usr/share/doc/quilt/quilt.pdf.gz`) ;

— `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 (consultez imake(1), xmkmf(1)) ;

3. D'autres paquets plus spécialisés mais similaires existent aussi comme `dh-make-perl`, `dh-make-php`, etc.

Les courtes descriptions données ci-dessus ne servent qu'à vous présenter ce que fait chaque paquet. Avant de continuer, veuillez lire la documentation de chaque programme pertinent, y compris ceux installés par les dépendances du paquet comme **make**, au moins celle concernant l'utilisation standard. Cela peut vous sembler fastidieux maintenant, mais plus tard vous serez *très* content de l'avoir fait. Si vous avez des questions particulières par la suite, vous devriez relire les documents mentionnés ci-dessus.

1.3 Documentation nécessaire au développement

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 (http://www.debian.org/doc/packaging-manuals/fhs/fhs-3.0.html) » ou FHS, qui définit où chaque fichier et répertoire doivent 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 (consultez les copies locales de `/usr/share/doc/debian-policy/policy.pdf.gz` et `/usr/share/doc/debian-policy/fhs/fhs-3.0.pdf.gz`) ;

— `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, adopter 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. (consultez la copie locale de `/usr/share/doc/developers-reference-fr/developers-reference.pdf`) ;

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

— Le tutoriel des Autotools (http://www.lrde.epita.fr/~adl/autotools.html) fournit un très bon tutoriel pour le système de construction GNU connu sous le nom de GNU 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 (consultez les copies locales de `/usr/share/doc/gnu-standards/standards.pdf.gz` et `/usr/share/doc/gnu-standards/maintain.pdf.gz`).

Si ce document contredit n'importe quelle documentation précédente, celle-ci est prioritaire. Veuillez signaler un bogue sur le paquet `maint-guide` avec **reportbug**.

Voici un autre tutoriel qui peut être lu en parallèle de ce document :

— tutoriel d'empaquetage Debian (http://www.debian.org/doc/packaging-manuals/packaging-tutorial/packaging-tutorial) ;

1.4 Où demander de l'aide

Avant de vous décider à poser publiquement une question, veuillez lire ces excellentes documentations :

— fichiers de `/usr/share/doc/`*`paquet`* pour tous les paquets appropriés ;

— contenu de **`man`** *`commande`* pour tous les paquets appropriés ;

— contenu de **`info`** *`commande`* pour tous les paquets appropriés ;

— contenu des archives de la liste de diffusion debian-mentors@lists.debian.org (http://lists.debian.org/debian-mentors/) ;

— contenu des archives de la liste de diffusion debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/) .

Vous pouvez utiliser les moteurs de recherche web plus efficacement en indiquant des chaînes de recherche comme `site:lists.debi` pour limiter le domaine.

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 disponible ou sur la toile, vous pouvez les poser de façon interactive :

— liste de diffusion debian-mentors@lists.debian.org (http://lists.debian.org/debian-mentors/) (cette liste est pour les débutants) ;
— liste de diffusion debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/) (cette liste est pour les spécialistes) ;
— IRC (http://www.debian.org/support#irc) , par exemple `#debian-mentors` ;
— équipes se concentrant sur un ensemble particulier de paquets (liste complète à https://wiki.debian.org/Teams (https://wiki.debian.org/Teams)) ;
— listes de diffusion de certaines langues comme debian-devel-{french,italian,portuguese,spanish}@lists.debian.org ou debian-devel@debian.or.jp (liste complète à https://lists.debian.org/devel.html (https://lists.debian.org/devel.html) et https://lists.debian.org/users.html (https://lists.debian.org/users.html)).

Les responsables Debian les plus expérimentés seront heureux de vous aider, si vous demandez correctement après avoir fait les efforts nécessaires.

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 Debian, chapitre 5.8. « Manipulation des bogues » (http://www.debian.org/doc/manuals/developers-reference/pkgs.html#bug-handling) est fortement recommandée.

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 ne soit parfaitement conforme aux règles Debian (une fois encore, lisez la *vraie documentation* pour les détails). Bonne chance !

Chapitre 2

Premiers pas

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Commencez par créer votre propre paquet (ou, encore mieux, par en adopter un).

2.1 Processus de construction de paquet Debian

Si vous faites un paquet Debian à partir d'un programme amont, le processus typique de construction de paquet Debian implique de créer plusieurs fichiers au nom spécifique pour chaque étape comme suit :

— obtenir une copie du logiciel amont, généralement distribuée sous la forme d'un fichier au format tar compressé :
  — *paquet*`-`*version*`.tar.gz`
— ajouter les modifications spécifiques à Debian au programme amont dans le répertoire `debian`, et créer un paquet source non natif (c'est-à-dire, le jeu de fichiers d'entrée utilisés pour la construction du paquet Debian) au format `3.0 (quilt)` :
  — *paquet*`_`*version*`.orig.tar.gz`
  — *paquet*`_`*version*`-`*révision*`.debian.tar.gz` 1
  — *paquet*`_`*version*`-`*révision*`.dsc`
— construire les paquets binaires Debian, qui sont des fichiers de paquet installable classique au format `.deb` (ou `.udeb` pour l'installateur Debian) à partir du paquet source Debian :
  — *paquet*`_`*version*`-`*révision*`_`*arch*`.deb`

Veuillez remarquer que le caractère séparant *`paquet`* de *`version`* a été modifié : le tiret (`-`) dans le nom de l'archive source a été remplacé par un tiret bas (_) dans les noms de fichier de paquet Debian.

Dans les noms de fichier précédents, la partie *`paquet`* du nom de fichier est remplacée par le **nom du paquet**, la partie *`version`* par la **version amont**, la partie *`révision`* par la **révision Debian** et la partie *`arch`* par l'**architecture du paquet**, conformément à la Charte Debian. 2

Chaque étape de ces grandes lignes sera expliquée avec des exemples détaillés dans les sections suivantes.

1. Les paquets source Debian non natifs au style plus ancien de format `1.0` utilisent à la place *`paquet`*`_`*`version`*`-`*`révision`*`.diff.gz`.
2. Consultez 5.6.1 « Source » (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Source) , 5.6.7 « Paquet » (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Package) et 5.6.12 « Version » (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Version) . L'**architecture du paquet** doit suivre la Charte Debian, 5.6.8 « Architecture » (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Architecture) et est automatiquement attribuée par le processus de construction du paquet.

2.2 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 ce qui suit :

— la commande **aptitude** ;
— la page des paquets Debian (http://www.debian.org/distrib/packages) ;
— la page web du système de suivi de paquets Debian (https://tracker.debian.org/) .

Si le paquet existe déjà, et bien, installez-le :-). S'il se trouve qu'il est **orphelin** (c'est à dire si son responsable est Debian QA Group (http://qa.debian.org/)), vous devriez pouvoir le reprendre s'il est toujours disponible. Vous pouvez aussi adopter un paquet dont le responsable a rempli une demande d'adoption (« Request for Adoption » ou **RFA**). 3

Plusieurs ressources d'état de propriété de paquet Debian existent :

— La commande **wnpp-alert** du paquet `devscripts` ;
— liste des paquets en souffrance et paquets souhaités (http://www.debian.org/devel/wnpp/) ;
— journal de rapports de bogue Debian : bogues du pseudo-paquet `wnpp` dans `unstable` (http://bugs.debian.org/wnpp) ;
— paquets Debian en manque d'amour (http://wnpp.debian.net/) ;
— naviguer parmi les bogues `wnpp` en fonction des debtags (http://wnpp-by-tags.debian.net/) .

En remarque, il est important de souligner que Debian possède déjà des paquets pour quasiment tous les types de programme, et que le nombre de paquets déjà dans l'archive Debian est bien plus important que le nombre de personnes ayant les droits suffisants pour envoyer les mises à jour. Par conséquent, contribuer aux paquets existants déjà dans l'archive est bien plus apprécié (avec plus de chances d'être parrainé) des autres développeurs [4]. Il est possible de contribuer de plusieurs façons comme suit :

— se charger de paquets orphelins, encore largement utilisés ;
— rejoindre des équipes d'empaquetage (http://wiki.debian.org/Teams) ;
— trier des bogues de paquets populaires ;
— préparer des envois de QA ou des NMU (http://www.debian.org/doc/developers-reference/pkgs.html#nmu-qa-upload) .

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 s'est déjà occupé de la configuration initiale 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 que le programme 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 souffrance et paquets souhaités (http://www.debian.org/devel/wnpp/) . 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 ne s'occupe ;
— le logiciel **doit** avoir une licence :
  — pour la section `main` (principale), la Charte Debian exige qu'il **soit totalement conforme aux principes du logiciel libre selon Debian** (« Debian Free Software Guidelines » ou DFSG (http://www.debian.org/social_contract#guidelines)) et qu'**il ne dépende pas de paquets hors de `main`** pour la compilation ou l'exécution. 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 ;

3. Consultez la référence du Développeur Debian 5.9.5 « Adoption de paquet » (http://www.debian.org/doc/manuals/developers-reference/-pkgs.html#adopting) .
4. Cela dit, il existera toujours des paquets qui vaudront la peine d'être empaquetés.

  - pour la section `non-free` (non libre), il peut être non conforme aux 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 **pas** introduire dans Debian de problèmes relatifs à la sécurité et à la maintenance :
  - le programme 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 sont d'accord pour la création du paquet et bienveillants envers Debian. Il est important de pouvoir consulter les auteurs en cas de problèmes spécifiques au programme, n'essayez donc pas de créer un paquet à partir d'un logiciel non maintenu ;
  - le programme ne devrait certainement **pas** être exécuté setuid root, ou encore mieux, il ne devrait pas être setuid ou setgid pour quoi que ce soit ;
  - le programme ne devrait ni être un démon, ni s'installer dans un répertoire `*/sbin`, ni ouvrir un port en tant que super-utilisateur.

Bien sûr, cette dernière remarque n'est qu'une mesure de sécurité, et n'a pour but que 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 empaqueter un logiciel de ce type.

En tant que nouveau responsable vous devriez acquérir de l'expérience dans l'empaquetage de paquets plus faciles plutôt que de créer des paquets compliqués :

- paquets simples :
  - paquet binaire unique, arch = all (collection de données comme par exemple des fonds d'écran),
  - paquet binaire unique, arch = all (exécutables écrits en langage interprété comme le shell POSIX) ;
- paquet de complexité intermédiaire :
  - paquet binaire unique, arch = any (exécutables binaires ELF provenant de langages comme C ou C++),
  - plusieurs paquets binaires, arch = any + all (paquets pour exécutables binaires ELF et documentation),
  - sources amont dans un autre format que `tar.gz` ou `tar.bz2`,
  - sources amont contenant du contenu non distribuable ;
- paquets plus compliqués :
  - paquet de module interpréteur utilisé par d'autres programmes,
  - paquet de bibliothèque générique ELF utilisé par d'autres paquets,
  - plusieurs paquets binaires dont un paquet de bibliothèque ELF ;
  - paquet source avec plusieurs sources amont,
  - paquets de modules du noyau,
  - paquets de correctifs du noyau,
  - paquets avec des scripts du responsable non triviaux.

Ce n'est pas si difficile d'empaqueter des paquets plus compliqués, mais cela exige un peu plus de connaissances. Vous devriez chercher de l'aide particulière pour chaque fonctionnalité compliquée. Par exemple, certains langages ont leur propre sous-charte :

- charte Perl (http://www.debian.org/doc/packaging-manuals/perl-policy/) ;
- charte Python (http://www.debian.org/doc/packaging-manuals/python-policy/) ;
- charte Java (http://www.debian.org/doc/packaging-manuals/java-policy/) .

Un autre vieux proverbe latin s'applique : « *fabricando fit faber* » (c'est en forgeant que l'on devient forgeron). La pratique et l'expérience de toutes les étapes de l'empaquetage sont *vivement* recommandées avec des paquets simples lors de la lecture de ce tutoriel. Une archive source amont triviale comme `hello-sh-1.0.tar.gz` créée comme suit peut servir de bon point de départ : [5]

5. Ne vous inquiétez pas du `Makefile` manquant. Vous pouvez installer la commande **hello** en utilisant simplement **debhelper** comme en Section 5.11, ou en modifiant le source amont pour ajouter un nouveau `Makefile` avec la cible `install` comme en Chapitre 3.

```
$ mkdir -p hello-sh/hello-sh-1.0; cd hello-sh/hello-sh-1.0
$ cat > hello <<EOF
#!/bin/sh
# (C) 2011 Truc Bidule, GPL2+
echo ”Bonjour !”
EOF
$ chmod 755 hello
$ cd ..
$ tar -cvzf hello-sh-1.0.tar.gz hello-sh-1.0
```

2.3 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 distribuées au format **tar**+**gzip** avec l’extension `.tar.gz`, au format **tar**+**bzip2** avec l’extension `.tar.bz2` ou au format **tar**+**xz** avec l’extension `.tar.xz`. Elles contiennent normalement un sous-répertoire nommé *`paquet-version`* avec toutes les sources dedans.

Si la dernière version de la source est disponible dans un dépôt de gestion de versions tel que Git, Subversion ou CVS, vous devez l’obtenir avec `git clone`, `svn co` ou `cvs 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` [6]), vous devriez le décompresser à l’aide des outils adéquats et le recompresser.

Si le programme est distribué avec du contenu non compatible avec les principes du logiciel libre selon Debian, vous devriez aussi le décompresser pour enlever ce contenu et le recompresser avec une version amont modifiée contenant `dfsg`.

