HORS SÉRIE - HORS SÉRIE - HORS SÉRIE - HORS SÉRIE - HORS SÉRIE

GNU LINUX MAGAZINE / FRANCE

France Métro : 6,40€ - DOM 6,95€ - BEL : 7,30€ - LUX : 7,30€ - PORT. CONT. : 7,30€ - CH : 13FS - CAN : 12$ - MAR : 65DH

L 15066 - 28 H - F: 6,40 € - RD

Janvier / Février 2007

HORS SÉRIE N°28

debian 4.0 Etch

ADMINISTRATION ET CONFIGURATION

◆ INCLUS ! Calendrier/Poster

- Mettre à jour et recompiller le noyau Linux
- Interview exclusive du président de Debian France
- Maîtriser le système de gestion de paquets
- Nettoyer son système
- Recompiler et reconstruire ses paquets
- Comprendre et utiliser le suivi de bogue
- Utiliser le détournement de fichiers

100% DEBIAN

ÉDITO

Bienvenue dans votre système Debian GNU/Linux !

Vous venez de fraîchement installer une distribution Debian ? Vous utilisez ce système depuis quelques mois ou bien plus ? Vous n'êtes pas utilisateur Debian mais vous avez sous votre responsabilité un serveur hébergé utilisant cette distribution ? Vous êtes simplement curieux ? Alors ce hors-série devrait vous plaire.

J'ai tenté de réunir dans ce magazine des informations détaillées sur le fonctionnement, l'utilisation et l'administration d'un système Debian. Tout ne peut bien sûr être dit en quelques dizaines de pages, mais les articles qui vont suivre devraient vous occuper pour un temps.

Je pense qu'il serait vain de cacher ma nette préférence pour la distribution Debian et c'est avec un très grand plaisir que je vous propose ce numéro. D'autant plus qu'au moment même où je rédige le premier brouillon de ce texte, on apprend par Andreas Barth sur la liste debian-devel-announce que Etch est maintenant complètement gelé. Comme vous n'avez sans doute pas encore lu l'article d'introduction, ceci signifie tout simplement que le délai avant la publication de Debian GNU/Linux 4.0, nom de code Etch, se compte en semaines (non, je ne prend pas trop de risques). Les manipulations décrites dans les pages de ce magazine ont été réalisées, pour la plupart, avec une distribution Etch, l'actuelle Testing en soft freeze. Elles sont et seront donc applicables durant une longue période.

Il m'est maintenant utile de préciser que ces explications sont également valables, à quelques nuances près, pour des distributions dérivées de Debian GNU/Linux, comme Ubuntu. Aucun test n'a été fait avec ces distributions qui sortent totalement du cadre de ce hors série.

Je profite de cet espace tout personnel pour remercier au passage Frédéric Le Roy qui a contribué à ce numéro et qui travaille actuellement à la rédaction d'un livre sous une licence Libre sur la découverte de Debian GNU/Linux. Mes remerciements se portent également à Julien Blache, président de l'association Debian France, qui a bien voulu prendre le temps de répondre à mes questions alors que, comme ne nombreux développeurs Debian, il travaille d'arrache pied à la finalisation de Etch. Ceux que je ne remercierais pas, en revanche, sont les quelques travailleurs de la SNCF de la région Alsace qui ont déposé un préavis de grève renouvelé au jour le jour sur plus d'une semaine, me faisant perdre un temps précieux sur ce projet et ayant mis à l'épreuve les nerfs de pas mal de monde, y compris de leurs collègues contrôleurs (en attendant le train, on a forcément le temps de discuter).

Sur ces belles paroles, je vous laisse découvrir par vous-même la richesse du projet, des distributions et des outils Debian.

Ce magazine possède en double page centrale un poster/calendrier 2007 qu'on pourrait sans doute qualifier « de coquin ». Cela fait bien longtemps que le directeur de publication entend mes supplications quant à son aval pour une telle opération. Eh bien, voilà ! Au bout de plus de huit ans, il a fini par craquer et céder à ma demande. Je suis toutefois conscient que cela peut ne pas plaire à tout le monde ; aussi veuillez accepter mes excuses si cela vous choque ou vous dérange. Le magazine a été mis en page de manière à ne pas souffrir de l'absence de ces pages qui sont donc amovibles sans perturber la lecture du magazine : soit pour les afficher, soit pour les jeter en me maudissant.

Denis Bodor

SOMMAIRE:

Linux Magazine France Hors Série
est édité par **Diamond Editions**
B.P. 20142 - 67603 Sélestat Cedex

Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09

E-mail :
cial@ed-diamond.com

Service commercial :
abo@ed-diamond.com

Site :
www.ed-diamond.com

Directeur de publication :
Arnaud Metzler

PRINTED IN Germany / Imprimé en allemagne / Dépôt légal : 3e Trimestre 1998 / N° ISSN : 1291-78 34 / Commission Paritaire : 09 03 K78 976 / Périodicité : Bimestrielle / Prix de vente : 6,40 Euros

RÉDACTION

Rédacteur en chef :
Denis Bodor

Conception graphique :
Fabrice Krachenfels

Responsable publicité :
Véronique Wilhelm
Tél. : 03 88 58 02 08

Service abonnement :
Tél. : 03 88 58 02 08

Impression : VPM DRUCK / www.vpm-druck.de

Distribution France :
(uniquement pour les dépositaires de presse)

MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12

Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04

Service des ventes : Distri-médias :
Tél. : 05 61 72 76 24

LE PROJET ET LES DISTRIBUTIONS DEBIAN

Tout bon magazine dédié à un sujet précis débute par une introduction théorique, un historique et une présentation du sujet. Ce hors-série ne déroge pas à la règle et, avant d'entrer dans le vif du sujet, je vais planter le décor.

Avant toute chose, précisons le sujet de ce numéro ou, plus exactement, ce qui n'en fera pas partie. Il ne s'agit pas d'une collection d'articles d'introduction GNU/Linux. Vous ne trouverez donc pas dans les pages qui suivent de guide d'installation ou de cours d'utilisation d'un système GNU/Linux. Ce hors-série s'adresse à des utilisateurs UNIX et GNU/Linux souhaitant en savoir plus sur le projet Debian et l'administration d'un système Debian GNU/Linux. Les utilisateurs déjà accoutumés aux systèmes issus du projet Debian ne devraient pas être en reste, puisque nous tenterons de couvrir un large panel de sujets, allant de l'utilisation du système de gestion de paquet à l'administration système en passant par les procédures d'installation alternatives.

Debian GNU/Linux est un système largement utilisé en production, mais également pour des systèmes *desktop* personnels. Ainsi, qu'il s'agisse d'un serveur dédié loué, d'une machine personnelle, d'un poste de travail ou d'une machine de développement, ce sont toujours les mêmes outils spécifiques qui sont utilisés et les mêmes règles qui s'appliquent.

Comment parler de Debian sans au moins citer ses dérivés dont le plus connu est, sans doute, la distribution Ubuntu ? Bien que les distributions soient différentes sur bien des points, une large masse de fonctionnalités et de spécifications sont identiques. Si vous êtes utilisateur d'une distribution Ubuntu, vous devrez trouver dans les pages qui vont suivre de quoi vous occuper sur votre système.

UN PEU D'HISTOIRE

Le nom Debian est la fusion des prénoms des créateurs du projet : Ian & Debra Murdock (mari et femme). Les toutes premières versions de Debian GNU/Linux furent diffusées en 1993. De 1994 à 1995, le projet fut soutenu directement par la FSF comme partie intégrante du projet GNU. Le projet a rapidement gagné en popularité et, en novembre 1995, il y avait déjà plus de 60 développeurs pour maintenir les paquets qui composaient alors la distribution.

Dès le départ, l'accent fut mis sur la nécessité de disposer d'un outil d'installation et de désinstallation des composants du système. `dselect` (ou l'horrible `dselect`, pour certains) fit son apparition cette année-là et marqua le début officiel de la mise en œuvre d'un système de gestion de paquets à ce jour inégalé (c'est un avis tout personnel), dans le monde des distributions GNU/Linux.

Ian Murdock arrêta de travailler activement sur le projet en mars 1996, un peu avant l'arrivée de la version 1.0. Cette version fut d'ailleurs renommée 1.1, suite à une erreur commise par un producteur de CD-Rom qui avait libellé 1.0 une version non stable.

La version 2.0 marqua le départ d'un autre point caractéristique du projet : le fait de produire une seule distribution avec un seul numéro de version pour une masse importante d'architectures différentes. Ceci peut sembler anodin, mais il faut bien comprendre que c'est tout le système de développement et de maintenance des paquets qui supporte, encore aujourd'hui, toutes les architectures. Un seul paquet source permet de produire les paquets binaires pour n'importe quelle architecture supportée. La version 2.0 de Debian (Hamm) sortit en juillet 1998 pour les architectures de processeurs Intel i386 et Motorola 68000. Cette version stable contenait plus de 1500 paquets maintenus par plus de 400 développeurs. La version 2.1 (Slink) ajouta le support officiel pour l'Alpha et le Sparc. Cette version intègre également APT (*Advanced Packaging Tool*), la nouvelle génération de l'interface de gestion de paquets.

La Debian 2.2 (Potato) est sortie le 15 août 2000, et, elle aussi, ajouta le support pour deux nouvelles architectures : PowerPC et ARM. La version 3.0 (Woody) ajouta IA-64, HP PA-RISC, MIPS et S/390. A ce moment, le projet compte 900 développeurs et 8 000 paquets.

La version 3.1 (Sarge) est l'actuelle distribution stable (au moment ou ce texte est rédigé) et la prochain mouture, 4.0 (Etch) est prévue pour le mois de décembre 2006. Peut-être l'annonce de sa disponibilité en tant que nouvelle version stable aura-t-elle été faite au moment où vous lirez ceci.

Au fil des versions et révisions, c'est non seulement le nombre d'architectures qui augmente (officiellement ou non officiellement supportées), mais également la stabilité et la maturité du système de gestion des paquets. Debian GNU/Linux est développé à la fois dans le but de fournir une distribution pour particuliers et pour professionnels. Les besoins en termes de stabilité et d'intégrité de la distribution sont, en effet, identiques.

LE CŒUR DU SYSTÈME : LA GESTION DE PAQUETS

Le projet Debian fut sans doute le premier à proposer une distribution capable de se mettre à jour intégralement jour après jour, mois après mois et même année après année. Les machines initialement installées avec une version 2.0 et régulièrement mises à jour sont légion. Le fait de pouvoir changer de matériel sans réinstallation (propre aux systèmes d'exploitation de bonne facture) couplé à un excellent système de gestion des paquets et des mises à jour permet cet état de fait. Réinstaller une distribution Debian ne doit pas arriver. Exactement de la même manière que le redémarrage d'un serveur ou d'un poste GNU/Linux n'est conditionné que par deux choses : un problème ou une mise à jour matériels ou un changement de noyau.

Le principal problème à résoudre dans une gestion de paquets se résume en un seul terme : dépendances. En effet, une application nécessite souvent, pour fonctionner, la présence de bibliothèques dans le système. La tâche du système de gestion de paquets est de proposer l'installation des paquets nécessaires au fonctionnement de l'application (ou directement résoudre les problèmes). Le paquet `gimp`, par exemple, contient l'application de retouche d'image The Gimp. Celle-ci nécessite la présence, entre autres, du paquet `libgtk2.0-common`, lui-même nécessitant la présence du paquet `libgtk2.0-0`, etc. La tâche de la gestion de paquets est de déterminer les dépendances, mais également les dépendances inverses. Si on désinstalle `libgtk2.0-0`, cela doit provoquer un avertissement suivi de la désinstallation de `libgtk2.0-common` et donc de `gimp`, car l'application ne peut fonctionner dans le cas contraire. Debian GNU/Linux est réputé pour la qualité de cette gestion. Il est très difficile de se retrouver avec un système « bancal » ou « cassé ».

La gestion des paquets d'une distribution Debian GNU/Linux repose sur le programme `dpkg`. Celui-ci est appelé par des outils APT ou des interfaces comme Aptitude (mode console) ou Synaptic (GUI). Si vous êtes utilisateur d'une distribution RedHat, ex-Fedora ou Mandriva par exemple, considérez que `dpkg` correspond à RPM.

Le format des paquets, le `.deb` est une archive `ar` standard contenant trois fichiers :

- `control.tar.gz` qui contient les méta-informations du paquet (sommes de contrôle, description du paquet, liste des dépendances, scripts d'installation/désinstallation, etc.).
- `data.tar.gz` qui contient les données de l'application ou de la bibliothèque à installer. Il s'agit d'une archive TAR compressée avec `gzip` contenant, tout simplement, l'arborescence complète et les fichiers du paquet tels qu'ils seront copiés sur le système.
- `debian-binary` qui est un simple fichier texte contenant le numéro de version du format. Actuellement, nous en sommes à la version 2.0 du format de paquet.

LES VERSIONS ET RÉVISIONS

Voici un point de confusion courant pour les utilisateurs débutants avec une distribution Debian. Le système de développement et de stabilisation d'une distribution se résume (grossièrement) au chemin suivi par un paquet dans le flux de stabilisation.

Trois « branches » (plus exactement des distributions, voir plus bas) majeures existent :

- **Stable** : là se trouvent les paquets maintenus pour l'actuelle distribution officielle. Au moment où ce texte est rédigé, l'actuelle distribution stable est la version 3.1, nom de code Sarge (voir tableau sur les versions et les noms de code).
- **Testing** : comprend les paquets non encore stabilisés et en cours de tests. Ils ne sont pas considérés comme utilisables sur un serveur en production ou une machine en laquelle on peut faire à 100% confiance. Gardez à l'esprit la définition d'un paquet dans Testing : « contient les paquets qui n'ont pas encore été acceptés dans la distribution stable, mais qui sont en attente de l'être ». Voilà pourquoi un certain nombre d'utilisateurs préfèrent la distribution Testing, les paquets sont presque stables, mais plus récents.
- **Unstable** : contient les paquets en développement. Le nom parle de lui-même (instable) et ces paquets sont à considérer comme absolument inappropriés pour une utilisation sûre du système. Il existe également une branche Experimental qu'on peut légitimement considérer comme la version « bricolo » d'un paquet. Utiliser une version d'Unstable n'est pas sûr, utiliser une version provenant d'Experimental est complètement stupide, sauf si vous êtes développeur Debian ou participez de près ou de loin au projet.

Les trois branches sont également les distributions (c'est là qu'on risque de perdre le fil) et donc des dépôts pour les paquets. Stable est la distribution officielle. Testing est la prochaine distribution en cours de finalisation. Testing devient Stable au moment où le processus de développement est terminé et qu'une distribution est officiellement produite.

NOTE

Un paquet est ici défini comme le nom du paquet + sa version. Un paquet vim peut être présent dans Stable, Testing et Unstable, mais vim:6.3-071+1sarge1 est le paquet dans Stable, vim:7.0-122+1 le paquet dans Testing et vim:7.0-164+1 le paquet dans Unstable. Ce ne sont pas les mêmes paquets.

INTRODUCTION

Le changement de Testing en Stable est transparent pour l'utilisateur d'une version Stable. Actuellement, Sarge est Stable, lorsque Etch (actuel Testing) deviendra Stable, la seule chose que vous verrez, c'est un gros lot de mises à jour. Votre distribution se transformera de Sarge en Etch, mais vous serez toujours utilisateur de Debian GNU/Linux en version Stable.

Le processus Debian pour que la distribution Testing devienne Stable est la suivante :

- Lorsque Testing devient suffisamment mature, le gel de la distribution commence. Il n'est pas possible d'arbitrairement dire « Stop, maintenant Testing devient Stable », car des paquets fonctionnels d'Unstable migrent en permanence vers Testing au fur et à mesure de leur stabilisation. Le début du gel de Testing commence par un allongement des délais de propagation (de sauts) d'Unstable vers Testing. Les paquets en provenance d'Unstable contiennent toujours des bogues. En allongeant le délai, on s'assure de limiter l'arrivée de nouveaux bogues dans Testing.

- On gèle Testing complètement. A ce moment, les paquets qui devaient encore arriver dans Testing sont bloqués (on ferme les vannes à bogues). Toute l'énergie des développeurs est alors concentrée sur la stabilisation de Testing et la correction des bogues. Une seule exception permet à un paquet d'Unstable d'arriver dans Testing à ce moment : le fait qu'il corrige un bogue critique pouvant empêcher la création d'une nouvelle distribution. Actuellement, le paquet `mediawiki1.7` par exemple, empêche la sortie d'Etch en tant que Stable. Il provoque une erreur de segmentation avec `php 5.2.0-5`. Dans ce cas précis, si la version du paquet actuellement dans Unstable corrige le problème, elle pourra passer dans Testing, même après le début du gel de la distribution. Une autre solution peut être de simplement éliminer `mediawiki1.7` de la distribution. Il faut savoir sacrifier (rétrograder) un paquet pour permettre la sortie d'une distribution.

- Une fois que le nombre de bogues non corrigés passe en deçà d'un certain niveau, Testing est prête pour publication. Une nouvelle distribution Debian GNU/Linux est officiellement annoncée et Testing devient Stable. La distribution se voit alors attribuée son numéro de version. L'ancienne distribution Stable devient obsolète, elle est retirée des dépôts et archivée. Les utilisateurs voient alors une grosse mise à jour de leur système arriver. Parallèlement, des images de CD-Rom et de DVD sont créées en guise de version stable de la distribution pour les nouveaux utilisateurs ou les nouvelles installations.

Stable est également mise à jour régulièrement, mais il ne s'agit que de corrections de bogues. Lorsque Testing devient Stable, la distribution n'est pas totalement exempte de bogues et même une fois la distribution publiée officiellement, de nouveaux bogues peuvent être signalés. A intervalles plus ou moins réguliers, une mise à jour mineure est publiée sous la forme d'images de CD ou DVD. L'actuelle version Stable est la 3.1r4. Nous en sommes donc à la 5ème série d'images officielles des versions d'installation de la distribution. Ceci permet aux nouvelles installations d'être directement à jour et ne pas voir l'utilisateur obligé de récupérer (via le Net) une quantité trop importante de paquets mis à jour post-installation.

En parallèle à la stabilisation de la distribution, le développement et l'amélioration de l'installeur suit son cours. Pour qu'une nouvelle distribution soit publiée, il faut donc également que l'installeur soit stabilisé. L'installeur assure la procédure de création d'un système Debian GNU/Linux sur les disques d'une machine. Il fournit également une interface conviviale à l'utilisateur afin de faciliter l'installation. Il doit suivre l'évolution des technologies et les fonctionnalités offertes par les systèmes GNU/Linux récents (LVM, RAID logiciel, etc.). Tout comme la distribution elle-même, l'installeur doit pouvoir gérer toutes les architectures officiellement supportées par le projet. On notera au passage qu'il n'y a pas si longtemps une décision importante a été prise : celle de réduite le nombre d'architectures supportées officiellement aux architectures récentes et encore « vivantes ».

Le support d'un trop grand nombre d'architectures est un réel problème. Un paquet peut parfaitement fonctionner sur x86, mais pas sur Alpha et donc bloquer la stabilisation de Testing. Le délai entre la version 3.0 (juillet 2002) et 3.1 (juin 2005) a provoqué bien des « râlements ». La stabilisation après le gel complet de la future 3.1 était bien trop longue et la version stable, 3.0, était presque devenue obsolète. Voilà pourquoi des décisions radicales ont dû être prises (même si une partie des développeurs Debian, les DD, a estimé que le problème provenait plus d'un manque de réactivité que du nombre trop important d'architectures supportées).

CONTRAT SOCIAL DEBIAN

Le projet Debian se distingue non seulement par divers choix techniques, mais également par des principes qu'on pourrait qualifier de philosophiques. Ces principes ont été détaillés de longue date (1997) dans un document appelé « contrat social Debian » et mis à jour en avril 2004. On peut voir ce contrat comme le « contrat de confiance » d'un certain détaillant de matériels électroménagers. Comme le précise le contrat social lui-même, il s'agit d'un ensemble de principes auxquels le projet tient fermement.

Le contrat se compose de cinq points :

1. Debian demeurera totalement libre. C'est la base du contrat. Les distributions Debian sont composées d'éléments libres selon les « Principes du Logiciel libre selon Debian » (voir plus loin). Point important, il est clairement spécifié : « Nous ne rendrons pas le système

dépendant d'un composant non libre. », ce qui ne va pas sans créer un certain nombre de problèmes, dont ceux posés par les pilotes de périphériques intégrant des éléments propriétaires ou non libres selon les principes du Logiciel libre selon Debian.

2. Nous donnerons nos travaux à la communauté des Logiciels libres. C'est une conséquence directe des principes du Logiciel libre. Les travaux en question sont l'ensemble des développements nécessaires pour construire et maintenir les distributions. Ceci inclut les utilitaires et applications de gestion de paquets, mais également les éléments satellites et l'information elle-même (corrections de bogues, améliorations, requêtes des utilisateurs, etc.).

3. Nous ne dissimulerons pas les problèmes. C'est le principe de « *Full disclosure* » qui s'oppose à la sécurité par obscurantisme (qui n'est pas vraiment possible avec le Logiciel libre). Ceci signifie la conservation de l'intégralité de la base de données concernant la gestion des bogues et sa mise à disposition publique. C'est la transparence qui garantit la qualité, aussi bien en termes de sécurité que de qualité de code.

4. Nos priorités sont nos utilisateurs et les Logiciels libres. Encore un principe important qui guide le développement et l'évolution des distributions. Ceux-ci sont dirigés par les besoins des utilisateurs (au sens large du terme). Ceci inclut le fait que Debian ne s'oppose ni aux travaux dérivés du projet, ni aux travaux non libres prévus pour fonctionner avec les systèmes Debian. Cette clause affirme donc le non-totalitarisme de Debian, puisqu'il n'y a clairement pas de restriction imposée (autre que celles des licences des Logiciels libres) concernant la réutilisation de tous les travaux du projet.

5. Travaux non conformes à nos standards sur les Logiciels libres. Cette dernière clause ou principe permet de prendre en compte la réalité de la vie. En s'en tenant uniquement aux quatre principes précédents et en ignorant le fait que des applications propriétaires (ou semi-propriétaires) existent et qu'un certain nombre d'utilisateurs en ont besoin, le projet Debian ne serait pas viable. Les distributions Debian rendent possible la prise en charge de ces applications par le système de paquet et leur intégration via deux sections spécifiques nommées `contrib` et `non-free`. Les paquets présents dans ces sections ne font pas partie du système Debian, mais sont configurés de manière à pouvoir être utilisés dans une distribution du projet.

Le contrat social Debian a aujourd'hui le même problème que bien des contrats : il ne pouvait pas, au moment de sa création, tenir compte de cas précis qui n'existaient pas encore ou qui ne pouvaient être envisagés. Qui aurait pu penser, il y a quelques année, que la prise en charge de périphériques sous Linux via des morceaux de code binaires et propriétaires aurait tendance à se généraliser ? Plusieurs cas se distinguent :

- le téléchargement d'un *firmware* propriétaire dans le périphérique nécessitant son fonctionnement ;
- la mise en œuvre de *wrappers* permettant d'utiliser un pilote propriétaire Windows avec Linux ;
- les licences incomplètes, restrictives ou non libres utilisées par les constructeurs sur les sources de leurs pilotes.

Le problème se pose déjà au niveau du développement du noyau Linux et une certaine tolérance existe. Malheureusement, il faudra clarifier les choses de manière définitive à un moment ou un autre. Il en va de même pour le projet Debian, qui devra prendre position et donc faire une mise à jour du contrat social en prenant compte des nouvelles réalités. Pour l'heure, rien n'est vraiment tranché.

LES PRINCIPES DU LOGICIEL LIBRE SELON DEBIAN

Pourquoi les documentations sous licence GNU FDL ne sont-elles pas systématiquement libres selon Debian ? La réponse se trouve dans la vision du Logiciel libre selon Debian. Cette vision est à la fois plus large, mais aussi plus précise que les simples quatre types de libertés qui caractérisent habituellement un Logiciel libre. Les principes du Logiciel libre selon Debian (DFSG : *Debian Free Software Guidelines*) sont les suivants :

- Redistribution libre et gratuite. C'est un composant de base qu'on retrouve dans les licences comme la GNU GPL.
- Code source. Encore une des quatre libertés de base.
- Applications dérivées : la licence doit autoriser les modifications et les applications dérivées, ainsi que leur distribution sous les mêmes termes que ceux de la licence du logiciel original. On retrouve également ce principe dans la GPL, mais pas nécessairement dans d'autres licences.
- Intégrité du code source de l'auteur. Debian prend en compte le fait que l'auteur ne souhaite pas que des modifications sur son œuvre existent, mais uniquement s'il est tout de même possible de créer des travaux dérivés. Ainsi, ma licence peut interdire la distribution de versions dérivées de mon code source, mais pas la distribution de mon code avec des patchs. De la même manière, je peux exiger que les versions dérivées portent un autre nom ou un autre numéro de version que mon original. Mon travail reste du Logiciel libre selon Debian.

INTRODUCTION

- Aucune discrimination de personne ou de groupe. Ceci peut semble normal pour un citoyen européen ou français, puisque la discrimination n'est pas légale (Article 225-1 du Code pénal français). Il n'en va pas forcément de même dans le reste du monde et si ma licence interdit aux personnes de plus d'1m90 d'utiliser mon logiciel, il ne sera pas libre selon Debian. La liberté fondamentale numéro 1 du Logiciel libre de pouvoir utiliser le logiciel couvre ce point, mais de manière très générale.
- Aucune discrimination de champ d'application. Il ne faut pas que ma licence interdise par exemple l'utilisation de mon logiciel pour des recherches sur la fusion froide (sic).
- Distribution de licence. La licence sous laquelle est diffusé mon logiciel est globale. Il ne faut pas d'obligation de se conformer à une autre licence.
- La licence ne doit pas être spécifique à Debian. Encore une fois, il faut que la licence soit globale. Je ne peux fournir de libertés qu'à Debian.
- La licence ne doit pas contaminer d'autres logiciels. Il ne s'agit pas ici de l'aspect « contaminant » de la GPL par inclusion de code libre dans un nouveau développement, mais du fait que les termes de la licence puissent se propager à d'autres applications. Par exemple, « L'utilisation de ce logiciel interdit toute installation de logiciels non-GPL sur le même système ». Un peu trop totalitaire pour être un Logiciel libre selon Debian.

Le problème posé par la licence FDL (*Free Documentation License*) concerne les sections non modifiables. La FDL prévoit, en effet, que l'auteur puisse placer des restrictions sur certaines parties de la documentation, interdisant leur modification. Suite à une réflexion puis un vote des développeurs Debian, la décision a été prise que seules les documentations sous FDL n'ayant pas de sections non modifiables sont libres selon les DFSG et donc peuvent être intégrées à l'archive principale de Debian. Les autres documentations devront alors être placées dans `non-free`. La résolution générale prend effet pour la distribution Etch, prochaine distribution Stable.

Les différents noms de code des versions de Debian GNU/Linux correspondent à des noms de personnages du dessin animé 3D *Toy Story* :

Version	Nom	Date	Personnage
0.01 à 0.91	...	août 1993 à janvier 1994	...
0.93R5	...	mars 1995	...
0.93R6	...	novembre 1995	...
1.1	Buzz	juin 1996	Le ranger de l'espaceNoyau linux 2.0
1.2	Rex	décembre 1996	Le tyrannosaure
1.3	Bo	juillet 1997	La bergère
2.0	Hamm	juillet 1998	Le cochon-tirelire
2.1	Slink	9 mars 1999	Le chien à ressort
2.2	Potato	15 août 2000	Monsieur Patate
3.0	Woody	19 juillet 2002	Le cow-boy
3.1	Sarge	6 juin 2005	Le chef des soldats
4.0	Etch	janvier 2007 (?)	L'écran magique
...	Lenny	...	La paire de jumelles

On remarquera qu'un nom de code ne sera jamais utilisé pour une distribution stable. Il s'agit de Sid, l'enfant des voisins qui casse tous les jouets dans *Toy Story*. Sid est également l'acronyme de « *Still In Development* » (toujours en développement). C'est tout simplement le nom donné à la version Unstable de la distribution.

ENFIN, UNE REPRÉSENTATION DE DEBIAN POUR LA FRANCE :

La création d'une représentation française de Debian n'est pas un projet récent. Mais aujourd'hui, ce projet se concrétise enfin grâce à quelques développeurs Debian motivés.

La gestation de l'association fut bien longue en raison de la charge de travail de chacun des participants, celle-ci provenant non seulement de leur activité professionnelle ou estudiantine, mais également de leur implication dans le projet Debian. Les premières discussions formelles à propos d'une association datent de l'été 2005 sur la liste initialement prévue pour l'organisation de *Solutions Linux* 2005. Après plusieurs mois de discussions, les différents initiateurs du projet, parmi lesquels J. Blache, R. Hertzog, A. Jarno ou encore S. Luther ont finalement mis un terme à la rédaction des statuts et du règlement intérieur.

AG constitutive de l'association Debian France (photo par Aurelien Jarno)

Le 7 Juillet 2006 à Vandœuvre-Lès-Nancy, en marge des *Rencontres Mondiales du Logiciel Libre* (RMLL), s'est déroulée l'Assemblée Générale Constitutive de l'Association Debian France. Les statuts ont, à cette occasion, été adoptés à l'unanimité. Enfin, l'association fut officiellement créée le 25 septembre 2006 lors de l'inscription au Tribunal d'instance de Colmar, avec la publication légale dans le quotidien *L'Alsace* à cette même date.

Depuis, l'association trouve sa vitesse de croisière, petit à petit, et on attend avec impatience ses premiers faits officiels, comme l'organisation du stand Debian lors de *Solutions Linux* 2007, les 30, 31 janvier et 1er février à Paris. L'occasion d'accueillir les nouveaux adhérents et de faire la démonstration de l'existence physique de Debian France.

Bon courage les gars... Mon bulletin d'adhésion est déjà parti.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

DEBIAN FRANCE : INTERVIEW DU PRÉSIDENT

Depuis septembre 2006, le projet Debian dispose d'une représentation officielle en France sous la forme de l'association Debian France. Son président, Julien Blache, a accepté de répondre à nos questions.

Q GLMF :

Peux-tu te présenter aux lecteurs ?

R JB :

J'utilise Debian depuis début 2000 et Linux depuis 1999. Je suis Développeur Debian depuis l'été 2000. En dehors des contributions techniques, je m'occupe, entouré d'une solide équipe de Développeurs Debian français et francophones, de la représentation du Projet Debian sur le salon *Solutions Linux* depuis plusieurs années.

Je suis actuellement en dernière année de cycle ingénieur à l'INSA Lyon ; je mène une activité professionnelle en parallèle avec ces études depuis 2 ans. Nous travaillons sur et avec Debian, avec un effectif qui est notamment composé de 3 Développeurs Debian ; nos collègues sont, comme nous, des spécialistes de la distribution.

Q GLMF :

Pourquoi créer une association Debian France ?

R JB :

Il y a plusieurs bonnes raisons. Cela faisait plusieurs années que j'évoquais la création d'une association avec les DD français, notamment pour ce qui concerne la représentation de Debian sur les salons comme *Solutions Linux* pour des raisons purement administratives.

Mais ce n'est qu'une des raisons. Rassembler les utilisateurs, contributeurs et entreprises intéressés par Debian doit permettre des rencontres et des projets. Cela doit aussi permettre de se compter.

Il est plus facile de représenter Debian, que ce soit sur un salon ou auprès d'une institution, lorsque l'on est organisé en une structure officielle et que l'on peut avancer des chiffres et quelques autres données.

La structure française permettra également (bientôt) aux utilisateurs français de faire des dons à l'association et/ou au Projet plus facilement qu'à l'heure actuelle via une structure forcément étrangère.

L'association utilisera ses fonds propres pour la représentation du Projet sur les salons et pour soutenir des projets ; elle acceptera également des dons pour le Projet Debian, les fonds constitués par ces dons étant sous l'autorité du *Project Leader*.

Parmi les actio ns concrètes que nous avons dans l'idée, nous espérons pouvoir héberger en France des morceaux de l'infrastructure Debian, par exemple.

Q GLMF :

Tu parles de serveurs qui pourraient être sous la responsabilité de Debian France ?

R JB :

Oui, tout à fait. Il serait bon que l'on commence à distribuer un peu l'infrastructure du Projet qui est pour l'instant très regroupée géographiquement. Ca n'est toutefois pas critique. Nous répondrons aux besoins si nous le pouvons, nous n'allons pas les créer !

Q GLMF :

Pourquoi une association en droit local Alsace-Moselle et non une association loi 1901 ?

R JB :

Pour des raisons purement géographiques. Il n'y a pas de raison particulière. Cela ne change pas grand-chose dans les faits pour l'usage que nous avons de l'association.

Q GLMF :

L'aspect « officiel » Debian est clairement mis en avant dans le règlement intérieur de l'association. Le Conseil d'Administration, par exemple, doit être composé pour au moins deux tiers de Développeurs Debian. Cela a-t-il été décidé de manière à ce que l'association fasse partie intégrante du projet ?

R JB :

En un sens, oui. L'association va accepter (ce n'est pas encore en place) des dons pour le Projet Debian, ce qui est différent des dons faits à l'association elle-même. Ces fonds (ou matériels) appartiennent au Projet Debian et sont directement sous l'autorité du Project Leader et l'association n'a pas à y toucher sans son ordre.

Cette restriction sur la composition du CA est une mesure de sécurité vis à vis de Debian et de ses fonds qui seront sous notre garde.

Nous ne souhaitions pas créer une association de Développeurs Debian, nous voulions créer une association ouverte à tous les utilisateurs et contributeurs du Projet. Cette clause, ainsi que la clause de veto en assemblée générale, permet cela en toute sécurité par rapport au Projet Debian.

Prendre le nom « Association Debian France » n'est pas anodin et l'association ne peut pas faire n'importe quoi ; nous avons mis les garde-fous qui nous semblaient nécessaires sans en faire trop.

Q GLMF :

On voit souvent le projet Debian comme un groupe de personnes davantage douées pour travailler que pour promouvoir une distribution. Le fait d'avoir un morceau de Debian en France signifie-t-il qu'il y aura également plus de communication à la manière d'autres associations diffusant régulièrement des communiqués et des annonces ?

R JB :

Le but de l'association est clairement de communiquer sur Debian, mais pas de se substituer au Projet. Nous communiquerons sur les actions de l'association et sur les communications de Debian. Attention également aux effets d'annonce, ce n'est pas le but ici, bien évidemment.

Pour résumer : communiquer, oui, sur-communiquer, non.

Q GLMF :

On a vu, il n'y a pas si longtemps, qu'il était relativement difficile pour un utilisateur de reconnaître ce qui représentait officiellement Debian. Je pense aux différents canaux #debian-fr dont celui sur OFTC. Le rôle de Debian France sera-t-il aussi de clarifier les choses au niveau de la francophonie ?

R JB :

Nous n'avons pas d'emprise sur les différents forums ou canaux IRC de support aux utilisateurs, pas plus au niveau français qu'au niveau mondial.

L'association a probablement un rôle à jouer, mais nous n'avons pas de projet dans cette direction pour le moment. Nous sommes ouverts à toutes les idées, sur ce point ou sur un autre.

Q GLMF :

Faut-il s'attendre à voir apparaître un site/blog Debian France et ses forums, ses tips, sa carte d'utilisateur, etc. ?

R JB :

Le blog est sans doute un bon moyen de communication aujourd'hui. C'est quelque chose qui peut se faire. La priorité pour le moment, outre les tracasseries administratives, est de mettre en place un site plus utile et fourni qu'à l'heure actuelle ; cela prendra sûrement la forme d'un *wiki*.

Un forum est envisageable à partir du moment où nous disposons d'une équipe de personnes motivées pour assurer son animation et sa maintenance au quotidien. Avis aux amateurs !

L'association Debian France, c'est un outil. Comme tous les outils, il ne s'use que si on s'en sert. Debian France, ce sera avant tout ce que ses adhérents en feront, avec les moyens que nous aurons, tous ensemble, rassemblés pour ce faire.

Q GLMF :

Il existe plusieurs sites francophones concernant le projet Debian et la distribution GNU/Linux comme andesi.org ou encore debian-fr.org. Y a-t-il déjà des projets d'unification de toutes ces documentations autour d'un site central ?

R JB :

Il n'y pas de projets précis, mais nous avons pris des contacts avant la création de l'association. J'invite évidemment les mainteneurs de ces ressources à travailler avec nous s'ils le souhaitent.

Q GLMF :

Merci pour ces réponses et bonne chance à l'association.

TOUR D'HORIZON DES OUTILS APT*

Lorsqu'on débute avec une distribution Debian, on fait rapidement connaissance avec les commandes APT, le système de gestion des paquets. Ceci se résume souvent à deux commandes alors que le nombre d'outils disponibles est bien plus important.

APT est l'acronyme de « *Advanced Packaging Tool* ». C'est tout simplement le système de gestion des paquets et des dépendances. Il n'existe pas de commande `apt`, mais des outils permettant d'interagir avec le système de gestion. Leur nom débute habituellement avec les trois fameuses lettres. Initialement, les outils du système APT étaient des interfaces pour `dpkg` l'installeur/désinstalleur de paquets. Aujourd'hui, les fonctionnalités d'APT sont également utilisées en arrière-plan par des outils graphiques comme Synaptic (via la `libapt`). Cette application tout comme Adept, l'équivalent KDE/Qt de Synaptic (GNOME/GTK+), sort du cadre de cet article. Nous nous intéresserons uniquement aux outils en ligne de commande ou en mode console.

SOURCES.LIST LA BASE DE TOUTES CHOSES

Pour que le système de gestion fonctionne, il faut qu'il puisse savoir où chercher et trouver des informations à jour sur les paquets disponibles. Les sources d'information sont définies dans le fichier `/etc/apt/sources.list`.

Sur un système fraîchement installé, on trouve habituellement dans le fichier quelque chose comme ceci :

```
deb http://ftp.fr.debian.org/debian/ stable main non-free contrib
deb-src http://ftp.fr.debian.org/debian/ stable main non-free contrib
```

La syntaxe utilisée est :

```
type URI distribution [composant1] [composant2] [...]
```

Le type peut être `deb` pour les sources de binaires ou `deb-src` pour les index des paquets sources. Si vous voulez construire vos propres paquets binaires à partir des sources, il vous faut une ligne de type `deb-src` pour chaque ligne `deb`.

L'URI ou *Universal Resource Identifier* est le chemin complet vers la ressource. Deux cas de figure se présentent : soit vous indiquez le chemin complet et vous terminez `distribution` par un `/` (slash), soit, comme dans l'exemple ci-dessus, vous précisez l'URI, `distribution` et un ou plusieurs composants. Dans la réalité, on trouve de tout, comme par exemple :

```
deb http://apt.bxlug.be/sarge/ main/
```

On voit clairement ici que `distribution` se termine par un `/`, mais qu'il s'agit en fait d'un composant. On voit également des choses comme ceci :

```
deb http://people.ubuntu.com/~doko/0002/ ./
```

L'URI précise l'emplacement direct des paquets et du fichier d'index `Packages.gz` et `distribution` pointe vers le répertoire courant. Cela fonctionne, mais il serait préférable de voir des architectures plus « propres » utilisées.

L'URI peut être, vous l'aurez compris, une URL d'un serveur HTTP ou FTP pour les téléchargements de paquets, mais il peut également s'agir d'un répertoire (`file`), d'une liaison sécurisée SSH (`ssh`) ou d'un lecteur CD-ROM ou DVD (`cdrom`). Dans ce dernier cas, il est conseillé d'utiliser l'outil `apt-cdrom`. Ce type d'URI permet de prendre en compte le changement de média et l'interaction avec l'utilisateur.

Pour un système de développement ou de tests, il n'est pas rare d'avoir des dizaines de lignes dans son `sources.list`. Il est toutefois possible de mettre un peu d'ordre en plaçant dans `/etc/apt/sources.list.d` plusieurs fichiers ayant l'extension `.list` et contenant chacun une série de ligne. On peut ainsi plus facilement organiser les sources, les gérer en groupe et les exporter vers d'autres systèmes Debian.

Le système APT se configure via deux fichiers importants. Le premier est `/etc/apt/apt.conf`. C'est le fichier de configuration principal. C'est ici que vous pourrez personnaliser le comportement de l'ensemble du système APT. Habituellement, la principale personnalisation concerne l'utilisation d'un serveur mandataire HTTP (*proxy*). Pour ce faire, il suffit d'ajouter dans votre `apt.conf` une seule ligne :

```
Acquire::http::Proxy "http://User:Pass@Serveur:Port";
```

La configuration se fait via des groupes d'options (ici `Acquire`) et des options séparées par deux double-points. Une ligne se termine toujours par un point-virgule. Il est également possible de spécifier un champ d'action pour les options avec la syntaxe :

```
APT {
  Get {
    Assume-Yes "true";
    Fix-Broken "true";
  };
};
```

Le groupe APT, contient une sous-section Get permettant de définir des options par défaut de la commande apt-get. Ici, nous choisissons de répondre automatiquement affirmativement et de tenter de réparer automatiquement un système dont les dépendances sont défectueuses. Pour connaître l'ensemble des options, consultez la page de manuel d'apt.conf et des différents outils APT.

L'autre fichier de configuration important est /etc/apt/preferences. C'est le fichier de contrôle des préférences du système APT. Son but est principalement de déterminer les priorités affectées aux distributions. En effet, si vous avez, dans votre sources.list plusieurs archives pour plusieurs distributions, il est important de spécifier l'ordre de priorité des versions pour l'installation des paquets.

Voici un exemple de fichier preferences :

```
Package: *
Pin: release a=unstable
Pin-priority: 80

Package: *
Pin: release a=experimental
Pin-priority: 70

Package: *
Pin: release a=testing
Pin-priority: 900

Package: *
Pin: release a=stable
Pin-priority: 500

Package: *
Pin: release v=2.2*
Pin-priority: 50

Package: *
Pin: release v=2.1*
Pin-priority: 40

Package: *
Pin: release v=2.0*
Pin-priority: 30
```

L'ordre dans lequel apparaissent les lignes n'est pas important. C'est la Pin-priority qui détermine l'ordre des sélections de paquets sur l'ensemble des distributions. Nous avons ici, dans l'ordre :

900 : *testing*

500 : *stable*

80 : *unstable*

70 : *experimental*

50 : 2.2

40 : 2.1

30 : 2.0

Les règles qui s'appliquent alors sont les suivantes :

1 - Ne jamais revenir en arrière : le système APT ne doit jamais installer une version inférieure d'un paquet à la place d'une version déjà installée.

2 - Installer la version qui possède la priorité la plus importante. Ici la version provenant de la distribution Testing, sauf si le paquet n'est pas présent dans Testing, mais dans Unstable et Stable. A ce moment-là, c'est la version dans Stable qui sera utilisée (priorité 500).

3 - Installer la version la plus importante si deux versions ont la même priorité. On prendra soin d'éviter ce genre de chose et de bien ordonner les priorités dans le fichier des préférences.

4 - Si deux paquets ont la même priorité et la même version, mais qu'ils sont différents, APT devra installer la version qui n'est pas celle déjà installée.

Une configuration comme celle donnée ici permet de maintenir un système de test relativement propre. Il s'agit d'une distribution Testing, puisque les paquets ayant la plus grande priorité proviennent de cette distribution. En parallèle, il nous est parfaitement possible d'installer un paquet provenant d'Unstable (voir syntaxe d'apt-get). Ensuite, petit à petit, la version installée sera rattrapée par celle de la distribution Testing.

Les priorités définies dans preferences s'étalent normalement entre 1 et 999. Une priorité au-delà de 1000 possède une propriété particulière. Elle permet d'enfreindre la règle numéro 1. Consultez la page de manuel d'apt_preferences pour avoir plus d'information sur l'interprétation des priorités.

Notez que des priorités peuvent s'appliquer à un ensemble réduit de paquets (voire un seul paquet). On peut ainsi forcer une version avec une priorité maximum en utilisant :

```
Package: vim
Pin: version 7.0*
Pin-Priority: 1001
```

La priorité 1001 permet de forcer APT à mettre à jour « vers le bas » (*downgrader*) le paquet si nécessaire. apt-get ne manquera pas de vous le signaler avec un message du type : Les paquets suivants seront mis à une VERSION INFÉRIEURE.

Dans tous les cas, réservez ce genre de choses à des installations de test et, en aucune façon, à des systèmes en production.

PAQUETS

APT-CACHE RECHERCHE ET INFORMATION

Lorsqu'on souhaite installer une application, encore faut-il connaître le nom du paquet concerné. L'intérêt premier d'apt-cache est donc la recherche de paquets. Cet outil, comme bien d'autres en provenance du projet Debian, utilise une syntaxe particulière :

```
commmande action argument(s)
```

Ainsi, la recherche de paquets se fait via apt-cache search motif où motif est une expression rationnelle (ou régulière). Une recherche simple peut être :

```
% apt-cache search ascii
```

qui ne liste pas moins de 227 paquets. On peut alors utiliser plusieurs expressions séparées par une espace :

```
% apt-cache search ascii lib
```

Nous voici arrivé à 52 résultats seulement. Mais nous n'avons pas utilisé pleinement les ressources à notre disposition et pouvons utiliser :

```
% apt-cache search  "^lib.*ascii"
liblingua-de-ascii-perl - convert German umlauts to and from
ascii
libtext-asciitable-perl - Perl module to create a table
using ASCII characters
semi - library to provide MIME feature for emacsen
```

Et si nous savons que les chaînes lib et ascii figurent dans le nom du paquet, nous pouvons limiter la recherche à celui-ci (et non plus au nom à la description du paquet) :

```
% apt-cache search --names-only "^lib.*ascii"
liblingua-de-ascii-perl - convert German umlauts to and from
ascii
libtext-asciitable-perl - Perl module to create a table
using ASCII characters
```

Les expressions rationnelles sont un des éléments les plus importants à maîtriser ou du moins à connaître pour un utilisateur GNU/Linux. Qu'il s'agisse d'une recherche avec apt-cache search, de développement Perl ou PHP ou encore de filtrage de mails avec procmail, elles fournissent une puissance importante. Si vous ne connaissez pas encore le sujet, je vous recommande fortement de consulter les différents tutoriels en ligne et de vous exercer avec divers outils (comme grep). Ce ne sera jamais du temps perdu.

apt-cache n'est pas qu'un outil de recherche. Il permet également d'obtenir des informations sur les paquets. La commande show permet ainsi d'afficher la description et d'autres éléments intéressants sur un ou plusieurs paquets :

```
% apt-cache show php5-ldap
Package: php5-ldap
Priority: optional
Section: web
Installed-Size: 96
Maintainer: Debian PHP Maintainers
  <pkg-php-maint@lists.alioth.debian.org>
Architecture: i386
Source: php5
Version: 5.2.0-6
Depends: libc6 (>= 2.3.6-6), libldap2 (>= 2.1.17-1),
  phpapi-20060613+lfs, php5-common (= 5.2.0-6)
Filename: pool/main/p/php5/php5-ldap_5.2.0-6_i386.deb
Size: 17228
MD5sum: f0bdb017e8ecc65cdec2a3d21f9d2092
SHA1: b6b31332f3d3ed1499040ba37eb9009b80e87e79
SHA256: 6e9a4db3dc26157496dd0bce416caa79bb5963519bcb7f5eb3d
df6ee25f7700b
Description: LDAP module for php5
 This package provides a module for LDAP functions in PHP
 scripts.

 PHP5 is an HTML-embedded scripting language. Much of its
 syntax is borrowed from C, Java and Perl with a couple of
 unique PHP-specific features thrown in. The goal of the
language
 is to allow web developers to write dynamically generated
 pages quickly.
```

Nous obtenons la description du paquet, mais également la version, le nom du paquet source, la section, le chemin et le nom de fichier du paquet dans l'archive Debian, diverses sommes de contrôles et *hashs* cryptographiques, la taille du paquet ou encore l'espace occupé après installation.

Il existe bien d'autres commandes (actions) utilisables avec apt-cache, même si search et show sont de loin les plus intéressantes :

- madison paquet permet de lister les versions disponibles pour un paquet (voir apt-sho-verions).
- stats retourne un nombre de statistiques intéressantes sur les paquets disponibles.
- depends liste les dépendances d'un paquet, les paquets suggérés, les éventuels conflits et les paquets remplacés.

Ce ne sont là que quelques exemples. La page de manuel d'apt-cache vous en apprendra bien plus.

APT-GET INSTALLATION/DÉSINSTALLATION

A présent, nous sommes en mesure de trouver les paquets qui nous intéressent, mais encore faut-il pouvoir les installer et les mettre à jour. Pour ce faire, c'est l'utilitaire apt-get qui entre en jeu. Dans un premier temps, apt-get update permet de récupérer les listes de paquets et leurs descriptions.

Depuis peu, la nouvelle version d'APT utilisée dans Etch ne récupère plus automatiquement le fichier `Packages` dans son ensemble, mais les fichiers présents dans `Packages.diff/`. Ceci permet de minimiser le temps de téléchargement pour l'utilisateur, mais surtout pour la création de miroirs d'archives Debian. C'est le fichier `Packages.diff/Index` qui contient la liste des différences avec une précédente version de `Packages`. Bien entendu, l'authenticité du fichier `Index` peut être vérifiée (voir plus loin).

Une fois la liste des paquets obtenue ou mise à jour, le système APT est en mesure d'associer un nom de paquet avec un chemin vers une ressource et un nom de fichier à récupérer. Dans le même temps, le système détermine si le paquet demandé nécessite l'installation d'autres paquets pour fonctionner. Si tel est le cas, les autres paquets sont également proposés à l'installation. Par extension, et comme on peut s'en douter, les paquets supplémentaires peuvent en nécessiter d'autres qui devront également être installés, et ainsi de suite.

Installer ou mettre à jour un paquet se fait en utilisant l'action `install` de l'outil `apt-get` :

```
% sudo apt-get install abiword
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Les paquets supplémentaires suivants seront installés :
  abiword-common libenchant1c2a libgnomecups1.0-1
  libgnomeprint2.2-0 libgnomeprint2.2-data
  libgnomeprintui2.2-0 libgnomeprintui2.2-common
Paquets suggérés :
  abiword-plugins abiword-plugins-gnome
Paquets recommandés :
  abiword-help gnome-icon-theme
Les NOUVEAUX paquets suivants seront installés :
  abiword abiword-common libenchant1c2a libgnomecups1.0-1
  libgnomeprint2.2-0 libgnomeprint2.2-data
  libgnomeprintui2.2-0 libgnomeprintui2.2-common
0 mis à jour, 8 nouvellement installés, 0 à enlever et 728
non mis à jour.
Il est nécessaire de prendre 5372ko dans les archives.
Après dépaquetage, 19,7Mo d'espace disque supplémentaires
seront utilisés.
Souhaitez-vous continuer [O/n] ?
```

Cet exemple le montre clairement, nous avons demandé l'installation du traitement de texte Abiword sous la forme du paquet du même nom. APT analyse les dépendances et affiche le nom de l'ensemble des paquets à installer. L'utilisateur est alors invité à confirmer l'opération. Si tel est le cas, les fichiers sont récupérés via les ressources spécifiées dans le fichier `sources.list`, puis installés.

La mise à jour d'un seul paquet est tout aussi simple, puisque, après avoir utilisé `apt-get update`, il suffit d'utiliser à nouveau `apt-get install` suivi du nom du paquet. Il existe cependant une méthode plus efficace pour garder un système à jour.

L'action `upgrade` permet de rechercher et mettre à jour automatiquement les paquets déjà installés. Tous les paquets, dont la mise à jour n'interfère pas avec d'autres paquets et ne nécessitant pas d'installation complémentaire, sont concernés. `apt-get upgrade` vous signalera les paquets non mis à jour avec un message comme « Les paquets suivants ont été conservés ». Si vous souhaitez élargir la mise à jour à l'ensemble des paquets, vous devez utiliser `apt-get dist-upgrade`. Attention, ne confirmez jamais la mise à jour avec `dist-upgrade` sans lire attentivement les messages à l'écran. Un certain nombre de paquets peuvent être désinstallés durant la mise à jour. Dans tous les cas, `apt-get` vous le signalera et vous n'aurez pas d'excuse en cas de problème.

NOTE

Il est possible de forcer l'installation d'un paquet, même si celui-ci est déjà installé et à jour. Il suffit d'utiliser l'option `--reinstall` avant de spécifier le nom du paquet.

`apt-get` permet également l'installation des sources dans le but de créer ses propres paquets Debian. L'action en question est `source`. Nous y reviendrons plus en détail dans un article spécialement dédié à la construction de paquets.

`apt-get remove` permet la suppression d'un paquet. Bien entendu, les dépendances sont gérées et, là encore, il est fortement recommandé de lire attentivement les messages à l'écran. Une option `--purge` permet de supprimer les fichiers de configuration du paquet. Par défaut, ceux-ci sont conservés pour une éventuelle réinstallation ultérieure (voir article sur le nettoyage du système). Purger les fichiers de configuration est parfois une bonne idée, mais mieux vaut réfléchir avant d'agir (ceci dit, je suppose que vous faites régulièrement des sauvegardes de votre `/etc` ;).

Enfin, dernière action intéressante, `clean` permet de vider le cache des fichiers téléchargés. En effet, les fichiers `.deb` récupérés sur Internet ne sont pas automatiquement supprimés après installation. Vous pouvez donc rapidement vous retrouver avec des centaines de mégaoctets de données dans `/var/cache/apt/archives`. `apt-get clean` permet donc de faire rapidement de la place.

NOTE

A propos du cache de fichiers, vous pouvez utiliser l'option `-d` ou `--download-only` permettant de récupérer un fichier sans l'installer. Ceci vous permettra, par exemple, de manipuler directement les fichiers via `dpkg`.

APT-SHOW-VERSIONS SUIVI DES VERSIONS

`apt-get dist-upgrade` est une bonne solution pour mettre à jour globalement son installation système. Cependant, l'opération manque cruellement de finesse, en particulier si vous n'utilisez pas la distribution Stable. Pour Testing et Unstable, il n'est pas rare que les utilisateurs rencontrent des problèmes et fassent des rapports de bogue. Il n'est donc peut-être pas souhaitable que vous installiez ces paquets bogués.

En parallèle à ces questions de mise à jour, il est toujours avantageux, lorsqu'on travaille avec plusieurs distributions, de connaître les différentes versions installables. On peut ainsi choisir d'installer une version spécifique avec `apt-get install paquet=version` ou encore une version provenant d'une distribution particulière avec, par exemple, `apt-get install paquet/unstable`.

Pour en apprendre plus à la fois sur les paquets à mettre à jour et les versions en présence dans les différentes distributions, vous devez installer le paquet `apt-show-versions`. Utilisez ensuite l'outil en commençant par initialiser et mettre à jour le cache avec `apt-show-versions -i` en tant qu'utilisateur `root`.

Dès lors, vous pourrez lister les paquets à mettre à jour sur votre système avec mention de l'actuelle version et celle pouvant être installée :

```
% apt-show-versions -u
[...]
libpng12-0/testing upgradeable from 1.2.8rel-5.2 to 1.2.13-4
eterm/testing upgradeable from 0.9.3-1 to 0.9.4.0debian1-2
xkb-data/testing upgradeable from 0.8-7 to 0.9-4
[...]
```

Vous pouvez également limiter la plage de recherche en utilisant une expression rationnelle :

```
% apt-show-versions  -u -r "vim.*per"
vim-perl/testing upgradeable from 1:7.0-035+1 to 1:7.0-122+1
```

Comme vous pouvez le voir, vous obtenez bien plus d'information qu'avec `apt-get upgrade`. Ceci vous permet de gérer plus finement l'étendue des mises à jours en fonction du moment et de votre charge de travail.

En plus de vous informer de la version à installer, `apt-show-versions` vous permet de regarder dans les distributions d'à côté :

```
% apt-show-versions -a vim
vim     1:7.0-035+1         install ok installed
vim     1:6.3-071+1sarge1   stable
vim     1:7.0-122+1         testing
vim     1:7.0-164+1         unstable
vim/testing upgradeable from 1:7.0-035+1 to 1:7.0-122+1
```

Vous pouvez ainsi, sur un système de test, essayer la nouvelle version de Vim sortie tout droit d'Unstable alors que votre système est basé sur Stable. Notez cependant que ce genre de chose peut facilement déboucher sur des problèmes importants de dépendance. Le système APT dispose d'un *solver*, mais il possède ses limites. Réservez donc ce genre de chose à un système dédié.

APT-FILE ET DPKG -S/-L RECHERCHE DE FICHIER

Les paquets sont des archives d'archives contenant les fichiers destinés à être copiés sur le système hôte. Il est donc légitime de se dire qu'il pourrait être possible et fort pratique de retrouver les paquet auquel appartient un fichier. La première solution pour cela consiste à utiliser `dpkg` :

```
% dpkg -S /usr/share/vim/vimrc
vim-common: /usr/share/vim/vimrc
```

Il est très facile, en cas de problème sur un fichier, de réinstaller le paquet correspondant et ainsi retrouver un système stable. De la même manière, on pourra avec l'option `-L` lister les fichiers installés par un paquet.

Malheureusement, `dpkg` possède deux limitations importantes. Premièrement, les recherches ne peuvent porter que sur les paquets installés. Impossible donc de trouver une solution de cette manière au problème « On me parle du fichier `/usr/truc`. Qu'est ce qui me manque ? ». Second problème, `dpkg -S` ou `-L` ne retournent pas toujours tous les fichiers installés. Exemple :

```
% dpkg -S /usr/bin/vim
dpkg: /usr/bin/vim introuvable.
```

`/usr/bin/vim` ne serait donc installé par aucun paquet. Nous en avons presque confirmation avec :

```
% dpkg -L vim
/.
/usr
/usr/bin
/usr/bin/vim.basic
/usr/share
/usr/share/lintian
/usr/share/lintian/overrides
/usr/share/lintian/overrides/vim
/usr/share/doc
/usr/share/doc/vim
```

L'explication est simple : le paquet `vim` ne contient pas `/usr/bin/vim`, mais installe un `/usr/bin/vim` :

```
% ls -l /usr/bin/vim
[...]21 2006-08-07 11:43 /usr/bin/vim -> /etc/alternatives/vim

% ls -l /etc/alternatives/vim
[...]17 2006-08-10 14:18 /etc/alternatives/vim -> /usr/bin/vim.perl

% ls -l /usr/bin/vim.perl
[...]1776704 2006-07-11 06:09 /usr/bin/vim.perl
```

`/usr/bin/vim` n'est qu'une succession de liens symboliques pointant finalement sur un binaire. Ainsi, on se rend compte que le paquet `vim` contient uniquement `vim.basic`, une version light de Vim. Notre binaire `vim.perl` est, en réalité, installé par `vim-perl`. Celui-ci, lors de l'installation, crée le lien `/etc/alternatives/vim` vers `/usr/bin/vim.perl`.

Pour régler nos deux problèmes en une fois, il nous suffit d'installer `apt-file`. Cet outil permet de rechercher les fichiers installés par des paquets et lister les fichiers d'un paquet, et ce, indépendamment du fait qu'il soit installé ou non :

```
% apt-file search -F /usr/bin/vim
vim: usr/bin/vim
vim-full: usr/bin/vim
vim-gnome: usr/bin/vim
vim-gtk: usr/bin/vim
vim-lesstif: usr/bin/vim
vim-perl: usr/bin/vim
vim-python: usr/bin/vim
vim-ruby: usr/bin/vim
vim-tcl: usr/bin/vim
```

Nous obtenons immédiatement la liste des paquets installant le fichier demandé. Notez l'utilisation de l'option `-F` permettant de ne pas développer la chaîne (`/usr/bin/vim.*`). Vous pouvez également utiliser une expression rationnelle pour vos recherches avec `-X` ou encore réduire la sortie aux simples noms de paquets avec `-l` (pour pouvoir réutiliser plus facilement la liste dans un script *shell*).

NOTE

`apt-file` semble nécessiter la présence de `/etc/apt/sources.list`. Si vous avez distribué vos sources APT dans des fichiers `/etc/apt/sources.list.d/`, vous rencontrerez certainement des problèmes (bogue #353275).

`apt-file` offre ainsi bien des services, mais n'oubliez pas de régulièrement remettre à jour le contenu des paquets avec `apt-file update`. Enfin, à l'instar de `dpkg -L`, vous pouvez lister le contenu d'un paquet (mais dans l'installeur) avec `apt-file list`.

APT-KEY ET APT-SECURE AUTHENTIFICATION DES PAQUETS

La prochaine distribution Stable (Etch) intégrera le système APT en version 0.6.x (contre 0.5.x dans l'actuel Stable). La grande nouveauté de cette version concerne la sécurité et l'authentification des sources, des archives et des paquets.

Le fonctionnement global repose sur deux concepts. Le premier est celui des sommes de contrôle et des hashs. Chaque archive Debian comprend un fichier `Release`. Celui-ci contient les sommes de contrôle MD5 pour différents fichiers importants dont `Packages.gz` ou `Sources.gz` :

```
964ffa005e2af942b784271935d5c47a5598a1b5 14939855 main/binary-i386/Packages
cb267d1a043eb68f754de5eb08b00b38cb830828  4534669 main/binary-i386/Packages.gz
2008097d54491ae3131a3487ddf17496c2ab3c39       95 main/binary-i386/Release
```

La somme de contrôle MD5 permet de vérifier l'intégrité des fichiers, mais il ne s'agit pas d'un hash cryptographique. Voilà pourquoi les hashs SHA1 et SHA256 sont également présents dans le fichier `Release`. On notera que les sommes MD5 ne sont plus utilisées par le système APT étant donné la faiblesse de l'algorithme, les problèmes de collision et le simple fait que MD5 est depuis longtemps censé être remplacé par SHA1.

Les sommes MD5 et les hashs SHA et SHA256 sont également présents dans les méta-informations des paquets :

```
% apt-cache show vim
Package: vim
Priority: optional
Section: editors
Installed-Size: 1408
[...]
MD5sum: cd291606a06d9c4b92c9e3b75f701353
SHA1: ae76724ab28a61fbb3311502d173bff8d714e18d
SHA256: 7aaac6985f610b8c3da29a36e181cdc5a2d4318a5994d257686
85b5c3c971844
[...]
```

Enfin, assurer l'intégrité est une chose, assurer la provenance de paquets, une autre. Pour ce faire, le fichier `Release` est accompagné de `Release.gpg` qui est le fichier de signature GnuPG. C'est le second concept mis en œuvre. Peut-être avez-vous eu un message comme celui-ci dernièrement, suite à un `apt-get update` :

```
[...]
Lecture des listes de paquets... Fait
W: Aucune clé publique n'est disponible pour la/les clé(s)
suivante(s):
A70DAF536070D3A1
[...]
```

Le message est clair, vous ne disposez pas de la clef publique permettant de vérifier la signature de l'archive. Il vous faut donc récupérer cette clef avec GnuPG :

```
% gpg --recv-keys 6070D3A1
% gpg --list-key 6070D3A1
pub   1024D/6070D3A1 2006-11-20 [expire: 2009-07-01]
uid    Debian Archive Automatic Signing Key
         (4.0/etch) <ftpmaster@debian.org>
```

Puis exporter une version ASCII de la clef pour la « tuyauter » vers `apt-key` :

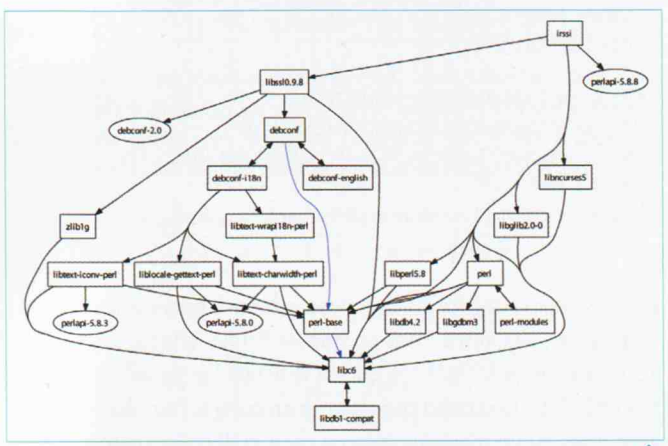
```
% gpg --export -a 6070D3A1 | sudo apt-key add -
OK
```

Dans le cas présent, le problème ne se pose pas si le paquet `debian-archive-keyring` est installé et à jour. Cependant, pour des archives non officielles, il peut être nécessaire d'ajouter de la sorte la clef publique.

Pour l'heure, la chaîne est la suivante :

- L'authenticité de `Release` peut être vérifiée grâce à sa signature GnuPG.
- On peut vérifier l'intégrité des fichiers `Packages` grâce aux hashs et les hashs sont forcément de confiance, puisqu'ils proviennent du fichier `Release`.
- Les fichiers `Packages` contiennent les hashs pour les différents paquets.

Il est donc très facile de vérifier chaque paquet et c'est précisément ce que fait la nouvelle version d'APT. La commande `apt-key` vous permet de gérer les différentes clefs disponibles. En installant `debian-archive-keyring`, vous importez directement les clefs publiques Debian dans la liste. Vous pouvez à tout moment consulter cette dernière avec `apt-key list`.



APT-RDEPENDS ANALYSE DES DÉPENDANCES

La sortie fournie par `apt-cache show` n'est pas toujours des plus faciles à lire en termes d'analyse des dépendances. Cet avis est également celui de Simon Law qui a donc décidé d'écrire un petit outil permettant de déterminer plus simplement les dépendances : `apt-rdepends`.