Comme exemple, le programme nommé **gentoo** (un gestionnaire de fichiers utilisant GTK+) sera utilisé. [7]

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’aucun message d’avertissement ne se produit, même *sans importance*, sinon les outils de décompression d’autres personnes pourraient ne pas gérer ces problèmes, et être incapables de les décompresser. La ligne de commande de votre interpréteur devrait ressembler à ceci :

```
$ 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 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 Section 3.3).

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

2.4 Systèmes de construction simples

Des programmes simples sont généralement fournis avec un fichier `Makefile` et peuvent être compilés en appelant simplement `make`. [8] 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`.

6. Vous pouvez identifier le format de l’archive en utilisant la commande **file** si l’extension du fichier ne suffit pas.
7. Ce programme est déjà empaqueté. La version actuelle (http://packages.qa.debian.org/g/gentoo.html) utilise Autotools comme structure de construction et est substantiellement différente des exemples suivants qui étaient basés sur la version 0.9.12.
8. Beaucoup de programmes récents sont livrés avec un script appelé `configure` qui crée un fichier `Makefile` personnalisé pour le système au moment de son exécution.

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.

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.

2.5 Systèmes de construction portables répandus

De nombreux logiciels libres sont écrits en C et C++. Beaucoup d'entre eux utilisent les Autotools ou CMake pour les rendre portables sur différentes architectures. Ces outils de construction doivent être utilisés pour créer les `Makefile` et d'autres fichiers sources nécessaires. Ensuite, de tels programmes sont construits en utilisant l'habituel `make; make install`.

Les Autotools sont les outils de construction GNU. Ils comprennent Autoconf, Automake, Libtool et gettext. Vous pouvez reconnaître de telles sources à l'aide des fichiers `configure.ac`, `Makefile.am` et `Makefile.in`. [9]

La première étape du travail des Autotools est généralement faite par les auteurs amont qui exécutent `autoreconf -i -f` dans le répertoire des sources et distribuent les fichiers créés avec les sources.

```
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**. Consultez `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 le répertoire des sources pour compiler le programme en une commande exécutable ***binaire***.

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

Vous pouvez modifier plein de choses dans le fichier `Makefile` ; vous pouvez, par exemple, modifier l'emplacement par défaut du répertoire d'installation en utilisant la commande `./configure --prefix=/usr`.

Bien que ce ne soit pas nécessaire, mettre à jour `configure` et les autres fichiers avec `autoreconf -i -f` peut améliorer la compatibilité des sources. 10

CMake est un système de construction alternatif. De tels sources peuvent être reconnus avec le fichier `CMakeLists.txt`.

2.6 Nom et version de paquet

Si le code source amont est distribué sous le nom de `gentoo-0.9.12.tar.gz`, vous pouvez prendre `gentoo` comme **nom de paquet** (source) et `0.9.12` comme **version amont**. Ces désignations seront aussi utilisées dans le fichier `debian/changelog` décrit plus loin dans Section 4.3.

9. Les Autotools sont trop importants pour être traités dans ce petit tutoriel. Cette section a seulement pour but de fournir les mots-clefs et les références. Veuillez vous assurer d'avoir lu le tutoriel des Autotools (http://www.lrde.epita.fr/~adl/autotools.html) et la copie locale de `/usr/share/doc/autotools-dev/README.Debian.gz` si vous avez besoin de les utiliser.

10. Vous pouvez automatiser ce processus en utilisant le paquet `dh-autoreconf`. Consultez Section 4.4.3.

Même si cette simple approche fonctionne la plupart du temps, vous pourriez devoir remplacer **nom de paquet** et **version amont** en renommant les sources amont afin de suivre la Charte Debian et les conventions existantes.

Le **nom de paquet** ne doit contenir que des lettres minuscules (`a-z`), des chiffres (`0-9`), les signes plus (`+`) ou moins (`-`) et des points (`.`). Il doit comporter au moins deux caractères, commencer par un caractère alphanumérique et ne doit pas être déjà utilisé. Il est préférable de garder sa longueur inférieure à trente caractères. 11

Si l'amont utilise quelques termes génériques comme `test-suite` comme nom, il vaut mieux les renommer pour identifier son contenu de façon explicite et éviter de polluer l'espace de noms. 12

La **version amont** ne devrait contenir que des caractères alphanumériques (`0-9A-Za-z`), plus (`+`), tildes (`~`) et points (`.`). Elle doit commencer par un chiffre (`0-9`). 13 Il est conseillé de garder sa longueur inférieure à huit caractères si possible. 14

Si l'amont n'utilise pas un système d'affectation de version classique comme `2.30.32` mais utilise plutôt une sorte de date comme `11Apr29`, un nom de code aléatoire ou une somme de hachage d'un système de gestion de versions dans la version, assurez-vous d'enlever ces parties de la **version amont**. Ces renseignements peuvent être enregistrés dans le fichier `debian/changelog`. Si vous devez inventer une version, utilisez le format `AAAAMMJJ`, par exemple `20110429`, comme numéro de version. Cela garantit que **dpkg** interprète correctement les futures versions comme des mises à niveau. Pour permettre des mises à niveau en douceur vers un schéma de version classique comme `0.1` dans l'avenir, utilisez plutôt la forme `0~AAMMJJ` comme `0~110429` pour la version amont.

Les chaînes de version 15 peuvent être comparées en utilisant dpkg(1) comme suit :

```
$ dpkg --compare-versions ver1 op ver2
```

La règle de comparaison de versions peut être résumée par :

— les chaînes sont comparées en commençant par le début ;
— les lettres sont plus grandes que les nombres ;
— les nombres sont comparés comme des entiers ;
— les lettres sont comparées dans l'ordre de leur code ASCII ;
— des règles particulières sont appliquées pour les points (`.`), plus (`+`) et tildes (`~`) comme suit :
`0.0` < `0.5` < `0.10` < `0.99` < `1` < `1.0~rc1` < `1.0` < `1.0+b1` < `1.0+nmu1` < `1.1` < `2.0`

Un cas délicat se produit quand une version amont `gentoo-0.9.12-ReleaseCandidate-99.tar.gz` est publiée comme une préversion de `gentoo-0.9.12.tar.gz`. Vous devez vous assurer que la mise à niveau fonctionne correctement en renommant les sources amont en `gentoo-0.9.12~rc99.tar.gz`.

2.7 Configuration de dh_make

Configurez les variables d'environnement de l'interpréteur de commandes `$DEBEMAIL` et `$DEBFULLNAME` de telle sorte que plusieurs outils de maintenance Debian identifient l'adresse électronique et le nom à utiliser pour les paquets : 16

```
$ cat >>~/.bashrc <<EOF
DEBEMAIL="your.email.address@example.org"
DEBFULLNAME="Prénom Nom"
export DEBEMAIL DEBFULLNAME
EOF
$ . ~/.bashrc
```

11. La longueur par défaut du champ de nom de paquet dans **aptitude** est de 30 caractères. Plus de 90 % des paquets ont un nom de paquet inférieur à 24 caractères.
12. Si vous suivez la référence du Développeur Debian 5.1. « Nouveaux paquets » (http://www.debian.org/doc/developers-reference/pkgs.html#newpackage) , le processus d'ITP devrait permettre de se rendre compte de ce genre de problème.
13. Cette règle plus stricte devrait permettre d'éviter toute confusion de noms de fichier.
14. La longueur par défaut du champ de version dans **aptitude** est de 10 caractères. La révision Debian précédée par un tiret en utilise au moins 2. Plus de 80 % des paquets ont une version amont inférieure à 8 caractères et une révision Debian inférieure à 2 caractères. Plus de 90 % des paquets ont une version amont inférieure à 10 caractères et une révision Debian inférieure à 3 caractères.
15. Les chaînes de version peuvent être **version amont** (*`version`*), **révision Debian** (*`révision`*) ou **version** (*`version-révision`*). Consultez Section 8.1 pour la façon dont la **révision Debian** est incrémentée.
16. Le texte qui suit présuppose que l'interpréteur de commandes que vous utilisez à la connexion est Bash. Si vous utilisez un autre interpréteur de commandes, par exemple zsh, il est nécessaire d'utiliser le fichier de configuration correspondant au lieu de `~/.bashrc`.

2.8 Paquet Debian non natif initial

Les paquets Debian normaux sont des paquets Debian non natifs réalisés à partir de programmes amont. Pour créer un paquet Debian non natif à partir d'une source amont `gentoo-0.9.12.tar.gz`, vous pouvez lui créer un paquet Debian non natif initial en appelant la commande **dh_make** comme suit :

```
$ cd ~/gentoo
$ wget http://www.example.org/gentoo-0.9.12.tar.gz
$ tar xvzf gentoo-0.9.12.tar.gz
$ cd 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. [17] Consultez dh_make(8) pour plus de précisions.

Vous devriez voir quelques questions sur le type de paquet vous voulez créer. Gentoo est un paquet binaire simple — il ne crée qu'un paquet binaire, c'est-à-dire 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`*. [18]

Cette exécution de **dh_make** crée une copie de l'archive amont en `gentoo_0.9.12.orig.tar.gz` dans le répertoire parent pour permettre ensuite la création d'un paquet source Debian non natif nommé `debian.tar.gz` plus tard :

```
$ 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 du nom de fichier `gentoo_0.9.12.orig.tar.gz` :

— le nom de paquet et la version sont séparés par le caractère tiret bas (_) ;

— la chaîne `.orig` est insérée 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 Chapitre 4 et Chapitre 5. Vous devriez aussi comprendre que l'empaquetage ne peut pas être un processus complètement automatisé. Vous aurez à modifier les sources amont pour Debian (consultez Chapitre 3). Après cela, vous devez utiliser les méthodes correctes pour construire les paquets Debian (Chapitre 6), les vérifier (Chapitre 7) et les envoyer (Chapitre 9). Toutes ces étapes seront expliquées.

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 compliquée puisqu'il pourrait utiliser d'anciennes techniques. Lors de l'apprentissage des bases, veuillez vous en tenir aux cas d'empaquetage récents ; des explications sont données plus loin en Chapitre 8.

Veuillez noter que le fichier source ne doit pas forcément contenir de système de construction comme en Section 2.4 et Section 2.5. Il peut simplement s'agir d'un ensemble de données graphiques par exemple. L'installation de fichiers peut être effectuée en utilisant juste les fichiers de configuration de `debhelper` comme `debian/install` (consultez Section 5.11).

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

18. Il y a plusieurs choix à ce moment : `s` pour un seul paquet binaire, `i` pour un paquet binaire indépendant de l'architecture, `m` pour plusieurs paquets binaires, `l` pour un paquet de bibliothèque, `k` pour un paquet de module du noyau, `n` pour un paquet de correctif du noyau et `b` pour un paquet `cdbs`. Ce document se concentre sur l'utilisation de la commande **dh** (du paquet `debhelper`) pour créer un seul paquet binaire, mais effleure aussi son utilisation pour les paquets binaires indépendants de l'architecture ou de plusieurs paquets binaires. Le paquet `cdbs` propose une autre infrastructure de scripts de paquet que la commande **dh** et sort du cadre de ce document.

Chapitre 3

Modification du code source

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

La place manque pour entrer dans *tous* les détails de modification des sources amont, mais voici quelques étapes basiques et quelques problèmes créant des difficultés.

3.1 Configuration de quilt

Le programme **quilt** fournit une méthode fondamentale pour enregistrer les modifications du code source amont pour l'empaquetage Debian. Il est utile d'avoir de légères personnalisations du paramétrage par défaut, configurez donc un alias **dquilt** pour l'empaquetage Debian en ajoutant les lignes suivantes à `~/.bashrc`. La deuxième ligne fournit les mêmes fonctionnalités de complétion de l'interpréteur pour la commande **dquilt** que pour la commande **quilt** :

```
alias dquilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
. /usr/share/bash-completion/completions/quilt
complete -F _quilt_completion -o filenames dquilt
```

Ensuite créez `~/.quiltrc-dpkg` comme suit :

```
d=. ; while [ ! -d $d/debian -a $(readlink -e $d) != / ]; do d=$d/..; done
if [ -d $d/debian ] && [ -z $QUILT_PATCHES ]; then
    # if in Debian packaging tree with unset $QUILT_PATCHES
    QUILT_PATCHES="debian/patches"
    QUILT_PATCH_OPTS="--reject-format=unified"
    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:diff_ctx=35: ↩
        diff_cctx=33"
    if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi
```

Consultez quilt(1) et `/usr/share/doc/quilt/quilt.pdf.gz` pour apprendre à utiliser **quilt**.

3.2 Correction de bogues amont

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

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

Corrigez l'erreur et enregistrez-la avec la commande **dquilt** sous `fix-gentoo-target.patch` : 1

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

Modifiez le fichier `Makefile` comme suit :

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

Demandez à **dquilt** de créer `debian/patches/fix-gentoo-target.patch` et ajoutez sa description conformément à DEP-3 : Directives pour l'étiquetage des correctifs (http://dep.debian.net/deps/dep3/) :

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

3.3 Installation des fichiers à leur emplacement

La plupart des logiciels tiers s'installent d'eux-mêmes dans le répertoire `/usr/local`. Dans Debian, il est réservé à l'usage privé de l'administrateur système, les paquets ne doivent donc pas utiliser de répertoires comme `/usr/local/bin`, mais devraient plutôt utiliser les répertoires système comme `/usr/bin`, conformément à la norme de hiérarchie des fichiers (FHS (http://www.debian.org/doc/packaging-manuals/fhs/fhs-3.0.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 (d'après la cible `install` du `Makefile`). Pour permettre à Debian de fournir des paquets installables préconstruits, il modifie le système de construction pour installer les programmes dans une image de l'arborescence de fichiers créée dans un répertoire temporaire plutôt que dans la destination réelle.

Ces deux différences entre l'installation normale du programme d'un côté et le système d'empaquetage Debian de l'autre 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 `Makefile` doit suivre les conventions GNU et gérer la variable `$(DESTDIR)` ; 2

— les sources doivent suivre la norme de hiérarchie des fichiers (« Filesystem Hierarchy Standard » ou FHS).