Le plus simple et le plus convaincant est encore de vous montrer un exemple :

```
% apt-rdepends bc
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
bc
  Dépend: libc6 (>= 2.3.2.ds1-4)
  Dépend: libncurses5 (>= 5.4-1)
  Dépend: libreadline4 (>= 4.3-1)
libc6
  Dépend: libdb1-compat
libdb1-compat
  Dépend: libc6 (>= 2.2.5-13)
libncurses5
  Dépend: libc6 (>= 2.3.2.ds1-4)
libreadline4
  Dépend: libc6 (>= 2.3.2.ds1-4)
  Dépend: libncurses5 (>= 5.4-1)
```

Non seulement `apt-rdepends` vous liste les dépendances de `bc`, mais également, récursivement, les dépendances des dépendances. Vous obtenez une liste complète et claire des liens existant entre les paquets.

L'option `-r` vous permet de lister les dépendances inverses ou, autrement dit, la liste complète des paquets qui ont besoin de celui spécifié en argument. Autant vous dire qu'il n'est ni utile, ni très rusé de tenter l'expérience avec le paquet `libc6`. Plus utile, l'option `-p` permet d'ajouter à la sortie une mention d'état (`Installed/NotInstalled`). Un `grep` sur la sortie est une bonne manière d'obtenir rapidement la liste des paquets à installer sans utiliser `apt-get`.

Enfin, dans le registre des options qui ne sont pas vraiment utiles, mais qu'on aime utiliser par curiosité, nous avons `-d` permettant de générer un fichier de graphe au format Dotty. On redirigera la sortie standard vers un fichier pour traiter les données par la suite avec la commande `dotty`. Plus amusant et, sans aucun doute, plus décoratif (mais moins que le poster/calendrier inclus dans ce magazine), nous pouvons directement traiter les données Dotty via `dot` pour produire un graphe PostScript A3 (en orientation paysage) que nous envoyons à l'imprimante par défaut :

```
% apt-rdepends -d irssi | \
 dot -Gsize="14.693,16.535" -Grotate=90 -Gratio=auto -Tps | \
 lpr
```

Disponible chez votre
marchand de journaux
et sur
http://www.ed-diamond.com

GNU LINUX MAGAZINE / FRANCE

En KIOSQUE

NUMERO 90

AU SOMMAIRE

NETTOYEZ VOTRE SYSTÈME

Au fil de la vie d'un système et dès les premières phases d'utilisation et de personnalisation, un phénomène se met en place : la prise de poids. Application inutilisée, fichier de configuration non purgé et données rémanentes finissent par prendre beaucoup de place.

Dans les magazines féminins, les conseils minceur et autres techniques de régime arrivent après les fêtes. Ici, ce sera avant. Preuve que GLMF n'est pas un vrai féminin ;)

Trêve de plaisanteries, l'heure est grave. Votre système est installé et mis à jour depuis des années, vous avez testé bien des paquets, changé de gestionnaire de fenêtres plusieurs fois et même oublié qu'un jour vous avez développé des applications en XXX (remplacez XXX par le langage dont vous avez tout oublié depuis). Au fil du temps, votre système Debian dispose de moins en moins d'espace disponible et `/usr` atteint les 4 Go !

Le système de gestion de paquets permet non seulement les installations/désinstallations propres, mais également l'analyse des dépendances et le suivi des applications et des configurations.

DEBORPHAN

Je l'ai expliqué en introduction du magazine, l'installation des paquets nécessaires au bon fonctionnement d'une application est automatique. Inversement, la suppression d'un des paquets nécessaires entraîne la suppression de l'application. Un autre type de situations peut se présenter : l'installation de l'application et des dépendances, puis la suppression de l'application. Ici, tous les paquets supplémentaires installés restent sur le système, car il n'y a pas de dépendance inverse.

Ainsi, à moins de suivre vous-même les dépendances lors de l'installation, vous ne savez pas nécessairement, des semaines plus tard, si tel paquet sert encore à quelque chose ou non.

Heureusement, Cris van Pelt a créé un petit utilitaire appelé `deborphan` (à présent maintenu par Peter Palfrader). Celui-ci permet de lister les paquets des sections `oldlibs` et `libs` vers lequel il n'existe pas de dépendance. Il s'agit donc de bibliothèques partagées qui ne sont plus nécessaires à aucune application empaquetée. Notez bien cette petite précision : le système de gestion de paquets est incapable d'avoir connaissance des applications installées manuellement. Si vous installez ou copiez des binaires manuellement sur le système, ceux-ci peuvent être dépendants de bibliothèques installées avec `apt-get` ou `dpkg`. Les retirer peut donc poser problème, mais là, c'est à vous de gérer.

L'utilisation de `deborphan` est très simple, puisqu'il suffit d'invoquer la commande pour obtenir la liste des paquets « orphelins» :

```
% deborphan
libzzip-0-12
libgnomecups1.0-1
libxul0d
libgnomeprintui2.2-common
libengine-pkcs11-openssl
```

Ce premier exemple est très intéressant, car il permet de voir un cas particulier. `deborphan` liste cinq paquets qui semblent inutiles. C'est parfaitement exact pour les quatre premiers, mais `libengine-pkcs11-openssl` présente une particularité. Il s'agit d'une bibliothèque dynamique requise par aucun paquet particulier et donc détectée comme inutile. Ce paquet contient cependant un greffon (*plugin*) pour OpenSSL permettant d'accéder aux périphériques répondant à la norme PKCS#11. Il n'est pas nécessaire au fonctionnement d'OpenSSL, mais peut l'être pour votre système (accès aux données X509 d'une *smartcard*).

Nous pouvons ensuite utiliser `deborphan` directement avec `apt-get` tout en prenant en compte le cas particulier :

```
% sudo apt-get remove --purge `deborphan -e libengine-
pkcs11-openssl`
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Les paquets suivants seront ENLEVÉS:
  libengine-pkcs11-openssl* libgnomecups1.0-1*
  libgnomeprintui2.2-common* libp11-0* libxul0d*
  libzzip-0-12*
0 mis à jour, 0 nouvellement installé, 6 à enlever et 734
non mis à jour.
Il est nécessaire de prendre 0o dans les archives.
Après dépaquetage, 22.8Mo d'espace disque seront libérés.
```

Attention au piège ! Le fait d'ignorer, via l'option `-e`, le paquet que nous souhaitons exclure provoque un effet de bord. Le paquet `libp11-0` nécessaire à `libengine-pkcs11-`

openssl est alors détecté comme inutile. Mais l'utilisation d'apt-get remove provoque sa suppression et donc celle de libengine-pkcs11-openssl.

La solution consiste à utiliser les options de gestion des paquets à conserver. deborphan est ainsi capable de maintenir une liste des paquets importants à ne jamais considérer comme orphelins. Ici nous utiliserons :

```
% sudo deborphan -A libengine-pkcs11-openssl
```

puis vérifions avec :

```
% deborphan -L
libengine-pkcs11-openssl
```

A présent, deborphan réagit comme espéré :

```
% deborphan -z
       72 libzzip-0-12
      664 libgnomecups1.0-1
    19828 libxul0d
     1416 libgnomeprintui2.2-common
```

Notez l'utilisation de l'option -z dans cette commande, permettant d'afficher la taille en kilooctets des paquets détectés.

La liste est maintenue de manière « interne » à deborphan. Il est possible d'y ajouter des éléments (-A, de les lister (-L), de supprimer des éléments (-R) ou encore de purger la liste (-Z). Autre option intéressante, nous pouvons spécifier un fichier contenant la liste avec -k :

```
% echo libengine-pkcs11-openssl > ~/non_orphan

% deborphan -k ~/non_orphan
libzzip-0-12
libgnomecups1.0-1
libxul0d
libgnomeprintui2.2-common

% deborphan -k ~/non_orphan -A libzzip-0-12

% deborphan -k ~/non_orphan
libgnomecups1.0-1
libxul0d
libgnomeprintui2.2-common

% cat ~/non_orphan
libengine-pkcs11-openssl
libzzip-0-12
```

Comme vous pouvez le voir, le fichier spécifié après -k peut être manipulé avec les options -A, -R et -Z ou un éditeur de texte. En ne spécifiant pas de fichier, c'est /var/lib/deborphan/keep qui est utilisé.

NOTE

La désinstallation de paquets orphelins peut rendre orphelins d'autres paquets.

En guise de conclusion en ce qui concerne deborphan, voici un petit script Bash permettant d'afficher la taille totale des paquets orphelins :

```
#!/bin/bash

sum=0;
for i in `deborphan -z | sed "s/^ *//;s/ .*//"`
do
  sum=$(( $sum + $i ))
done
echo $sum Ko
```

PURGE DES FICHIERS DE CONFIGURATION

L'installation et la mise à jour d'applications, en plus que tout ce que cela implique en termes de dépendances, doit prendre en compte la distinction entre l'application elle-même, la documentation et les fichiers de configuration. Ce dernier élément est très important. La configuration des outils, démons et applications est une tâche souvent fastidieuse et longue. Il ne serait pas acceptable de perdre tout ce temps et cette énergie lors d'une mise à jour.

C'est pourquoi le créateur et responsable d'un paquet doit spécifier l'emplacement et le nom du ou des fichiers de configuration. On retrouve cette information dans le fichier conffiles de l'archive control.tar.gz placée dans un paquet .deb. Ainsi, le système de gestion est en mesure de mettre à jour un paquet sans modifier ces fichiers. Mieux encore, il est capable de détecter d'éventuelles modifications apportées par l'utilisateur, émettre un avertissement et demander quel comportement adopter. Vous l'avez sans doute déjà constaté lors d'un apt-get upgrade.

La distinction entre l'application elle-même et les fichiers de configuration permet également de supprimer les binaires tout en conservant la configuration. Ainsi, une réinstallation ultérieure du paquet évite de reconfigurer l'ensemble sur le système hôte. C'est le comportement par défaut de la commande apt-get. Si l'on souhaite également désinstaller la configuration, il faut ajouter l'option --purge. Dans le cas d'une désinstallation automatique via la gestion des dépendances, il n'est pas rare de laisser « traîner» la configuration sur le système.

Les fichiers de configuration sont habituellement de simples fichiers textes formatés et n'occupent donc pas beaucoup de place. Cependant, leur présence pour un grand nombre de paquets désinstallés peut être dérangeant et ajouter de la confusion. dpkg permet de lister les paquets installés ou en partie. Ainsi, en n'affichant que les paquets marqués rc, on peut lister les configurations orphelines :

PAQUETS

```
% dpkg -l | grep ^rc
rc libgii0          0.9.1-0.2     General Input Interface
runtime libraries
rc mplayer-custom   1.0-pre8-0.0  The Ultimate Movie Player
For Linux
```

Les configurations peuvent être purgées en utilisant l'option `-P` de `dpkg`. Ainsi, par extension, on peut utiliser le script suivant :

```
for pkg in $(dpkg -l | grep ^rc | cut -d " " -f 3)
 do
  dpkg -P $pkg
done
```

Les fichiers de configuration ne sont pas les seuls éléments de données qui sont purgés avec l'option `--purge` d'`apt-get`. Certains paquets installent, par exemple, des utilisateurs ou des groupes. D'autre part, l'utilisation de cette option n'implique pas forcément la suppression de la configuration. Ainsi, en remplacement ou en complément à `dpkg -l | grep ^rc`, on peut utiliser `deborphan --find-config` qui couvre un plus vaste nombre de possibilités (mais implique l'installation de `deborphan` qui a besoin, par exemple, de `dialog`).

`dpkg` et `grep` liste les paquets à purger, `cut` récupère leur nom et `dpkg -P` les purge. Mais là encore, l'automatisation complète n'est parfois pas souhaitable et mieux vaudra archiver les fichiers de configuration auparavant ou procéder manuellement.

Enfin, l'utilisation de `dpkg -L` suivi du nom du paquet permet de connaître l'emplacement et le nom du ou des fichiers de configuration :

```
% deborphan --find-config
lessdisks

% dpkg -L lessdisks
/etc/lessdisks-install.conf
```

POPULARITY-CONTEST

Je l'ai dit plus haut, plus un système est vieux, plus l'espace disque se réduit naturellement. Nous venons de couvrir des problèmes propres à la gestion des paquets, mais il nous reste à régler le problème le plus important : celui entre la chaise et le clavier.

Un bon système gestion de paquets permet de tester rapidement n'importe lequel d'entre eux. Cela permet également d'en installer un très grand nombre en se disant allègrement « je désinstallerai ça plus tard ». Et le paquet demeure sur le système et finit par être oublié.

Un projet appelé « *Debian Popularity Contest* » a été mis en place de longue date. Son but est de déterminer quels paquets, en dehors de ceux indispensables au fonctionnement de la distribution, sont importants pour les utilisateurs. A l'heure actuelle, une distribution Debian GNU/Linux dans son intégralité occupe plus de vingt CD-Roms d'installation. Bien que le Net fournisse le média par excellence pour installer GNU/Linux, il est important de pouvoir minimiser le nombre d'images ISO qu'un utilisateur doit télécharger. Mieux encore, il doit être possible d'installer un système viable, mais surtout satisfaisant avec un seul CD. Il faut donc déterminer quels paquets doivent trouver leur place sur ce premier CD d'installation.

Le résultat des travaux se présente sous la forme d'un paquet `popularity-contest` contenant un script installé par défaut. Celui-ci est lancé régulièrement via `crond` et va scanner l'ensemble du système pour maintenir une liste de paquets sous la forme :

```
1163915259 1153751347 dpkg /usr/bin/dpkg-query
1163915259 1155493105 bash /bin/sh
1163915258 1155493112 login /bin/su
```

Les deux premières valeurs sont respectivement la date d'accès et la date de modification au format UNIX du binaire spécifié à la fin de la ligne. Le nom du paquet concerné est indiqué à la troisième position. Le fichier est automatiquement créé de manière hebdomadaire et se trouve dans `/var/log/popularity-contest`. Cette liste est alors envoyée par mail, « anonymisée » et utilisée à des fins statistiques. La configuration de `popularity-contest` se fait via `/etc/popularity-contest.conf` et peut être désactivée en spécifiant `PARTICIPATE="no"`.

Mais il y a mieux encore. Les paquets pour lesquels le fichier de référence n'a pas été accédé depuis plus d'un mois sont marqués, en fin de ligne, par le tag `<OLD>`. Un simple coup d'œil dans le fichier journal ou en sélectionnant les lignes sur la sortie standard lors de l'exécution du script, nous permet d'obtenir un résultat intéressant :

```
1160423386 1143974170 oops /usr/sbin/oops <OLD>
1160423385 1153752326 xinetd /usr/sbin/xinetd <OLD>
1160423383 1155493244 snmpd /usr/sbin/snmpd <OLD>
1160423381 1153752262 smartmontools /usr/sbin/smartctl <OLD>
```

Le format utilisé pour la date est un *timestamp* UNIX, le nombre de secondes écoulées depuis le 1er janvier 1970 à minuit. Il existe plusieurs manières de convertir ce format en une date lisible par un humain. La plus simple est d'utiliser la commande `date` avec :

```
% date -d '1970-01-01 UTC 1160423386 seconds' '+%d/%m/%Y'
09/10/2006
```

ou

```
% date -d '@1160423386' '+%d/%m/%Y'
09/10/2006
```

On peut réutiliser cette technique dans un script Perl de manière à obtenir un outil directement utilisable via l'entrée standard :

```
#!/usr/bin/perl
mainloop: while (<>)
{
  my $ligne=$_;
  @tligne = split(' ', $ligne);
  $date=`date -d '\@$tligne[0]' '+%d/%m/%Y'`;
  chomp($date);
  print "$date $tligne[1]\n"
}
```

Ensuite, un enchaînement de redirection permet d'obtenir une liste lisible et cohérente :

```
% sudo popularity-contest | grep "<OLD>" | \
  cut -d " " -f 1,3 | sort -n -r | \
  perl /chemin/vers/script.pl
24/10/2006 postgresql-7.4
24/10/2006 libssl-dev
21/10/2006 libdbd-sqlite3-perl
12/10/2006 openvpn
[...]
15/08/2003 libfile-chdir-perl
26/04/2003 libdevel-symdump-perl
13/06/2002 libdigest-nilsimsa-perl
14/03/2001 libdigest-hmac-perl
20/09/2000 gettyps
18/08/2000 libio-socket-multicast-perl
28/06/1997 libconvert-binhex-perl
```

Il est recommandé de ne pas se fier aveuglément aux informations retournées par `popularity-contest` et ceci concerne surtout les bibliothèques dynamiques qui ne sont jamais accédées directement. D'autre part, une utilisation maladroite de `touch` et tout est perdu. Il n'en reste pas moins utile de procéder de la sorte et on gagnera sans doute quelques mégaoctets, en particulier avec les jeux.

CRUFT

Il est parfois nécessaire d'installer sur un système des applications qui n'existent pas sous la forme de paquets. Il s'agit, le plus souvent, d'applications ou de SDK propriétaires, mais aussi de suites logicielles et de chaînes de compilation en Logiciel Libre.

Il n'est pas possible d'utiliser les outils de gestion de paquets pour ce type d'installation. A moins d'utiliser des outils comme `stow`, la seule solution consiste à prendre le problème à l'envers. Pour connaître les fichiers installés ou copiés de cette manière, il suffit de lister tous les fichiers ne faisant pas partie d'un quelconque paquet.

C'est précisément ce que permet de faire `cruft`. Son utilisation est simplissime, mais le dépouillage de la sortie nécessitera quelques manipulations et tris.

RECOMPILER ET RECONSTRUIRE SES PAQUETS

La volonté de recompiler ses applications alors qu'elles existent déjà en version binaire sous forme de paquets peut être motivée par plusieurs choses. L'optimisation des binaires et l'analyse des sources sont deux bons exemples.

Pour une machine personnelle, le fait de disposer rapidement de paquets binaires s'installant de manière quasi automatique est un grand avantage. Lorsqu'on parle de serveur en production, la rapidité d'installation compte souvent moins que la qualité des paquets et l'intégration des applications. Un très bon exemple pour montrer les différences en termes de besoins concerne le noyau Linux : modules et image initramfs pour un système *desktop*, mais noyau statique pour un serveur.

Il en va de même avec la recompilation des applications, utilitaires et démons. Utiliser les options d'optimisation du compilateur GNU peut s'avérer très intéressant sans pour autant que cela ne devienne une lubie comme pour d'autres distributions. Il arrive, en effet, un moment où il faut se demander si le temps CPU utilisé pour compiler un nouveau binaire est rentable par rapport au gain obtenu par la mise en œuvre de ce dernier à court terme. En d'autres termes, il n'est pas intéressant de recompiler tous les paquets du système.

LA MÉTHODE CLASSIQUE : DPKG-BUILDPACKAGE

Dans votre fichier `/etc/apt/sources.list`, vous avez sans doute défini des sources `deb` pour les paquets binaires, mais aussi des sources `deb-src`. Ceci permet au système APT de déterminer l'emplacement des différents composants d'un paquet source :

- L'archive originale appelée version « amont » (*upstream version*). Il s'agit de la version des sources telle que développée par les membres du projet d'origine ou le développeur de l'application.
- Le résumé du contenu du code source stocké dans un fichier comportant une extension `.dsc`. Celui-ci intègre différentes informations comme les paquets nécessaires à la construction du paquet binaire ou différentes sommes de contrôle.
- Un fichier compressé généré par `patch` comportant l'extension `.diff.gz`. C'est le fichier « de différences » à appliquer sur les sources en version amont et permettant d'ajouter les modifications Debian (répertoires et fichiers supplémentaires, mais aussi correctifs et mises à jour spécifiques).

Construire un paquet à partir des sources en utilisant les outils de base de Debian est relativement simple, même si le processus se décompose en une succession d'étapes relativement pénibles. Nous verrons par la suite qu'il existe des solutions permettant de simplifier la vie des utilisateurs qui reconstruisent régulièrement ou systématiquement les paquets.

Je vais utiliser un exemple concret pour illustrer mes propos. Le but est de construire, à partir des sources, le paquet contenant `bc` (la calculatrice en ligne de commande). La tâche n'est pas démesurée, loin de là, mais ce qui va être dit est également valable pour recompiler de grosses applications comme The Gimp ou encore Evolution.

Compiler une application avec un système Debian nécessite deux éléments :

- une chaîne de compilation fonctionnelle ;
- l'installation des paquets contenant les fichiers d'en-tête permettant l'utilisation des bibliothèques partagées.

La première étape consiste donc à installer le méta-paquet `build-essential`. Son installation provoquera l'installation des compilateurs C et C++, les en-têtes de la bibliothèque standard C GNU, de `make` et des outils de construction de paquets Debian (`dpkg-dev`).

La seconde étape consiste à installer les dépendances nécessaires à la reconstruction d'un paquet binaire. Tout comme les dépendances entre paquets, ces dépendances sont prises en charge par le système APT. Vous pouvez obtenir la liste des dépendances de construction/compilation avec `apt-cache showsrc` suivi du nom du paquet concerné :

```
% apt-cache showsrc bc
[...]
Maintainer: John Hasler <jhasler@debian.org>
Build-Depends: bison, debhelper (>= 4), file, flex,
  libreadline5-dev | libreadline-dev, texi2html, texinfo
Architecture: any
[...]
```

Pour installer automatiquement ces paquets, il suffit d'utiliser `apt-get build-dep` et le nom du paquet. Enfin, on télécharge les éléments du paquet source avec `apt-get source bc` :

```
% apt-get source bc
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Nécessité de prendre 387ko dans les sources.
Réception de: 1 http://ftp.fr.debian.org testing/main bc 1.06-20 (dsc) [736B]
Réception de: 2 http://ftp.fr.debian.org testing/main bc 1.06-20 (tar) [279kB]
Réception de: 3 http://ftp.fr.debian.org testing/main bc 1.06-20 (diff) [107kB]
387ko réceptionnés en 0s (391ko/s)
```

`apt-get` télécharge les trois fichiers nécessaires en fonction des informations présentes dans le fichier `Sources`, lui-même lu grâce aux informations de votre `sources.list`.

```
gpg: Signature faite le sam 07 oct 2006 19:39:32
  CEST avec la clé RSA ID 0AC5BE61
gpg: Impossible de vérifier la signature:
  clé publique non trouvée
```

Le contenu du fichier `bc_1.06-20.dsc` est authentifié grâce à sa signature GnuPG. Ici, cette vérification échoue, car nous ne possédons pas la clef publique du responsable du paquet dans notre trousseau. Un simple `gpg --recv-key 0AC5BE61` et le message n'apparaît plus, preuve que la signature est bonne. Le fichier `bc_1.06-20.dsc` contient également les sommes de contrôle des fichiers `bc_1.06.orig.tar.gz` et `bc_1.06-20.diff.gz`.

```
dpkg-source: extracting bc in bc-1.06
dpkg-source: unpacking bc_1.06.orig.tar.gz
dpkg-source: applying ./bc_1.06-20.diff.gz
```

Enfin, les sources en version amont sont désarchivés dans un répertoire `bc-1.06` et le patch Debian est appliqué. Nous obtenons, au final, une version Debian des sources dans ce répertoire. À partir de ce moment, nous sommes en mesure de construire un paquet binaire à partir de ces données.

Remarquez qu'`apt-get source` fait un certain nombre de manipulations à votre place. S'il vous arrive d'endommager le contenu du répertoire des sources Debian, il n'est pas nécessaire de refaire appel à `apt-get`. Vous pouvez, tout simplement, supprimer le répertoire en question, puis utiliser :

```
% dpkg-source -x bc_1.06-20.dsc
dpkg-source: extracting bc in bc-1.06
dpkg-source: unpacking bc_1.06.orig.tar.gz
dpkg-source: applying ./bc_1.06-20.diff.gz
```

La construction à proprement parler se fait avec l'outil `dpkg-buildpackage`. Il suffit d'entrer dans le répertoire des sources et d'utiliser :

```
% dpkg-buildpackage -b -rfakeroot
dpkg-buildpackage: source package is bc
dpkg-buildpackage: source version is 1.06-20
[...]
dh_testdir
./configure --prefix=/usr --with-readline
```

`dpkg-buildpackage` se charge de tout. Il lance la configuration des sources selon les besoins spécifiés dans les fichiers de configuration du paquet, puis passe à la compilation proprement dite :

```
[...]
/usr/bin/make CFLAGS="-O2 -g -Wall -D_POSIX_SOURCE -DDOT_IS_LAST"
make[1]: entrant dans le répertoire " /mnt/six4/temp/bc-1.06 "
/usr/bin/make  all-recursive
make[2]: entrant dans le répertoire " /mnt/six4/temp/bc-1.06 "
Making all in lib
make[3]: entrant dans le répertoire " /mnt/six4/temp/bc-1.06/lib "
if gcc -DHAVE_CONFIG_H  -I. -I. -I.. -I. -I.. -I./../h
  -g -O2 -Wall -funsigned-char -O2 -g -Wall -D_POSIX_SOURCE
  -DDOT_IS_LAST -MT getopt.o -MD -MP -MF ".deps/getopt.Tpo"
[...]
make[3]: quittant le répertoire " /mnt/six4/temp/bc-1.06/dc "
Making all in doc
make[3]: entrant dans le répertoire " /mnt/six4/temp/bc-1.06/doc "
make[3]: Rien à faire pour " all ".
```

En fin de processus, divers outils propres à la distribution Debian sont utilisés pour vérifier l'état des données, nettoyer les sources ou encore configurer les documentations :

```
[...]
dh_testdir
dh_testroot
dh_clean -k
[...]
dh_builddeb -pbc
dpkg-deb: construction du paquet "bc" dans "../bc_1.06-
20_i386.deb".
dh_testdir -pdc
```

La construction du paquet est la dernière étape :

```
[...]
dpkg-deb: construction du paquet "dc" dans "../dc_1.06-20_i386.deb".
dpkg-genchanges -b
dpkg-genchanges: envoi d'un binaire - aucune inclusion de code source
dpkg-buildpackage: binary only upload (no source included)
 signfile ../bc_1.06-20_i386.changes
```

Si vous disposez d'une installation fonctionnelle de GnuPG, la phrase de passe protégeant votre clef privée vous sera demandée afin de signer le fichier bc_1.06-20_i386.changes.

Nous avons ici utilisé deux options avec dpkg-buildpackage. La première, -b, permet de demander uniquement la construction d'un paquet binaire. En l'absence de cette option, de nouveaux fichiers sources seront fabriqués. Bien entendu, l'archive des sources amont ne changera pas, mais le patch et le fichier de description (.dsc) seront mis à jour. Il n'est habituellement pas nécessaire de procéder de la sorte, sauf si vous devenez le nouveau responsable du paquet. On remarquera, à ce propos, que la procédure de signature du fichier de description ne fonctionnera pas étant donné que vous ne disposez pas de la clef publique du responsable officiel (John Hasler pour ce qui concerne le paquet bc).

Enfin, l'option -r et l'argument fakeroot permettent de construire le paquet sans être super-utilisateur. En effet, le paquet binaire contient une arborescence qui sera copiée sur la racine du système de fichier lors de l'installation. Il faut donc, si nécessaire, pouvoir donner aux fichiers de cette arborescence les privilèges et « appartenances » au root. C'est précisément ce que permet de faire fakeroot. L'option -r sert simplement à préciser le programme permettant d'obtenir les privilèges root.

Au terme du processus, vous trouverez le ou, dans le cas présent, les paquets binaires dans le répertoire parent du répertoire des sources. Il vous suffira alors de lancer l'installation avec dpkg -i et le nom du ou des fichiers .deb :

```
% sudo dpkg -i ../bc_1.06-20_i386.deb
(Lecture de la base de données...
  131920 fichiers et répertoires déjà installés.)
Préparation du remplacement de bc 1.06-20
  (en utilisant ../bc_1.06-20_i386.deb) ...
Dépaquetage de la mise à jour de bc ...
Paramétrage de bc (1.06-20) ...
```

SIMPLIFIEZ-VOUS LES CHOSES AVEC APT-SRC

La reconstruction d'un paquet binaire à partir des sources Debian via la méthode classique peut être vite lassante. Pour nous faciliter les choses, l'utilitaire apt-src a été créé par Joey Hess. L'un de ses principaux avantages est de pouvoir être utilisé par un utilisateur standard. Comme vous l'aurez compris en lisant ce qui vient d'être détaillé, l'installation des sources d'un paquet n'est pas dépendant d'une arborescence prédéfinie comme pour les paquets binaires (/var/cache/apt et /var/lib/apt).

NOTE

Certaines actions d'apt-src nécessitent le passage en root. L'utilitaire gère automatiquement cela en vous demandant votre mot de passe (si vous êtes dans la liste des sudoers).

apt-src automatise donc tout simplement les étapes décrites plus haut. La commande apt-src install bc (après un apt-src update) donnera les mêmes résultats qu'apt-get source bc, installant dans le répertoire courant une arborescence de sources patchés prêts à être compilés. apt-src prend également en charge l'installation des dépendances binaires et de construction. Ainsi, le simple fait de demander l'installation des sources d'un paquet provoque l'installation des dépendances et des paquets -dev nécessaires.

Immédiatement après, et sans changer de répertoire, nous utiliserons la commande apt-src build bc pour obtenir nos deux paquets bc et dc que nous pourrons installer dans la foulée (via sudo).

NOTE

On peut également utiliser l'option -i permettant d'installer automatiquement le ou les paquets construits. A utiliser avec précaution cependant. Comme dans le cas de bc, cette option utilisée avec des paquets sources produisant plusieurs paquets binaires provoquera l'installation de tous les paquets binaires générés (ici bc, mais aussi dc). Ce qui n'est pas nécessairement souhaitable.

Mais le principal avantage d'apt-src est le suivi des sources installés. En effet, n'ayant pas d'emplacement spécifiquement dédié, les arborescences sources peuvent rapidement se retrouver un peu partout dans le système de fichiers. En particulier, si nous n'êtes pas particulièrement rigoureux dans vos manipulations. Ainsi, apt-src list vous permettra d'obtenir les emplacements de toutes les arborescences sources déjà installées :

```
% apt-src list
i bc    1.06-20  /mnt/temp/bc-1.06
i wcalc 2.2.2-2  /home/denis/DEB/wcalc-2.2.2
```

On pourra supprimer une arborescence en utilisant apt-src remove et le nom du paquet. L'action version suivie du nom d'un paquet permettra d'afficher le numéro de version des sources installés.

Enfin, on notera la possibilité de renseigner `apt-src` sur la présence d'une arborescence installée indépendamment avec l'action `import`. L'action `upgrade` permettra de mettre à jour l'ensemble des arborescences sources installées. `apt-src` devient rapidement plus agréable à utiliser que le couple `apt-get source` et `dpkg-buildpackage`. Mais si vous êtes un accro de la reconstruction, il y a peut être mieux...

ENCORE PLUS SIMPLE : APT-BUILD

Alors qu'`apt-src` permet « d'en mettre un peu partout », l'utilitaire `apt-build` est bien plus rigoureux et installe une hiérarchie qui lui est propre dans `/var`. Ainsi, les paquets sources seront par défaut téléchargés, placés et construits dans `/var/cache/apt-build/build`. Ceci peut-être modifié lors de l'installation d'`apt-build`. L'outil de configuration des paquets (`debconf`) vous demandera de préciser certains éléments à ce moment.

Ainsi, vous serez amené à préciser (ou laisser les valeurs pas défaut) différents éléments de configuration :

- Le répertoire de stockage des arborescences sources.
- Le répertoire de stockage des paquets binaires produits.
- Le niveau d'optimisation entre `-O1` et `-O3`. Un haut niveau d'optimisation provoque un temps de compilation plus important, mais aussi, parfois, certains problèmes de stabilité.
- L'ajout ou non d'une ligne `deb file:/var/cache/apt-build/repository apt-build main` dans votre `sources.list` (notez que la configuration ne prend pas en compte la possibilité de répartir les sources dans des fichiers `/etc/apt/sources.list.d/`.
- Les options et arguments à utiliser avec le compilateur GNU `gcc` (voir ci-après).
- Les options à utiliser pour `make`.
- L'architecture de base du système parmi plusieurs options possibles qui apparaîtront en fonction du CPU détecté. Notre système de test est un AMD. Le système de configuration nous propose donc, k6, athlon-tbird, athlon-mp, athlon-fx, etc. Si votre architecture n'est pas listée, vous devrez en choisir une quelconque, puis éditer le fichier `/etc/apt/apt-build.conf` manuellement et, en particulier, la ligne `mtune`. Il s'agit en réalité de l'option `-mtune` passée à `gcc` et permettant de spécifier un paramètre réglant l'optimisation de l'ordonnancement des instructions pour un processeur donné. Le compilateur est, en effet, capable de produire un code binaire qui sera mieux ordonnancé via l'utilisation des pipelines. Le gain de performance peut être, selon les applications, tout à fait significatif.

Revenons un instant sur l'option `-mtune`. Précisons d'emblée que, dixit la documentation GCC, l'option `-mcpu` est obsolète et synonyme de `-mtune` pour les architectures i386 et x86-64. Il est important de le préciser, car certaines versions des pages de manuel ne sont pas à jour. Le point important est de bien comprendre que cette option n'est pas dépendante de l'architecture dans le sens où elle ne casse pas la compatibilité. Un fichier source compilé avec l'option `-mtune=athlon-xp` fonctionnera parfaitement sur un i586.