Les programmes qui utilisent GNU **autoconf** suivent les conventions GNU automatiquement, de telle sorte qu'ils peuvent être faciles à empaqueter. Sur cette base et d'autres paramètres, on estime que le paquet `debhelper` fonctionnera pour 90 % des paquets sans modification intrusive de leur système de construction. L'empaquetage n'est donc pas aussi compliqué qu'il n'y paraît.

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

1. Le répertoire `debian/patches` devrait maintenant exister si vous avez exécuté **dh_make** comme décrit auparavant. Cet exemple de manipulation le crée seulement si vous mettez à jour un paquet existant.
2. Consultez les normes GNU de codage : 7.2.4 DESTDIR : prise en charge des installations détournées (http://www.gnu.org/prep/standards/html_node/-DESTDIR.html#DESTDIR) .

Pour un paquet source créant un seul paquet binaire, le répertoire temporaire utilisé par la commande **dh_auto_install** sera configuré en `debian/paquet`.[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 des chemins relatifs au répertoire racine plutôt qu'au répertoire de travail.

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 à partir du paquet `.deb` sous le répertoire racine. 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 commandes exécutables lors de « make install »
BIN     = /usr/local/bin
# Emplacement des icônes lors de « make install »
ICONS   = /usr/local/share/gentoo
```

Les fichiers sont configurés pour s'installer sous `/usr/local/`. Conformément aux explications précédentes, ce répertoire est réservé pour les utilisations locales de Debian, changez donc ces chemins comme suit :

```
# Emplacement des commandes exécutables lors de « make install »
BIN     = $(DESTDIR)/usr/bin
# Emplacement des icônes lors de « make install »
ICONS   = $(DESTDIR)/usr/share/gentoo
```

Les emplacements exacts qui devraient être utilisés pour les exécutables, icônes, documentation, etc., sont décrits dans la norme de hiérarchie des fichiers (FHS). Vous devriez la consulter et lire les sections relatives à votre paquet.

Dès lors, les commandes exécutables devraient être installées 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` et ainsi de suite. Remarquez qu'il n'y a pas de page de manuel mentionnée dans le fichier `Makefile` de `gentoo`, mais comme la Charte Debian exige 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 ci-dessus. 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 --include='*.[c|h]' -e 'usr/local/lib' .
```

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 en remplaçant dans ces lignes, `usr/local/lib` par `usr/lib`. Cela peut être automatisé comme suit :

```
$ sed -i -e 's#usr/local/lib#usr/lib#g' \
       $(find . -type f -name '*.[c|h]')
```

Afin de confirmer toutes les substitutions, vous pouvez procéder de façon interactive comme suit :

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

Ensuite, vous devriez trouver la cible `install` (chercher la ligne qui commence par `install:` fonctionne en général) et renommez toutes les références aux répertoires autres que ceux définis au début du `Makefile`.

À l'origine, la cible `install` de `gentoo` était de la forme :

3. Pour un paquet source créant plusieurs paquets binaires, 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 `paquet-1_*.deb` et `paquet-2_*.deb`

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

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

Corrigez ce bogue amont et enregistrez la modification avec la commande **dquilt** sous `debian/patches/install.patch` :

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

Dans votre éditeur, modifiez cela pour le paquet Debian comme suit :

```
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 remarqué qu'il y a maintenant une commande `install -d` avant les autres dans la règle. Elle n'existait pas dans le `Makefile` d'origine 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 l'installation se fait dans une arborescence spécifique nouvellement créée, chacun de ces répertoires doit être créé.

D'autres choses peuvent être ajoutées 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
```

Vérifiez soigneusement, et si tout est bon, demandez à **dquilt** de créer le correctif `debian/patches/install.patch` et ajoutez sa description :

```
$ dquilt refresh
$ dquilt 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 spécifiques à Debian comme `debian/patches/fix-gentoo-targe patch`, envoyez-les au responsable amont pour qu'elles puissent être intégrées dans la version suivante du programme et que tout le monde puisse en profiter. Évitez aussi de faire des corrections spécifiques à Debian ou Linux — ou même UNIX ! 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 : les 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 le système Debian. Dans ce cas, remplacez-la par une bibliothèque qui existe dans Debian, et qui sert à la même chose.

Supposons qu'une ligne de `Makefile` (ou `Makefile.in`) du programme comme suit :

```
LIBS = -ltruc -lbidule
```

Si votre programme ne compile pas depuis que la bibliothèque `truc` n'existe plus et que son équivalent est fourni par la bibliothèque `truc2` sur le système Debian, vous pouvez corriger ce problème de construction en `debian/patches/truc2.patch` en modifiant `truc` en `truc2` : [5]

```
$ dquilt new truc2.patch
$ dquilt add Makefile
$ sed -i -e 's/-ltruc/-ltruc2/g' Makefile
$ dquilt refresh
$ dquilt header -e
... description du correctif
```

5. Si des modifications de l'API existent entre les bibliothèques `truc` et `truc2`, les modifications nécessaires au code source doivent être faites pour correspondre à la nouvelle API.

Chapitre 4

Fichiers nécessaires dans le répertoire `debian`

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Un nouveau sous-répertoire se trouve dans le répertoire des sources du programme, nommé `debian`. Certains 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. 1

4.1 `control`

Ce fichier contient plusieurs valeurs utilisées par **dpkg**, **dselect**, **apt-get**, **apt-cache**, **aptitude** et d'autres outils de gestion de paquets pour gérer le paquet. Cela 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 (>=10)
 6 Standards-Version: 3.9.4
 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 7 sont les informations de contrôle pour le paquet source. Les lignes 9 à 13 sont les informations de contrôle pour le paquet binaire.

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 plusieurs parties : `main` (logiciels libres), `non-free` (logiciels non libres), et `contrib` (logiciels libres qui dépendent de logiciels non libres). Chacune d'entre elles est divisée en sections qui

1. Dans ce chapitre, le préfixe `debian/` est omis pour simplifier l'écriture des fichiers du répertoire `debian` quand la signification n'est pas ambiguë.

classent les paquets en catégories grossières. Entre autres existent `admin` pour les programmes destinés aux administrateurs, `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. 2

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 : 3

— la priorité `optional` fonctionne habituellement pour les nouveaux paquets qui ne sont pas en conflit avec d'autres se réclamant de priorité `required`, `important` ou `standard`.

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 distribuer les courriers relatifs aux bogues. Évitez d'utiliser les virgules, esperluettes (&) ou parenthèses.

La ligne 5 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. 4 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 séparés par des virgules ; lisez ci-après les explications sur les dépendances entre paquets binaires pour mieux comprendre la syntaxe de ces lignes :

— pour tous les paquets empaquetés avec la commande **dh** dans le fichier `debian/rules`, `debhelper (>=9)` doit faire partie du champ `Build-Depends` pour être conforme à la Charte Debian au sujet de la cible `clean` ;

— les paquets source de paquets binaires avec `Architecture: any` sont reconstruits par les empaqueteurs automatiques (« autobuilder »). Puisque la procédure des serveurs d'empaquetage automatique installe seulement les paquets indiqués dans le champ `Build-Depends` avant d'exécuter `debian/rules build` (consultez Section 6.2), ce champ `Build-Depends` doit indiquer pratiquement tous les paquets nécessaires et `Build-Depends-Indep` est rarement utilisé ;

— pour les paquets source de paquets binaires dont tous sont `Architecture: all`, le champ `Build-Depends-Indep` peut indiquer tous les paquets nécessaires à moins qu'ils ne 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. 5

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
```

2. Consultez la Charte Debian, 2.4 « Sections » (http://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections) et la liste des sections dans `sid` (http://packages.debian.org/unstable/) .

3. Consultez la Charte Debian, 2.5 « Priorités » (http://www.debian.org/doc/debian-policy/ch-archive.html#s-priorities) .

4. Consultez la Charte Debian, 7.7 « Relations entre paquets source et binaires — Build-Depends, Build-Depends-Indep, Build-Conflicts, Build-Conflicts-Indep » (http://www.debian.org/doc/debian-policy/ch-relationships.html#s-sourcebinarydeps)

5. Cette situation quelque peu étrange est une fonctionnalité bien expliquée dans la Charte Debian, note de bas de page 55 (http://www.debian.org/doc/debian-policy/footnotes.html#f55) . 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) .

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.

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 les architectures pour lesquelles le paquet binaire peut être compilé. Cette valeur est en général un des suivantes en fonction du type de paquet binaire : [6]

— `Architecture: any`
  — le paquet binaire créé dépend de l'architecture, en général dans un langage compilé ;
— `Architecture: all`
  — le paquet binaire créé est indépendant de l'architecture, en général du texte, des images ou des scripts en langage interprété.

La ligne 10 est laissée telle quelle car c'est écrit en C. dpkg-gencontrol(1) indiquera la valeur d'architecture appropriée pour chaque machine sur laquelle ce paquet source 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 Section 4.4 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`, `Breaks`, `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 (consultez dpkg(8), dselect(8), apt(8), aptitude(1), etc.)

Voici une description simplifiée des relations entre paquets : [7]

— `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 généralement 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 par défaut les paquets recommandés avec le paquet (mais l'utilisateur peut désactiver ce comportement). **dpkg** ignorera 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, il ne lui sera probablement pas proposé 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** ignoreront ce champ ;
— `Pre-Depends`
  cela est plus fort que `Depends`. Le paquet ne sera pas installé avant que les paquets dont il pré-dépend ne soient installés et *correctement configurés*. Utilisez-le *très* rarement et seulement après en avoir discuté sur la liste de discussion debian-devel@lists.debian.org (http://lists.debian.org/debian-devel/) . Autrement dit : ne l'utilisez pas du tout ; :-)

6. Consultez la Charte Debian, 5.6.8 « Architecture » (http://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-Architecture) pour de plus amples précisions.

7. Consultez la Charte Debian, 7 « Déclaration de relations entre paquets » (http://www.debian.org/doc/debian-policy/ch-archive.html#s-priorities) .

— `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 ;

— `Breaks`
les paquets énumérés seront cassés une fois que le paquet ait été installé. En général, une entrée `Breaks` indique qu'elle s'applique aux versions antérieures à une certaine valeur. La résolution de conflit se fait en utilisant des gestionnaires de paquets de haut niveau pour généralement mettre à niveau les paquets énumérés ;

— `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 virtual-package-names-list.txt.gz (http://www.debian.org/doc/packaging-manuals/virtual-package-names-list.txt) . 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, séparées 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. La restriction de chaque paquet particulier est indiquée entre parenthèses après son nom, et devrait contenir une relation dans la liste suivante suivie par une valeur de 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.

dh_shlibdeps(1) calcule les dépendances à des bibliothèques partagées pour les paquets binaires. Il crée une liste d'exécutables ELF et de bibliothèques partagées trouvées pour chaque paquet binaire. Cette liste est utilisée en remplacement de `${shlibs:Depends`

dh_perl(1) calcule les dépendances de Perl. Il crée une liste de dépendances vers `perl` ou `perlapi` pour chaque paquet binaire. Cette liste est utilisée en remplacement de `${perl:Depends}`.

Certaines commandes de `debhelper` peuvent rendre le paquet créé dépendant de paquets supplémentaires. Toutes ces commandes créent une liste de paquets nécessaires pour chaque paquet binaire. Cette liste est utilisée en remplacement de `${misc:Depends`

dh_gencontrol(1) crée `DEBIAN/control` pour chaque paquet binaire en substituant `${shlibs:Depends}`, `${perl:Depends}`, `${misc:Depends}`, etc.

Cela 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 9 est l'URL de la page d'accueil. Supposons qu'il s'agisse de http://www.obsession.se/gentoo/.

La ligne 12 est la description courte. Les terminaux sont larges de 80 colonnes par convention, aussi elle 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. [8]

Le champ `Vcs-*` pour documenter le système de gestion de versions (VCS) peut être inséré entre les lignes 6 et 7. [9] Considérons que le paquet `gentoo` a son VCS localisé sur le service Git d'Alioth en `git://git.debian.org/git/collab-maint/gentoo`

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

8. Ces descriptions sont en anglais. Les traductions de ces descriptions sont fournies par Le projet de traduction de descriptions de Debian — DDTP (http://www.debian.org/intl/l10n/ddtp) .

9. Consultez la référence du Développeur Debian, 6.2.5. « Emplacement du système de gestion de versions » (http://www.debian.org/doc/manuals/developers-reference/best-pkging-practices.html#bpp-vcs) .

```
 1 Source: gentoo
 2 Section: x11
 3 Priority: optional
 4 Maintainer: Prénom Nom <votre.adresse.mail@example.org>
 5 Build-Depends: debhelper (>=10), xlibs-dev, libgtk1.2-dev, libglib1.2-dev
 6 Standards-Version: 4.0.0
 7 Vcs-Git: https://anonscm.debian.org/git/collab-maint/gentoo.git
 8 Vcs-browser: https://anonscm.debian.org/git/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 the
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 an 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 utilizes the GTK+ toolkit
30  for its interface.
```

(Les numéros de ligne ont été ajoutés.)

4.2 `copyright`

Ce fichier contient des renseignements sur le copyright et la licence des sources amont. La Charte Debian, 12.5 « Informations sur le copyright » (http://www.debian.org/doc/debian-policy/ch-docs.html#s-copyrightfile) impose son contenu et la DEP-5 : `debian/copyright` analysable automatiquement (http://dep.debian.net/deps/dep5/) fournit des directives pour son format.

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` publié sous GPL-2.

You must fill in missing information to complete this file, such as the place you got the package from, the actual copyright notice, and the license. For certain common free software licenses (GNU GPL-1, GNU GPL-2, GNU GPL-3, LGPL-2, LGPL-2.1, LGPL-3, GNU FDL-1.2, GNU FDL-1.3, Apache-2.0, 3-Clause BSD, CC0-1.0, MPL-1.1, MPL-2.0 or the Artistic license), you can just refer to the appropriate file in the `/usr/share/common-licenses/` directory that exists on every Debian system. Otherwise, you must include the complete license.