Nous avons également l'option `-march`. Celle-ci permet d'obtenir des binaires utilisant les instructions spécifiques d'un processeur ou d'une famille de processeurs. MMX, SSE, SSE2, 3dNOW!, *enhanced* 3dNOW! ou encore SSSE3 sont des sets d'instructions. SSSE3, par exemple, est spécifique aux Intel Xeon et Intel Core 2 (mais il faudra attendre GCC version 4.3 pour les utiliser). Pour autoriser le compilateur GNU à se servir de ces jeux d'instructions spécifiques, il suffit de préciser l'architecture avec `-march`. Notez que l'option implique automatiquement un `-mtune` identique.

On peut également jouer sur les deux options pour obtenir un binaire, par exemple, capable de fonctionner sur toute architecture supportant les instructions MMX (`-march=pentium-mmx`), mais optimisé pour l'ordonnanceur d'un Athlon XP (`-mtune=athlon-xp`. Il en résultera un binaire optimisé pour Athlon XP qui ne pourra fonctionner que sur processeurs MMX. Dans la pratique, cela reste très dépendant du code source. Un simple `hello World` fonctionnera sans doute très bien sur un Pentium même compilé avec `-march=athlon-xp` (non testé).

Remarquez que c'est bien `-mtune` qui est utilisé par `apt-build`. L'option `-march` a été retirée avec la version 0.11.5, car elle posait des problèmes sur certaines architectures. En effet, par exemple, il ne semble pas y avoir de valeur pour cette option concernant les architectures PPC. On utilise alors `-mcpu` (eh oui !), qui reste valable pour cette architecture.

Pour obtenir des binaires optimisés et spécifiques à votre machine, il est donc conseillé d'utiliser `-march` en guise d'option pour `gcc` et s'assurer que `-mtune` soit vide ou identique.

Pour connaître les bonnes options, commencez par déterminer avec exactitude le modèle de processeur de votre machine :

```
% cat /proc/cpuinfo
processor  : 0
vendor_id  : AuthenticAMD
cpu family : 6
model              : 8
model name : AMD Athlon(tm) XP 2400+
stepping   : 1
cpu MHz            : 1993.744
cache size : 256 KB
[...]
```

Les indications importantes sont `cpu family` et `model`. Ensuite, le plus simple est de consulter les documentations des distributions basées fortement sur les sources. Gentoo est un excellent exemple : **http://fr.gentoo-wiki.com/HOWTO_CFLAGS**. De là, on tire la ligne d'options idéale pour `gcc` : `-march=athlon-xp -pipe -fomit-frame-pointer` (l'option `-O2` est spécifiée par ailleurs dans la configuration d'`apt-build`).

Votre `apt-build` est prêt. Il ne vous reste plus qu'à lancer la construction et l'installation du paquet avec :

```
% sudo apt-build install bc
--> Installing build dependencies (for bc) <--
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
[...]
--> Updating package lists <--
[...]
--> Downloading source bc (1.06-20) <--
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
[...]
--> Building bc <--
dpkg-buildpackage: source package is bc
dpkg-buildpackage: source version is 1.06-20
[...]
-> Cleaning up object files <--
Cleaning in directory .
dh_testdir
[...]
--> Moving packages to repository <--
--> Building repository <--
[...]
Les NOUVEAUX paquets suivants seront installés:
  bc
0 mis à jour, 1 nouvellement installés, 0 à
  enlever et 110 non mis à jour.
Il est nécessaire de prendre 0o/66.2ko dans les archives.
Après dépaquetage, 193ko d'espace disque
  supplémentaires seront utilisés.
Sélection du paquet bc précédemment désélectionné.
Dépaquetage de bc (à partir de .../archives/bc_1.06-20_i386.deb) ...
Paramétrage de bc (1.06-20) ...
```

En une seule commande, les sources sont téléchargés et compilés avec nos options, les paquets binaires sont créés et le paquet demandé est installé. Notez que `bc` et `dc` sont créés, mais que seul `bc` est installé sur le système.

! ATTENTION

Il est important de noter que les lignes qui défilent lors de la compilation des sources ne semblent pas faire mention de nos options d'optimisation. Comme le précise le `README.Debian`, c'est parfaitement normal. Les lignes que vous voyez sont les commandes utilisées par `make`, mais le wrapper fait son travail et les binaires sont bien produits tels qu'attendu.

Un autre élément important avant la grosse surprise concerne la priorité des paquets. Il est important de placer dans son `/etc/apt/preferences` quelques lignes comme :

```
Package: *
Pin: release a=apt-build
Pin-Priority: 800
```

Ainsi, vous serez sûr que les paquets construits sont bien ceux qui seront installés automatiquement. Vous pouvez, bien entendu, utiliser l'action `build-source` en lieu et place d'`install` pour installer ensuite le ou les paquets présents dans `/var/cache/apt-build/repository` avec `dpkg -i`. Mais vous perdez alors tout l'avantage du système de gestion de dépendances.

Mais venons-en à la partie la plus amusante. Vous n'êtes pas satisfait du manque d'optimisation et vous êtes à deux doigts de reconstruire un par un tous les paquets installés ? `apt-buid` a une option pour vous : `apt-build world`. Cette commande magique doit être précédée de la génération d'une liste de paquets avec :

```
% dpkg --get-selections | \
awk '{if ($2=="install") print $1}' \
> /etc/apt/apt-build.list
```

Il vous faut ensuite supprimer de cette liste de paquets ceux qu'il n'est pas souhaitable de reconstruire. Soit parce que c'est inutile, comme le compilateur GCC qui utilise son propre compilateur, soit parce que Debian prévoit un autre système, comme pour le noyau Linux. Il existe également des paquets `non-free`, comme `acroread`, pour lesquels on ne dispose pas des sources.

Ceci fait, vous n'avez qu'à utiliser `apt-build world` et faire preuve de beaucoup de patience. Au terme du processus, vous devrez avoir, sur votre système, tous les paquets spécifiés dans la liste en version recompilée et optimisée pour votre architecture.

Avec ce type d'utilisation d'`apt-build`, on se rapproche clairement des optimisations de binaires, telles qu'on en trouve dans des distributions comme Gentoo. On notera cependant qu'on est loin de la souplesse des outils de ce type de distributions, et pour cause, Debian n'est pas une distribution source.

Disponible chez votre
marchand de journaux
et sur
http://www.ed-diamond.com

misc

Multi-System & Internet Security Cookbook

En KIOSQUE

NUMERO 29

DEBTAGS : DES ÉTIQUETTES SUR LES PAQUETS

Comment trouver un paquet qui répond à vos besoins dans une distribution ? La prochaine distribution Debian Stable sera sans doute constituée de quelques 22 CD (ou 3 DVD) et les outils de recherche basés sur les descriptions de paquets montrent déjà leurs limites. Il faut trouver une autre solution.

Quel est le problème ? La réponse tient en une seule commande :

```
% apt-cache stats
Nombre total de paquets : 29777 (1191k)
  Paquets ordinaires : 23139
  Paquets entièrement virtuels : 1092
  Paquets virtuels simples : 1043
  Paquets virtuels mixtes : 554
  Manquants : 3949
```

29777 paquets, soit une meule de foin énorme où chaque utilisateur cherche son aiguille. Les outils comme `apt-cache search` bien que très utiles ne sont pas conçus pour trouver un paquet autrement que par leur nom ou un mot qui puisse se trouver dans leur description. Comment trouver, en effet, tous les éditeurs de texte en mode console sans en connaître le nom ? Vous me direz qu'il suffit de faire `apt-cache search text editor`. Cela fonctionne, nous obtenons une liste, mais le résultat n'est pas celui attendu.

Une première solution pour régler le problème est celle jusqu'alors utilisée. Elle consiste à faire usage de sections et de classer les paquets à l'intérieur. Il existe une trentaine de sections définies arbitrairement. Elles permettent par exemple à l'outil `deborphan` de fonctionner en analysant les dépendances des paquets pour les sections « oldlibs » et « libs ».

Cependant, le système de sections n'est pas suffisant. Non seulement un paquet ne peut se trouver que dans une seule section, mais le nombre de sections est insuffisant. 30000 paquets dans 30 sections nous donnent 1000 paquets par section, ce qui n'avantage pas l'utilisateur. Pire encore, en augmentant le nombre de sections (si c'est possible), nous réduisons certes le nombre de paquets pour chacune d'elles, mais comment l'utilisateur fera-t-il pour choisir parmi 150 sections de 200 paquets ?

Cet état de fait est tellement évident qu'un rapport de bogue de niveau « *wishlist* » a été introduit dans le BTS Debian il y a fort longtemps. Le bogue 144046 date de janvier 2002.

LA SOLUTION VENUE D'INDE

Enrico Zini, développeur Debian, s'est penché sur le problème et a fait une découverte très intéressante : les problèmes de classification de logiciels sont identiques aux problèmes de classification des livres. Or, justement, les bibliothèques ne datant pas d'hier, un grand nombre de recherches scientifiques à propos de la classification des ouvrages ont été faites. Si les problèmes sont les mêmes, les solutions le sont également. Comme le fait remarquer avec humour Enrico Zini dans l'un de ces documents présentant sa solution, un mathématicien bibliothécaire indien a initié un projet secret de classification des logiciels pour Debian dès 1931. Il n'a simplement pas utilisé les mots « logiciels » et « Debian », mais « livres » et « bibliothèque » (sans doute pour coder ses travaux !). Ce mathématicien bibliothécaire est Shiyali Ramamrita Ranganathan. Enrico Zini a eu l'idée d'appliquer les cinq lois de Ranganathan (voir encart) aux paquets/logiciels d'une distribution Debian :

1. Les logiciels sont faits pour être utilisés. Ceci signifie que les utilisateurs souhaitant se servir des logiciels doivent pouvoir avoir connaissance de leur existence. Si un logiciel n'est pas connu, il est inutile et cela viole cette première loi.

2. A chaque utilisateur son logiciel. L'utilisation de tous les logiciels n'est pas appropriée à tous les utilisateurs. L'utilisateur ne doit pas simplement trouver un logiciel, il doit trouver le logiciel qui lui convient.

3. A chaque logiciel son utilisateur. Inversement un logiciel ne peut s'adapter à tout utilisateur. La relation est à double sens.

4. Épargnons le temps de l'utilisateur. L'utilisateur ne doit pas passer du temps à trouver le logiciel dont il a besoin ni à apprendre le fonctionnement du système de classification plus que nécessaire. Le but est de trouver le logiciel, pas de devenir un spécialiste du classement.

5. Une distribution est un organisme en développement. Une distribution n'est pas un système figé. Elle évolue en permanence et les règles de classement des logiciels doivent pouvoir s'adapter à cette évolution.

NOTE

Il est amusant de remarquer que ces lois qui peuvent donc s'appliquer aux logiciels d'une distribution fonctionnent également parfaitement avec les sites Web. Le WWW peut être comparé, en effet, à une gigantesque bibliothèque. Les moteurs de recherche les plus populaires actuellement, indépendamment de la méthode de tri des résultats, en sont encore à la recherche textuelle. Beaucoup d'utilisateurs remarquent ainsi que nous touchons déjà aux limites de ce type de recherche et l'utilisation des travaux de Ranganathan représente une solution. Reste à trouver comment mettre en œuvre ces principes de classement à l'échelle planétaire, de manière multilingue et communautaire/collaborative.

Utilisant ces lois en guise d'axiomes, Ranganathan créa un système de classification adapté appelé « classification à facettes » dont la définition est : une classification à facettes est une méthode de classification utilisant des aspects, des propriétés et des caractéristiques clairement définis, mutuellement exclusifs et collectivement exhaustifs pour un sujet donné. Voilà qui n'est pas très clair.

SHIYALI RAMAMRITA RANGANATHAN

Shiyali Ramamrita Ranganathan (1892-1972) était un mathématicien et un bibliothécaire indien. Il est principalement connu pour ses travaux sur le catalogage et la classification. Il est célèbre en tant qu'inventeur de la classification à facettes et pour avoir édicté les cinq lois de Ranganathan :

1. Les livres sont faits pour être utilisés
2. A chaque lecteur son livre
3. A chaque livre son lecteur
4. Épargnons le temps du lecteur
5. Une bibliothèque est un organisme en développement

Ranganathan était considéré comme une « bête de travail » à un niveau pathologique (Workaholic en anglais). Comme le précise Wikipédia, un workaholic (de l'anglais *work*, travail, et du suffixe *-holic*) est une personne dépendant au travail ou accro au travail, à un degré qui relève de la dépendance compulsive ou névrotique. Il s'agit d'une réelle maladie particulièrement répandue au Japon, où l'on a même inventé un mot pour désigner le décès qui peut en découler : *Karoshi*.

En classification à facettes, nous avons plus qu'un jeu de catégories ou sections. Nous en avons un pour chaque aspect du logiciel. Chaque aspect ou facette du logiciel possède ainsi sa collection de catégorie. Cela va bien plus loin que le simple fait de classer un logiciel dans plusieurs catégories. Voici les facettes définies par Ranganathan pour classifier ses ouvrages (nos logiciels) :

P : la Personnalité. C'est l'objet principal du logiciel, ce qu'il est. C'est la facette principale.

M : la Matière. La substance qui constitue le logiciel. En d'autres termes, de quoi il est fait.

E : l'Énergie. L'opération en relation avec le logiciel. Ce que l'on peut faire avec.

S : l'eSpace. La couverture géographique du logiciel. Cette facette ne s'applique pas vraiment à un logiciel, mais on peut résumer en disant qu'elle décrit où nous utilisons le logiciel (même si la réponse est toujours la même).

T : le Temps. La période chronologique. Cette facette s'applique également difficilement aux logiciels. Ceci pourrait, dans Debian, être en rapport avec le moment où le logiciel est prêt à être utilisé.

L'enchaînement de ces facettes est dite «PMEST», en rapport avec les lettres qui identifient les noms des facettes (c'est plus évident en anglais : Personality, Matter, Energy, Space, Time).

Enrico Zini a donc utilisé PMEST pour créer un système de classement appelé DebTags. Voici comment Enrico décrit la manière dont PMEST est utilisé dans son système :

- «Le paquet comprend un programme qui peut utiliser des informations qui ne sont pas stockées localement sur la machine» Est-ce que l'enregistrement de ces informations est l'objet principal du programme ? Si c'est le cas, la Personnalité du programme doit être classifiée dans une catégorie en relation avec le réseau. Sinon, c'est l'Energie du programme qui est classifiée dans une telle catégorie.
- Le paquet contient du code qui utilise l'appel système `socket()`. Ceci concerne la facette Matière du paquet, la technologie qu'il utilise et les fonctions qu'il met en œuvre.
- Le paquet analyse des journaux d'un pare-feu. Ceci implique que la Personnalité et l'Energie du programme sont en relation avec le réseau mais pas la Matière. Si on regarde de près, le programme ne fait que parcourir des données au format texte.

STRUCTURE DE DEBTAGS

Enrico Zini est parti de PMEST mais a élargi le nombre de facettes puisque, justement PMEST est un concept destiné à pouvoir être étendu (cinquième loi de Ranganathan). L'implémentation de DebTags utilise donc :

- «Facets» : les facettes. Ce sont les différents aspects étudiés des paquets. On retrouve la liste des facettes avec `grep "^Facet: " /var/lib/debtags/vocabulary` ou `debtags tagcat | grep "^Facet"`. On y retrouve des facettes «interface», «works-with», «made-of» ou encore «protocol».

- «Tags» : les étiquettes. Ce sont les catégories permettant de classifier les paquets dans chaque facette. `debtags tagcat` permet de lister les tags qui apparaissent sous la forme du couple facette::catégorie. La catégorie est un mot anglais qui ne s'applique donc que dans la facette concernée.
- «Vocabulary» : le lexique (la traduction en «vocabulaire» ne s'applique pas correctement ici). C'est le lieu de stockage des facettes et de tags. Le lexique contient la liste des facettes et des tags, une description pour chacun d'eux, la nature de la facette, et des informations de gestion (état). Le lexique est stocké dans un dépôt SVN et copié sur votre système lors d'un `debtags update`. Vous pouvez avoir un aperçu de son contenu en utilisant la commande `grep-dctrl -FFacet -r . /var/lib/debtags/vocabulary`. Le lexique n'est pas figé et peut faire l'objet d'ajouts et de changements. Sa gestion est plus ou moins ouverte permettant ainsi à chacun de proposer des changements.
- «Tag Database» : C'est le cœur du système. Là où sont stockés les tags pour chaque paquet. Là encore, cette base de données est mise à jour avec `debtags update` et stockée localement (`/var/lib/debtags/package-tags`).

UTILISATION DE DEBTAGS

A présent que vous connaissez les tenants et les aboutissants du système, il est temps d'en faire usage. Commencez donc par installer le paquet `debtags` qui vous fournira la commande du même nom. La première étape est sans doute de voir à quoi ressemble la mise en œuvre des tags. Pour cela, utilisez l'action `show` en précisant un ou plusieurs noms de paquets :

```
denis@raven:~ % debtags show vim
Name: vim
Priority: not available
Section: editors
[...]
 This package contain a version of vim compiled
 with a rather standard set of features. See the
 other vim-* packages if you need more (or less).
Tags: devel::editor, interface::text-mode,
  uitoolkit::ncurses, use::editing, works-with::text
```

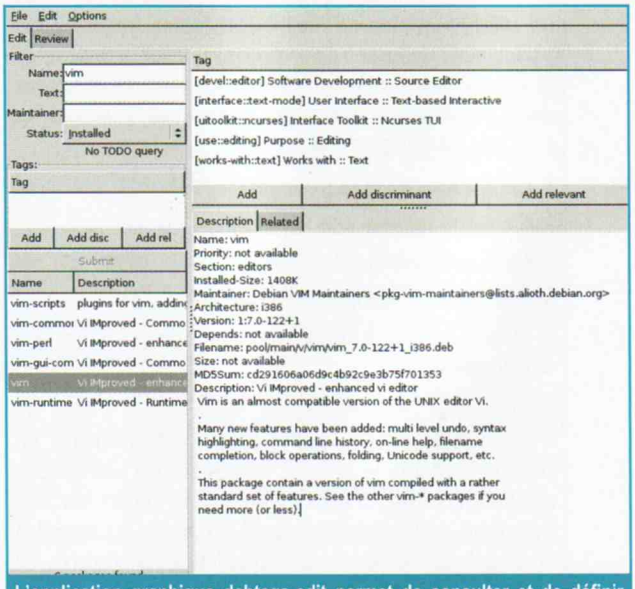
Cinq tags sont définis :

`devel::editor` : `devel` est une facette Energie, ce qu'on peut faire avec le logiciel, ici de l'édition pour le développement. `interface::text-mode` : La facette est une Personnalité qui caractérise son utilisation, une interface en mode texte. `uitoolkit::ncurses` : Vim est composé de Matière utilisant un toolkit d'interface `ncurses`. `use::editing` : Encore une facette Personnalité qui indique à quoi sert le paquet, l'édition. `works-with::text` : Une autre facette Energie qui nous indique ce que manipule le programme dans le paquet, le texte.

Passons à la recherche. Nous allons considérer que Vim n'est pas un bon choix pour nous (c'est une fiction bien sûr) et décidons de chercher quelque chose qui nous conviendra mieux. Nous voulons un éditeur de texte, lié au développement d'applications et ayant une interface en mode texte. Aussi nous utilisons :

```
% debtags search 'devel::editor &&
  interface::text-mode && works-with::text'
[...]
vim - Vi IMproved - enhanced vi editor
[...]
elvis-tiny - Tiny vi compatible editor for the base system
nvi - 4.4BSD re-implementation of vi
vile - VI Like Emacs - vi work-alike
[...]
jed - editor for programmers (textmode version)
[...]
lpe - Lightweight Programmer's Editor
mped - Minimum Profit, a programmer's text editor
xemacs21-bin - highly customizable text editor
[...]
```

Nous n'avons que l'embarras du choix. `DebTags` n'est pas en mesure de prendre en compte l'aspect personnel des préférences de l'utilisateur. Il faut donc s'en tenir au descriptif des paquets et aux essais pour cela.



L'application graphique debtags-edit permet de consulter et de définir des tags pour les paquets si vous n'appréciez pas les interfaces console ou ligne de commande

Poussons plus loin dans la recherche et imaginons qu'en fait vous ne vouliez pas un éditeur en mode console, mais autre chose. Vous pourriez, par exemple, rechercher ce qui fonctionne sous X. Pour cela, nous devons trouver le tag qui convient. Comme `debtags` est bien fait, il propose un système de recherche dans les tags. N'oubliez pas la quatrième loi de Ranganathan : «Épargnons le temps de l'utilisateur» :

```
% debtags tagsearch interface
devel::ui-builder - User Interface
interface (facet) - User Interface
interface::3d - Three-Dimensional
interface::TODO - Need an extra tag
interface::commandline - Command Line
interface::daemon - Daemon
interface::framebuffer - Framebuffer
interface::shell - Command Shell
interface::special:not-applicable - Not applicable
interface::special:not-yet-tagged - Not yet tagged
interface::special:todo - Need an extra tag
interface::svga - Console SVGA
interface::text-mode - Text-based Interactive
interface::web - World Wide Web
interface::x11 - X Window System
uitoolkit (facet) - Interface Toolkit
```

Nous avons une facette `interface` et une catégorie qui conviennent parfaitement : `x11`, ce qui nous donne le tag `interface::x11`. Notez que `debtags` nous affiche également la facette `uitoolkit` qui nous permettrait de choisir un toolkit spécifique parmi ceux qui seraient proposés par `debtags tagsearch uitoolkit`.

Nous pourrions donc utiliser `debtags search 'devel::editor && interface::x11 && works-with::text'`. Mais attention ! Ce n'est pas ce que nous avons dit vouloir rechercher : un éditeur qui ne fonctionne pas en mode console. Ce qui ne signifie pas pour autant qu'il doit fonctionner sous X. La bonne commande est donc :

```
% debtags search 'devel::editor && works-with::text
  && !interface::text-mode'
[...]
vim-scripts - plugins for vim, adding bells and whistles
[...]
```

Le paquet `vim-scripts` n'était pas listé précédemment. Il ne constitue bien entendu pas un choix valide dans notre recherche, mais il s'agissait d'une simple excuse pour démontrer la souplesse du système de recherche. Il est en effet possible d'utiliser des opérations de logique booléenne comme le ET (`&&`), le OU (`||`) et le NON (`!`). De plus, il est possible d'utiliser des parenthèses pour créer des groupes.

Enfin, en plus de l'action `tagsearch`, il est possible d'utiliser `tagsehow` pour avoir plus d'informations sur un tag :

```
% debtags tagshow interface::x11
Tag: interface::x11
Description: X Window System
 X Window System
```

Cela permet de s'assurer de la correspondance entre ce que l'on recherche et l'utilisation d'un tag.

A VOUS DE PARTICIPER

Si je vous ai détaillé les bases théoriques de DebTags, ce n'était pas simplement pour que vous compreniez mieux l'utilisation de la commande `debtags` mais surtout pour vous permettre de participer à l'effort d'étiquetage. Jugez plutôt :

```
% debtags stats
Total count of packages: 29777
Total count of packages (according to APT): 29777
Total count of packages (according to Debtags): 11549
Number of facets: 28
Number of tags: 582
Number of packages with tags,
          but no special::not-yet-tagged tags: 11549 (100.0%)
Number of packages with
          special::not-yet-tagged tags: 0 (0.0%)
Number of packages with only
          special::not-yet-tagged tags: 0 (0.0%)
Number of packages with no tags: 0 (0.0%)
```

Au moment où cet article est rédigé, seuls 11549 paquets sur 29777, soit moins de 40%, sont «tagués». Il est pourtant très facile de donner un peu de son temps pour ce travail. Une page Web existe et référence les efforts en cours dans ce sens : **http://debtags.alioth.debian.org/**.

Prenez le temps de vous pencher sur la lecture des quelques pages de ce site et éventuellement d'y consacrer un peu plus de temps encore pour une participation. Il n'est pas nécessaire d'être développeur Debian pour aider et la tâche est relativement facile.

LE SAVIEZ-VOUS ? DEBIAN, CE N'EST PAS QUE GNU/LINUX

Debian GNU/Hurd est une distribution basée sur le micro-noyau multi-serveur GNU/Hurd. Ce système, à la fois présenté comme l'avenir du Logiciel Libre et un éternel chantier, dispose d'un système d'installation et de gestion de paquets Debian. Pour l'heure, la distribution est destinée aux architectures de type i386, mais pourrait, un jour, s'étendre à d'autres plateformes. Le site est **http://www.es.debian.org/ports/hurd/**.

Debian GNU/NetBSD, comme son nom l'indique, est une distribution Debian pour NetBSD. NetBSD supporte un nombre impressionnant d'architectures et fonctionne donc sur du matériel que Linux ne fait pas fonctionner. Le but du projet est d'avoir une distribution portable permettant à n'importe quel utilisateur Debian de faire usage de NetBSD. Vous en saurez plus en visitant **http://www.debian.org/ports/netbsd/**.

Debian GNU/w32 n'est pas un projet officiel Debian. L'objectif est de fournir le système de gestion de paquets Debian et le portage de la plupart des paquets dans un environnement Microsoft Windows avec la bibliothèque Cygnus/RedHat `cygwin`. Bien entendu, on ne peut parler de distribution, même si le but est de permettre à un utilisateur Debian de s'y retrouver. Le site officiel est : **http://debian-cygwin.sourceforge.net/**.

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

CRÉEZ VOTRE PAQUET DEBIAN

Debian, comme toute distribution, utilise un système de gestion de paquets. Afin de ne pas corrompre le travail de ce dernier, il est indispensable de ne par perturber son fonctionnement avec des installations manuelles parasites. Il existe plusieurs solutions permettant d'éviter les interférences, mais la meilleure reste la création de paquets.

Les explications qui vont vous être données ci-après ne sont qu'une base de travail vous permettant de créer un paquet à partir des sources officielles d'un projet. Cette base n'a pour but qu'une seule chose : installer proprement une application ou des données qui ne sont pas encore empaquetées. Si vous souhaitez créer des paquets pour la distribution et/ou les diffuser via le Net, il vous faudra étudier davantage et sans doute faire appel à l'aide d'un développeur Debian (DD) pour vérification (et éventuellement vous parrainer lors de la soumission du paquet dans la distribution).

LA MÉTHODE SALE

Qui ne connaît pas la bonne vieille méthode pour avoir une application dernier cri sur son système ? Le très classique :

```
./configure
make
make install
```

Dans le pire des cas, on utilisera même l'option `--prefix=/usr` pour bien intégrer l'application compilée dans le système. Par défaut, en effet, un script généré par `autoconf` produit un `Makefile` qui installe les éléments dans `/usr/local` lors d'un `make install`. Remarquez que pour une installation « sauvage », le `/usr/local` est encore la meilleure solution, puisque ce répertoire est justement destiné à regrouper les éléments locaux du système, c'est-à-dire, les éléments qui ne sont pas, par défaut, gérés par l'UNIX installé.

Les problèmes soulevés par ce type de méthode sont multiples :

- `/usr/local` n'est pas toujours dans le chemin de recherche pour les binaires, ni pour les bibliothèques partagées.
- Ce type d'installation est indépendant du système de paquets. Vous ne savez donc pas ce qui est installé, quelle version est utilisée et de quelles bibliothèques ont besoin les programmes dans `/usr/src/bin`. Il est donc parfaitement possible que vous supprimiez une dépendance via le système de gestion des paquets sans aucun avertissement de sa part.
- Vous pouvez gravement perturber le fonctionnement du système en installant des binaires ou des bibliothèques de cette manière si des versions concurrentes existent. Même la reconstruction de paquets peut être perturbée et les binaires liés aux bibliothèques placées dans `/usr/local/lib`. Je ne parle même pas des fichiers de données, des macros M4, des fichiers d'en-tête, etc.

Si vous tenez absolument à tester une application de cette manière, il existe des solutions comme l'utilisation d'un environnement « chrooté » ou des outils de traçage d'installation (préférez la première solution). Dans tous les cas, s'il s'agit d'aller au-delà des simples essais. Vous devrez intégrer le logiciel proprement dans le système et donc passer par la création d'un paquet Debian.

L'EXEMPLE

Pour baser les explications qui vont suivrent sur un exemple concret, nous allons créer une application très basique, utilisant néanmoins un système moderne de configuration et de compilation : le couple `automake`/`autoconf`. Bien entendu, si vous comptez empaqueter votre propre application qui utilise déjà ce type de support, sautez directement au chapitre suivant.

Notre projet d'application se présente sous la forme de l'arborescence suivante :

```
.
|-- AUTHORS
|-- ChangeLog
|-- Makefile.am
|-- NEWS
|-- README
|-- configure.ac
`-- src
    |-- Makefile.am
    |-- afficher.c
    |-- afficher.h
    `-- main.c
```

Commençons par les fichiers les plus simples : `AUTHORS`, `ChangeLog`, `NEWS` et `README` contiennent des données au format texte dont la nature est en rapport direct avec leur nom (ces fichiers peuvent être vide).

Passons maintenant aux fichiers plus importants, à commencer par les sources dans le sous-répertoire `src` :

```
/* main.c */
#include <stdio.h>
#include <stdlib.h>

#include "afficher.h"

int main(int argc, char **argv) {
  afficher("Bonjour le monde");
  exit(EXIT_SUCCESS);
}
```

```
/* afficher.c */
#include <stdio.h>
#include <stdlib.h>

#include "afficher.h"

int afficher(char *str) {
  printf("%s\n", str);
  return(0);
}
```

```
/* afficher.h */
#ifndef AFFICHER_H
#define AFFICHER_H

int afficher(char *str);

#endif
```

Rien de bien fantastique. Il s'agit d'un simple exemple de type `Hello World` à peine plus évolué. Reste à voir maintenant le contenu des fichiers à l'attention des outils de configuration et compilation :

`src/Makefile.am` est un fichier à destination d'`automake` permettant de produire le fichier `src/Makefile.in` qui sera lui-même utilisé, plus tard, par le fameux script `configure` pour enfin produire un `Makefile`. Voici son contenu qui ne devrait pas nécessiter d'autres explications que celles en commentaire :

```
# binaire programme
bin_PROGRAMS=bonjour

# liste des sources
bonjour_SOURCES= \
          main.c \
          afficher.c afficher.h
```

`Makefile.am` est le fichier pour `automake` placé à la racine du projet. Comme il n'y a pas grand-chose à faire à ce niveau, il se contente de spécifier le sous-répertoire à traiter :

```
SUBDIRS=src
```

Enfin, `configure.ac` est le fichier principal à destination d'`autoconf`. Il contient diverses macros permettant de produire le script de configuration des sources, `configure`.

```
AC_PREREQ(2.60)
# Nom du projet
AC_INIT(bonjour, 0.1, lefinnois@lefinnois.net)
AM_INIT_AUTOMAKE

# test CC et make
AC_PROG_CC
AC_PROG_MAKE_SET

# on génère quoi ?
AC_CONFIG_FILES([
          Makefile
          src/Makefile
          ])
# go go go !
AC_OUTPUT
```

Et voilà, il n'en faut pas plus pour pouvoir générer notre archive de sources. Il suffit en effet, d'invoquer les commandes :

```
% aclocal
% autoconf
% automake -a -c
configure.ac: installing `./install-sh'
configure.ac: installing `./missing'
Makefile.am: installing `./INSTALL'
Makefile.am: installing `./COPYING'

% ls configure
configure
```

Remarquez les options `-a` et `-c` utilisées avec `automake` permettent respectivement d'ajouter automatiquement les fichiers manquants, et ce, en les copiant (plutôt qu'en utilisant des liens symboliques. Les fichiers `README`, `ChangeLog`, etc. sont également de la partie, mais nous avons pris la peine de les créer au préalable et de le compléter avec quelques informations.

A partir de ces sources, nous allons maintenant produire notre fichier d'archive ou *tarball* source. Pour le reste des manipulations propres à Debian, cette archive sera considérée comme les sources amonts (voir introduction). Pour produire notre archive, nous utiliserons une cible spécifique du `Makefile` que nous produirons en utilisant :

```
% ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
[...]
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: executing depfiles commands

% make distcheck
[...]
==========================================
bonjour-0.1 archives ready for distribution:
bonjour-0.1.tar.gz
==========================================

% src/bonjour
Bonjour le monde
```

Comme le précise le message, l'utilisation de la cible `distcheck` nous permet d'obtenir `bonjour-0.1.tar.gz` directement nommé et numéroté d'après le nom du

projet et la version spécifiés dans le `configure.ac`. A des fins de tests, vous pouvez désarchiver le tarball source dans un autre répertoire (ou sur une autre machine) pour en vérifier le bon fonctionnement. Il ne devrait cependant pas y avoir de problème, `make distcheck` étant précisément dédié à cela.

Si vous vous intéressez à l'utilisation d'`automake` et d'`autoconf` pour vos projets, je vous conseille la lecture des différents articles d'Y. Mettier publiés sur le sujet dans GLMF, articles d'où est humblement tiré notre sujet d'expérimentation.

ALLONS-Y !

Nous avons maintenant un exemple de sources amont et le tarball correspondant. La première chose à faire pour débuter la création d'un paquet est de vérifier que nous disposons de tous les outils nécessaires. Il s'agit bien entendu des différents éléments de compilation fournis par `build-essential` qui installera un compilateur C et C++, la commande `make` et le paquet `dpkg-dev`. Ce dernier est celui contenant les outils de fabrication de paquets Debian. Il installera également `binutils` ou encore `patch`.

Il est également conseillé d'installer les paquets `dh-make` permettant de créer une base pour la construction de paquets, `autoconf` et `automake`, `debhelper` qui regroupe un lot de scripts très utiles, `fakeroot` que vous connaissez déjà ou encore `lintian`, le testeur de paquets. Une installation complète et fonctionnelle de GnuPG est également souhaitée afin de signer le paquet source finale.

Créez ensuite votre environnement de travail sous la forme d'un simple répertoire vide. Là, copiez le tarball source et désarchivez-le. Nous sommes prêts à « débianiser » les sources. Lancez :

```
% dh_make -e lefinnois@lefinnois.net \
 -f ../bonjour-0.1.tar.gz
Type of package: single binary, multiple binary,
  library, kernel module or cdbs?
[s/m/l/k/b] s

Maintainer name : Denis Bodor
Email-Address   : lefinnois@lefinnois.net
Date            : Mon, 18 Dec 2006 15:30:38 +0100
Package Name    : bonjour
Version         : 0.1
License         : blank
Type of Package : Single
Hit [enter] to confirm:

Done. Please edit the files in the debian/
subdirectory now. bonjour uses a configure script,
so you probably don't have to edit the Makefiles.
```

`dh_make` est très pratique, car il permet de presque automatiquement « débianiser » des archives sources. Il fait la plupart du travail à notre place. `dh_make` est extrêmement bien conçu et un peu intrusif. Vous remarquerez, qu'ici, seule l'adresse mail a été précisée, pourtant mon nom apparaît comme par enchantement. En fait, `dh_make` a récupéré mon *login* et parcouru les ressources à sa disposition (`/etc/passwd`, LDAP, NIS, YP, etc.) pour retrouver les informations manquantes. Il n'y a pas de magie dans Debian, juste la compétence des développeurs.

`dh_make` a également procédé à un certain nombre de manipulations. Premièrement, il a copié notre tarball source en `bonjour_0.1.orig.tar.gz` respectant ainsi les conventions et règles en usage pour les paquets source. `dh_make` a également créé un répertoire `debian` dans l'arborescence des sources et l'a peuplé de nombreux fichiers. La description et l'étude de ces fichiers est l'objet du présent article.

NOTE

Il n'est possible de « débianiser » des sources amont qu'une seule fois. Si vous faîtes une erreur de manipulation ou détruisez un fichier, vous ne pouvez pas invoquer la commande une seconde fois. Le résultat ne serait pas garanti. Il faut donc faire très attention et faire des sauvegardes avant les manipulations critiques. Dans le cas contraire, vous n'aurez pas d'autres choix que de tout recommencer ou tenter de corriger le problème manuellement.

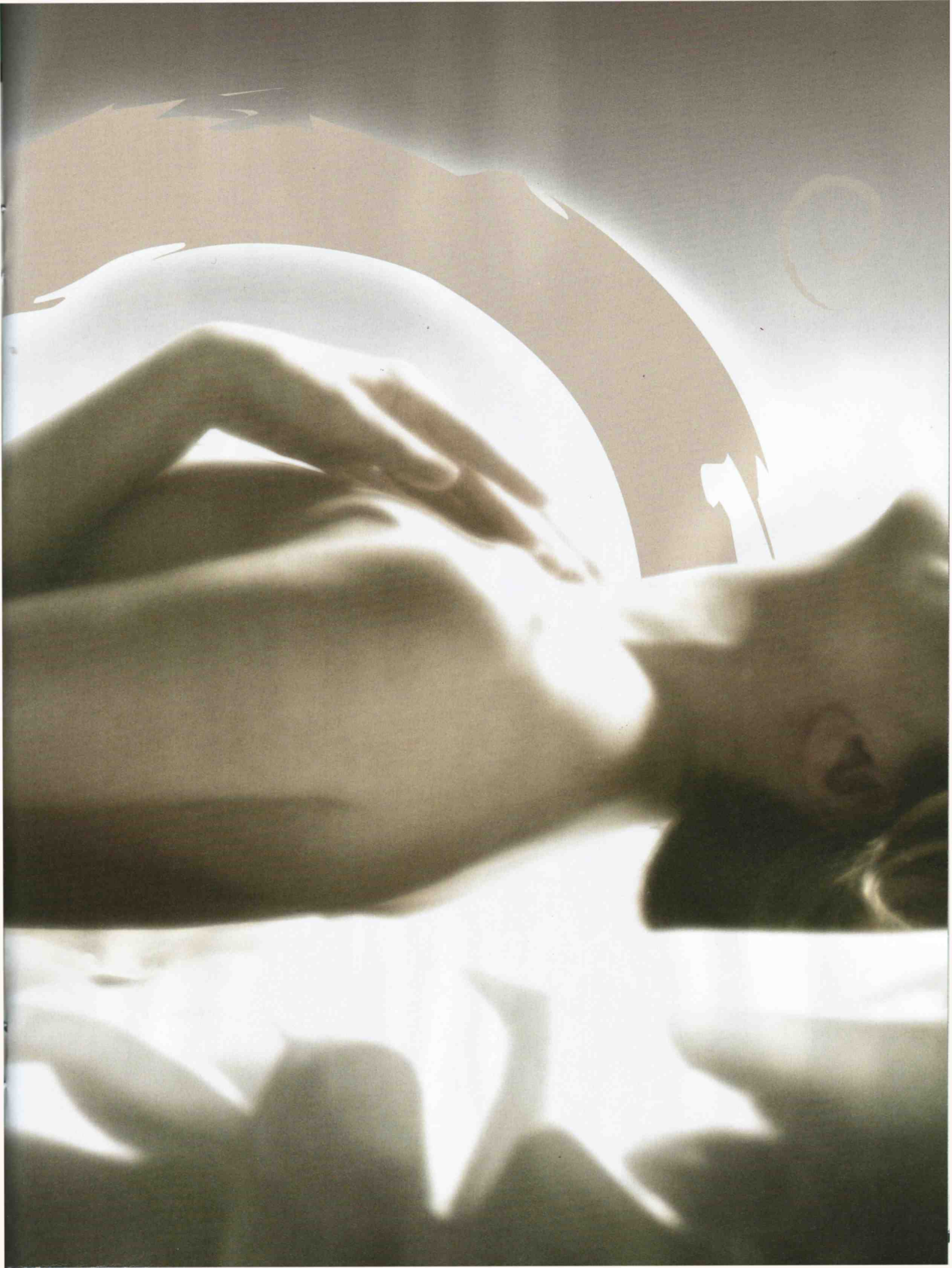
Remarquez que la partie sur l'utilisation d'`automake`/`autoconf` en début d'article n'était ni innocente, ni inutile. `dh_make` est en mesure de détecter ce type de configuration et s'en accommode très bien. Cela vous évite également de devoir adapter les `Makefile` aux besoins propres à une distribution Debian comme le respect de la hiérarchie du système de fichiers (Debian FHS), etc.

Nous sommes maintenant prêts pour la suite qui, comme nous l'indique `dh_make`, semble simple : éditez les fichiers dans `debian/`.

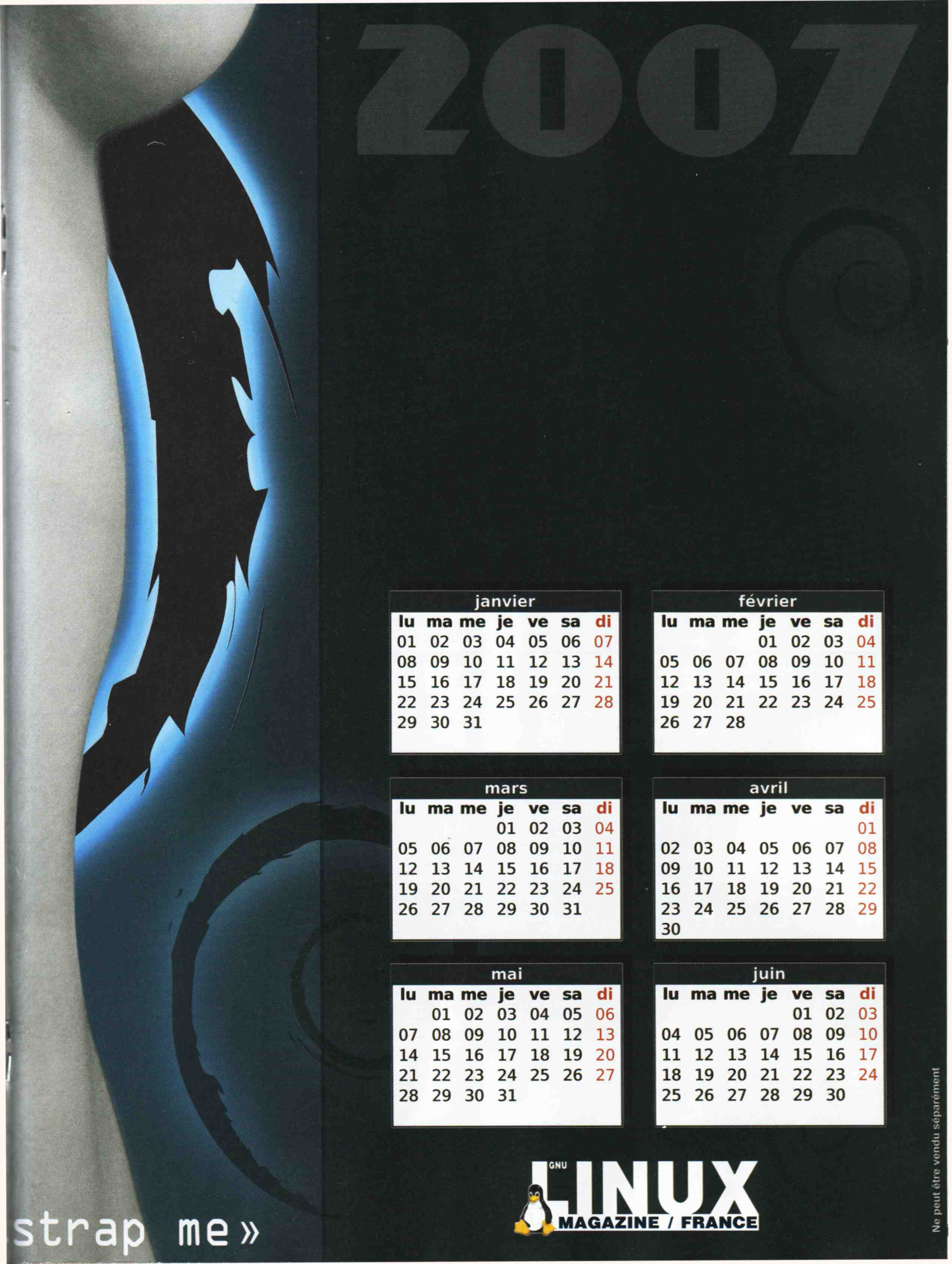
CONTROL

Le fichier `debian/control` contient les informations « administratives » sur le paquet. Ces informations sont utilisées par le système APT et l'utilitaire `dpkg`. `dh_make` a créé un fichier pour nous. Nous n'avons plus qu'à le compléter :

```
Source: bonjour
Section: unknown
Priority: optional
Maintainer: Denis Bodor <lefinnois@lefinnois.net>
Build-Depends: debhelper (>= 5), autotools-dev
Standards-Version: 3.7.2

Package: bonjour
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: <insert up to 60 chars description>
 <insert long description, indented with spaces>
```



« deboots



2007

janvier

lu	ma	me	je	ve	sa	di
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

février

lu	ma	me	je	ve	sa	di
			01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

mars

lu	ma	me	je	ve	sa	di
			01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

avril

lu	ma	me	je	ve	sa	di
						01
02	03	04	05	06	07	08
09	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

mai

lu	ma	me	je	ve	sa	di
	01	02	03	04	05	06
07	08	09	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

juin

lu	ma	me	je	ve	sa	di
				01	02	03
04	05	06	07	08	09	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

strap me »

GNU LINUX MAGAZINE / FRANCE

GNU LINUX MAGAZINE / FRANCE

juillet						
lu	ma	me	je	ve	sa	di
						01
02	03	04	05	06	07	08
09	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

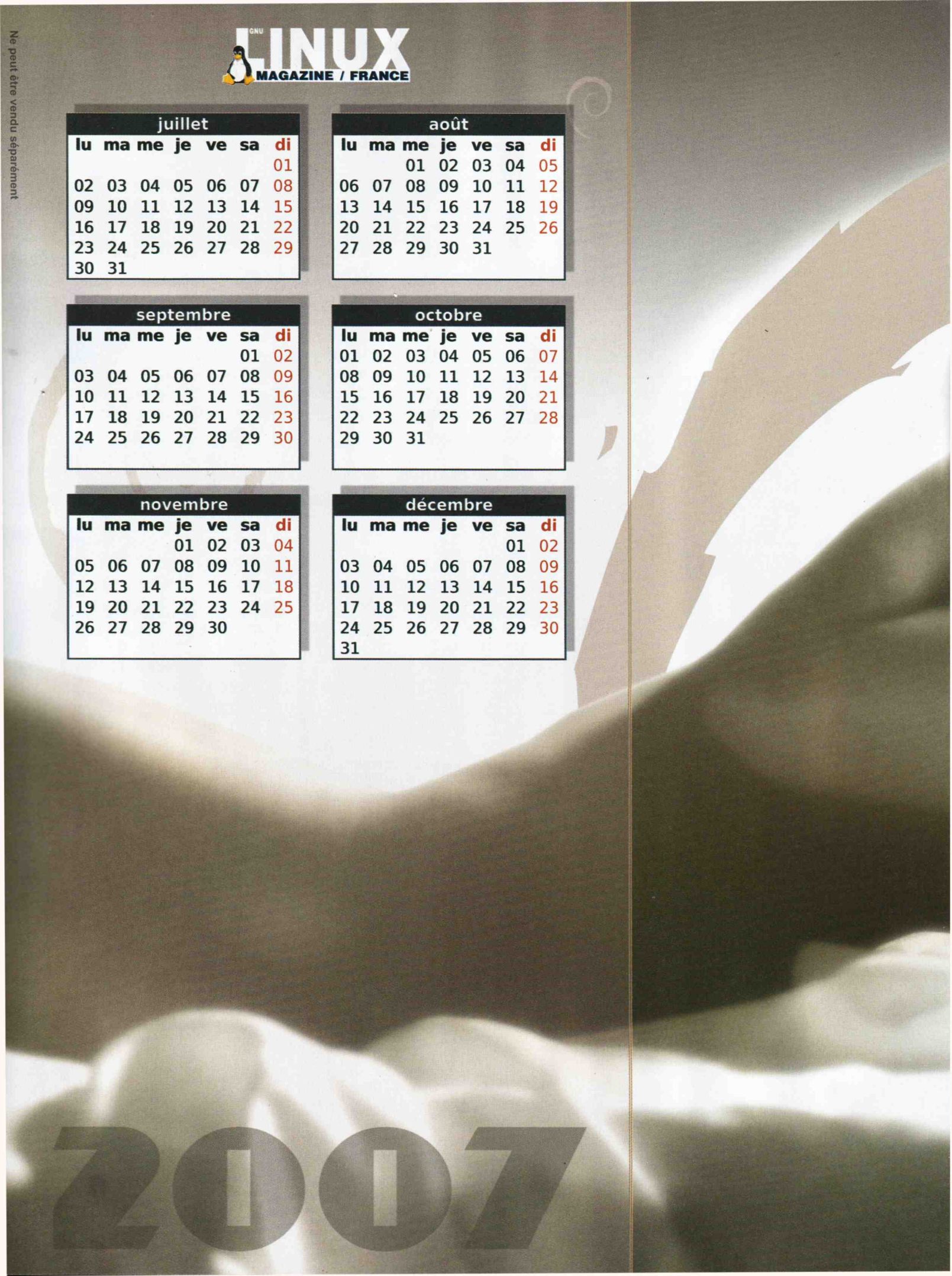
août						
lu	ma	me	je	ve	sa	di
		01	02	03	04	05
06	07	08	09	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

septembre						
lu	ma	me	je	ve	sa	di
					01	02
03	04	05	06	07	08	09
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

octobre						
lu	ma	me	je	ve	sa	di
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

novembre						
lu	ma	me	je	ve	sa	di
			01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

décembre						
lu	ma	me	je	ve	sa	di
					01	02
03	04	05	06	07	08	09
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						



La première ligne indique le nom du paquet source. Elle est automatiquement complétée, tout comme la valeur pour `Maintainer` (le responsable du paquet, vous), `Standards-Version` (qui est la version de la charte Debian), `Package` (le nom du paquet binaire), `Priority` (l'importance du paquet, `optional` est parfait dans la plupart des cas) et `Architecture`. Ce dernier point est, dans le cas présent bien déterminé, mais il est possible qu'un programme ne fonctionne que sur un type d'architecture donné. Il faut alors le spécifier manuellement.

`Section` détermine la section et sous-section dans laquelle placer le paquet. Comme le précise l'article sur le système Debtags, l'utilisation des sections n'est pas idéale. Cependant, en tant que responsable de paquet, même amateur, vous devez vous plier à la règle. Vous pouvez prendre connaissance des différents choix possibles en utilisant `cat /var/lib/apt/lists/*_Packages | grep "^Section" | sort | uniq`. Trois sections sont disponibles, `main` (qui est sous-entendu en l'absence de mention spécifique), `contrib` (logiciel dépendant de logiciel non libres) et `non-free` (logiciel non libre au sens DFSG). A vous de déterminer au mieux où placer votre paquet en ayant conscience des limitations du système. Ici, nous utiliserons `utils` (sous-entendu `main/utils` donc).

`Description` permet de préciser la description courte (60 caractères maxi). C'est celle qui apparaît, par exemple, avec `apt-cache search` ou `dpkg -l`. Juste en dessous se trouve la description longue, affichée lors d'un `apt-cache show`. Ici, un certain nombre de règles sont d'usage :

- La première colonne de chaque ligne est vide.
- La description doit tenir en un paragraphe
- Si vous utilisez plusieurs paragraphes, n'utilisez pas de ligne vide, mais une ligne contenant un simple point dans la seconde colonne.

Nous utiliserons donc ici :

```
Description: The classic french helloworld program
 The helloworld program produces a familiar, friendly greeting
 in french.
```

Passons maintenant aux deux éléments plus délicats. `Depends` renseigne sur les dépendances de votre paquet binaire. Vous n'avez, en principe, pas à toucher la valeur actuelle définie par `dh_make`. En effet, `${shlibs:Depends}`, par exemple, est une variable qui sera complétée lors de la première construction et analyse de votre paquet. Un script spécifique analysera le binaire (via `ldd` entre autres) et déterminera les noms des paquets contenant les bibliothèques dynamiques utilisées. Il en va de même pour `${misc:Depends}`. Pour un programme aussi simple que celui utilisé ici, vous n'avez pas à vous inquiéter. En revanche, le script d'analyse, `dh_shlibdeps`, sera incapable de donner des résultats satisfaisants avec un programme plus complexe chargeant à la demande des bibliothèques partagées.

NOTE

`Depends` permet de définir les dépendances de votre paquet. Celles qui s'installeront lorsqu'on l'installera via `apt-get` ou `aptitude`. Mais il existe aussi d'autres manières de lier votre paquet avec d'autres, de manière mois radicale que `Depends`. Ainsi, nous avons :

- `Recommends` pour spécifier les paquets qui ne sont pas vraiment indispensables, mais qui sont normalement utilisés avec le logiciel.
- `Suggests` pour les paquets qui contiennent des données ou des applications qui fonctionnent parfaitement avec le logiciel. La nuance avec `Recommends` est subtile.

Nous avons aussi des liens spéciaux entre les paquets :

- `Conflicts` permet de ne pas installer votre paquet à moins que celui ou ceux spécifiés ici ne soient désinstallés. C'est le cas, par exemple, si votre paquet fournit un programme qui remplit les mêmes fonctions que celui installé par un autre paquet : un super-serveur `inetd`, par exemple.
- `Provides` est en rapport avec les paquets virtuels. Le paquet `exim4`, par exemple, fournit le paquet virtuel `mail-transport-agent` qui est nécessaire au fonctionnement du système. Le paquet `ssmtp` fait de même. Le simple fait d'installer l'un de ces paquets satisfait les dépendances liées.

Enfin, `Build-Depends` informe sur les dépendances de construction du paquet. C'est cette liste qui sera utilisée par `apt-get build-dep`. Pour déterminer le contenu de ce champ, le « Guide du nouveau responsable Debian » fournit une astuce. Désarchivez votre tarball source dans un répertoire temporaire, puis utilisez dans ce dernier les commandes :

```
% strace -f -o /tmp/log ./configure

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

Dans le cas présent, vous devez obtenir une sortie comme ceci :

```
base-files (>= 3.1.16 ), binutils (>= 2.17-2 ),
coreutils (>= 5.96-5 ), gcc-4.1 (>= 4.1.1-13 ),
libacl1 (>= 2.2.41-1 ), libattr1 (>= 2.4.32-1 ),
libc6 (>= 2.3.6.ds1-4 ), libc6-dev (>= 2.3.6.ds1-4 ),
libncurses5 (>= 5.5-5 ), libselinux1 (>= 1.32-3 ),
libsepol1 (>= 1.14-1 ), locales (>= 2.3.6.ds1-4 )
```

La plupart de ces paquets ne nécessitent pas de mention particulière puisqu'ils seront déjà installés avec debhelper. Une autre solution consiste à compiler le logiciel, puis à utiliser la commande objdump -p sur le binaire en redirigeant la sortie vers grep NEEDED. Il suffit alors de rechercher à quel paquet appartiennent les bibliothèques listées avec dpkg -S et vérifier s'il existe un paquet -dev correspondant. Les deux méthodes peuvent être utilisées de concert, mais comprenez bien que c'est bien vous qui devrez finalement tirer les conclusions qui s'imposent et définir les dépendances de construction. Notre ligne sera ici : Build-Depends: debhelper (>= 5), autotools-dev, libc6-dev (>= 2.3.6.ds1-4)

RULES

Le fichier debian/rules est un Makefile utilisé par dpkg-buildpackage pour créer le paquet. On y trouve ainsi différentes cibles comme build:, install: ou binary-arch:. Le travail du fichier rules est de centraliser les informations et le lancement des scripts de construction et de vérification (dh_*).

On retrouve dans ce fichier le lancement du script autoconf, ./configure avec l'option --prefix=/usr. La variable CFLAGS permet également de passer des options au compilateur.

Le fichier rules est le cœur du système de création de paquet. Bien entendu, pour quelque chose d'aussi simple qu'un « bonjour monde », il n'y a pas grand-chose à dire ou faire. Cependant, vous remarquerez que les scripts dh_* inutilisés sont présents sous la forme de commentaires. Libre à vous donc de parcourir les pages de manuel des différents scripts.

AUTRES FICHIERS

Le fichier copyright résume les informations sur le projet amont comme l'auteur du logiciel, les sites Web ou FTP d'où vient le logiciel, etc. Complétez simplement le fichier en suivant les indications placées là par dh_make.

changelog résume l'évolution du paquet au cours de ses différents changements de versions. Là encore, dh_make vous a facilité le travail en proposent un canevas.

README.Debian est le document texte qui décrit les différences entre la version standard du logiciel et la version Debian. Le fichier peut être supprimé s'il n'y a rien à dire.

conffiles.ex devra être renommé conffiles et devra contenir la liste (un élément par ligne) des fichiers à considérer comme des éléments de configuration (voir article sur le détournement).

docs liste les fichiers de documentation se trouvant à la racine de l'arborescence source. Ils seront déplacés ensuite dans /usr/share/doc/nom_du_paquet.

manpage.1.ex est un exemple de page de manuel au format roff. Normalement, votre logiciel doit avoir une page de manuel.

dirs contient la liste des répertoires qui doivent exister sur le système cible, mais que le paquet ne va pas créer.

CONSTRUCTION

Il est maintenant temps de lancer la création du paquet ou plus exactement des paquets, puisque ceci concerne à la fois le paquet binaire et un paquet source propre et distribuable. Placez-vous dans le répertoire des sources amont et utilisez dpkg-buildpackage -rfakeroot. Si vous avez lu l'article sur la reconstruction de paquets, cette commande vous est sans doute familière. Notez cependant l'absence de l'option -b limitant le processus de construction au paquet binaire.

```
% dpkg-buildpackage -rfakeroot
dpkg-buildpackage: source package is bonjour
dpkg-buildpackage: source version is 0.1-1
dpkg-buildpackage: source changed by Denis Bodor
dpkg-buildpackage: host architecture i386
[...]
dh_testdir
dh_testroot
rm -f build-stamp
dh_clean
 dpkg-source -b bonjour-0.1
dpkg-source: building bonjour using existing bonjour_
0.1.orig.tar.gz
dpkg-source: building bonjour in bonjour_0.1-1.diff.gz
[...]
```

Dès le lancement, les scripts vont commencer par construire le paquet source en utilisant le tarball amont et en produisant un patch (fichier diff) compressé Gzip.

```
[...]
dh_testdir
# Add here commands to configure the package.
./configure --host=i486-linux-gnu --build=i486-linux-gnu
[...]
[...]
```

Le script configure est ensuite lancé en utilisant les paramètres permettant l'installation dans une arborescence correspondant aux besoins de la distribution. Suit, directement après, la compilation elle-même :

```
[...]
Making all in src
make[2]: entrant dans le répertoire " /mnt/HS28ex/bonjour-
0.1/src "
if i486-linux-gnu-gcc -DPACKAGE_NAME=\"bonjour\"
 -DPACKAGE_TARNAME=\"bonjour\" -DPACKAGE_VERSION=\"0.1\"
 -DPACKAGE_STRING=\"bonjour\ 0.1\"
 -DPACKAGE_BUGREPORT=\"lefinnois@lefinnois.net\"
 -DPACKAGE=\"bonjour\" -DVERSION=\"0.1\" -I. -I.
 -Wall -g -O2 -MT main.o -MD -MP -MF ".deps/main.Tpo"
 -c -o main.o main.c[...]
[...]
```

Les scripts prennent ensuite le « dépouillage » du logiciel en charge en se chargeant de répartir les éléments un peu partout dans le système, en fonction des données spécifiées dans les fichiers de configuration de debian/.

```
[...]
touch build-stamp
 fakeroot debian/rules binary
```

```
dh_testdir
dh_testroot
dh_clean -k
dh_installdirs
[...]
dh_testdir
dh_testroot
dh_installchangelogs ChangeLog
dh_installdocs
dh_installexamples
dh_installman
dh_link
dh_strip
dh_compress
dh_fixperms
dh_installdeb
dh_shlibdeps
dh_gencontrol
dh_md5sums
dh_builddeb
[...]
```

Enfin, il est temps de construire le fichier `.deb`, puis d'authentifier les fichiers importants, à commencer par `bonjour_0.1-1.dsc` :

```
[...]
dpkg-deb: construction du paquet "bonjour"
  dans "../bonjour_0.1-1_i386.deb".
tar: -: file name read contains nul character
 signfile bonjour_0.1-1.dsc
Vous avez besoin d'une phrase de passe pour déverrouiller la
clé secrète pour l'utilisateur: " Denis Bodor
lefinnois@lefinnois.net "
clé de 1024 bits DSA, ID 43C4B3C8, créée le 2000-05-14
Entrez la phrase de passe: **************************
[...]
```

Si une version utilisable de GnuPG est détectée, une clef secrète est recherchée pour l'utilisateur spécifié dans le fichier `debian/control`. Si la clef est trouvée, la phrase de passe est demandée pour signer le fichier `.dsc`. L'opération est renouvelée pour le fichier `bonjour_0.1-1_i386.changes` tiré du `changelog` original du paquet source :

```
[...]
 dpkg-genchanges
dpkg-genchanges: including full source code in upload
dpkg-buildpackage: full upload (original source is included)
 signfile ../bonjour_0.1-1_i386.changes

Vous avez besoin d'une phrase de passe pour déverrouiller la
clé secrète pour l'utilisateur: " Denis Bodor
lefinnois@lefinnois.net "
clé de 1024 bits DSA, ID 43C4B3C8, créée le 2000-05-14
Entrez la phrase de passe: **************************
```

La procédure terminée, vous devez trouver les fichiers suivants dans le répertoire parent :

`bonjour_0.1.orig.tar.gz` : le tarball des sources amont

`bonjour_0.1-1.diff.gz` : le patch à appliquer aux sources amont pour obtenir les sources Debian.

`bonjour_0.1-1.dsc` : le fichier de description à destination de la commande `dpkg-source -x` permettant, à partir des sources amont et du patch, d'obtenir l'arborescence des sources Debian prêtes à être compilées/construites.

`bonjour_0.1-1_i386.changes` : la liste des modifications apportées au paquet source.

`bonjour_0.1-1_i386.deb` : le paquet binaire.

Avant d'utiliser notre paquet binaire, nous allons vérifier notre travail. Pour cela, faisons appel à `lintian`. Cet outil, comme `linda`, recherche les erreurs courantes et vous informe de ses découvertes :