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

```
 1 Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
 2 Upstream-Name: gentoo
 3 Upstream-Contact: Emil Brink <emil@obsession.se>
 4 Source: http://sourceforge.net/projects/gentoo/files/
 5
 6 Files: *
 7 Copyright: 1998-2010 Emil Brink <emil@obsession.se>
 8 License: GPL-2+
 9
10 Files: icons/*
11 Copyright: 1998 Johan Hanson <johan@tiq.com>
```

```
12 License: GPL-2+
13
14 Files: debian/*
15 Copyright: 1998-2010 Josip Rodin <joy-mg@debian.org>
16 License: GPL-2+
17
18 License: GPL-2+
19  This program is free software; you can redistribute it and/or modify
20  it under the terms of the GNU General Public License as published by
21  the Free Software Foundation; either version 2 of the License, or
22  (at your option) any later version.
23  .
24  This program is distributed in the hope that it will be useful,
25  but WITHOUT ANY WARRANTY; without even the implied warranty of
26  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
27  GNU General Public License for more details.
28  .
29  You should have received a copy of the GNU General Public License along
30  with this program; if not, write to the Free Software Foundation, Inc.,
31  51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
32  .
33  On Debian systems, the full text of the GNU General Public
34  License version 2 can be found in the file
35  '/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 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=medium
2
3   * Initial release. (Closes: #nnnn)  <nnnn is the bug number of your ITP>
4
5  -- Josip Rodin <joy-mg@debian.org>  Mon, 22 Mar 2010 00:37:31 +0100
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`, et l'urgence devrait être en medium à moins qu'il n'existe une raison particulière pour une autre valeur.

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 juste en dessous de la première ligne qui commence avec

une astérisque (*). Vous pouvez utiliser dch(1) pour le modifier. Vous pouvez éditer cela manuellement avec un éditeur de texte à condition de suivre les conventions de formatage utilisées par dch(1).

Pour éviter d'envoyer un paquet accidentellement avant qu'il ne soit terminé, il vaut mieux modifier la valeur de distribution à `UNRELEASED` qui n'est pas valable.

Vous obtiendrez quelque chose comme :

```
1  gentoo (0.9.12-1) UNRELEASED; 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 +0100
8
```

(Les numéros de ligne ont été ajoutés.)

Une fois satisfait de toutes les modifications, correctement documentées dans `changelog`, vous devriez modifier la valeur de distribution d'`UNRELEASED` à la valeur de distribution cible `unstable` (ou même `experimental`). 10

Vous pouvez en apprendre plus sur la mise à jour du fichier `changelog` plus loin en Chapitre 8.

4.4 `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`

Tous les fichiers `rules`, comme n'importe quel `Makefile`, sont constitués de plusieurs règles, chacune d'entre elles définissant une cible et comment la réaliser. 11 Une nouvelle règle commence avec la déclaration de sa cible en première colonne. Les lignes suivantes commençant par une tabulation (caractère ASCII 9 : TAB) indiquent ce qui doit être réalisé pour cette cible. Les lignes vides et celles qui commencent par dièse (#) sont traitées comme des commentaires et sont ignorées. 12

Une règle que vous voulez exécuter est appelée par le nom de sa cible comme un argument de la ligne de commande. Par exemple `debian/rules` *`build`* et `fakeroot make -f debian/rules` *`binary`* exécutent respectivement les règles pour les cibles *`build`* et *`binary`*.

Voici une explication simplifiée des cibles :

— `clean` (obligatoire) : pour nettoyer tous 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 ;
— `build-arch` (obligatoire) : pour construire les programmes compilés dépendants de l'architecture à partir des sources dans l'arborescence de construction ;
— `build-indep` (obligatoire) : pour construire les documents formatés indépendants de l'architecture 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, une combinaison des cibles `binary-arch` et `binary-inde`

10. Si vous utilisez la commande `dch -r` pour faire cette dernière modification, n'oublier pas de sauver le fichier `changelog` explicitement dans l'éditeur.
11. Vous pouvez apprendre les bases pour écrire un `Makefile` dans la référence Debian, 12.2. « Make » (http://www.debian.org/doc/manuals/debian-reference/ch12#_make) . La documentation complète est disponible en http://www.gnu.org/software/make/manual/html_node/index.html et dans le paquet `make-doc` de la partie `non-free` de l'archive.

12. La Charte Debian, 4.9 « Script de construction principal : debian/rules » (http://www.debian.org/doc/debian-policy/ch-source.html#s-debianrules) explique les détails.
13. Cette cible est utilisée par `dpkg-buildpackage` comme en Section 6.1.

— `binary-arch` (obligatoire) : pour créer tous les paquets binaires dépendants de l'architecture (`Architecture: any`) dans le répertoire parent ; 1[4]

— `binary-indep` (obligatoire) : pour créer tous les paquets binaires indépendants de l'architecture (`Architecture: all`) dans le répertoire parent ; 1[5]

— `get-orig-source` (optionnelle) : pour obtenir la dernière version du paquet source d'origine à partir d'une archive amont.

Vous vous sentez sans doute submergé pour l'instant, mais les choses vont vraiment se simplifier à l'examen du fichier `rules` que **dh_make** donne par défaut.

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 # See debhelper(7) (uncomment to enable)
 3 # output every command that modifies files on the build system.
 4 #DH_VERBOSE = 1
 5
 6 # see FEATURE AREAS in dpkg-buildflags(1)
 7 #export DEB_BUILD_MAINT_OPTIONS = hardening=+all
 8
 9 # see ENVIRONMENT in dpkg-buildflags(1)
10 # package maintainers to append CFLAGS
11 #export DEB_CFLAGS_MAINT_APPEND  = -Wall -pedantic
12 # package maintainers to append LDFLAGS
13 #export DEB_LDFLAGS_MAINT_APPEND = -Wl,--as-needed
14
15
16 %:
17         dh $@
```

(Les numéros de ligne ont été ajoutés et certains commentaires réduits. Dans le vrai fichier `rules`, l'espace de début est une tabulation.)

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 4 peut être décommentée pour configurer la variable `DH_VERBOSE` à 1, afin que la commande **dh** affiche les commandes **dh_*** qu'elle exécute. Vous pouvez également ajouter ici une ligne `export DH_OPTIONS=-v`, afin que chaque commande **dh_*** affiche les commandes qu'elle exécute. Cela permet de comprendre ce qui se passe exactement derrière ce simple fichier `rules`, et de déboguer ses problèmes. Ce nouveau **dh** est conçu pour constituer un élément essentiel des outils `debhelper` sans rien vous cacher.

Lines 16 and 17 are where all the work is done with an implicit rule using the pattern rule. The percent sign means "any targets", which then call a single program, **dh**, with the target name. 1[6] The **dh** command is a wrapper script that runs appropriate sequences of **dh_*** programs depending on its argument. 1[7]

— `debian/rules clean` exécute `dh clean`, qui exécute à son tour :

14. Cette cible est utilisée par `dpkg-buildpackage -B` comme en Section 6.2.
15. Cette cible est utilisée par `dpkg-buildpackage -A`.
16. Cela 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 Debconf9 par l'auteur de `debhelper`. Sous `Lenny`, **dh_make** créait un fichier `rules` beaucoup plus compliqué avec des règles explicites et de nombreux scripts **dh_*** énumérés pour chacune, la plupart n'étant plus nécessaires maintenant (et montrant l'âge du paquet). La nouvelle commande **dh** plus simple libère le responsable de ce travail de routine « manuel ». Vous gardez les pleins pouvoirs de personnalisation du processus avec les cibles `override_dh_*`. Consultez Section 4.4.3. Il se base uniquement sur le paquet `debhelper` et ne rend pas obscur le processus de construction comme le paquet `cdbs` a tendance à le faire.
17. You can verify the actual sequences of **dh_*** programs invoked for a given *`target`* without really running them by invoking `dh` *`target`* `--no-act` or `debian/rules --` *`'target`* `--no-act'`.

```
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 : 1[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
dh_installemacsen
dh_installifupdown
dh_installinfo
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 ;

18. L'exemple suivant suppose que `debian/compat` a une valeur au moins égale à 9 pour éviter d'invoquer une commande d'assistance Python automatiquement.

— `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. [19] Les plus remarquables d'entre elles valent la peine d'être (très) brièvement présentées de façon simplifiée, dans l'hypothèse d'un environnement de construction basé sur un `Makefile` : [20]

— **dh_auto_clean** exécute normalement ce qui suit si un `Makefile` fournit la cible `distclean` : [21]

```
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 un `Makefile` fournit la cible `test` : [22]

```
make test
```

— **dh_auto_install** exécute normalement ce qui suit si un `Makefile` fournit la cible `install` (la ligne est coupée pour la lisibilité) :

```
make install \
  DESTDIR=/chemin/vers/paquet_version-révision/debian/paquet
```

Toutes les cibles ayant besoin de la commande **fakeroot** contiendront **dh_testroot**, qui se terminera en erreur si vous n'utilisez pas cette commande afin de vous faire passer pour le superutilisateur.

Le plus important à propos du fichier `rules` créé par **dh_make**, c'est qu'il ne s'agit que d'une 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.

Bien que la cible `install` ne soit pas obligatoire, elle est prise en charge. `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 : [23]

— ajout de l'assistance pour la commande **dh_python2** (la meilleure solution pour Python) : [24]
  — ajout de `python` à `Build-Depends`,
  — utilisation de `dh $@ --with python2`,
  — conséquence : gestion des modules Python avec la structure `python` ;
— ajout de l'assistance pour la commande **dh_pysupport** (obsolète) :
  — ajout de `python-support` à `Build-Depends`,
  — utilisation de `dh $@ --with pysupport`,

19. Pour obtenir des renseignements complets sur le rôle exact de tous ces scripts **dh_*** et sur leurs options, veuillez lire leur page de manuel respective et la documentation de `debhelper`.
20. 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.
21. En fait, il recherche la première cible disponible dans le `Makefile` parmi `distclean`, `realclean` et `clean`, et l'exécute.
22. En fait, il recherche dans le `Makefile` la première cible disponible parmi `test` et `check`, et l'exécute.
23. 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`*.
24. L'utilisation de la commande **dh_python2** est préférable à **dh_pysupport** ou **dh_pycentral**. N'utilisez pas la commande **dh_python**.

  - conséquence : gestion des modules Python avec la structure `python-support` ;
- ajout de l'assistance pour la commande **dh_pycentral** (obsolète) :
  - 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 pour 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 et des formats avec TeX ;
- ajout de l'assistance pour les 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 pour 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 les paquets de modules du noyau ;
- ajout de l'assistance pour les 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 pour les 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 pour la commande **dh_girepository** :
  - ajout de `gobject-introspection` à `Build-Depends`,
  - utilisation de `dh $@ --with gir` à la place,
  - conséquence : calcul des dépendances pour les paquets embarquant des données d'introspection GObject et génération de la variable de substitution `${gir:Depends}` pour les dépendances du paquet ;
- ajout de l'assistance pour 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 un fichier de configuration `debian/`*`paquet`*`.bash-completi`

De nombreuses commandes **dh_*** invoquées par la nouvelle commande **dh** peuvent être personnalisées par les fichiers de configuration correspondants dans le répertoire `debian`. Consultez Chapitre 5 et la page de manuel de chaque commande pour la personnalisation de telles fonctionnalités.

Certaines commandes **dh_***, invoquées à l'aide de la nouvelle commande **dh**, doivent parfois être exécutées avec des arguments supplémentaires, exécuter d'autres commandes avec elles ou être ignorées. Pour cela, créez une cible `override_dh_`*`toto`* avec sa règle dans le fichier `rules` définissant une cible `override_dh_`*`toto`* pour remplacer la commande **dh_*toto***. Son rôle est fondamentalement de dire *exécute-moi à la place*. [25]

Les commandes **dh_auto_*** ont tendance à en faire plus que ce qui a été présenté ici (très) simplement, pour prendre en compte toutes les situations délicates. Évitez d'utiliser les cibles `override_dh_*` pour remplacer `dh_*` par des commandes équivalentes simplifiées (à part pour la cible `override_dh_auto_clean`) car elles pourraient contourner des fonctionnalités intelligentes de `debhelper`.

Ainsi, par exemple pour conserver les données de configuration dans le répertoire `/etc/gentoo` au lieu du répertoire consacré `/etc` du dernier paquet `gentoo` utilisant Autotools, il suffit de remplacer l'option `--sysconfig=/etc` donnée par la commande **dh_auto_configure** à la commande **./configure** avec :

```
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 l'appel de la commande **./configure** puisqu'elle remplacera seulement l'option `--sysconfig` et conservera toutes les autres options à propos de la commande **./configure**.

Si le `Makefile` des sources de `gentoo` nécessite de préciser `build` comme cible pour la construction [26], vous pouvez créer une cible `override_dh_auto_build` pour cela.

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

Cela garantit l'exécution de `$(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 au lieu d'utiliser les cibles `distclean` ou `clean`, vous pouvez créer une cible `override_dh_auto_clean` pour l'activer.

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

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. [27]

```
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 Section 4.4.1 (à part `get-orig-source`) peut être rendue difficile. Veuillez limiter les cibles explicites aux cibles `override_dh_*`, et à celles complètement indépendantes, si possible.

25. Avec `Lenny`, pour modifier le comportement d'un script **dh_*** il fallait trouver et adapter la ligne pertinente du fichier `rules`.
26. **dh_auto_build** sans autre option exécutera la première cible du `Makefile`.
27. Les fichiers `debian/changelog` et `debian/NEWS` sont toujours installés automatiquement. Le journal de modification amont est trouvé 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`

The **dh_make** command had major updates since this old document was written. So some parts of this document aren't applicable any more.

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

The **debmake** command is used in place of the **dh_make** command in my new Guide for Debian Maintainers (https://www.debian.org/-doc/devel-manuals#debmake-doc) .

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 présentera globalement le rôle de ces fichiers et leur format. Veuillez consulter la Charte Debian (http://www.debian.org/doc/devel-manuals#policy) et la référence du développeur Debian (http://www.debian.org/-doc/devel-manuals#devref) pour les principes d'empaquetage.

The **dh_make** command will create some template configuration files under the `debian` directory. Take a look at all of them.

Dans ce chapitre, le préfixe `debian/` est omis pour simplifier l'écriture des fichiers du répertoire `debian` quand la signification n'est pas ambiguë.

Certains modèles de fichiers de configuration de `debhelper` risquent de ne pas être créés par la commande **dh_make**. Dans ce cas, vous devez les créer avec un éditeur.

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

— renommez les fichiers de configuration pour utiliser le véritable nom du paquet binaire à la place de *`paquet`* ;
— 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` (consultez Section 4.1) ;
— si nécessaire, modifiez le fichier `rules` (consultez Section 4.4).

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 `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
-----------------
<possible notes regarding this package - if none, delete this file>
 -- Josip Rodin <joy-mg@debian.org>, Wed, 11 Nov 1998 21:02:14 +0100
```

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

5.2 `compat`

Le fichier `compat` définit le niveau de compatibilité de `debhelper`. Actuellement, vous devriez le définir à `debhelper` v10 comme suit :

```
$ echo 10 > debian/compat
```

Vous pouvez utiliser le niveau de compatibilité v9 dans certaines circonstances pour une compatibilité avec les anciennes versions. Cependant, utiliser un niveau inférieur à v9 n'est pas recommandé et devrait être évité pour les nouveaux paquets.

5.3 `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 ces fichiers de configuration comme des conffiles. 1 Lors de la mise à jour d'un paquet, une question sera posée, permettant de garder les anciens fichiers de configuration ou non.

dh_installdeb(1) considère *automatiquement* tous les fichiers du répertoire `/etc` comme des conffiles, 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 devant contenir des conffiles 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 conffiles, parce que **dpkg** va alors interroger les utilisateurs pour vérifier les modifications tout le temps.

Si le programme que vous empaquetez force tous les utilisateurs à modifier les fichiers de configuration du répertoire `/etc`, il y a deux façons classiques de ne pas les marquer comme conffiles, et garder **dpkg** silencieux :

— créer un lien symbolique dans le répertoire `/etc` vers un fichier du répertoire `/var` créé par les scripts du responsable ;
— créer un fichier généré par les scripts du responsable dans le répertoire `/etc`.

Pour obtenir des renseignements sur les scripts du responsable, consultez Section 5.18.

5.4 *`paquet`*`.cron.*`

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

— *`paquet`*`.cron.hourly` — installé comme `/etc/cron.hourly/`*`paquet`* et exécuté une fois par heure ;
— *`paquet`*`.cron.daily` — installé comme `/etc/cron.daily/`*`paquet`* et exécuté une fois par jour ;
— *`paquet`*`.cron.weekly` — installé comme `/etc/cron.weekly/`*`paquet`* et exécuté une fois par semaine ;
— *`paquet`*`.cron.monthly` — installé comme `/etc/cron.monthly/`*`paquet`* et exécuté une fois par mois ;

1. Consultez dpkg(1) et la Charte Debian, D.2.5 « Conffiles » (http://www.debian.org/doc/debian-policy/ap-pkg-controlfields.html#s-pkg-f-Conffiles) .

— *paquet*`.cron.d` — installé comme `/etc/cron.d/`*paquet* et exécuté à tout autre moment.

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

Aucun fichier `cron.*` n'est nécessaire pour configurer la rotation des journaux ; pour cela, voyez dh_installlogrotate(1) et logrotate(8).

5.5 `dirs`

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

Les fichiers énumérés dans un fichier `install` n'ont pas besoin que leurs répertoires soient créés avant, consultez Section 5.11.

Il est préférable d'essayer d'exécuter 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 dans la liste du fichier `dirs`.

5.6 *paquet*`.doc-base`

Si votre paquet est fourni avec une autre documentation que des pages de manuel et d'information, 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 obtenir plus de renseignements sur le format de ce fichier, consultez install-docs(8) et la copie locale `/usr/share/doc/doc-base/doc-base.html/index.html` du manuel Debian de `doc-base`.

Pour obtenir plus de renseignements sur l'installation de documentation supplémentaire, consultez Section 3.3.

5.7 `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`, d'autres fichiers sont aussi inclus :

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

5.8 `emacsen-*`

Si votre paquet fournit des fichiers Emacs qui peuvent être compilés en bytecode 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 *paquet*`.examples`

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

5.10 *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 ? :-)

Please read dh_installinit(1) dh_installsystemd(1) to provide startup script.

The *package*`.default` file will be installed as `/etc/default/`*package*. This file sets defaults that are sourced by the init script. This *package*`.default` file is most often used to set some default flags or timeouts. If your init script has certain configurable features, you can set them in the *package*`.default` file, instead of in the init script itself.

Si le programme amont fournit un fichier pour le script d'initialisation, vous avez le choix de l'utiliser ou non. Si vous n'utilisez pas leur script d'initialisation, créez-en un nouveau en *paquet*`.init`. Cependant, si le script d'initialisation amont semble fonctionnel et s'installe au bon endroit, il vous faudra tout de même configurer le lien symbolique `rc*`. Pour cela, il faut 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 `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`. Ils sont installés par dh_install(1). [2] 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'une espace puis du répertoire d'installation (par rapport au répertoire d'install). Par exemple, si un binaire `src/`*truc* n'est pas installé, le fichier `install` pourrait ressembler à :

```
src/truc usr/bin
```

Cela signifie que lors de l'installation du paquet, il y aura une commande exécutable `/usr/bin/`*truc*.

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 plusieurs 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`).

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

5.12 *paquet*.info

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

5.13 *paquet*.links

Pour créer des liens symboliques supplémentaires dans les répertoires de construction du paquet en tant que responsable du paquet, vous devriez les installer avec dh_link(1) en énumérant leurs chemins complets de fichiers source et destination dans un fichier *paquet*.links.

5.14 {*paquet*.,source/}lintian-overrides

Si lintian signale des erreurs dans son diagnostic 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 le manuel utilisateur de Lintian (https://lintian.debian.org/manual/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/ par la commande **dh_lintian**.

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

5.15 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 pages de manuel. Ils doivent être copiés et modifiés pour chaque commande à laquelle il manque une page de manuel. Veuillez vous assurer d'avoir effacé les modèles non utilisés.

5.15.1 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 brève description sur la façon d'éditer ce genre de fichiers.

Le nom de fichier de la page de manuel devrait donner 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	Remarques
1	Commandes utilisateur	Commandes ou scripts exécutables
2	Appels système	Fonctions fournies par le noyau
3	Appels de bibliothèque	Fonctions des bibliothèques système
4	Fichiers spéciaux	Situés généralement dans /dev
5	Formats de fichier	Par exemple le format de /etc/passwd
6	Jeux	Jeux ou autres programmes frivoles
7	Macropaquets	Comme les macros de **man**
8	Administration système	Programmes normalement exécutés par le superutilisateur
9	Sous-programmes du noyau	Appels non standard 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. 3

5.15.2 `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
```

5.15.3 `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
```

5.16 *paquet*`.manpages`

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

Pour installer `docs/gentoo.1` en tant que page de manuel du paquet `gentoo`, créez le fichier `gentoo.manpages` contenant :

```
docs/gentoo.1
```

3. Remarquez que la page de manuel initiale créée par **help2man** prétendra que des renseignements supplémentaires sont disponibles dans le système d'information. Si la commande n'a pas non plus de page **info**, vous devriez modifier manuellement la page créée par la commande **help2man**.

5.17 NEWS

La commande dh_installchangelogs(1) l'installe.

5.18 {post,pre}{inst,rm}

Les 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 obtenir plus de renseignements, consultez 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 créant vos propres 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 niveau vers de nouvelles versions devraient être silencieuses et non intrusives (les utilisateurs existants ne devraient pas remarquer la mise à niveau à part en constatant la résolution de vieux bogues et peut-être l'arrivée de nouvelles fonctionnalités).

Lorsque la mise à niveau est forcément intrusive (par exemple des fichiers de configuration dispersés dans plusieurs répertoires personnels avec des structures complètement différentes), vous pouvez envisager en dernier recours de basculer le paquet vers un état de repli sûr (par exemple en désactivant un service) et en fournissant la documentation adéquate conformément à la Charte Debian (`README.Debian` et `NEWS.Debian`). N'embêtez pas l'utilisateur avec des notes **debconf** déclenchées par les scripts du responsable lors des mises à niveau.

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

Ces scripts du responsable font partie des améliorations de Debian qui expliquent **pourquoi les gens choisissent Debian**. Prenez garde de ne pas les transformer en source d'agacement.

5.19 *paquet*.symbols

L'empaquetage de bibliothèque n'est pas facile pour un mainteneur débutant, et devrait être évité. Cela dit, si le paquet fournit des bibliothèques, des fichiers `debian/`*`paquet`*`.symbols` devraient exister. Consultez Section A.2.

5.20 TODO

La commande dh_installdocs(1) l'installe.

5.21 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 le service Debian Package Tracker (https://tracker.debian.org/) .

Voici ce qu'il contient :

4. Malgré l'utilisation de raccourcis **bash** pour présenter les noms des fichiers `{pre,post}{inst,rm}`, vous devriez utiliser une syntaxe pure POSIX pour les scripts du responsable pour être compatibles avec **dash** comme interpréteur de commande du système.

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

Normalement, avec un fichier `watch`, la page servie à l'URL `http://sf.net/gentoo` est téléchargée pour chercher des liens sous la forme `<a href=...>`. Le nom de base (juste après la dernière `/`) de chaque URL liée est comparé au motif de l'expression rationnelle Perl `gentoo-(.+)\.tar\.gz` (consultez perlre(1)). Parmi les fichiers correspondants, celui avec le plus grand numéro de version est téléchargé puis le programme **uupdate** est exécuté pour créer une arborescence source mise à jour.

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 correspondant au 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 les motifs `watch` de la forme `http://sf.net/`*`projet`*`/`*`nom-d-archive`*`-(.+)\.tar\.gz`. Le but est de résoudre le problème lié au fréquent changement d'URL SourceForge.

Si la source propose une signature cryptographique pour l'archive, il est recommandé de vérifier son authenticité en utilisant l'option `pgpsigurlmangle` comme expliqué dans uscan(1).

5.22 `source/format`

Dans le fichier `debian/source/format`, il devrait y avoir une seule ligne indiquant le format voulu du paquet source (consulter 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`.

5.23 `source/local-options`

Lors de la maintenance de paquet Debian avec un système de gestion de version, il faut généralement créer une branche (par exemple, `upstream`) pour suivre les sources amont, et une autre branche (typiquement `master` avec Git) pour suivre le paquet Debian. Pour cette dernière, il est préférable de garder les sources amont non modifiés aux côtés des fichiers `debian/*` du paquet Debian pour faciliter la fusion avec les nouveaux sources d'amont.

Après la construction d'un paquet, les sources sont normalement laissés modifiés (« patched »). `dquilt pop -a` doit être exécuté avant d'envoyer vers la branche `master`. Créer un fichier optionnel `debian/source/local-options` contenant `unapply-patches` permet d'automatiser cela. Ce fichier n'est pas inclus dans le paquet source créé et modifie seulement le comportement de construction local. Ce fichier peut aussi contenir `abort-on-upstream-changes` (consultez dpkg-source(1)).

```
unapply-patches
abort-on-upstream-changes
```

5. Consulter DebSrc3.0 (http://wiki.debian.org/Projects/DebSrc3.0) pour un résumé de la conversion des sources aux 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.

5.24 source/options

Les fichiers créés automatiquement dans l'arborescence source peuvent être assez agaçants pour l'empaquetage puisqu'ils sont à l'origine de grands fichiers de correctif sans signification. Des modules personnalisés comme **dh_autoreconf** permettent d'atténuer ce problème comme décrit en Section 4.4.3.

Vous pouvez fournir une expression rationnelle Perl en argument de l'option `--extend-diff-ignore` de dpkg-source(1) pour ignorer les modifications faites aux fichiers créés automatiquement lors de la création du paquet source.

Comme solution générique pour aborder ce problème de fichiers créés automatiquement, vous pouvez enregistrer ces arguments d'option de **dpkg-source** dans le fichier `source/options` du paquet source. L'exemple suivant permet de sauter la création de fichiers de correctif pour `config.sub`, `config.guess` et `Makefile` :

```
extend-diff-ignore = "(^|/)(config\.sub|config\.guess|Makefile)$"
```

5.25 patches/*

L'ancien format source `1.0` créait un unique et gros fichier `diff.gz` contenant les fichiers de maintenance du paquet dans `debian` et les correctifs aux sources. Un tel paquet est un peu délicat à inspecter et à comprendre pour chaque modification de l'arbre source par la suite. Ce n'est pas 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 tous contenus dans le répertoire `debian` sont empaquetés dans le fichier `debian.tar.gz`. Puisque la commande **dpkg-source** peut gérer les correctifs au format **quilt** dans le source `3.0 (quilt)` sans le paquet `quilt`, il n'est pas nécessaire d'ajouter `quilt` à `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]

En Chapitre 3, 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 **dquilt** conformément à la description en Section 3.1.

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
$ dquilt import ../toto.patch
$ dquilt push
$ dquilt refresh
$ dquilt header -e
... description du correctif
```

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

7. Plusieurs méthodes de maintenance des ensembles de correctifs ont été proposées et sont utilisées dans les paquets Debian. Le système **quilt** est le système de maintenance conseillé. Parmi d'autres, **dpatch**, **dbs** et **cdbs** existent. La plupart d'entre eux conservent de tels correctifs dans des fichiers `debian/patches/*`.