```
% lintian -i bonjour_0.1-1_i386.changes

W: bonjour: binary-without-manpage bonjour
N:
N:   Each binary in /usr/bin, /usr/sbin, /bin, /sbin or
N:   /usr/games should have a manual page
N:
N:   Note, that though the `man' program has the
N:   capability to check for several program names
N:   in the NAMES section, each of these programs
N:   should have its own manual page (a symbolic
N:   link to the appropriate manual page is sufficient)
N:   because other manual page viewers such as xman
N:   or tkman don't support this.
N:
N:   Refer to Policy Manual, section 12.1 for details.
N:
E: bonjour: spelling-error-in-description french French
N:
N:   Lintian found a spelling error in the package
N:   description. Lintian has a list of common misspellings
N:   that it looks for; it does not have a dictionary like
N:   a spelling checker does.
```

Notre paquet ne contient pas de page de manuel. Nous aurions dû en créer une en tant que responsable du paquet. De plus, nous avons fait une erreur d'anglais relativement classique dans la description du paquet (fort intéressant pour un français).

Il ne nous reste plus qu'à corriger les problèmes et reconstruire le paquet. Enfin, nous pouvons l'installer avec `dpkg -i`.

CONCLUSION

Cet article n'a pas la prétention de faire de vous un responsable de paquet Debian, mais simplement de vous faire découvrir, à l'aide d'un exemple simple, les principes utilisés. Il existe une documentation officielle Debian expliquant, plus en détail et en français, les étapes et les manipulations présentées ici. C'est le *Guide du nouveau responsable Debian*. Vous la trouverez sur http://www.debian.org/doc/manuals/maint-guide/.

Il ne s'agit que d'un guide et il est loin de détailler tous les cas de figure. Voyez ce document comme la seconde marche (la première étant cet article) d'un grand escalier où il est facile de trébucher.

Après avoir lu le *Guide du nouveau responsable Debian*, la meilleure solution est d'utiliser `apt-get source` à tout va, afin d'étudier le `debian/rules` de paquets importants. Tout comme avec la programmation, observer le travail des autres est la meilleure source d'apprentissage qui soit...

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

ÉTUDE DU DÉMARRAGE D'UN SYSTÈME DEBIAN

Le démarrage d'un système GNU/Linux est une procédure complexe reposant sur un mécanisme en partie spécifique à la distribution. Nous allons, dans cet article, détailler les étapes du démarrage d'un système Debian GNU/Linux afin de vous permettre de mieux comprendre la structure du système.

Les informations qui vont suivre sont, en grande partie, génériques à l'ensemble des systèmes GNU/Linux. Les distributions suivant l'évolution des fonctionnalités du noyau et des utilitaires, on retrouve souvent les même techniques et processus mis en œuvre. Cependant, certaines spécificités font de Debian GNU/Linux une distribution plus structurée (à mon goût) que d'autres et donc plus facile à comprendre.

ET LE COURANT FUT

Dès l'allumage d'une machine x86 de type PC (attention aux machines Apple/Intel qui sont maintenant une catégorie x86 à part), la machine et la mémoire sont dans un état indéterminé. Le processeur exécute alors automatiquement un code placé en ROM (ou en mémoire Flash) : le BIOS. Celui-ci initialise les composants les plus importants de la machine durant une procédure appelée « POST » (pour *Power-On Self Test*). L'ensemble des systèmes vitaux, dont la mémoire et les contrôleurs de disques sont inspectés. C'est à ce stade qu'un éventuel problème est rapporté à l'utilisateur via différent « bips », codes et messages d'erreur à l'écran.

Ceci fait, il parcourt ensuite les périphériques de stockage ou autre (*boot* PXE) qu'il peut utiliser pour exécuter le démarrage du système. Dans le cas des périphériques de stockage, le premier secteur est inspecté à la recherche d'un code de boot. Dans le cas d'un disque dur, ce secteur est le MBR (*Master Boot Record*) qui, une fois chargé et exécuté, chargera le vrai secteur de boot sur la première partition marquée « active ».

BOOT

Le secteur de boot prend le relais (via le MBR). Dans le cas d'un système GNU/Linux, c'est la première étape du chargement du *bootloader*. L'un de ceux-là, GRUB (*GRand Unified Bootloader*), est composé d'un minimum de deux éléments logiciels. Nous avons d'une part le code de boot qui doit tenir dans les 512 octets du premier secteur et qui se nomme `stage1`. Celui-ci à pour seul but de trouver et charger le second élément, `stage2`, le code principal du bootloader.

En effet, un bootloader aussi complet, souple et configurable que GRUB ne peut pas « tenir » dans 512 malheureux octets. Le code principal de GRUB est en mesure de lire divers supports de stockage et différents types de partitions. Il est également en mesure de charger un fichier de configuration regroupant les options de démarrage qui apparaissent sous la forme d'un menu à l'écran. A la demande de l'utilisateur ou automatiquement (après un délai configuré), GRUB charge le code principal du système d'exploitation, le noyau. Il existe également un certain nombre de `jfs_*_1_5`. Il s'agit de codes plus réduits (moins le 10 Ko) pouvant être placés dans une zone juste après le MBR. Cette zone d'une taille correspondant au nombre de secteurs par tête moins 1 est habituellement inutilisée. Ceci permet de placer `stage2` sur un système de fichiers correspondant à la version de `stage1.5` (ReiserFS, XFS, JFS, etc.).

NOTE

GRUB procède également à un certain nombre de vérifications et passe le processeur en mode protégé. Au démarrage de la machine, le processeur est dans un mode particulier appelé « mode réel ». Il s'agit d'un mode correspondant au fonctionnement des processeurs 8086. Eh oui, lorsque vous mettez la machine sous tension, votre beau Core 2 Duo E6600 tout neuf ne vaut guère plus qu'un 8086 cadencé à 2,4 GHz. C'est `stage1` qui remet les choses au goût du jour.

GRUB n'est, bien sûr, pas le seul bootloader. LILO ou SYSLinux sont deux autres implémentations. L'utilisation de GRUB reste cependant majoritaire, bien que les besoins en « boot graphique » de certains utilisateurs poussent les distributions orientées *desktop* vers des solutions différentes. Enfin, précisons que la notion de « bootloader », sans être spécifique aux plateformes PC, est un concept caractéristique de ces architectures. En effet, le BIOS est sans doute la pire invention du monde informatique en raison du nombre de limitations imposées dès son invention. D'autres architectures comme les Sparc ou les Mac utilisent un *firmware* intégrant, entre autres, un grand nombre des fonctionnalités offertes par

les bootloaders. Il est logique, en effet, d'estimer que la place d'un bootloader est en ROM et non sur un disque, du moins la plus grande partie.

BOOTLOADER ET NOYAU

GRUB charge le noyau Linux sous la forme d'un fichier image et saute à la première instruction de cette image. Le noyau est compressé (Zlib) et l'en-tête d'image contient le code permettant la décompression et le chargement en mémoire.

Si une image de disque d'initialisation (*Init RAM Disk*) est spécifiée dans la configuration de GRUB, celui-ci la charge également en mémoire et informe le noyau de sa présence au moment du démarrage. Ce RAM Disk maintenant au format `initramfs` est toujours appelé `initrd`, ce qui correspond au nom du format utilisé avec les versions plus anciennes du noyau Linux. C'est le code décompressé du noyau qui va alors monter le RAM Disk comme un système de fichiers contenant l'arborescence racine du système.

Le noyau Linux peut également intégrer le support d'un ou plusieurs systèmes de stockage (SCSI, IDE, SATA) et systèmes de fichiers (ext3, ReiserFS, etc.). Il peut alors se passer de RAM Disk et utiliser directement la partition désignée comme système de fichiers racine. Nous verrons par ailleurs en quoi cela peut être encore utile aujourd'hui.

Le système de fichier racine fourni par un RAM Disk est temporaire. Il sert principalement au chargement de modules noyau permettant la prise en charge d'autres systèmes de fichiers, de périphériques nécessaires au démarrage du système et à la prise en charge de fonctionnalités particulières (RAID logiciel, LVM, etc.). Une fois le processus de démarrage suffisamment avancé, le système de fichier racine est changé (via `pivot_root` qui ne fait qu'appeler la fonction du même nom). On permute ainsi le RAM Disk et le vrai système de fichiers racine du système pour poursuivre le chargement et terminer le processus de boot. C'est également le système en RAM Disk qui monte les pseudo-systèmes de fichiers comme `/proc`, `/sys`, `/tmp` ou `/dev`.

INITRAMFS

Le remplaçant de l'`initrd` au format CRAMFS (*Compressed* ROM/RAM FS), appelé `initramfs` a fait son apparition dans le noyau Linux de développement 2.5.46. Contrairement à son prédécesseur, l'`initramfs` n'est pas vraiment un système de fichier. Il n'est donc pas possible de le monter avec `mount -ro loop image /point/de/montage`.

Il s'agit en réalité d'une archive `cpio` compressée avec Gzip. Le noyau, lors du démarrage, ne monte donc pas directement le système de fichier, mais crée un RAM Disk, puis copie le contenu de l'archive `cpio`. L'`initramfs` est un système utilisant `ramfs` qui présente l'avantage d'avoir un taille dynamique (la mémoire est allouée en fonction des besoins).

Si vous souhaitez inspecter le contenu d'une image `initramfs`, créez un répertoire, placez-vous dedans et utilisez `gunzip -c /chemin/initrd.img | cpio -iv`. Vous retrouverez ainsi toute l'arborescence du futur RAM Disk temporaire dans le répertoire courant. Vous pourrez alors inspecter le script *shell* `init` à la racine de l'arborescence pour en apprendre plus sur le déroulement du processus en ce qui concerne l'`initramfs`.

Dans tous les cas, une fois les fonctions d'initialisation exécutée par le noyau, il est temps de lancer le premier programme utilisateur : `init`.

INIT

Pour démarrer tous les services ou utilitaires qui composent un système GNU/Linux, un programme unique est utilisé. C'est le processus ayant le PID 1, le parent de tous les processus du système. Son fichier de configuration est `inittab` qui décrit la liste des programmes à lancer, ce sont les services.

L'`init` le plus largement utilisé avec les systèmes GNU/Linux est un clone de l'Init utilisé dans UNIX System-V (Système cinq) développé par AT&T et rendu public en janvier 1983. Ce système utilisant des niveaux d'exécution (*runlevel*) est depuis longtemps critiqué. On cherche donc, plus ou moins activement, à le remplacer.

NOTE

La dernière version 6.10 de la distribution Ubuntu basée sur Debian annonce parmi ses nouveautés l'utilisation d'un remplaçant au vieillissant Init System-V appelé « Upstart ». Bien qu'il s'agisse effectivement d'un code différent destiné à gérer les services avec une philosophie novatrice, Upstart se borne pour l'heure à singer le comportement de l'Init classique. Le renouveau du système de gestion des services n'est pas pour aujourd'hui, en tout cas, pas chez Ubuntu.

`init` est un programme relativement simple. Il prend, entre autres comme argument, une valeur numérique (ou « S/s ») et l'utilise pour déterminer les programmes ou scripts qui doivent être lancés. Cette valeur est le runlevel dans lequel le système passe alors. Les scripts

lancés sont spécifiés dans le fichier /etc/inittab. Le choix d'utiliser un runlevel donné pour tel ou tel « état » du système est complètement arbitraire, mais les distributions s'entendent pour la plupart sur quelques points :

0 : Arrêt du système.

1 : Mode mono-utilisateur (habituellement pour réparer le système).

2 : Fonctionnement normal.

6 : Redémarrage du système.

S : est utilisé pour le démarrage du système. Dans ce runlevel, le système est dit « *on boot* ». Si vous avez la main dans ce runlevel, c'est qu'un problème est survenu lors du démarrage (habituellement lors de la vérification des systèmes de fichiers avec fsck).

SCRIPTS D'INIT

Dans un système Debian, les scripts à utiliser pour chaque niveau d'exécution sont lancés via un seul et même script shell : /etc/init.d/rc qui prend en argument le numéro de runlevel. Ce script utilise alors l'arborescence /etc/rcN.d/ où N est le runlevel. Ces répertoires contiennent des liens symboliques pointant vers des fichiers stockés dans /etc/init.d. C'est le nom du lien symbolique qui détermine le comportement du script.

Si le nom du lien débute par un « S », le script dans /etc/init.d/ est lancé avec l'argument start, s'il s'agit d'un « K », l'argument est stop. Une valeur numérique suit ce premier caractère. Il est utilisé pour déterminer l'ordre dans lequel exécuter les scripts lors d'un passage d'un niveau à un autre. Les liens débutants par un « K » sont lancés en premier, puis ceux débutant pas un « S ».

On notera également, comme il est précisé dans /etc/init.d/rc, qu'un script configuré pour lancer un service dans le runlevel précédent ne sera pas lancer à nouveau lors du passage au nouveau runlevel.

Un script d'init est relativement complet. Il prend en argument une action qui peut être start ou stop, mais également reload pour demander au démon qu'il gère de recharger sa configuration ou encore restart pour provoquer un arrêt suivi d'un redémarrage du démon. Il peut également vérifier un certain nombre de points de configuration comme l'existence d'une variable d'environnement évaluée après le chargement d'un autre script présent dans /etc/default. Ceci permet, par exemple, de conditionner le démarrage du service en fonction d'options comme start_smartd=yes (pour le service de surveillance des disques). C'est également là qu'on trouvera ces options spécifiques au système local, comme les options à passer au serveur OpenSSH, l'utilisation de FetchMail en démon, la langue pour GDM ou encore l'emplacement du fichier de configuration de Nagios.

Pour lancer et arrêter un démon (exécutable binaire du service), les scripts utilisent start-stop-daemon. Ce petit programme permet de lancer des exécutables tout en gardant leur trace en stockant leur numéro de processus dans un fichier .pid placé dans /var/run. Ainsi, start-stop-daemon est en mesure de tuer le processus correspondant au service ou de lui demander de recharger sa configuration en lui envoyant un signal HUP. start-stop-daemon présente un avantage évident : il rend générique le comportement des binaires et sert de « *wrapper* » à ces derniers. C'est ce qui rend si facile l'écriture de nouveau scripts d'init.

Vous l'aurez compris, les scripts d'init servent non seulement pour le démarrage du système et la gestion des runlevel, mais également pour lancer et stopper des services à la convenance de l'administrateur système.

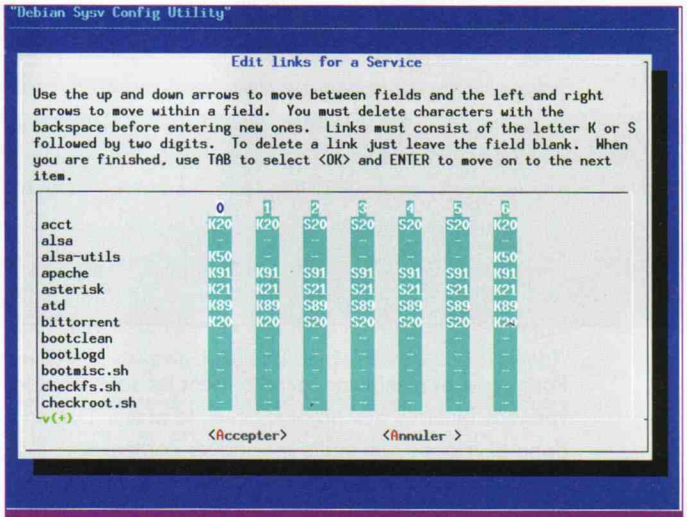
Si vous souhaitez créer un script d'init spécifique, je vous recommande la lecture et l'étude du script exemple /etc/init.d/skeleton. Très complet et parfaitement commenté, il constitue comme son nom l'indique le squelette pour tout nouveau script d'init. Bien entendu, il convient de respecter l'intégrité du système Debian et donc d'éviter au maximum l'ajout de fichier indépendant de tous paquets. Reportez-vous à l'article sur la création de paquet le cas échéant.

Un reproche qui est souvent fait aux distributions Debian concerne l'interface de gestion des runlevels. L'édition et la mise à jour des liens symboliques entre /etc/init.d et les /etc/rc*.d sont souvent réputées pénibles et peu ergonomiques. Pourtant, de base, le système Debian contient un utilitaire permettant de manipuler facilement ces liens. La commande update-rc.d permet de facilement gérer des configurations personnalisées. Par exemple, update-rc.d ssh remove supprimera complètement le service OpenSSH du système d'init. Notez que /etc/init.d/ssh doit avoir été supprimé au préalable (ce qui n'est pas « bien » au sens Debian) ou que vous devez utiliser l'option -f pour forcer la suppression. L'option -n est également très intéressante, puisqu'elle permet de simuler l'action à des fins de test.

update-rc.d monservice defaults permet de créer les liens automatiquement pour un démarrage du service dans les runlevels 2, 3, 4 et 5 ainsi qu'un arrêt de ce service dans les runlevels 0, 1 et 6. Par défaut, le code numérique est égal à 20, mais vous pouvez le préciser en fin de ligne de commande (et même en définir un différent pour start et stop). Enfin, dernier exemple, vous pouvez personnaliser à l'extrême en utilisant, update-rc.d monservice start 20 2 3 4 5 stop 50 0 1 6. Ici, monservice sera presque configuré comme

par défaut (démarrage en 2, 3, 4 et 5 et arrêt en 0, 1 et 6) à la différence que nous utilisons des valeurs numériques différentes pour `start` (20) et `stop` (50). Consultez la page de manuel de `update-rc.d` pour en savoir plus.

Si l'outil livré par défaut ne vous convient pas, vous pouvez toujours opter pour des interfaces plus conviviales (en mode console toujours). Les paquets `sysv-rc-conf` et `sysvconfig` fournissent une interface permettant une vision plus globale et plus illustrée des liens pour les scripts d'init. Les deux outils proposent sensiblement la même chose et il faudra les tester pour se faire un avis de leur ergonomie. Personnellement, je préfère largement configurer les scripts d'init avec le bon vieux `update-rc.d`.



Sysvconfig permet de configurer les runlevels et scripts d'init via une interface à menu. On aime ou on n'aime pas.

POURQUOI NE PAS UTILISER D'INITRD ?

Le principal intérêt d'un RAM Disk de démarrage réside dans le fait d'ajouter dynamiquement des fonctions dans le noyau. Autrement dit, de charger des modules pour pouvoir supporter le matériel sur lequel on boote.

Cependant, il faut savoir que le support des modules chargeables dynamiquement n'est pas nécessairement une bonne chose. Dans le cas d'un serveur dont vous connaissez parfaitement la configuration matérielle, il n'est pas nécessaire de faire appel à ce type de fonctionnalités. En effet, la configuration d'un serveur ne change pas fréquemment et la généricité du système n'est absolument pas intéressante, au contraire. Tout le support du matériel peut donc être inclus statiquement dans une image du noyau qui est alors compilée sans le support des modules.

Dernier avantage pour un serveur, l'absence de fonctionnalité de chargement dynamique de modules empêche l'installation de la pire forme de rootkit qui soit : celle d'un module noyau. L'utilisation d'un noyau statiquement compilé est donc un gain de sécurité non négligeable. C'est d'ailleurs la raison pour laquelle les noyaux installés par les hébergeurs sur leurs serveurs loués sous Debian GNU/Linux ne sont pas ceux de la distribution, mais des versions spécifiquement prévues.

Bien entendu, sans support des modules, il faut inclure dans le noyau le support du contrôleur de disque, de l'interface réseau, du système de fichier utilisé, etc. Ce dernier point fait que le noyau est alors capable de trouver lui-même la partition racine et monter seul le système de fichier correspondant. Le RAM disk de démarrage perd alors tout son intérêt.

QUICK TIP : COMMENT FAIRE UNE CAPTURE D'UNE CONSOLE EN FRAMEBUFFER ?

Vous vous demandez peut-être comment a été faite la capture qui se trouve quelques pages plus loin dans l'article sur la console BootSplash. Forcément, vous avez dû feuilleter le magazine et vous arrêter sur la capture (tout le monde aime les captures d'écran). Voici la méthode.

Le principe du *frame buffering* est un mappage mémoire. `/dev/fb*` fournit une méthode d'accès depuis l'espace utilisateur. Ainsi, il nous est possible de très simplement récupérer les valeurs brutes de ce qui se trouve à l'écran avec `sudo cat /ded/fb0 > ecran.raw`.

Ces données brutes ne sont pas utilisables directement avec une application de retouche ou un convertisseur de formats graphiques, car elles ne comprennent pas d'informations sur la résolution de l'écran. Il nous faut donc passer par un petit script Perl :

```
#!/usr/bin/perl -w
$w = shift || 320;
$h = shift || 240;
$pixels = $w * $h;
open OUT, "|pnmtopng" or die "Can't pipe pnmtopng: $!\n";
printf OUT "P6%d %d\n255\n", $w, $h;
while ((read STDIN, $raw, 2) and $pixels--) {
   $short = unpack('S', $raw);
   print OUT pack("C3",
      ($short & 0xf800) >> 8,
      ($short & 0x7e0) >> 3,
      ($short & 0x1f) << 3);
}
close OUT;
```

Il vous faudra l'utilitaire `pnmtopng` du paquet `netpbm` pour le faire fonctionner. A cette seule condition, vous pourrez utiliser ensuite :

```
% ./script.pl 1024 768 < ecran.raw > ecran.png
```

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

LE NOYAU LINUX ET DEBIAN

Avec une distribution aussi structurée et organisée que Debian GNU/Linux, le noyau occupe une place importante. En effet, celui-ci ne peut être considéré comme un simple paquet applicatif ou serveur. Son installation doit être aussi simple que celle d'un paquet mais sa reconstruction doit également permettre une grande souplesse.

Pour gérer et manipuler correctement les sources et les versions binaires des noyaux Linux, le projet Debian a donc développé des outils spécifiques comme `make-kpkg`. Mais avant d'entrer dans le détail de la recompilation du noyau «façon Debian», voyons déjà l'aspect le plus simple : l'installation et la mise à jour.

L'installation d'un noyau, ou plus exactement la mise à jour (puisque le système ne serait pas fonctionnel dans le cas contraire) se fait très simplement en utilisant `apt-get`. Mais encore faut-il choisir la bonne version pour votre machine. Le projet Debian met à disposition plusieurs variantes compilées et empaquetées du noyau Linux pour chaque version stable de ce dernier.

Ainsi, pour la version 2.6.18 on trouve pas moins de neuf paquets :

```
% apt-cache search --names-only "linux-image-2.6.18"
linux-image-2.6.18-3-486
        - Linux 2.6.18 image on x86
linux-image-2.6.18-3-686
        - Linux 2.6.18 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.18-3-686-bigmem
        - Linux 2.6.18 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.18-3-k7
        - Linux 2.6.18 image on AMD K7
linux-image-2.6.18-3-vserver-686
        - Linux 2.6.18 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.18-3-vserver-k7
        - Linux 2.6.18 image on AMD K7
linux-image-2.6.18-3-xen-686
        - Linux 2.6.18 image on PPro/Celeron/PII/PIII/P4
linux-image-2.6.18-3-xen-k7
        - Linux 2.6.18 image on AMD K7
linux-image-2.6.18-3-xen-vserver-686
        - Linux 2.6.18 image on PPro/Celeron/PII/PIII/P4
```

Les différents paquets se distinguent par le choix de la plate-forme dans la configuration des sources du noyau avant compilation. Le choix est relativement simple si vous connaissez votre matériel (ce que j'espère pour vous). Les paquets `*xen*` et `*vserver*` sont des versions «patchées» spécialement conçues pour fonctionner dans un environnement de para-virtualisation de type Xen ou Linux-Vserver.

NOTE

Les paquets `*smp*` destinés aux architecture multiprocesseurs n'existent plus. Toutes les versions binaires du noyau supportent à présent le SMP (Symmetric MultiProcessing). Une différenciation était faite car le support SMP utilisé sur un système monoprocesseur dégradait les performances avec les précédentes versions du noyau, ce qui n'est plus le cas à présent.

Un paquet `linux-image-2.6.X` n'est pas un paquet comme les autres. Lors de son installation, il va générer un élément indispensable au fonctionnement et surtout au démarrage du système : l'image d'un disque mémoire d'initialisation `initramfs`. En effet, si vous jetez un œil au contenu du fichier `linux-image-2.6.18-3-k7_2.6.18-7_i386.deb` par exemple, vous constaterez que le fichier `/boot/initrd.img-2.6.18-3-k7` n'est pas inclus. C'est, en effet, lors de l'utilisation d'`apt-get install` que le fichier est créé :

```
Finding valid ramdisk creators.
Using mkinitramfs-kpkg to build the ramdisk.
```

`mkinitramfs-kpkg` est appelé pour créer le RAM Disk. Il ne s'agit en réalité que d'un «wrapper» pour `mkinitramfs`. Ceci est très important car l'image `initramfs` dépend fortement de votre configuration. Ainsi, si vous avez installé une solution de RAID logiciel, il est impératif que les scripts présents dans le RAM Disk configurent et activent le RAID avant le montage et le passage sur le système de fichier racine. Si vous changez votre configuration vers du RAID logiciel (par exemple comme détaillé dans l'article présent dans le GLMF HS 18), vous devrez utiliser `update-initramfs` afin de prendre le changement en compte pour le prochain démarrage et donc mettre à jour votre image `initramfs`.

L'installation d'une nouvelle version du paquet noyau provoque également l'installation d'une nouvelle arborescence de modules noyaux dans `/lib/modules`. Le nom du répertoire créé dépend de la version du noyau et de sa révision. Ceci est fixé lors de la compilation du noyau et de la construction du paquet (voir plus loin). Accessoirement, si les sources utilisées pour la

construction du noyau empaqueté sont installées, un lien symbolique sera ajouté entre /lib/modules/2.6.*/source et /usr/src/linux-source-2.6.*. Dans le cas contraire, apt-get ne manquera pas de signaler le problème (qui n'en est pas vraiment un) :

```
Hmm. The package shipped with a symbolic link /lib/
modules/2.6.18-3-k7/source
However, I can not read the target: Aucun fichier ou
répertoire de ce type
Therefore, I am deleting /lib/modules/2.6.18-3-k7/source
```

Ce message est parfaitement normal lorsque vous installez un noyau provenant d'une distribution Debian puisque vous ne disposez pas de l'emplacement de construction original (/build/buildd/linux-2.6-2.6.18/debian/build/build-i386-none-k7 dans le cas présent).

RECOMPILATION D'UN NOYAU DEBIAN

En principe, la recompilation d'un noyau n'est pas strictement nécessaire. Seuls certains cas particuliers d'optimisation ou d'activation de fonctionnalités justifient cela. Néanmoins, il est important pour un administrateur système de connaître le modus operandi façon Debian afin de respecter, le cas échéant, l'intégrité de la distribution. Trois paquets au minimum sont importants pour procéder :

- les sources du noyau : linux-source-2.6.*
- le paquet kernel-package fournissant, entre autres, le script de construction du paquet binaire make-kpkg
- les fichiers d'en-têtes le la Libncurses, libncurses5-dev permettant la configuration du noyau via l'interface menuconfig (Libncurses/Dialog). Notez que cela n'est pas strictement nécessaire puisque le noyau dispose également d'une interface de configuration en ligne de commande par question/réponse. Enfin, si vous souhaitez quelque chose de plus convivial à la sauce GTK+ (make gconfig) ou Qt (make xconfig) il vous faudra installer d'autres paquets -dev.

Voyons tout d'abord la manière la plus classique de recompiler et reconstruire un paquet binaire pour le noyau. Notez qu'il est possible de procéder aux manipulations qui suivent en tant qu'utilisateur normal (via fakeroot) mais j'ai pris le parti d'aller au plus simple en prenant l'identité root. En effet, la logique veut qu'une telle manipulation est faite dans le but d'installer un nouveau composant essentiel au système et donc qu'elle relève de la responsabilité du super-utilisateur. Le seul intérêt d'utiliser fakeroot est de se prémunir contre d'éventuelles erreurs de manipulations (pour les commandes sans fakeroot). Ce qui ne doit de toute façon pas arriver lorsque le super-utilisateur reconstruit un noyau.

Commencez donc par installer les trois paquets cités plus haut puis placez-vous dans le répertoire /usr/src où vous trouverez linux-source-2.6.18.tar.bz2. Attention, il ne s'agit pas des sources d'origine (sources vanilla) du noyau Linux, contrairement à ce que le nom du fichier laisse entendre, mais une version spécifique à Debian et respectant *normalement* les DFSG (Debian Free Software Guidelines ou principes du Logiciel libre selon Debian).

Désarchivez ensuite les sources avec tar xfj, vous obtiendrez une arborescence dans linux-source-2.6.18. Entrez dans ce répertoire.

Plusieurs arguments peuvent être utilisés pour personnaliser la version et la révision du noyau ainsi que celle du paquet à construire. Dans un premier temps, nous utiliserons les valeurs par défaut nous permettant d'obtenir une version locale non conflictuelle capable de cohabiter avec une version du noyau identique à celle construite. Vous remarquerez, par exemple, que les modules installés par un paquet officiel sont placés dans /lib/modules/2.6.18-3-k7 qui correspond au paquet linux-image-2.6.18-3-k7. Le suffixe -3-k7 est ajouté par le responsable du paquet Debian précisément pour permettre aux utilisateurs de construire facilement leur noyau sans conflit.

Notre première manipulation sera absolument sans intérêt si ce n'est didactique. Nous allons construire un paquet en tous points identique à celui fourni par Debian.

Dans le répertoire des sources du noyau, copiez le fichier de configuration du noyau fourni par Debian de /boot/config-2.6.18-3-k7 en .config et lancez un make oldconfig pour vérification. Normalement rien ne vous sera demandé, oldconfig étant normalement utilisé pour adapter une configuration d'une précédente version du noyau à une nouvelle en interrogeant l'utilisateur sur les fonctionnalités non configurées au besoin.

Ceci fait, il ne vous reste plus qu'à lancer le processus de construction avec :

```
# make-kpkg --initrd kernel_image
```

Notez la présence de l'option --initrd, indispensable pour générer toutes les actions nécessaires pour le RAM Disk d'initialisation lors de l'installation du paquet. Si, dans le cas présent, vous oubliez cette option, le système sera incapable de démarrer (le support IDE/SATA/SCSI est, par défaut, compilé sous forme de modules). Après un délai dépendant à la fois de votre ou vos processeurs et de la mémoire disponible, vous obtiendrez le fichier linux-image-2.6.18_2.6.18-10.00.Custom_i386.deb dans /usr/src.

C'est, bien entendu, le paquet Debian correspondant à votre noyau fraîchement compilé que vous pourrez installer avec `dpkg -i`.

A ce stade, vous disposez normalement de deux noyaux 2.6.18. Le premier installé depuis le dépôt Debian et compilé par le responsable officiel et le votre :

```
% dpkg -l "linux-image-2.6.18*"
ii linux-image-2.6.18      2.6.18-10.00.Custom
          Linux kernel binary image for version 2.6.18
ii linux-image-2.6.18-3-k7 2.6.18-7
          Linux 2.6.18 image on AMD K7
```

La version Debian de notre noyau est `2.6.18-10.00.Custom` et celle officielle, `2.6.18`. Le paquet Debian quant à lui possède respectivement les valeurs `2.6.18-7` et `2.6.18-3-k7`, valeur que l'on retrouve sur le système une fois ce noyau booté avec :

```
% uname -s -r
Linux 2.6.18-3-k7
```

Nous allons maintenant construire un troisième noyau bien plus personnalisé. Première étape, dans le `/usr/src/linux-source-2.6.18`, nettoyez les sources et les informations de `make-kpkg` avec `make-kpkg clean`.

Vous pouvez maintenant utiliser les deux options importantes de `make-kpkg` :

`--revision` est destinée au système de paquet. Par défaut, la révision utilisée est `$(version)-10.00.Custom` soit le numéro de version du noyau plus une chaîne de caractères. C'est la version Debian telle qu'affichée par `dpkg -l` et qui vous permettra d'installer plusieurs noyaux concurrents sans conflit et sans que le système APT ne considère le nouveau paquet comme une mise à jour. Ce numéro de révision doit être composé uniquement de caractères alphanumériques et des caractères «+» et «.». De plus, il doit impérativement comporter un chiffre. Attention, lors de la première utilisation de cette option, sa valeur est enregistrée, au moins jusqu'à l'utilisation de `make-kpkg clean`. Sa valeur est stockée dans `conf.vars` et elle ne sera révisée que si `stamp-configure` et `stamp-debian` sont supprimés. Plus clairement, sans `make-kpkg clean`, vous ne pouvez pas utiliser cette option plusieurs fois de suite. En revanche, cela vous permettra de relancer une compilation partielle sur la même base de configuration Debian en omettant `--revision`.

`--append-to-version` vous permet d'influer sur la vraie version du noyau et donc sur le nom du chemin de recherche des modules (variable `EXTRAVERSION` dans le `Makefile` du noyau). C'est la valeur qui apparaît avec `uname -r`. Il est important de prendre cela en compte car, si vous utilisez comme argument `-3-k7` vous allez créer un conflit avec le paquet noyau déjà installé puisqu'ils partageront tous deux le même répertoire de modules, le même nom pour l'image RAM Disk et le fichier du noyau dans `/boot`. Ce numéro de version ne doit contenir que des caractères alphanumériques minuscules et les caractères «-», «.» et «+».

Relancez donc la configuration du noyau et apportez vos modifications, comme par exemple, la désactivation des fonctionnalités qui ne vous servent pas (PCMCIA, ISDN, FireWire, RAID, LVM, etc.). Enfin, relancez la construction avec :

```
# make-kpkg clean && make-kpkg \
--append-to-version=-hs28-1-k7 \
--initrd --revision=1.0.lef kernel_image
```

NOTE

l'utilisation de `make-kpkg clean` est indispensable après avoir utilisé `make menuconfig` car cette dernière commande crée un fichier `include/linux/version.h` qui ne tient pas compte de votre `--append-to-version`. `make-kpkg clean` permet de nettoyer les sources, et donc le fichier `version.h`. `make-kpkg` ne crée le fichier que s'il n'existe pas déjà, mais ne le mettra pas à jour s'il existe.

Après un délai normalement plus court puisque vous avez désactivé des fonctionnalités, vous obtiendrez le fichier `linux-image-2.6.18-hs28-1-k7_1.0.lef_i386.deb`. Installez-le comme à l'habitude puis observez la sortie de la commande `dpkg -l` :

```
ii  linux-image-2.6.18      2.6.18-10.00.Custom
          Linux kernel binary image for version 2.6.18
ii  linux-image-2.6.18-3-k7  2.6.18-7
          Linux 2.6.18 image on AMD K7
ii  linux-image-2.6.18-hs28- 1.0.lef
          Linux kernel binary image for version 2.6.18-hs28-1-k7
```

Et de trois ! Idem pour `/lib/modules` :