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

Chapitre 6

Construction du paquet

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Tout devrait maintenant être prêt pour construire le paquet.

6.1 Reconstruction complète

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

— le paquet `build-essential` ;
— les paquets énumérés dans le champ `Build-Depends` (consultez Section 4.1) ;
— les paquets énumérés dans le champ `Build-Depends-indep` (consultez Section 4.1).

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

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

Cela 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`) ;
— création du fichier `.dsc` ;
— création du fichier `.changes`, en utilisant **dpkg-genchanges**.

Si le résultat de la construction est satisfaisant, signez les fichiers `.dsc` et `.changes` avec votre clef privée GPG avec la commande **debsign**. Vous devez entrer votre phrase secrète deux fois. 1

Pour un paquet Debian non natif, par exemple `gentoo`, vous verrez les fichiers suivants dans le répertoire parent (`~/gentoo`) après la construction des paquets :

1. Cette clef GPG doit être signée par un développeur Debian pour être connectée au réseau de confiance et doit être enregistrée dans le trousseau Debian (http://keyring.debian.org) . Cela permet à vos paquets d'être acceptés dans l'archive Debian. Consultez la page de création d'une nouvelle clef GPG (http://keyring.debian.org/creating-key.html) et le wiki Debian à propos de la signature de clef (http://wiki.debian.org/Keysigning) .

— `gentoo_0.9.12.orig.tar.gz`
C'est le code source amont d'origine, simplement renommé pour être conforme aux normes Debian. Il a été créé au début avec la commande `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).

— `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 correctif **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.ds`

— `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 décrit 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`.
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-changes@lists.debian.org (http://lists.debian.org/debian-devel-changes/) .

Les fichiers `gentoo_0.9.12-1.dsc` et `gentoo_0.9.12-1_i386.changes` doivent être signés en utilisant la commande **debsign** avec votre clef privée GPG du répertoire `~/.gnupg/`, avant de les téléverser dans l'archive FTP de Debian. La signature GPG fournit la preuve que ce sont vraiment vos fichiers par l'utilisation de votre clef publique GPG.

La commande **debsign** peut être utilisée pour signer avec votre identifiant de clef secrète GPG (utile pour les paquets parrainés) dans le fichier `~/.devscripts` comme suit :

```
DEBSIGN_KEYID=Votre_ID_de_clef_GPG
```

Les longues chaînes de chiffres dans les fichiers `.dsc` et `.changes` sont les sommes SHA1 et SHA256 des fichiers indiqués. Les personnes téléchargeant vos fichiers peuvent les vérifier avec 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 plusieurs architectures. 3

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`, (consultez Section 4.1).

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

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

2. 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 `dquilt pop -a` après l'extraction normale.

3. 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.

Cela fera tout le nécessaire pour créer entièrement les paquets binaires dépendant 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 (consultez Section 6.1), 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. [4] Cette distinction 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-In` du fichier `debian/control` (consultez Section 4.1).

6.3 Commande debuild

Vous pouvez automatiser le processus de construction en utilisant la commande **dpkg-buildpackage** et empaqueter ensuite avec la commande **debuild**, consultez debuild(1).

La commande **debuild** exécute la commande **lintian** pour une analyse statique après la construction du paquet Debian. La commande **lintian** peut être personnalisée dans `~/.devscripts` avec ce qui suit :

```
DEBUILD_DPKG_BUILDPACKAGE_OPTS="-us -uc -I -i"
DEBUILD_LINTIAN_OPTS="-i -I --show-overrides"
```

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

```
$ debuild
```

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

```
$ debuild -- clean
```

6.4 Paquet `pbuilder`

Pour un environnement de construction propre (**chroot**) permettant de vérifier les dépendances de construction, `pbuilder` est très utile. [5] Cela garantit une construction propre des sources en construction automatique sous `sid` pour différentes architectures et évite une erreur 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). [6]

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` ou `/etc/pbuilderrc` pour intégrer ce qui suit :

4. 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.
5. Comme le paquet `pbuilder` est en constante évolution, vous devriez vérifier les possibilités réelles de la configuration en consultant la dernière documentation officielle.
6. Consultez http://buildd.debian.org/ pour obtenir plus de renseignements sur le service de construction automatique de paquets Debian.

```
AUTO_DEBSIGN=${AUTO_DEBSIGN:-no}
HOOKDIR=/var/cache/pbuilder/hooks
```

D'abord, le système **chroot** local de `pbuilder` doit être initialisé :

```
$ sudo pbuilder create
```

Si vous possédez déjà le paquet source terminé, exécutez les commandes 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_version.dsc
```

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

Les signatures GPG des fichiers `.dsc` et `.changes` peuvent être créées avec :

```
$ cd /var/cache/pbuilder/result/
$ debsign toto_version_arch.changes
```

Si vous possédez une arborescence des sources à jour, mais n'avez pas créé le paquet source correspondant, exécutez plutôt les commandes suivantes dans le répertoire des sources ayant un répertoire `debian` :

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

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 : [7]

```
#!/bin/sh
set -e
install_packages() {
        apt-get -y --allow-downgrades install "$@"
        }
install_packages lintian
echo "+++ lintian output +++"
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 "+++ end of lintian output +++"
```

Un environnement `sid` à jour est nécessaire pour construire correctement les paquets destinés à `sid`. En pratique, `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. [8] Dans ces cas-là, le fait d'utiliser `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.

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

7. Cela suppose 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`.
8. Il existe des restrictions pour de telles mises à jour de paquet `stable`.

6.5 Commandes git-buildpackage et similaires

Si les développeurs amont utilisent un système de gestion de versions (VCS) [9] pour maintenir leur code source, vous devriez envisager aussi de l'utiliser. Cela rend la fusion et la sélection (« cherry-picking ») des correctifs amont plus faciles. De nombreux paquets de scripts d'enrobage sont disponibles pour la construction de paquets Debian pour chaque système de gestion de versions :

— `git-buildpackage` : assistants pour les paquets Debian gérés dans des dépôts Git ;
— `svn-buildpackage` : assistants pour la maintenance de paquets Debian avec Subversion ;
— `cvs-buildpackage` : scripts pour paquets Debian pour les arbres de sources CVS.

L'utilisation de `git-buildpackage` devient assez populaire parmi les développeurs Debian pour gérer les paquets Debian avec le serveur Git sur alioth.debian.org (http://alioth.debian.org/) . [10] Ce paquet fournit plusieurs commandes pour *automatiser* les activités d'empaquetage :

— gbp-import-dsc(1): import a previous Debian package to a Git repository.
— gbp-import-orig(1): import a new upstream tar to a Git repository.
— gbp-dch(1): generate the Debian changelog from Git commit messages.
— git-buildpackage(1) : construire des paquets Debian à partir d'un dépôt Git ;
— git-pbuilder() : construire des paquets Debian à partir d'un dépôt Git en utilisant **pbuilder** ou **cowbuilder**.

Ces commandes utilisent trois branches pour suivre les activités d'empaquetage :

— `main` pour l'arborescence source de paquet Debian ;
— `upstream` pour l'arborescence source amont ;
— `pristine-tar` pour l'archive amont générée par l'option `--pristine-tar`. [11]

`git-buildpackage` peut être configuré dans `~/.gbp.conf`. Consultez gbp.conf(5). [12]

6.6 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 [13] :

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

Ou faites simplement comme suit 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.

9. Consultez Systèmes de contrôle de versions (http://www.debian.org/doc/manuals/debian-reference/ch10#_version_control_systems) pour obtenir plus de renseignements.
10. Le wiki Debian sur Alioth (http://wiki.debian.org/Alioth) documente la façon d'utiliser le service alioth.debian.org (http://alioth.debian.org/) .
11. L'option `--pristine-tar` invoque la commande **pristine-tar** qui peut générer une copie exacte de l'archive amont d'origine en n'utilisant qu'un petit fichier binaire de delta et le contenu de l'archive amont, qui est normalement gardé dans une branche `upstream` du système de gestion de versions.
12. Voici quelques ressources disponibles sur la toile pour les utilisateurs avancés :

— Construction de paquets Debian avec git-buildpackage (`/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) ;
— git-dpm : paquets Debian dans le gestionnaire Git (http://git-dpm.alioth.debian.org/) ;

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

6.7 Hiérarchie des commandes

Voici un résumé sommaire de la façon dont beaucoup de commandes s'assemblent de manière hiérarchique. Il y a plusieurs manières de faire la même chose :

— `debian/rules` = script du mainteneur pour la construction du paquet ;
— **dpkg-buildpackage** = partie principale de l'outil de construction de paquet ;
— **debuild** = **dpkg-buildpackage** + **lintian** (construction avec des variables d'environnement vérifiées) ;
— **pbuilder** = partie principale de l'outil pour l'environnement chroot de Debian ;
— **pdebuild** = **pbuilder** + **dpkg-buildpackage** (construction dans le chroot) ;
— **cowbuilder** = accélération de l'exécution de **pbuilder** ;
— **git-pbuilder** = syntaxe d'utilisation aisée de ligne de commande pour **pdebuild** (utilisée par **gbp buildpackage**) ;
— **gbp** = gestion des sources Debian dans le dépôt git ;
— **gbp buildpackage** = **pbuilder** + **dpkg-buildpackage** + **gbp**.

Bien que les commandes de haut niveau telles que **gbp buildpackage** et **pbuilder** garantissent un environnement de construction de paquet parfait, il est essentiel de comprendre comment les commandes de bas niveau, telles que `debian/rules` et **dpkg-buildpackage**, sont exécutées en dessous.

Chapitre 7

Contrôle des erreurs du paquet

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Quelques techniques sont à connaître pour rechercher des erreurs sur le paquet avant de l'envoyer sur les archives publiques.

Il est aussi conseillé d'effectuer des essais sur une machine différente de la vôtre. Vous devez observer de près chaque alerte ou erreur pour tous les tests décrits ici.

7.1 Modifications suspectes

Si vous trouvez un nouveau fichier de correctif autocréé comme `debian-changes-*` dans le répertoire `debian/patches` après la construction de votre paquet Debian non natif au format `3.0 (quilt)`, il y a des chances que vous ayez accidentellement modifié quelques fichiers ou que le script de construction ait modifié les sources amont. Si c'est de votre faute, corrigez votre erreur. Si c'est provoqué par le script de construction, corrigez l'erreur à la racine avec **dh-autoreconf** comme en Section 4.4.3 ou contournez-la avec `source/options` comme en Section 5.24.

7.2 Vérification de l'installation d'un 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
```

Pour éviter les problèmes d'installation sur différents systèmes, vous devez vérifier qu'il n'y a pas de nom de fichier en conflit avec ceux existants dans d'autres paquets à l'aide du fichier `Contents-`*`i386`* téléchargé depuis l'archive Debian. La commande **apt-file** peut être pratique pour cela. S'il existe des collisions, veuillez prendre les mesures nécessaires pour éviter ces vrais problèmes : en renommant le fichier, en déplaçant le fichier commun dans un paquet différent dont plusieurs paquets peuvent dépendre, en utilisant le mécanisme d'alternatives (consultez update-alternatives(1)) en coordination avec les responsables des autres paquets concernés, ou encore en déclarant une relation de `Conflicts` dans le fichier `debian/control`.

7.3 Vérification des scripts du responsable d'un paquet

Tous les scripts du responsable, (c'est-à-dire les fichiers `preinst`, `prerm`, `postinst` et `postrm`), sont difficiles à écrire correctement, à moins qu'ils n'aient été générés automatiquement par les programmes `debhelper`. Ne les utilisez donc pas si vous êtes un responsable débutant (consultez Section 5.18).

Si le paquet utilise des scripts du responsable non triviaux, veuillez les essayer non seulement pour le processus d'installation, mais aussi les processus de suppression, purge et 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-révision_i386.deb
```

Les séquences suivantes devraient être essayées :

— installation de la version précédente (si elle existe) ;
— mise à niveau depuis la version précédente ;
— rétrogradation (« downgrade ») à la version précédente (optionnel) ;
— purge ;
— installation du nouveau paquet ;
— suppression (« remove ») du paquet ;
— nouvelle installation du paquet ;
— 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. N'oubliez pas d'essayer aussi les mises à jour à partir de cette version.

Même si la réinstallation vers une version antérieure n'est pas officiellement gérée, il est préférable de la permettre.

7.4 Utilisation de `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** utilise les balises suivants :

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

En cas d'avertissements, mettez au point le paquet pour les éviter ou vérifiez qu'ils sont infondés. S'ils sont infondés, configurez les fichiers `lintian-overrides` comme décrit en Section 5.14.

Vous pouvez reconstruire le paquet avec **dpkg-buildpackage** et lancer **lintian** en une seule commande, si vous utilisez debuild(1) ou pdebuild(1).

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 Section 6.3.

7.5 Commande debc

La commande debc(1) permet d'énumérer les fichiers du paquet Debian binaire.

```
$ debc paquet.changes
```

7.6 Commande debdiff

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

```
$ 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 source et pour vérifier d'éventuelles modifications faites par inadvertance lors de la mise à jour des paquets binaires, comme par exemple des fichiers déplacés ou enlevés involontairement.