```
2.6.18/
2.6.18-3-k7/
2.6.18-hs28-1-k7/
```

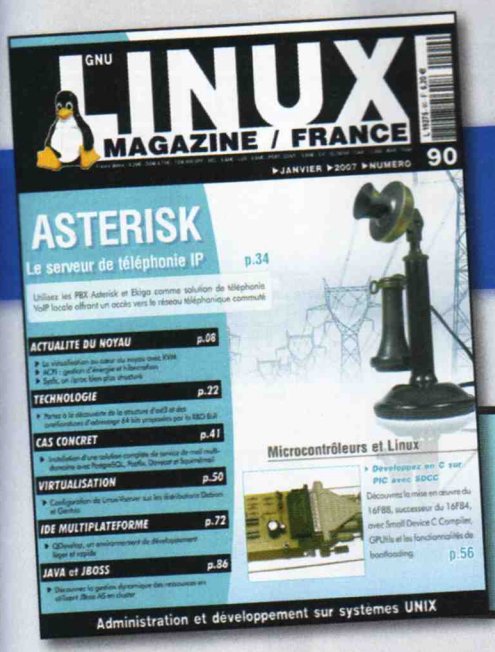
Vous êtes maintenant en mesure de construire autant de paquets qu'il vous plaira jusqu'à obtenir une version qui vous convienne. Vous pouvez, par exemple, créer une version entièrement statique du noyau et donc vous abstenir d'utiliser `--initrd`. Idéal pour un serveur.

COMPILATION DE MODULES

Entendez par «compilation de modules», la prise en charge de modules au format source qui ne sont pas intégrés dans la branche de développement officielle du

Abonnez - vous !

11 Numéros de Linux Magazine



1 an de bonne lecture, bien UNIX, bien technique... Bref...

LES 3 BONNES RAISONS DE VOUS ABONNER !

- Ne manquez plus aucun numéro
- Recevez Linux Magazine chaque mois chez vous, ou dans votre entreprise
- Economisez 15,20 €/an ! (soit plus de 2 magazines offerts !)

- Des offres de couplage sont disponibles
- Retrouvez les Tarifs étrangers hors France Métro sur www.ed-diamond.com

BON D'ABONNEMENT À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

*DIAMOND Éditions - LINUX MAGAZINE - BP 20142 - 67603 SÉLESTAT Cedex

11 Numéros de Linux Magazine à 53€ Soit Votre Linux Magazine à 4,82€

(Tarif au numéro dans le cadre d'un abonnement France Métro)

Offre France Métro

Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

LES 4 FAÇONS DE VOUS ABONNER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 15h -18h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

❍ Oui, je souhaite m'abonner à Linux Magazine pour 11 numéros

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Offres collectionneurs

Les anciens numéros !

TOUJOURS DISPONIBLES !

BON DE COMMANDE

À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

*DIAMOND Éditions - LINUX MAGAZINE - BP 20142 - 67603 SÉLESTAT Cedex

Bon de commande Linux Magazine

Référence	Prix / N°s	Qté.	Total
Linux Magazine 80 Run in memory	5,95 €		
Linux Magazine 81 Comment fonctionnent les générateurs de nombres pseudo-aléatoires	5,95 €		
Linux Magazine 82 eCos, une autre solution libre pour systèmes embarqués	5,95 €		
Linux Magazine 83 Greylist Eliminez le SPAM à la racine	5,95 €		
Linux Magazine 84 Déploiement de hotspots Wifi sécurisés	5,95 €		
Linux Magazine 85 Firewall: Netfilter & NuFW	6,20 €		
Linux Magazine 86 Serveur SMTP: Routage des mails avec Postfix	6,20 €		
Linux Magazine 87 Le point sur Mono .NET Java et les Brevets	6,20 €		
Linux Magazine 88 Sécurité: Smartcards & Tokens	6,20 €		
Linux Magazine 89 Utilisation avancée de XEN	6,20 €		

Bon de commande Linux Magazine Hors Série

LM HS 10 Installation de votre serveur internet	5,95 €		
LM HS 12 Firewall votre meilleur ennemi Acte 1	5,95 €		
LM HS 13 Firewall votre meilleur ennemi Acte 2	5,95 €		
LM HS 15 GIMP et la Photo	5,95 €		
LM HS 16 Kernel (1)	5,95 €		
LM HS 17 Kernel (2)	5,95 €		
LM HS 18 Haute Disponibilité	5,95 €		
LM HS 19 The Gimp 2.0	5,95 €		
LM HS 20 PHP 5	5,95 €		
LM HS 21 Recyclez vos PC	6,40 €		
LM HS 22 GIMP et le Web	6,40 €		
LM HS 23 Linux et electronique	6,40 €		
LM HS 24 Linux Embarqué	6,40 €		
LM HS 25 Linux Embarqué 2	6,40 €		
LM HS 26 Spécial The GIMP	6,40 €		
LM HS 27 Électronique et Linux	6,40 €		
		TOTAL	
		Frais de port France Metro	+ 3,81 €
		Frais de port Etranger	+ 5,34 €
		TOTAL	

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ **Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions***

☐ **Paiement par carte bancaire :**

N° Carte :

Expire le : **Cryptogramme Visuel :** Voir image ci-dessous

Date et signature obligatoire : 200

LES 4 FAÇONS DE COMMANDER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 15h -18h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

noyau linux. C'est le cas, par exemple, du support pour LIRC (récepteur infra-rouge), des Webcam spca5xx ou encore des modems USB ADSL Eagle. Les raisons de la non-inclusion dans les sources du noyau peuvent être diverses, code trop «sale» et problèmes de compatibilité avec Linus, problèmes liés aux licences ou encore utilisation d'objets binaires propriétaires (firmwares).

Quoi qu'il en soit, Debian fournit quelques-uns de ces modules en version source et parfois binaire. Dans le cas de la version binaire, il faudra s'assurer de la compatibilité avec votre version du noyau. Les paquets sont normalement judicieusement nommés, mais ce n'est pas toujours le cas. D'autre part, ces version binaires s'installeront dans le répertoire des modules propres à la version Debian du noyau. Si vous avez compilé votre propre version, votre noyau ne les prendra pas en compte.

Heureusement, pour régler le problème, nous disposons aussi des sources de ces modules en version empaquetée. C'est le cas, par exemple, du module pour LIRC, `lirc-modules-source`, que nous allons utiliser en guise d'exemple.

Nous venons de compiler notre noyau ou allons maintenant nous attaquer au module pour LIRC. Pour cela, il nous suffit tout d'abord d'installer les sources du modules avec `apt-get install lirc-modules-source`, ce qui aura pour effet d'ajouter un fichier `lirc-modules.tar.gz` dans notre `/usr/src`. En désarchivant celui-ci, nous obtenons une arborescence `/usr/src/modules/lirc` contenant les sources adaptées par Debian.

Entrez alors dans `/usr/src/linux-source-2.6.18` et utilisez :

```
# make-kpkg --append-to-version=-hs28-1-k7 \
modules_image
[...]
exec debian/rules  DEBIAN_REVISION=1.0.lef
          APPEND_TO_VERSION=-hs28-1-k7  modules_image
[...]
Module /usr/src/modules/lirc processed fine
```

Notez que nous devons réutiliser `--append-to-version`. `--revision` pour sa part n'est pas nécessaire puisqu'il est enregistré dans les sources du noyau suite à la compilation précédente. Vous pouvez d'ailleurs le constater dans les premières lignes qui défilent à l'écran. La page de manuel de `make-kpkg` est d'ailleurs très claire sur ce point : «Notez aussi qu'une fois que vous avez utilisé --append_to_version toto pour la configuration ou la construction du kernel-image, vous devez aussi utiliser la même option lors de lancements ultérieurs de make-kpkg (par exemple, pour construire des modules indépendants, ou autres). make-kpkg ne se souvient pas de l'argument toto à chacun des lancements de la commande (ce comportement est différent de --revision, qui est lui persistant lors des différents lancements)». C'est le seul point délicat de l'opération.

Une fois la compilation terminée, vous trouverez un fichier `lirc-modules-2.6.18-hs28-1-k7_0.8.0-9+1.0.lef_i386.deb` dans `/usr/src`. Il s'installera comme précédemment avec `dpkg`. On retrouvera les modules compilés dans `/lib/modules/2.6.18-hs28-1-k7/misc`. Enfin, avec `dpkg -l` on retrouve bien notre nom de paquet `lirc-modules-2.6.18-hs28-1-k7` et sa version `0.8.0-9+1.0.lef` avec notre argument pour l'option passée précédemment via `--revision`.

Je sais que j'insiste sur ce point, mais il faut bien comprendre que c'est la source de nombreux problèmes pour les utilisateurs qui construisent leur noyau avec une distribution Debian et compatible. La seule maîtrise de ces gestions de versions suffit à éviter 95% des problèmes et donc des plaintes dans les groupes de discussion, les listes de diffusion et les forums.

Si vous voulez construire un paquet binaire pour le module de manière à ce qu'il s'ajoute aux modules installés par un paquet noyau Debian, vous devez utiliser `--append-to-version=` suivi d'une partie de la valeur retournée par `uname -r` (`-1-k7` par exemple). Ceci fonctionne, mais il est fortement conseillé de compiler son noyau personnel et ensuite ses paquets de modules pour s'assurer d'une certaine cohérence. Des problèmes liés au compilateur peuvent apparaître, mieux vaut une configuration homogène et qui ne laisse pas planer de doute sur la responsabilité d'un développeur Debian (via un rapport de bogue, par exemple).

PATCH DU NOYAU

Un certain nombre de fonctionnalités d'un noyau Linux, lorsqu'elles ne sont pas intégrées dans les sources officielles, ne peuvent pas prendre la forme de modules. Soit parce qu'il s'agit d'un choix du développeur, soit parce que c'est techniquement impossible.

La solution pour inclure les fonctionnalités proposées passe donc par la modification des sources du noyau avant compilation. Debian prend parfaitement en charge ce type de modification sous la forme de paquets contenant un patch. Il existe un certain nombre de ces paquets mais vous ne devez jamais considérer ceci comme un gage de sécurité de la part de Debian. Ceci concerne, par exemple, le paquet `kernel-patch-2.6-reiser4` qui précise dans son descriptif que, non seulement il ne s'applique qu'à des noyaux en version 2.6.8 et 2.6.8.1, mais qu'il n'est absolument pas recommandé d'en faire usage sur une machine de production.

Quoi s'il en soit, l'application du patch avant compilation est conditionnelle. Dans la configuration Debian du patch,

on trouve les différentes versions du noyau auxquelles il peut s'appliquer. Si nous prenons le cas de `kernel-patch-bootsplash`, on retrouve une variable `KVERSIONS` dans le fichier `/usr/src/kernel-patches/all/apply/bootsplash` qui ne spécifie aucune version au-delà de 2.6.16. En fonction de la version du noyau compilé, c'est l'un des patchs présents dans `/usr/src/kernel-patches/diffs/bootsplash` qui sera appliqué.

NOTE

Bien que cela ne soit pas recommandé, il est parfois possible de forcer l'application d'un patch pour une version non prévue du noyau. L'utilitaire `patch` permet en effet une certaine tolérance (ce qui n'est pas le cas du code lui-même). En modifiant le contenu du fichier dans `kernel-patches/all/apply/` on peut donc «forcer» l'application d'un patch en trompant le test et en ajoutant soi-même la nouvelle version. Une solution plus propre est, bien entendu, de générer un nouveau fichier `.diff`, de le tester et de le proposer au responsable du paquet.

A ce point de l'article, l'installation d'un patch est relativement simple. Il suffit, en effet, d'installer le paquet correspondant, comme `kernel-patch-bootsplash`. Automatiquement une nouvelle arborescence `/usr/src/kernel-patches` est créée et peuplée. A l'instar des modules, l'ensemble des patchs sont installés dans cet unique répertoire. En revanche, il ne sont pas appliqués automatiquement mais doivent être clairement spécifiés dans la ligne de commande de `make-kpkg`. Ainsi pour notre exemple, nous utilisons :

```
# make-kpkg clean && make-kpkg \
--append-to-version=-hs28-1-k7 \
--added-patches bootsplash \
--initrd kernel_image
```

Nous constatons parmi les différentes étapes de la construction, l'application correcte du patch :

```
START applying bootsplash patch (Bootsplash)
Testing whether "Bootsplash" patch for 2.6.15 applies (dry run):
"Bootsplash" patch for 2.6.15 succeeded
Removing empty files:
Done.
```

Le message dépend du patch, tous ne sont pas aussi explicites quant à leur application. Notez que si vous appliquez plusieurs patchs, il faut les lister tous en argument de `--added-patches` séparés par des virgules. Le ou les patchs sont appliqués lors de la phase de configuration des sources, il sont «désappliqués» (unpatch) lors d'un `make-kpkg clean`.

Certains patchs appliquent simplement ces changements dans le code, mais également dans la configuration. Dans le cas de `kernel-patch-debianlogo` par exemple, le fichier de configuration du noyau est également changé. La configuration des sources ayant lieu après l'application du patch, le changement est détecté (comme avec un `oldconfig`) et le système demande votre intervention :

```
[...]
Standard 224-color Linux logo (LOGO_LINUX_CLUT224) [Y/n] y
Debian GNU/Linux Open Use logo (LOGO_LINUX_DEBIAN) [Y/n]
(NEW)
```

Les problèmes surviennent avec les machines limitées en ressources. En effet, sur un pauvre AMD Duron il serait intéressant de ne pas avoir à recompiler l'intégralité du noyau dans le cas où une version précédente identique mais non patchée est déjà compilée. Malheureusement, il ne semble pas y avoir de solution «propre». Il est impératif d'utiliser `make-kpkg clean` avant d'utiliser à nouveau `make-kpkg kernel_image`, et donc, de recommencer un cycle complet de compilation de longue durée. Ceci vous incitera sans doute à consulter l'ensemble des patchs intéressants ou nécessaires pour votre configuration AVANT de perdre du temps avec la construction d'un noyau «juste pour voir». Bien entendu, avec une configuration plus récente, voire dernier cri, la compilation n'est plus un problème. Il semblerait que les nouveaux processeurs fassent réellement des merveilles dans ce domaine (les utilisateurs Gentoo sont aux anges).

CONCLUSION

Comme précisé en début d'article, la recompilation du noyau, mais également de tous paquets binaires, doit être motivée. Dans la plupart des cas, elle ne sera pas nécessaire. Les distributions Debian sont déjà suffisamment modulaires pour offrir la majorité des fonctionnalités qu'un utilisateur puisse attendre. Vous remarquerez toutefois que, même pour ces manipulations qui restent donc rares, les outils développés et proposés sont exemplaires.

Si vous comptez vous pencher davantage sur la construction de noyaux, et passer du temps à expérimenter les différents patchs, modules et paquets, je vous conseille vivement de vous inscrire aux différentes listes de diffusion des développeurs Debian. Vous pouvez également surveiller régulièrement le système de suivi de bogues (voir article sur le sujet dans le présent numéro). Vos résultats, vos problèmes et vos solutions peuvent être utiles, tout comme le temps que vous consacrerez à la tâche.

EN BREF : LE PATCH BOOTSPLASH

Dans le précédent article, il est fait mention du patch kernel-patch-debianlogo et kernel-patch-bootsplash. Tous deux utilisent la capacité qu'a le noyau livré par Debian de démarrer avec une console en FrameBuffer. Voilà l'occasion rêvée, mais un peu hors sujet, de parler du second patch : BootSplash.

Le *frame buffering* est une méthode qui vous permet d'utiliser une carte graphique. La technique consiste à « mapper » en mémoire les valeurs des pixels qu'affiche la carte graphique. Écrire dans cette zone mémoire revient à écrire sur l'écran. Bien entendu, cela implique un mode spécifique définissant la résolution et la profondeur de couleurs. Le noyau binaire par défaut livré par Debian permet d'utiliser le frame buffering. Il suffit de passer au noyau le paramètre `vga=` suivi d'une valeur spécifiant le mode à utiliser.

On caractérise souvent le fait d'avoir la fonctionnalité de frame buffering activée par la présence d'une petit mascotte de Linux à l'écran (Tux). Le nombre de Tux indiquant, par la même occasion, le nombre de processeurs trouvés au démarrage. Vous n'avez pas de Tux en utilisant par exemple `vga=791` (1024x768, couleurs 16 bits) ? C'est normal, l'option de configuration `CONFIG_LOGO` n'est pas définie par défaut dans un noyau Debian. Mais, il y a mieux (selon les goûts de chacun) que le Tux. Il y a BootSplash, un système de démarrage graphique reposant sur les fonctionnalités de frame buffering.

La première chose à faire pour profiter de ce système qui existe sous forme de paquet est de patcher votre noyau avec `kernel-patch-bootsplash` comme expliqué précédemment. Compilez le noyau, construisez le paquet et installez-le.

Il vous faudra ensuite installer la partie « *user space* » prenant la forme du paquet `bootsplash` qui permet de piloter l'affichage. Le but est simple et multiple :

- Afficher un *splash screen* au démarrage.
- Offrir une console agrémentée de graphismes (fond, logo, etc.).
- Permettre d'afficher durant le démarrage du système une barre de progression indiquant les étapes en cours et le lancement des différents services (idem lors de l'arrêt du système).

BootSplash utilise un système de thèmes. Il est donc possible de personnaliser l'ensemble. Par défaut, un thème Debian est installé dans `/etc/bootsplash/themes/debian` et un lien symbolique `current` est créé pointant vers ce répertoire. Lors de l'installation du paquet `bootsplash`, il vous est demandé s'il faut activer BootSplash et s'il est nécessaire de reconstruire le RAM Disk de démarrage. Ceci est, bien entendu, indispensable. Le fichier `/etc/default/bootsplash` contient différents paramètres conditionnant l'utilisation de BootSplash et sa configuration (taille, thème, etc.) et il doit être présent.

Une fois tout ceci installé, vous pourrez modifier votre `lilo.conf` ou le `menu.lst` de GRUB pour ajouter les arguments `vga=791` et `splash=silent` à passer au noyau (pensez à garder une porte de sortie en conservant une entrée pour le précédent noyau).

Dès le premier démarrage, vous devez voir apparaître un magnifique écran de démarrage et finir sur une console fort sympathique (bien qu'un peu chargée graphiquement) :

Plein de graphiques, mais pas de barre de progression. Elle reste désespérément vide. En effet, le système d'Init doit piloter la progression de la barre pour que cela fonctionne. Seul problème, `/etc/init.d/rc` supporte `usplash` (un autre système du même type), mais pas BootSplash. Installez alors le paquet source de `sysv-rc-bootsplash`. Dans `sysv-rc-bootsplash-1.0.4`, vous trouverez deux fichiers : `rc.bootsplash` la version BootSplash et `rc.sysv-rc`, le `rc` original. Mais il ne s'agit pas du bon fichier. Copiez donc `/etc/init.d/rc` en `rc.sysv-rc`, reconstruisez le paquet et installez-le. Tadaaaa ! Le tour est joué.

Pour avoir d'autres thèmes, ajoutez une entrée à votre `/etc/apt/source.list` : `deb http://bootsplash.de/ unstable main`

Il vous suffit alors, après installation d'un thème, de modifier le `/etc/default/bootsplash`. N'oubliez pas d'utiliser `update-initramfs -u` pour mettre à jour le RAM Disk de démarrage.

NOTE

Il vous est, bien entendu, possible d'utiliser `usplash`. Celui-ci nécessite également le patch BootSplash du noyau et le paquet `bootsplash`, mais évite de devoir jouer avec le script `rc`. Cependant, vous en conviendrez avec moi, le thème par défaut livré sous la forme du paquet `debian-edu-artwork-usplash` est bien moins spectaculaire.

COMPRENDRE ET UTILISER LE SUIVI DE BOGUES

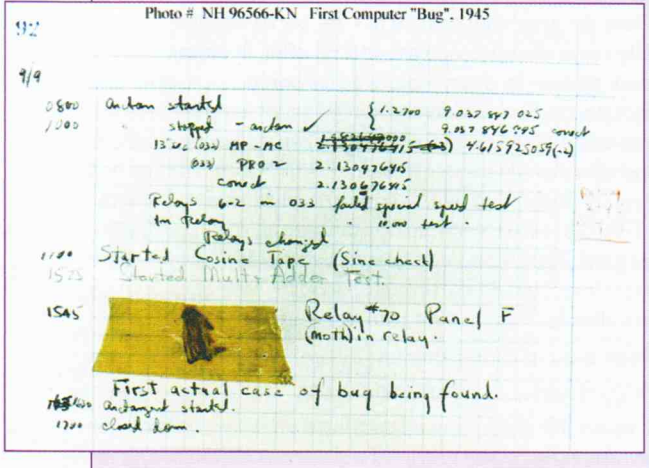
Selon Wikipédia, « un bogue informatique est une anomalie dans un programme informatique l'empêchant de fonctionner correctement. [Sa] gravité peut aller de bénigne (défauts d'affichage mineurs) à majeure (explosion du vol 501 de la fusée Ariane 5) ».

Je n'ai jamais eu connaissance du fait qu'une distribution Debian soit impliquée de près ou de loin dans le fonctionnement d'une fusée Ariane 5. Il n'en reste pas moins que les bogues (de l'anglais « bugs ») représentent le problème majeur de tout système informatique et qu'une distribution Debian est précisément un système informatique.

Votre distribution est donc pleine de bogues. Mais ne vous inquiétez pas, le projet Debian dispose d'un système très évolué permettant de lutter efficacement contre ce problème. Mieux encore, vous pouvez aider à la tâche via l'infrastructure en question, si ce n'est pour éliminer les bogues, au moins pour les trouver et les dénoncer.

LE PREMIER BOGUE INFORMATIQUE

Le terme est parfois faussement attribué à Grace Hopper : une anecdote raconte qu'elle aurait découvert qu'un insecte (bug), coincé entre deux contacts du relais qui faisait fonctionner l'appareil, était la raison du mauvais fonctionnement d'un des premiers ordinateurs électromécaniques.



En 1946, Hopper a rejoint la faculté de Harvard au laboratoire où elle a continué son travail sur Mark II et Mark III. Elle a attribué une erreur dans Mark II à un papillon nocturne pris dans un relais, créant le terme bug. L'insecte fut enlevé avec soin et placé dans un journal de bord (photo). Cette première anomalie a popularisé l'expression bug ou bogue pour représenter les erreurs dans un programme. (source Wikipédia)

L'infrastructure Debian permettant le travail collaboratif contre les bogues est appelée (très judicieusement) le « *Debian bug-tracking system* » ou BTS Debian pour les intimes. Sa partie la plus visible et la plus publique est le site Web **bugs.debian.org**. Là, vous trouverez tous les rapports, les résumés et les discussions en rapport avec les bogues.

Il faut bien comprendre que le BTS d'une distribution n'est, en principe, pas prévu pour servir les projets amonts, les projets produisant les applications et outils intégrés dans la distribution. Le BTS Debian ne se substitue pas aux infrastructures des projets amonts, il est complémentaire. C'est à la charge du responsable d'un paquet de déterminer si le bogue est spécifique à Debian, s'il peut être corrigé « localement » ou si la correction est le travail du projet amont. Des situations critiques peuvent même conduire à la suppression pure et simple d'un paquet. Ce fut le cas avec OpenWebmail, un Webmail écrit en Perl et les quelques messages dans les commentaires du BTS parlent d'eux-mêmes : « *The code is rife with vulnerabilities, and needs to be audited line by line ; I'm not sure this is likely anytime soon. I think we should remove it.* » (Andrew Pollock – 28 avril 2005).

Pour utiliser le BTS Debian, il faut tout d'abord comprendre son organisation et la classification des bogues.

FONCTIONNEMENT GÉNÉRAL DU BTS

Le BTS Debian est une énorme base de données. Il existe une entrée pour chaque bogue. Ceux-ci sont identifiés par un numéro et se rapportent nécessairement à un paquet. Lorsqu'un utilisateur ou un développeur fait un rapport de bogue, une entrée et un identifiant numérique sont créés.

Une entrée dans le BTS peut être dans l'un des quatre états possibles :

- *Open* : Le bogue est « ouvert », il n'est pas encore corrigé.
- *Forwarded* : Le bogue a été rapporté aux développeurs du projet amont afin qu'ils règlent le problème. La suite n'est plus dépendante du responsable du paquet. Souvent le projet Debian attend simplement une nouvelle version du source amont.
- *Resolved/Done* : Le bogue est éliminé, le problème est corrigé.

- *Pending* : Le travail est en cours, un développeur Debian a pris en charge le problème et tente de le résoudre. Il n'est pas nécessaire qu'une autre personne passe du temps dessus. Cet état « en cours » indique également que le problème est en phase d'être résolu et qu'il s'agit d'une question de temps et de délai de propagation de la correction. En réalité, il ne s'agit exactement d'un état du bogue, mais d'une étiquette concernant le rapport de bogue. Cependant, « pending » est utilisé comme état du bogue sur le site Web du BTS (**http://www.debian.org/Bugs/**).

Pour classifier les bogues, il faut déterminer leur niveau de gravité. Il existe sept niveaux, mais seuls les quatre premiers d'entre eux sont utilisables pour un rapport de bogue. Les trois autres permettent de qualifier un bogue de manière spécifique, car, souvent, il est si grave qu'il a des effets sur l'ensemble de la distribution ou qu'il nécessite un comportement spécial de la part de développeurs Debian. N'utilisez pas ces niveaux de gravité pour vos rapports de bogue. Si le problème est effectivement très important, un développeur Debian requalifiera le bogue de manière adéquate.

Les quatre premiers niveaux, en commençant par le moins important, sont :

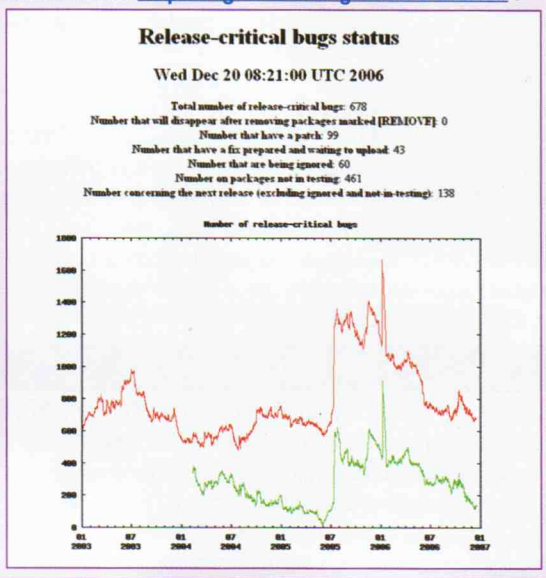
- *wishlist* : Il ne s'agit pas vraiment d'un bogue, mais d'un souhait comme l'ajout d'une fonctionnalité ou une modification dans l'organisation des fichiers du paquet. Il peut également s'agir d'un bogue réel, mais trop compliqué à résoudre. Si Windows était un paquet Debian, un rapport de bogue de niveau wishlist pourrait être : « empêcher les *spywares* d'infecter la machine » ;)
- *minor* : Le bogue ou le problème n'empêchent pas le fonctionnement et l'intérêt du paquet. Il s'agit souvent de bogues faciles à résoudre. Un exemple est le bogue 354054 concernant le paquet `vim`. Les fichiers source d'octave ne sont pas correctement indentés. Ceci n'interdit pas l'utilisation de Vim, mais c'est embêtant.
- *normal* : C'est la valeur par défaut qui s'applique à la plupart des bogues.
- *important* : C'est le plus haut niveau de gravité pour un rapport de bogue fait par un utilisateur. Il concerne un problème qui nuit gravement à l'utilité du paquet. Il n'en devient pas pour autant inutilisable. Par exemple, le bogue 306539, toujours concernant Vim, précise que la console est « en vrac » à la sortie de l'éditeur avec des locales UTF-8 sous certaines conditions. On voit bien qu'il s'agit de problèmes très gênants, mais qui ne surviennent que dans des conditions très précises.

Les trois autres niveaux de gravité, toujours du plus faible au plus important, sont :

- *serious* : Le paquets n'est pas distribuable, car il viole sévèrement la charte Debian pour la prochaine distribution stable (**http://release.debian.org/etch_rc_policy.txt**). Le document précise clairement ce qui cause une « sévère violation de la charte Debian ». Toujours concernant Vim, le bogue 381526, est une bonne démonstration. Une précédente version du paquet utilisait `/usr/share/doc/vim` comme un répertoire, celui-ci était utilisé ensuite par le paquet `vim-doc`. Cependant, une nouvelle version du paquet `vim` utilisait le répertoire de documentation comme un lien symbolique, mais `dpkg` ne fait pas le changement répertoire vers lien symbolique. Une erreur survenait donc lors de la mise à jour du paquet.
- *grave* : Un bogue de ce niveau rend le paquet inutilisable (ou presque). Il peut s'agir de pertes de données ou de problèmes de sécurité. Ce qui différencie principalement ce niveau du suivant est l'influence du bogue. Ici, elle se limite à un utilisateur. Soit, il perd des données personnelles, soit une faille de sécurité le met en danger (rend possible l'accès à ses données). Exemple : le bogue 320017, découlant de la découverte par Georgi Guninski d'une faille de sécurité dans Vim.
- *critical* : Voici le top du top dans la catégorie « bogues catastrophiques ». Un bogue de ce niveau de gravité concerne une perte de données pour le système (et non juste pour l'utilisateur du paquet), une faille de sécurité globale au système, le fait qu'il rende inutilisable d'autres paquets ou, enfin, qu'il bloque le fonctionne du système. Le bogue 330474 concernant le MUA Mutt est explicite : le simple fait d'ouvrir une mbox réécrivait la valeur du champ `Content-Length` dans l'en-tête des messages à zéro. La conséquence directe était, bien entendu, l'écrasement du corps du message dès la réouverture de la mbox. C'est une perte de données grave pouvant influer sur l'ensemble du système (serveurs POP3/IMAP, MTA, Cron, etc.).

LES BOGUES RC

Un bogue RC est un « *Release critical bug* », un bogue empêchant la sortie d'une nouvelle distribution stable de Debian. Ces bogues de gravité critical, grave et serious ne doivent pas apparaître dans une distribution stable. Ils doivent être corrigés ou bien les paquets en cause doivent être supprimés. Le compte de ces bogues est tenu en permanence et un résumé est donné sur **http://bugs.debian.org/release-critical/** :



Les bogues RC sont très importants et, habituellement, toute l'énergie des développeurs Debian leur est consacrée. Cependant, il arrive que le ou les bogues ne puissent pas être corrigés localement (à Debian) et requièrent l'action des développeurs amonts. Il faut alors que le développeur Debian en charge du paquet incite les membres du projet amont à plus de réactivité, sachant que les intérêts de chacun peuvent être différents.

Ce n'est que lorsque le nombre de bogues RC passe en dessous d'un certain seuil que la décision est prise de publier une nouvelle distribution stable.

PRENDRE CONNAISSANCE DES BOGUES EN COURS

Si vous rencontrez un problème avec un paquet ou un logiciel installé via un paquet, vous devez consulter le BTS Debian après avoir, bien entendu, analysé le problème localement. La méthode la plus simple est d'utiliser votre navigateur Web et de le pointer sur **http://www.debian.org/Bugs/** (ou **http://bugs.debian.org** qui vous redirigera). Là, vous trouverez l'interface Web d'interrogation du BTS.

Il vous suffit alors de préciser le paquet, la distribution et les options qui vous intéressent (version du paquet, niveau de gravité, état du bogue, etc.).

Une autre solution plus rapide consiste à préciser le nom du paquet dans l'URL. Par exemple : **http://bugs.debian.org/mutt**. Vous serez alors redirigé directement vers **http://bugs.debian.org/cgi-bin/pkgreport.cgi?pkg=mutt;dist=unstable**. Notez cependant que la distribution concernée est Unstable, ce qui n'est pas toujours souhaitable.

Si vous avez besoin d'avoir des informations sur un bogue précis, vous pouvez préciser dans l'URL, en lieu et place du nom du paquet, le numéro du bogue : `http://bugs.debian.org/376136`. Vous aurez alors le détail de tout ce qui concerne ce bogue ainsi que les discussions s'y rapportant (dont le rapport de bogue initial).

REPORTBUG

Pour simplifier les choses et permettre aux utilisateurs et aux développeurs de gérer les bogues plus facilement, un utilitaire écrit en Python a été créé : `reportbug`. Il vous permettra non seulement de prendre connaissance des bogues rapportés, mais également d'ajouter des informations et d'en signaler de nouveaux.

ATTENTION

`reportbug` n'est pas un jouet. Il permet d'interagir avec le BTS Debian et donc avec l'ensemble des développeurs et des responsables de paquets. Lisez attentivement les recommandations et les indications à l'écran, ne faites pas d'essais stupides, soyez clairs dans vos explications et vérifiez absolument tout ce que vous annoncez.

Pour faire connaissance avec `reportbug`, commencez par installer le paquet correspondant avec `apt-get install reportbug`. Dans les explications qui vont suivre, si vous souhaitez essayer, utilisez systématiquement l'option `-d` ou `--debug`. Ceci changera le comportement final de l'utilitaire, mais non son utilisation. Aucun vrai rapport de bug ne sera envoyé, vous seul le recevrez.

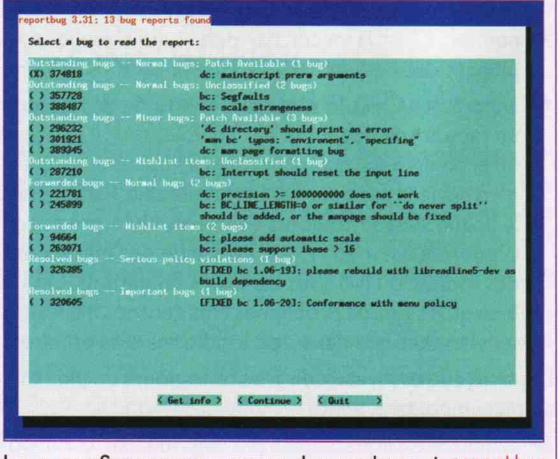
A présent que vous disposez de l'outil, nous allons voir un cas pratique (simulé, bien entendu). Notre paquet sera `bc` et notre rapport de bogue concernera l'ajout de couleurs dans les résultats affichés par l'utilitaire. Nous lançons donc `reportbug` (avec l'option `-d`) :