7.7 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
```

Le nouveau format source `3.0` garde les modifications dans plusieurs fichiers de correctifs comme décrit en Section 5.25. Vous pouvez aussi suivre les modifications de tous les fichiers `debian/patches/*` en utilisant **interdiff**.

7.8 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

Mise à jour de paquet

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Après la publication d'un paquet, il sera rapidement nécessaire de le mettre à jour.

8.1 Nouvelle révision Debian

Soit un rapport de bogue numéroté `#654321`, concernant votre paquet et décrivant un problème que vous pouvez résoudre. Voici ce que vous devez faire pour créer une nouvelle révision du paquet :

— pour un nouveau correctif :
  — configurer le nom du correctif : `dquilt new` *`nomdubogue.patch`*,
  — déclarer le fichier à modifier : `dquilt add` *`fichier-bogué`*,
  — corriger le problème dans le paquet source pour le bogue amont,
  — l'enregistrer en *`nomdubogue.patch`* : `dquilt refresh`,
  — ajouter sa description : `dquilt header -e` ;
— pour la mise à jour d'un correctif :
  — rappeler le correctif *`toto.patch`* existant : `dquilt pop` *`toto.patch`*,
  — corriger le problème dans l'ancien *`toto.patch`*,
  — mettre à jour *`toto.patch`* : `dquilt refresh`,
  — mettre à jour sa description : `dquilt header -e`,
  — appliquer tous les correctifs en enlevant les approximations (*fuzz*) : `while dquilt push; do dquilt refresh; done` ;
— ajouter une nouvelle révision au début du fichier `changelog` Debian, par exemple avec `dch -i`, ou explicitement avec `dch -v` *`version-révision`*, et ajoutez ensuite les commentaires en utilisant votre éditeur favori ; 1
— ajouter une courte description du bogue et de la solution dans l'entrée du changelog, suivie par `Closes: #654321`. De cette manière, le rapport de bogue sera *automagiquement* 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 qui a été fait en Section 6.1 et Chapitre 7 ;

— une fois satisfait, modifier la valeur de distribution dans `changelog` d'`UNRELEASED` à la valeur de distribution cible `unstable` (ou même `experimental`) ; 2

— envoyer le paquet comme en Chapitre 9. La différence est que cette fois, l'archive source originale ne sera pas incluse, car elle n'a pas été modifiée et est déjà dans l'archive Debian.

Un cas délicat peut se produire quand vous faites un paquet local pour expérimenter l'empaquetage avant d'envoyer la version normale vers l'archive officielle, par exemple *`1.0.1-1`*. Pour des mises à niveau plus en douceur, il vaut mieux créer une entrée de `changelog` avec une chaîne de version comme *`1.0.1-1~rc1`*. Vous pouvez nettoyer le `changelog` en fusionnant ces entrées de modification en une unique entrée pour le paquet officiel. Consultez Section 2.6 pour l'ordre des chaînes de version.

8.2 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-ancienneversion 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.

8.3 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_ancienneversion.debian.tar.gz` depuis la nouvelle arborescence source décompressée. 3 Bien sûr, vous devez vous occuper de quelques routines évidentes :

— création d'une copie des sources amont dans un fichier `toto_nouvelleversion.orig.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 corrigent des bogues et ferment les rapports associés en ajoutant `Closes: #numéro_de_bogue`,

  — description succincte des modifications *à la nouvelle version amont* par le responsable qui corrigent des bogues et ferment les rapports associés en ajoutant `Closes: #numéro_de_bogue` ;

— application de tous les correctifs en enlevant les approximations (« *fuzz* ») : `while dquilt push; do dquilt 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 :

  — `dquilt delete` pour l'enlever ;

— si un correctif appliqué aux sources entre en confit avec les nouvelles modifications des sources amont :

2. Si vous utilisez la commande `dch -r` pour faire cette dernière modification, n'oublier pas de sauver le fichier `changelog` explicitement dans l'éditeur.
3. Si un paquet *`toto`* est empaqueté avec l'ancien format `1.0`, ce peut plutôt être réalisé en exécutant `zcat /chemin/vers/toto_ancienneversion.diff.gz|patch -p1` depuis la nouvelle arborescence source décompressée.

  - `dquilt 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` ;
  - `dquilt refresh` pour mettre à jour le correctif ;
- continuer comme d'habitude avec `while dquilt push; do dquilt refresh; done`.

Cette méthode peut être automatisée avec uupdate(1) :

```
$ apt-get source toto
...
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 dquilt push; do dquilt refresh; done
$ dch
... documentation des modifications réalisées
```

Si le fichier `debian/watch` est configuré comme décrit en Section 5.21, la commande **wget** est inutile. Exécutez simplement uscan(1) dans le répertoire *`toto-ancienneversion`* à la place de la commande **uupdate**. Les sources mises à jour seront *automagiquement* recherchées, téléchargées, et la commande **uupdate** sera exécutée. [4]

Vous pouvez publier ces sources mises à jour en recommençant ce qui a été fait en Section 6.1, Chapitre 7 et Chapitre 9.

8.4 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, le faire permet de profiter de tout le potentiel du système `debhelper` moderne et du format source `3.0` : [5]

- si vous devez, pour quelque raison que ce soit, recréer des fichiers modèles qui avaient été effacés, vous pouvez exécuter **dh_make** à nouveau depuis le répertoire des sources Debian, avec l'option `--addmissing`. Puis modifiez-les de façon adéquate ;
- 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**, consultez les documents suivants pour comprendre ses variables de configuration `DEB_*` :
  - copie locale de `/usr/share/doc/cdbs/cdbs-doc.pdf.gz`,
  - le système de compilation usuel Debian (CDBS), FOSDEM 2009 (http://meetings-archive.debian.net/pub/debian-meetings/-2009/fosdem/slides/The_Common_Debian_Build_System_CDBS/) ;
- 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 ;

4. 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.

5. Si votre parrain ou d'autres responsables s'opposent à la mise à jour du style d'empaquetage existant, ne vous embêtez pas à argumenter. Il y a des choses plus importantes à faire.

— si vous avez un paquet source `1.0` avec un 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` créé 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 en utilisant un autre système de correctif comme `dpatch`, `dbs` ou `cdbs` avec `-p0`, `-p1` ou `-p2`, convertissez-le à `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**, supprimez-le et utilisez le nouveau format source `3.0 (native)`.

Vous devriez consulter les propositions d'améliorations Debian (DEP — Debian Enhancement Proposals) (http://dep.debian.net/) et adopter les propositions marquées « ACCEPTED ».

Les autres tâches décrites en Section 8.3 sont aussi à effectuer.

8.5 Conversion en UTF-8

Si les documents amonts sont encodés avec d'anciens jeux de caractères, les convertir en UTF-8 peut être utile :

— avec iconv(1) pour convertir l'encodage de fichiers texte :

```
iconv -f latin1 -t utf8 truc_entrée.txt > truc_sortie.txt
```

— avec w3m(1) pour convertir les fichiers HTML en fichier texte UTF-8. Assurez-vous d'exécuter cette commande avec des paramètres régionaux en UTF-8 :

```
LC_ALL=en_US.UTF-8 w3m -o display_charset=UTF-8 \
        -cols 70 -dump -no-graph -T text/html \
        < truc_entrée.html > truc_sortie.txt
```

8.6 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 soi, mais des personnes ont parfois utilisé `dch` au lieu de `dch -i`) ;

— 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 bogue refermés sont correctement listés dans les champs `Closes`, que les champs `Maintainer` et `Changed-By` correspondent, que le fichier est signé avec GPG, etc.

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

Chapitre 9

Envoi de paquet

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Debian now requires source-only uploads for normal upload. So this page is outdated.

Maintenant que votre nouveau paquet a été testé en détail, vous voulez le publier sur une archive publique afin de le partager.

9.1 Envoi vers l'archive Debian

Une fois devenu un développeur Debian officiel, 1 vous pouvez envoyer le paquet sur les archives Debian. 2 Vous pouvez le faire vous-même, mais il est plus facile d'utiliser les outils automatiques existants, comme dupload(1) ou dput(1). Nous décrirons la façon de faire avec **dupload**. 3

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 file de téléchargement qui sera utilisée par défaut. `anonymous-ftp-master` est la principale, mais il est possible que vous souhaitiez en utiliser une autre. [4]

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

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

dupload vérifie que les sommes SHA1 et SHA256 des fichiers correspondent à celles du fichier `.changes`. Dans le cas contraire, un avertissement proposera de le reconstruire comme décrit en Section 6.1 pour pouvoir téléverser le fichier correctement.

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**. [5] Par exemple en utilisant `hello.commands` :

1. Consultez Section 1.1.

2. Des archives publiquement accessibles comme http://mentors.debian.net/ qui fonctionnent à peu près de la même façon que l'archive Debian fournissent un espace d'envoi accessible publiquement aux non-DD (développeurs Debian). Vous pouvez configurer vous-même une archive équivalente en utilisant les outils de http://wiki.debian.org/HowToSetupADebianRepository. Cette section est donc utile aussi aux non-DD.

3. 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 d'origine les services relatifs à Ubuntu.
4. Consultez la référence du Développeur Debian, 5.6. « Envois de paquets » (http://www.debian.org/doc/manuals/developers-reference/pkgs.html#upload) .
5. Consultez ftp://ftp.upload.debian.org/pub/UploadQueue/README. Vous pouvez aussi utiliser la commande **dcut** du paquet `dput`.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Uploader: Machin Bidule <Machin.Bidule@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: GnuPG v1.4.10 (GNU/Linux)

[...]
-----END PGP SIGNATURE-----
```

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

Lors du premier envoi d'un paquet vers l'archive, il faut aussi inclure les sources `orig.tar.gz` d'origine. Si le numéro de révision Debian de ce paquet n'est ni `1` ni `0`, vous devez passer l'option `-sa` à la commande **dpkg-buildpackage**.

Pour la commande **dpkg-buildpackage** :

```
$ dpkg-buildpackage -sa
```

Pour la commande **debuild** :

```
$ debuild -sa
```

Pour la commande **pdebuild** :

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

D'un autre côté, l'option `-sd` forcera l'exclusion des sources `orig.tar.gz` originales.

9.3 Versions non envoyées

Si vous créez plusieurs enregistrements dans `debian/changelog` en omettant des téléversements, vous devez créer un fichier `*_.changes` approprié incluant toutes les modifications depuis le dernier envoi. Cela peut être fait avec l'option `-v` de **dpkg-buildpackage** suivie de la version, par exemple *`1.2`*.

Pour la commande **dpkg-buildpackage** :

```
$ dpkg-buildpackage -v1.2
```

Pour la commande **debuild** :

```
$ debuild -v1.2
```

Pour la commande **pdebuild** :

```
$ pdebuild --debbuildopts "-v1.2"
```

Annexe A

Empaquetage avancé

La réécriture de ce tutoriel avec des contenus à jour et des exemples pratiques supplémentaires est disponible sur Guide du responsable Debian (https://www.debian.org/doc/devel-manuals#debmake-doc) . Veuillez utiliser ce nouveau tutoriel comme document principal.

Voici quelques conseils et indications pour des sujets d'empaquetage avancé auxquels vous serez sans doute confrontés. La lecture de toutes les références suggérées ici est vivement recommandée.

Vous pouvez avoir besoin d'éditer vous-mêmes les fichiers de modèle d'empaquetage produits par la commande **dh_make** pour répondre à des préoccupations de ce chapitre. La nouvelle commande **debmake** devrait le faire de meilleure façon.

A.1 Bibliothèques partagées

Avant d'empaqueter des bibliothèques partagées, vous devriez lire les références essentielles suivantes avec attention :

— Charte Debian, 8 « Bibliothèques partagées » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html) ;
— Charte Debian, 9.1.1 « Structure de système de fichiers » (http://www.debian.org/doc/debian-policy/ch-opersys.html#s-fhs) ;
— Charte Debian, 10.2 « Bibliothèques » (http://www.debian.org/doc/debian-policy/ch-files.html#s-libraries) .

Voici quelques conseils simplifiés à l'extrême pour commencer :

— les bibliothèques partagées sont des fichiers objet ELF contenant du code compilé ;
— les bibliothèques partagées sont distribuées en fichiers `*.so` (pas en fichiers `*.a` ni en fichiers `*.la`) ;
— les bibliothèques partagées sont surtout utilisées pour partager du code commun à plusieurs exécutables avec le mécanisme **ld** ;
— les bibliothèques partagées sont parfois utilisées pour fournir plusieurs greffons à un exécutable avec le mécanisme **dlopen** ;
— les bibliothèques partagées exportent des symboles qui représentent des objets compilés comme des variables, des fonctions et des classes ; elles permettent qu'on y accède à partir des exécutables liés ;
— le SONAME d'une bibliothèque partagée `libtruc.so.1` : `objdump -p libtruc.so.1 | grep SONAME` ; 1
— le SONAME d'une bibliothèque partagée correspond normalement au nom du fichier de bibliothèque (mais pas toujours) ;
— le SONAME d'une bibliothèque partagée liée à `/usr/bin/truc` : `objdump -p /usr/bin/truc | grep NEEDED` ; 2
— `libtruc1` : le paquet de bibliothèque pour la bibliothèque partagée `libtruc.so.1` avec la version `1` d'ABI SONAME ; 3

1. Alternative : `readelf -d libtruc.so.1 | grep SONAME`.

2. Alternative : `readelf -d libtruc.so.1 | grep NEEDED`.

3. Consultez la Charte Debian, 8.1 « Bibliothèques partagées au moment de l'exécution » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-runtime) .

— les scripts du responsable du paquet de bibliothèque doivent appeler **ldconfig** dans des circonstances particulières pour créer les liens symboliques nécessaires au SONAME ; [4]

— `lib`*`truc1`*`-dbg` : le paquet de symboles de débogage qui contient les symboles de débogage du paquet de bibliothèque partagée `lib`*`truc1`* ;

— `lib`*`truc`*`-dev` : le paquet de développement qui contient les fichiers d'en-têtes, etc., de la bibliothèque partagée `lib`*`truc`*`.so.`*`1`* ; [5]

— les paquets Debian ne devraient normalement pas contenir de fichiers d'archive Libtool `*.la` ; [6]

— les paquets Debian ne devraient normalement pas utiliser RPATH ; [7]

— bien qu'il soit dépassé et que ce ne soit qu'une référence secondaire, le guide d'empaquetage de bibliothèques Debian (http://www.netfort.gr.jp/~dancer/column/libpkg-guide/libpkg-guide.html) peut encore être utile.

A.2 Gestion de `debian/`*`paquet`*`.symbols`

Lors de l'empaquetage d'une bibliothèque partagée, il faut créer un fichier `debian/`*`paquet`*`.symbols` pour gérer la version minimale associée à chaque symbole pour les modifications d'ABI rétrocompatibles sous le même SONAME de la bibliothèque pour le même nom de paquet de bibliothèque partagée. [8] Vous devriez lire les références essentielles suivantes avec attention :

— Charte Debian, 8.6.3 « Le système de symboles » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-symbols) ; [9]

— dh_makeshlibs(1) ;

— dpkg-gensymbols(1) ;

— dpkg-shlibdeps(1) ;

— deb-symbols(5).

Voici un exemple grossier de la manière de créer le paquet `libtruc1` à partir de la version amont `1.3` avec le fichier `debian/libtruc1.symbols` adéquat :

— préparer le squelette d'arborescence source Debian en utilisant le fichier amont `libtruc-1.3.tar.gz` :

  — s'il s'agit du premier empaquetage du paquet `libtruc1`, créer le fichier vide `debian/libtruc1.symbols` ;

  — si la version amont précédente `1.2` a été empaquetée en tant que paquet `libtruc1` contenant le fichier `debian/libtruc1.symbols` adéquat dans son paquet source, le réutiliser ;

  — si la version amont précédente `1.2` n'a pas été empaquetée avec le fichier `debian/libtruc1.symbols` adéquat, le créer comme fichier `symbols` à partir de tous les paquets binaires disponibles de même nom de paquet de bibliothèque partagée contenant le même SONAME de la bibliothèque, par exemple de versions `1.1-1` et `1.2-1` : 10

```
$ dpkg-deb -x libtruc1_1.1-1.deb libtruc1_1.1-1
$ dpkg-deb -x libtruc1_1.2-1.deb libtruc1_1.2-1
$ : > symbols
$ dpkg-gensymbols -v1.1 -plibtruc1 -Plibtruc1_1.1-1 -Osymbols
$ dpkg-gensymbols -v1.2 -plibtruc1 -Plibtruc1_1.2-1 -Osymbols
```

— essayer de construire depuis l'arborescence source avec des outils comme **debuild** et **pdebuild** (en cas d'échecs dus à des symboles manquants, etc., c'est qu'il y a eu des modifications d'ABI non rétrocompatibles qui nécessitent de changer le nom de paquet de bibliothèque partagée en quelque chose comme `libtruc1a` avant de recommencer) :

4. Consultez la Charte Debian, 8.1.1 « ldconfig » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-ldconfig) .
5. Consultez la Charte Debian, 8.3 « Bibliothèques statiques » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-static) et la Charte Debian, 8.4 « Fichiers de développement » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-dev) .
6. Consultez le wiki Debian sur l'objectif de publication relatif à la suppression des fichiers `*.la` (http://wiki.debian.org/ReleaseGoals/LAFileRemoval) .
7. Consultez le wiki Debian sur les problèmes avec RPATH`*.la` (http://wiki.debian.org/RpathIssue) .
8. Les modifications d'ABI non rétrocompatibles devraient normalement nécessiter une mise à jour du SONAME de la bibliothèque et du nom de paquet de la bibliothèque partagée.
9. Pour les bibliothèques C++ et d'autres cas lorsque le suivi de symboles individuels est trop compliqué, suivez plutôt la Charte Debian, 8.6.4 « Le système de shlibs » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-shlibdeps) .
10. Toutes les versions précédentes des paquets Debian sont disponibles sur http://snapshot.debian.org/ (http://snapshot.debian.org/) . La révision Debian est supprimée de la version pour faciliter le rétroportage du paquet : `1.1` << `1.1-1~bpo70+1` << `1.1-1` et `1.2` << `1.2-1~bpo70+1` << `1.2-1`

```
$ cd libtruc-1.3
$ debuild
...
dpkg-gensymbols: warning: some new symbols appeared in the symbols file: ...
 see diff output below
--- debian/libtruc1.symbols (libtruc1_1.3-1_amd64)
+++ dpkg-gensymbolsFE5gzx        2012-11-11 02:24:53.609667389 +0900
@@ -127,6 +127,7 @@
  truc_get_name@Base 1.1
  truc_get_longname@Base 1.2
  truc_get_type@Base 1.1
+ truc_get_longtype@Base 1.3-1
  truc_get_symbol@Base 1.1
  truc_get_rank@Base 1.1
  truc_new@Base 1.1
...
```

— si un différentiel est affiché par **dpkg-gensymbols** comme ci-dessus, extraire le fichier `symbols` adéquat mis à jour du paquet binaire généré de la bibliothèque partagée : 11

```
$ cd ..
$ dpkg-deb -R  libtruc1_1.3_amd64.deb libtruc1-tmp
$ sed -e 's/1\.3-1/1\.3/' libtruc1-tmp/DEBIAN/symbols \
       >libtruc-1.3/debian/libtruc1.symbols
```

— construire les paquets pour la publication avec des outils comme **debuild** et **pdebuild** :

```
$ cd libtruc-1.3
$ debuild -- clean
$ debuild
...
```

En plus des exemples précédents, il faut vérifier plus profondément la compatibilité d'ABI et changer vous-même les versions de certains symboles si nécessaire. 12

Bien que ce ne soit qu'une référence secondaire, le wiki Debian sur l'utilisation de fichiers symboles (http://wiki.debian.org/-UsingSymbolsFiles) et les pages web qui y sont liées peuvent être utiles.

A.3 Multiarchitecture

La fonctionnalité de multiarchitecture introduite dans Debian Wheezy intègre la prise en charge pour l'installation interarchitecture de paquets binaires (en particulier entre `i386` et `amd64`, mais aussi d'autres combinaisons) dans `dpkg` et `apt`. Vous devriez lire les références suivantes avec attention :

— wiki Ubuntu sur la spécification multiarchitecture (https://wiki.ubuntu.com/MultiarchSpec) (amont) ;

— wiki Debian sur le multiarchitecture et son implémentation (http://wiki.debian.org/Multiarch/Implementation) (situation dans Debian).

Elle utilise un triplet du style `i386-linux-gnu` et `x86_64-linux-gnu` pour le chemin d'installation des bibliothèques partagées. Le chemin de triplet réel est défini dynamiquement dans la variable `$(DEB_HOST_MULTIARCH)` par la commande dpkg-architecture(1) pour chaque construction de paquet binaire. Par exemple, le chemin d'installation de bibliothèques multiarchitectures est modifié comme suit : 13

11. La révision Debian est supprimée de la version pour faciliter le rétroportage du paquet : `1.1` << `1.3` << `1.3-1~bpo70+1` << `1.3-1`

12. Consultez la Charte Debian, 8.6.2 « Modifications d'ABI dans les bibliothèques partagées » (http://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-updates) .

13. Certains anciens chemins de bibliothèques particulières comme `/lib32/` et `/lib64/` ne sont plus utilisés.

Ancien chemin	Chemin multiarchitecture i386	Chemin multiarchitecture amd64
`/lib/`	`/lib/i386-linux-gnu/`	`/lib/x86_64-linux-gnu/`
`/usr/lib/`	`/usr/lib/i386-linux-gnu/`	`/usr/lib/x86_64-linux-gnu/`

Voici quelques exemples typiques de scénarios de découpage de paquet multiarchitecture pour les paquets suivants :

— une bibliothèque source `lib`*`truc`*`-1.tar.gz` ;
— un outil source *`bidule`*`-1.tar.gz` écrit en langage compilé ;
— un outil source *`machin`*`-tar.gz` écrit en langage interprété :

Paquet	Architecture	Multi-Arch	Contenu du paquet
`lib`*`truc1`*	any	same	la bibliothèque partagée, co-installable
`lib`*`truc1`*`-dbg`	any	same	les symboles de débogage de bibliothèque partagée, co-installable
`lib`*`truc`*`-dev`	any	same	les fichiers d'en-tête, etc., de bibliothèque partagée, co-installable
`lib`*`truc`*`-tools`	any	foreign	les programmes de prise en charge en cours d'exécution, non co-installable
`lib`*`truc`*`-doc`	all	foreign	les fichiers de documentation de bibliothèque partagée
`bidule`	any	foreign	les fichiers du programme compilé, non co-installable
`bidule``-doc`	all	foreign	les fichiers de documentation du programme
`machin`	all	foreign	les fichiers du programme interprété

Veuillez remarquer que le paquet de développement devrait contenir un lien symbolique vers la bibliothèque partagée associée **sans numéro de version**. Par exemple : `/usr/lib/x86_64-linux-gnu/libtruc.so` → `libtruc.so.1`.

A.4 Construction d'un paquet de bibliothèque partagée

Un paquet de bibliothèque partagée peut être construit en activant la prise en charge multiarchitecture en utilisant dh(1) comme ceci :

— mettre à jour `debian/control` :
  — ajouter `Build-Depends: debhelper (>=10)` à la section du paquet source,
  — ajouter `Pre-Depends: ${misc:Pre-Depends}` pour chaque paquet binaire de bibliothèque partagée,
  — ajouter la définition `Multi-Arch:` à chaque section de paquet binaire ;
— définir `debian/compat` à « 10 » ;
— ajuster le chemin classique `/usr/lib/` en `/usr/lib/$(DEB_HOST_MULTIARCH)/` multiarchitecture pour tous les scripts d'empaquetage :
  — appeler d'abord `DEB_HOST_MULTIARCH ?= $(shell dpkg-architecture -qDEB_HOST_MULTIARCH)` dans `debian/rules` pour définir la variable `DEB_HOST_MULTIARCH`,
  — remplacer `/usr/lib/` par `/usr/lib/$(DEB_HOST_MULTIARCH)/` dans `debian/rules`,
  — si `./configure` est utilisé par la cible `override_dh_auto_configure` dans `debian/rules`, assurez-vous de le remplacer par `dh_auto_configure --`, [14]
  — remplacer toutes les occurrences de `/usr/lib/` par `/usr/lib/*/` dans les fichiers `debian/`*`truc`*`.install` ;

14. Sinon, les arguments `--libdir=\$${prefix}/lib/$(DEB_HOST_MULTIARCH)` et `--libexecdir=\$${prefix}/lib/$(DEB_HOST_MULTIAR` peuvent être ajoutés à `./configure`. Veuillez remarquer que `--libexecdir` indique le chemin par défaut pour installer les programmes exécutables démarrés par d'autres programmes plutôt que par des utilisateurs. Sa valeur Autotools par défaut est `/usr/libexec/` mais sa valeur Debian par défaut est `/usr/lib/`.

— générer les fichiers comme `debian/truc.links` à partir de `debian/truc.links.in` dynamiquement en ajoutant un script à la cible `override_dh_auto_configure` dans `debian/rules` :

```
override_dh_auto_configure:
        dh_auto_configure
        sed 's/@DEB_HOST_MULTIARCH@/$(DEB_HOST_MULTIARCH)/g' \
                debian/truc.links.in > debian/truc.links
```

Veuillez vous assurer d'avoir vérifié que le paquet de bibliothèque partagée ne contient que les fichiers attendus, et que le paquet -dev fonctionne toujours.

Tous les fichiers installés simultanément par des paquets multiarchitectures au même endroit devraient avoir exactement le même contenu. Vous devez faire attention aux différences générées par l'ordre des octets de données (boutisme) et par les algorithmes de compression.

A.5 Paquet Debian natif

Si un paquet est maintenu uniquement pour Debian ou simplement pour un usage local, ses sources peuvent contenir tous les fichiers `debian/*`. Il existe deux façons de l'empaqueter.

Vous pouvez créer l'archive amont en excluant les fichiers `debian/*` et l'empaqueter comme un paquet Debian non natif comme dans Section 2.1. C'est la méthode normale que certaines personnes préconisent d'utiliser.

L'autre méthode est d'utiliser la manière de procéder pour un paquet Debian natif.

— créer le paquet source Debian natif au format `3.0 (native)` en utilisant un seul fichier tar compressé où tous les fichiers sont intégrés :

  — *paquet_version*`.tar.gz`

  — *paquet_version*`.dsc`

— construire les paquets binaires Debian à partir du paquet source Debian natif :

  — *paquet_version_arch*`.deb`

Par exemple, si vous avez des fichiers source dans `~/mypackage-1.0` sans les fichiers `debian/*`, vous pouvez créer un paquet natif Debian en utilisant la commande **dh_make** comme suit :

```
$ cd ~/monpaquet-1.0
$ dh_make --native
```

Alors le répertoire `debian` et son contenu sont créés comme en Section 2.8. Aucune archive compressée n'est créée puisqu'il s'agit d'un paquet Debian natif, mais c'est la seule différence. La suite de l'empaquetage est à peu près similaire.

Après exécution de la commande **dpkg-buildpackage**, les fichiers suivants apparaîtront dans le répertoire parent :

— `monpaquet_1.0.tar.gz`
  c'est l'archive compressée du code source créé à partir du répertoire `monpaquet-1.0` par la commande **dpkg-source** (il ne se termine pas par `orig.tar.gz`) ;

— `monpaquet_1.0.dsc`
  c'est un résumé du contenu du code source comme dans un paquet Debian non natif (il n'y a pas de révision Debian) ;

— `monpaquet_1.0_i386.deb`
  c'est le paquet binaire terminé comme dans un paquet Debian non natif (il n'y a pas de révision Debian) ;

— `monpaquet_1.0_i386.changes`
  ce fichier décrit toutes les modifications effectuées dans la version actuelle du paquet comme dans un paquet Debian non natif (il n'y a pas de révision Debian).