```
% reportbug -d
Please enter the name of the package in which you have
found a problem, or type 'other' to report a more
general problem.
>
```

NOTE

La toute première fois que vous lancerez `reportbug`, un message vous demandera dans quel mode d'utilisation vous souhaitez travailler. Le mode `novice` est le choix par défaut et une excellente solution pour débuter.

Viendra ensuite la question de l'interface à utiliser. Vous aurez le choix entre `text` (monde console en ligne de commande) et `urwid` (en console, mais avec une interface à menus). Dans le reste de l'article, nous utiliserons l'interface `text` (pour des raisons de démonstration, mais également parce que les couleurs de l'interface à menus sont particulièrement horribles). Notez que l'interface à menus nécessite l'installation du paquet `python-urwid`. Dans le cas contraire, `reportbug` se rabat sur l'interface `text`.



Le « configurateur » vous demandera si `reportbug` dispose de temps en temps d'un accès à Internet. Dans la plupart des cas, la réponse est affirmative. L'option inverse n'est pas traitée dans cet article. Je pars du principe que l'interaction avec le BTS nécessite une telle connectivité.

Offres de couplage

Lisez-vous RÉGULIÈREMENT :

100 % Linux

et / ou

100 % Sécurité

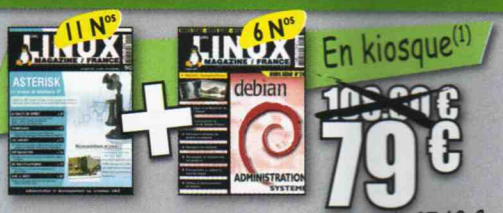
et / ou

100 % pratique

et / ou

Apprivoisez votre pingouin !

Si OUI, alors ces offres d'abonnement à tarif préférentiel vous sont destinées...



En kiosque(1) ~~108,60 €~~ **79 €**

soit une économie de 27.60 €

En kiosque(3) ~~154,60 €~~ **105 €**

soit une économie de 49,60 €

En kiosque(2) ~~115,20 €~~ **83 €**

soit une économie de 33,20 €

En kiosque(4) ~~190,80 €~~ **129 €**

soit une économie de 61,30 €

(1) Pour 11 N°s Linux Magazine + 6 N°s Linux Mag HS - (2) Pour 11 N°s Linux Magazine + 6 N°s MISC - (3) Pour 11 N°s Linux Magazine + 6 N°s MISC + 6 N°s Linux Mag. HS - (4) Pour 11 N°s Linux Magazine + 6 N°s MISC + 6 N°s Linux Mag. HS + 6 N°s Linux Pratique

OFFRE DE COUPLAGE À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

*DIAMOND Éditions - LINUX MAGAZINE - BP 20142 - 67603 SÉLESTAT Cedex

☐ **OUI, je m'abonne et désire profiter des offres spéciales de couplage**

Référence de l'offre :	Prix	Qté.	Total
☐ 11 N°s Linux Mag. + 6 N°s Linux Mag HS	79 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC	83 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC + 6 N°s Linux Mag HS	105 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC + 6 N°s Linux Mag HS + 6 N°s Linux Pratique	129 €		
OFFRES VALABLES UNIQUEMENTS EN FRANCE MÉTRO.		TOTAL	

Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

LES 4 FAÇONS DE VOUS ABONNER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 15h -18h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Boostez votre collection !

Avez-vous L'ÂME du COLLECTIONNEUR ?

Vous recherchez un magazine en particulier? Allez sur www.ed-diamond.com pour voir le sommaire détaillé de chaque magazine et ensuite... Boostez votre collection avec les "POWER PACKS x5", soit 5 Linux Magazine pour 15€ et les "POWER PACKS x10", soit 10 Linux Magazine pour 25€ à choisir dans la liste ci-dessous :

LES 4 FAÇONS DE COMMANDER !

- Par courrier postal en nous renvoyant le bon ci-dessous.
- Par le Web, sur notre site : www.ed-diamond.com.
- Par téléphone (paiement C.B.) entre 9h-12h & 15h -18h au 03 88 58 02 08.
- Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

Choisissez vos numéros dans le tableau ci-dessous*

* Seuls les numéros, ci-dessous, sont disponibles pour une commande de POWER PACKS par x5 et x10

N°06	GNOME - The Gimp	N°36	Linux embarqué : Le projet mGlinux	N°63	Les threads : kernel 2.6 et 2.4
N°07	Dopez Linux	N°37	L'impression sous Linux	N°64	Adamoto
N°08	Le futur résolument objet	N°38	Le desktop Shell : Enlightenment	N°65	Théorie et pratique : Supervision avec Nagios
N°09	Prêt pour le jeu !	N°39	Sécurité : Patchez votre noyau !	N°66	Créez votre Distribution Live
N°10	The HURD : 100% GNU	N°40	MySQL : la base de donnée OpenSource	N°67	C# .NET
N°11	Exclusif : l'avenir de G.N.O.M.E	N°41	Steganographie ou l'art de la dissimulation de données	N°68	Le crash disque vous guette
N°12	NT et Linux : Guerre ou complément ?	N°42	Développez vos pilotes de périphérique	N°69	La réponse de Sun à Linux ? SOLARIS 10
N°13	Cryptage : la clé de la sécurité	N°43	Administrez facilement votre réseau SNMP	N°70	Découvrez et comprenez la technologie GRID
N°14	XFree 4.0 : le futur à notre portée	N°44	Comprenez NetBios pour Maîtriser l'interopérabilité windows GNU Linux	N°71	Présentation et installation du Hurd
N°15	Passez à la vitesse supérieure	N°45	Cohabitation : UnDNS Bind dans un reseau Windows 2000	N°72	Services Web... C/C++ et gSOAP
N°16	OpenSources : Est-ce suffisant?	N°46	Debian : Utilisez Samba avec le support ACL	N°73	Compression théorie algorithmes et programmation
N°17	Linux : Système embarqué	N°47	GNUstep : le petit frère de Mac OS X ?	N°74	VFS : Système de fichiers virtuel
N°18	Spécial interview : l'avenir de Linux	N°48	Caudium, votre prochain serveur Web !	N°75	Tuning de code
N°19	Dossier spécial : Postgre SQL 7.0	N°49	Après MySQL & PostgreSQL SAP DB : La base de données libre & puisante	N°76	Algorithmes évolutionnistes
N°21	Le protocole Internet du 21e siècle : IPv6	N°50	Créez un album Photo avec PHP ...et sans MySQL	N°77	Systèmes de fichiers chiffrés
N°22	Le multi-hreading : Une manière moderne de programmer le Multitâche	N°51	Boostez votre site Web avec XML grâce à XSLT, CSS & XPath	N°78	Bluetooth, Spécifications, protocoles et configuration
N°23	Débugger sous Linux	N°52	Linux Temps réel où en est-on aujourd'hui !	N°79	Sécurisation du Noyau avec PAX
N°24	Palm et Linux	N°53	Linux sur PDA : Linux dans votre poche !		
N°25	Kernel 2.4.0	N°54	Maîtrisez LVM		
N°26	<Dossier> XML </Dossier>	N°55	Intelligence Artificielle : Principes & programmation de jeux de stratégie classique		
N°27	Les systèmes de fichiers journalisés	N°56	Développez vos applications Mozilla avec XPFE & XPCOM		
N°28	Scripting : la force d'Unix	N°57	Maîtrisez la gestion... Slots & Signaux ... des événements en C++		
N°29	L.F.S. Linux From Scratch	N°58	Djbdns enfin une alternative viable à BIND !		
N°30	Le chiffrement des données	N°59	Zopix, Créez un CD "Live" Zope en 10 minutes !		
N°31	VPN et tunneling	N°60	JBoss serveur d'applications J2EE OpenSource		
N°32	Changez de coquille	N°61	Découvrez MySQL 5 et les procédures stockées		
N°34	XSL - FO : TeX Killer ?	N°62	Créez votre OS, principe et implémentation		
N°35	QoS et iproute : optimisation et contrôle du trafic IP				

NUMÉROS LINUX MAGAZINE ÉPUISÉS

N°01, N°02, N°03, N°04, N°05, N°20, N°33.

BON DE COMMANDE POWER PACKS À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

*LINUX MAGAZINE - BP 20142 - 67603 SELESTAT CEDEX

	OUI, je désire acquérir un POWER PACK X5	1er 1PP* X5	2ème 2PP* X5	3ème 3PP* X5
Cochez ici ▲ POWER PACKS X5	1, Linux Magazine N°			
	2, Linux Magazine N°			
	3, Linux Magazine N°			
	4, Linux Magazine N°			
	5, Linux Magazine N°			
	Total par série de POWER PACKS X5 :	15 €	30 €	45 €
	OUI, je désire acquérir un POWER PACK X10	1er 1PP* X10	2ème 2PP* X10	3ème 3PP* X10
Cochez ici ▲ POWER PACKS X10	1, Linux Magazine N°			
	2, Linux Magazine N°			
	3, Linux Magazine N°			
	4, Linux Magazine N°			
	5, Linux Magazine N°			
	6, Linux Magazine N°			
	7, Linux Magazine N°			
	8, Linux Magazine N°			
	9, Linux Magazine N°			
	10, Linux Magazine N°			
	Total par série de POWER PACKS X10 :	25 €	50 €	75 €
	Les Hors Séries et numéros spéciaux sont exclus des POWER PACKS. Montant TOTAL 15€ + 3,81€ de frais de port. Le TOTAL s'élève à 18,81€ pour l'achat d'un POWER Pack x5.		TOTAL :	
			Frais de port :	+3,81€
	SEULEMENT EN FRANCE MÉTROPOLITAINE !		TOTAL :	

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

On vous demandera ensuite votre nom pour le faire figurer dans les rapports et les discussions. N'oubliez pas, la convention en anglais veut que la forme soit le prénom, puis le nom. Vient ensuite l'adresse mail à utiliser pour vous joindre.

Enfin, dernière étape, il vous faudra renseigner reportbug sur le fait que vous disposez ou non d'un MTA configuré sur votre machine. Il peut s'agir d'un vrai MTA (Exim, Postfix, Sendmail, etc.), mais aussi d'un MTA léger comme ssmtp ne faisant que relayer à un serveur de mails, mais utilisable, comme Sendmail, en ligne de commande. Si vous répondez négativement, reportbug se chargera d'utiliser le serveur de mail lui-même pour envoyer vos rapports.

Toutes ces informations sont stockées dans un fichier ~/.reportbugrc que vous pourrez consulter ou éditer par la suite. Vous pouvez également le supprimer pour que le lancement de reportbug provoque une nouvelle configuration.

Comme le message le précise, il nous faut renseigner le paquet concerné :

```
> bc
*** Welcome to reportbug.  Use ? for help at prompts. ***
Detected character set: ISO-8859-15
Please change your locale if this is incorrect.

Using 'Denis Bodor <lefinnois@lefinnois.net>' as your from
address.
Getting status for bc...
Verifying package integrity...
Checking for newer versions at packages.debian.org...
```

reportbug commence par analyser le système et le paquet installé. Il détermine ainsi le jeu de caractères utilisé, votre adresse mail, l'intégrité du paquet et s'il existe un paquet plus récent que le vôtre. Ce dernier point est important pour le BTS, il serait stupide de rapporter un bogue déjà corrigé par une version plus récente déjà disponible.

L'étape suivante est la consultation des rapports de bogues déjà présents dans le BTS :

```
Querying Debian BTS for reports on bc (source)...
13 bug reports found:

Outstanding bugs -- Normal bugs; Patch Available (1 bug)
   1) #374818: dc: maintscript prerm arguments
Outstanding bugs -- Normal bugs; Unclassified (2 bugs)
   2) #357728: bc: Segfaults
   3) #388487: bc: scale strangeness
Outstanding bugs -- Minor bugs; Patch Available (3 bugs)
   4) #296232: 'dc directory' should print an error
   5) #301921: 'man bc' typos: "environent", "specifing"
   6) #389345: dc: man page formatting bug
Outstanding bugs -- Wishlist items; Unclassified (1 bug)
   7) #287210: bc: Interrupt should reset the input line
Forwarded bugs -- Normal bugs (2 bugs)
   8) #221781: dc: precision >= 1000000000 does not work
   9) #245899: bc: BC_LINE_LENGTH=0 or similar for ``do never spl
Forwarded bugs -- Wishlist items (2 bugs)
  10) #94664: bc: please add automatic scale
  11) #263071: bc: please support ibase > 16
Resolved bugs -- Serious policy violations (1 bug)
  12) #326385 [FIXED bc 1.06-19]: please rebuild with libreadline
Resolved bugs -- Important bugs (1 bug)
  13) #320605 [FIXED bc 1.06-20]: Conformance with menu policy
(1-13/13) Is the bug you found listed above [y|N|m|r|q|s|f|?]?
```

reportbug consulte le serveur BTS et affiche les résultats. Il est IMPÉRATIF de bien lire ces rapports. Dénoncer un bogue alors qu'il est déjà dans le BTS est une perte de temps pour vous, mais surtout pour les développeurs Debian. En fin de liste, reportbug attend vos ordres. Vous pouvez entrer un numéro correspondant à la ligne qui vous intéresse pour avoir plus d'informations sur un bogue. Si vous avez le moindre doute sur le fait que le résumé affiché puisse être en rapport avec votre bogue, consultez le détail ! Il n'est pas rare que des bogues présents dans le BTS soient fusionnés, car ils sont les facettes d'un même problème. A vous d'estimer correctement cela pour ne pas faire perdre du temps à tout le monde.

CONSULTATION ET AJOUT D'INFORMATIONS SUR UN BOGUE

En entrant un numéro de ligne et en validant, vous affichez le rapport de bogue complet. Celui-ci résume les manipulations et les résultats obtenus par l'utilisateur ayant fait le rapport, mais également des indications sur son système. C'est le *pager* par défaut qui sera utilisé pour afficher le rapport (more ou less). Une fois que vous quittez le pager, les ordres utilisables sont différents. Vous êtes dans l'interface de gestion d'un bogue rapporté. Une validation (ou l'ordre N) permet de voir le message suivant concernant ce bogue. Là encore, il est IMPÉRATIF de lire l'ensemble des discussions. Cela vous donnera davantage d'indications sur les conditions de l'apparition du bogue et les avis de chacun des intervenants. La répétition est une perte de temps.

L'ordre b permet de lancer le navigateur par défaut sur la page Web sans sortir de reportbug. Vous aurez alors l'ensemble des messages sur une seule page bien plus facile à lire.

NOTE

Quelle que soit l'invite présentée par reportbug, vous pouvez à tout moment utiliser ? pour obtenir un bref résumé des ordres ou commandes utilisables. Dans l'invite, figure toujours en majuscule l'ordre par défaut qui sera donné par simple validation.

L'ordre x vous permet d'ajouter un message à la discussion afin de fournir des informations supplémentaires. Inutile de préciser, je pense, que les messages « ça marche pas chez moi non plus », « chez moi ça marche » ou « sur Ubuntu|Mandriva|FC|RHEL, ça marche », dignes des pires forums, ne sont même pas à envisager. Si vous n'avez rien de concret à ajouter, n'ajoutez rien. Vous serez invité à préciser un sujet pour votre message, celui-ci doit être cours et décrire le plus précisément possible l'information que vous ajoutez.

Si le rapport initial concerne une autre version du paquet que celle que vous utilisez, reportbug vous demandera automatiquement si vous avez constaté le même problème. Quelle que soit la réponse, l'éditeur de texte par défaut sera lancé afin de vous permettre de détailler vos manipulations, essais et constatations. En quittant celui-ci, reportbug reprendra la main et un certain nombre de possibilités s'offriront à vous. L'ordre par défaut (Y) provoque l'envoi du message dans le BTS (ou à vous-même avec l'option -d).

```
Does this bug still exist in version 1.06-20 of this package?
 [y|N|q|?]? N
Spawning vim...
Report will be sent to Denis Bodor <lefinnois@lefinnois.net>
Submit this report on bc (e to edit) [Y|n|a|c|e|i|l|m|p|q|?]? Y
Sending message via /usr/sbin/sendmail...
Bug report submitted to: Denis Bodor <lefinnois@lefinnois.net>
```

Voici un exemple de ce qui sera envoyé par mail :

```
Package: bc
Followup-For: Bug #357728

Problem gone

-- System Information:
Debian Release: 4.0
  APT prefers testing
  APT policy: (900, 'testing'),
              (500, 'stable'),
                (80, 'unstable'),
                (1, 'experimental')
Architecture: i386 (i686)
Shell:  /bin/sh linked to /bin/bash
Kernel: Linux 2.6.15-1-k7
Locale: LANG=fr_FR@euro,
        LC_CTYPE=fr_FR@euro (charmap=ISO-8859-15)

Versions of packages bc depends on:
ii  libc6        2.3.6.ds1-8 GNU C Library: Shared libraries
ii  libncurses5  5.5-5       Shared libraries for terminal hand
ii  libreadline5 5.2-1       GNU readline and history libraries

bc recommends no packages.

-- no debconf information
```

On retrouve le numéro du bogue, le message (Problem gone) qui est ici un bel exemple de ce qu'il ne faut pas faire, la version de la distribution (4.0 qui correspond ici à la Testing/Etch), vos préférences APT (voir article sur le sujet), l'architecture, le *shell* utilisé, la version du noyau, les locales et les versions des dépendances.

AJOUT D'UN RAPPORT

Si vous avez bien tout consulté et que votre bogue n'est pas déjà référencé, vous pouvez ajouter une entrée dans le BTS. Si vous êtes dans le détail d'un bogue, utiliser o pour revenir à la liste principale. L'ajout d'une entrée se fait via n. reportbug consultera les versions installées des dépendances du paquet et vous serez invité à décrire brièvement le problème :

```
Maintainer for bc is 'John Hasler <jhasler@debian.org>'.
Looking up dependencies of bc...

Please briefly describe your problem (you can elaborate in a
moment; an empty response will stop reportbug). This should be a
concise summary of what is wrong with the package, for example,
"fails to send email" or "does not start with -q option
specified."
> Add color
```

Viendra ensuite le moment de préciser le niveau de gravité du bogue en spécifiant le numéro dans la liste proposée. Ici, notre souhait de voir de la couleur dans bc est clairement à qualifier de wishlist. L'éditeur par défaut est alors lancé et vous pourrez détailler la demande ou les constatations.

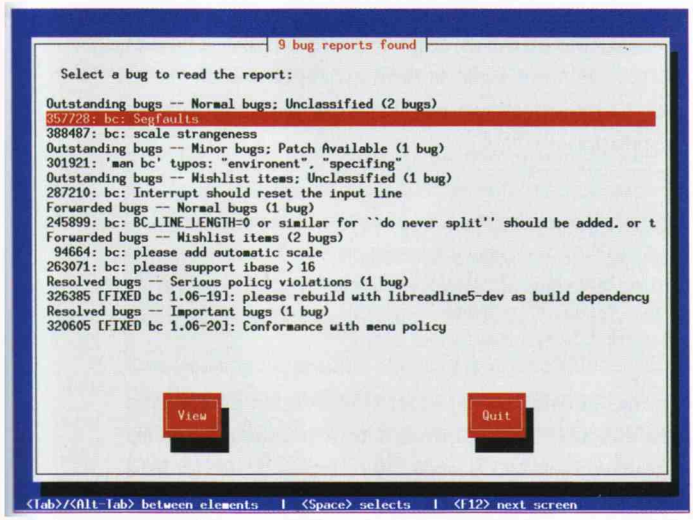
S'il s'agit d'un bogue, essayez de préciser le plus fidèlement possible les conditions dans lesquelles il apparaît sans pour autant saturer le message d'informations ennuyeuses. Expliquez également les essais que vous avez faits et éventuellement l'analyse complète du problème (utilisation de gdb, strace, etc.). Pensez aux personnes qui liront le message et tenteront de corriger le problème. N'oubliez pas non plus les informations automatiquement ajoutées. Inutile, par exemple, de préciser la version de la libC ou encore de votre noyau.

Tout comme avec l'ajout d'informations sur un rapport existant, dès que vous quittez l'éditeur, la solution par défaut est l'envoi du rapport. Consultez cependant les autres options proposées comme l'ajout d'un fichier joint ou d'un texte complémentaire, la sauvegarde du rapport au lieu de l'envoi, etc.

Une fois l'envoi effectué, attendez confirmation de l'intégration du rapport dans le BTS. Normalement, dans le délai maximum d'une heure, vous devez recevoir un message précisant le numéro d'identification attribué à votre rapport. Vous pourrez alors suivre l'évolution des choses via la page Web ou reportbug.

AUTRES MÉTHODES ET OUTILS

Si vous rencontrez des problèmes avec un paquet et souhaitez simplement savoir si les développeurs Debian sont au courant du problème, il n'est pas nécessaire d'utiliser un navigateur Web. L'utilitaire `querybts`, livré avec le paquet `reportbug` permet de consulter les entrées dans le BTS pour le paquet spécifié en argument. L'utilitaire est écrit en Python et utilise le module `python-newt` offrant ainsi une interface à menus conviviale en mode console.



`popbugs` est également un outil très intéressant pour la simple consultation. Il est livré avec le paquet `debian-goodies`. Celui-ci permet très simplement (exécuté sans aucun argument) d'ouvrir une page Web, spécialement générée avec votre navigateur par défaut, contenant la liste des bogues RC (*Release Critical*). `popbugs` filtre automatiquement les résultats en fonction des paquets que vous avez installés sur votre système (via les journaux de `popularity-contest`, voir article sur le nettoyage du système). Notez que vous pouvez spécifier un fichier en sortie avec `-o` plutôt que l'ouverture du fichier temporaire dans le navigateur.

`rc-alert`, fourni par le paquet `devscripts`, permet de faire la même chose, mais en mode console. Ainsi, la simple invocation de `rc-alert` affichera la liste des bogues critiques pour la publication de la distribution dans l'ordre des paquets installés :

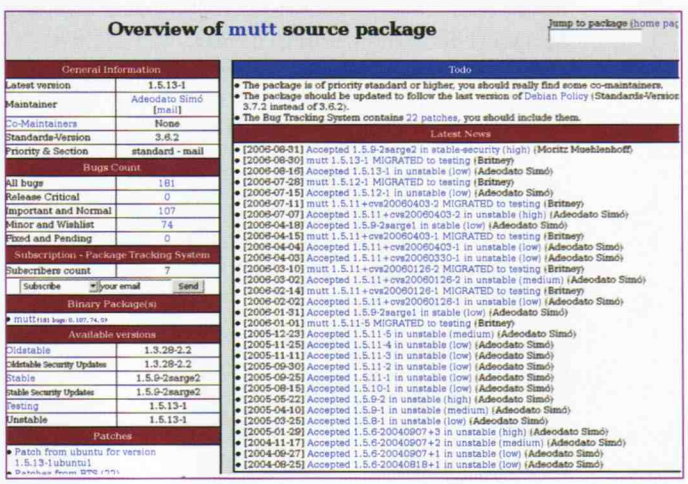
```
Package: wmaker
Bug:     397412
Title:   wmaker: Wmaker crash on creating desktop
Flags:   [   MR   ] (moreinfo [needed], unreproducible)

Package: wodim
Bug:     401791
Title:   wodim upgrade breaks xcdroast
Flags:   [        ] (none)
```

Si un bogue particulier vous intéresse ou qu'il semble important pour vous, vous pouvez suivre par mail les discussions s'y rapportant. Il vous suffit pour cela d'envoyer un message à **XXXXXX-subscribe@bugs.debian.org** où XXXXXX est le numéro identifiant le bogue. Ceci évite d'utiliser un quelconque outil et vous permettra, par exemple, de suivre l'évolution du bogue depuis un système Webmail ou mobile.

Vous pouvez également choisir d'améliorer le système APT en demandant l'affichage des bogues de niveau critical, serious et grave lors de l'utilisation d'`apt-get`. Ainsi, lors de l'installation ou de la mise à jour d'un paquet, vous serez prévenu d'un éventuel problème :

```
% sudo apt-get install wodim
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
[...]
Lecture des champs des paquets... Fait
Lecture de l'état des paquets... Fait
Récupération des rapports de bogue... Fait
Analyse des informations Trouvé/Corrigé... Fait
Bogues de gravité serious sur wodim
    (5:1.0-1 -> 5:1.0-1) <done>
 #401308 - wodim does not write to DVD-R
           (Corrigé: 9:1.1.0-1 1.1.0-1)
 #399995 - wodim: Error trying to dist-upgrade
           (Corrigé: cdrkit/9:1.1.0-1)
Résumé:
 wodim (2 bogues)
Êtes-vous certain de vouloir installer/mettre à
  jour les paquets ci-dessus ? [Y/n/?/...]
```

Enfin, pour conclure, vous pouvez également suivre l'évolution complète et consulter l'historique d'un paquet via le suivi de qualité Debian. Il suffit pour cela de pointer votre navigateur sur **http://packages.qa.debian.org** et de spécifier un nom de paquet. Vous aurez alors sous les yeux toute l'histoire de toutes les versions du paquet et un grand nombre d'informations s'y rattachant.



Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

DÉTOURNEMENT DE FICHIERS : AJOUTEZ VOTRE GRAIN DE SEL DANS LE SYSTÈME

Le fait d'utiliser une distribution basée sur des paquets binaires peut parfois être problématique. La rigidité induite interdit toute modification manuelle. Sauf si on utilise une bonne distribution et qu'on connaît les bonnes techniques.

Pourquoi donc vouloir changer manuellement les fichiers installés par un paquet ? La première réponse pourrait être celle des développeurs Debian eux-mêmes : Exim, Qmail ou Postfix, par exemple, sont des MTA pouvant remplacer le classique Sendmail. Cependant, un certain nombre d'outils et de scripts utilisent encore `/usr/sbin/sendmail` ou `/usr/lib/sendmail`. Pour contourner le problème, la solution la plus classique (et celle utilisée en réalité dans le cas des MTA), consiste à spécifier les paquets comme étant en conflit. Ainsi l'installation du paquet `ssmtp` provoquera la désinstallation de `exim4`, `exim4-base`, etc.

Une autre solution est d'utiliser un « détournement » (*diversion* en anglais). Ceci permet d'obliger `dpkg` à installer un fichier à un autre emplacement et sous un autre nom que celui prévu à l'origine par le responsable du paquet. Le détournement est contrôlé via l'outil `dpkg-divert` et peut être utile pour deux types de manipulations.

DÉTOURNEMENTS DEBIAN

Premièrement, nous avons celles des développeurs Debian responsables de paquets. Nous avons, par exemple, le cas de `xinetd` et `openbsd-inetd` (qui remplace depuis peu `netkit-inetd`). Les deux paquets peuvent être installés de concert, mais un seul super-serveur peut être raisonnablement utilisé. La solution consiste donc à utiliser un détournement du fichier `/etc/init.d/inetd`. Lorsque vous installez `xinetd`, ce fichier est détourné. L'original devient `/etc/init.d/inetd.real` et un nouveau fichier `/etc/init.d/inetd` est installé contenant ceci :

```
#!/bin/sh
# /etc/init.d/inetd has been diverted
# by the xinetd package. The inetd service
# is provided by xinetd, which means inetd
# doesn't need to be run.
#
# See /etc/init.d/xinetd,
# or /etc/init.d/inetd.real.

exit 0
```

Il ne s'agit pas du fichier originellement livré par le paquet `openbsd-inetd` qui, pour sa part, est renommé en `inetd.real`. Cette opération est d'ailleurs indiquée lors de l'installation d'`xinetd` :

```
[...]
Paramétrage de xinetd (2.3.14-1) ...
Stopping internet superserver: xinetd.
Adding `diversion of /etc/init.d/inetd
  to /etc/init.d/inetd.real by xinetd'
Starting internet superserver: xinetd.
```

Si nous désinstallons le paquet `xinetd`, l'opération inverse sera effectuée. Vous pouvez, à tout moment, lister les détournements actifs avec `dpkg-divert --list`. La liste des détournements actifs est stockée dans `/var/lib/dpkg/diversions`.

DÉTOURNEMENTS PAR LE SYSADMIN

L'autre type de manipulation concerne les administrateurs système. Vous avez sans doute déjà remarqué que la mise à jour d'un paquet entraîne parfois un avertissement concernant un fichier de configuration modifié manuellement et dont le sort est laissé à la discrétion du super-utilisateur.

C'est le système de gestion des fichiers de configuration (`conffiles`) qui permet cela. Prenons, par exemple, le fichier `/etc/crontab`. Si vous le modifiez manuellement, puisqu'une mise à jour du paquet le fait également, celui-ci ne sera pas simplement écrasé. Un message d'avertissement apparaîtra durant la phase d'installation vous proposant plusieurs solutions (conservation, écrasement, abandon de l'installation et examen du fichier) :

```
Fichier de configuration "/etc/crontab"
 ==> Modifié (par vous ou par un script) depuis l'installation.
 ==> Le distributeur du paquet a fourni une version mise à jour.
   Que voulez-vous faire? Vos options sont les suivantes:
    Y ou I : installer la version du responsable du paquet
    N ou O : garder votre version actuellement installée
      D    : afficher les différences entre les versions
      Z    : suspendre ce processus pour examiner la situation
```

```
 L'action par défaut garde votre version actuelle.
*** crontab (Y/I/N/O/D/Z) [défaut=N] ?
```

Ceci arrive car /etc/crontab est listé dans le conffiles du paquet binaire. Si ce n'était pas le cas, ce fichier serait tout simplement remplacé et vous perdriez vos ajouts/modifications.

Si l'avertissement vous dérange et que vous estimez savoir ce que vous faites concernant la configuration, le détournement est une excellente solution. Pour cela, il suffit d'utiliser dpkg-divert ainsi (en root) :

```
# dpkg-divert --add --rename --divert /etc/crontab.real /etc/crontab
Adding `local diversion of /etc/crontab to /etc/crontab.real'
```

Nous demandons plusieurs choses à dpkg-divert et, en premier lieu, le fait d'ajouter un détournement avec --add. Dans un second temps, nous voulons que, pour le système de gestion de paquets, tout ce qui concerne le fichier /etc/crontab soit détourné vers /etc/crontab.real. L'option --rename nous permet de profiter de l'opération pour directement renommer crontab en crontab.real.

Il ne nous reste, ensuite, plus qu'à copier crontab.real en crontab pour le modifier par la suite ou créer notre propre version du fichier en partant de zéro. Lors d'une mise à jour du paquet cron, si le fichier de configuration de crond doit être modifié (en raison d'un changement de syntaxe run-parts), c'est crontab.real qui subira les changements.

Bien entendu, ce type d'opération implique une grande responsabilité de l'administrateur système, puisqu'il devra opérer lui-même les changements après avoir comparé la version détournée et sa version du fichier.

Pour supprimer un détournement, il suffit d'utiliser la même commande en remplaçant --add par --remove. Notez cependant qu'un certain nombre de sécurités sont mises en place. Ainsi, vous ne pouvez pas provoquer le changement de nom de fichier inverse si votre version du fichier existe encore. dpkg-divert refusera simplement d'écraser bêtement votre version.

Pour simplifier les choses, la syntaxe de dpkg-divert possède des options par défaut. En utilisant simplement dpkg-divert /etc/crontab, le détournement installé se fera vers /etc/crontab.distrib et le fichier crontab ne sera pas renommé.

NOTE

Le comportement du système APT vis-à-vis des détournements dépendra de sa configuration en ce qui concerne les conffiles. Il est parfaitement possible dans notre exemple que vous obteniez automatiquement un fichier crontab.distrib.dpkg-new alors que, sans le détournement, APT vous aurait interrogé.

Le détournement de fichiers par l'administrateur système peut également concerner des fichiers binaires. Un exemple récurrent concerne gcc. L'idée est de mettre en place un fichier /usr/bin/gcc qui soit un script, un *wrapper* contenant :

```
#!/bin/sh
/usr/bin/gcc.distrib -Wall -O2 "$@"
```

Il suffit d'utiliser dpkg-divert pour ajouter le détournement, puis nommer votre script /usr/bin/gcc. Ainsi, chaque utilisation du compilateur provoquera l'utilisation des options spécifiées et, surtout, une mise à jour changera gcc.distrib. Le système peut évoluer et votre personnalisation reste fixe.

D'autres options existent pour dpkg-divert, comme --package permettant de préciser le nom d'un paquet étant exempté de détournement. Ceci permet de choisir quel paquet aura une influence sur le fichier concerné. Enfin, --local, qui fait partie des options par défaut si dpkg-divert est utilisé sans autres arguments que --add et un nom de fichier, permet de préciser que le détournement concerne toutes les versions du paquet et non pas simplement celle en cours.

L'intérêt de dpkg-divert n'est sans doute pas évident au premier abord, mais il suffit de réfléchir un peu : binaires, doubles configurations, données dans /usr/share, fichiers de configuration à centraliser (avec liens symboliques)... les applications ne manquent pas. Pour résumer, dpkg-divert est un excellent moyen de garder son système à la fois à jour et personnalisé.

Un bel exemple d'utilisation est la personnalisation d'applications comme Iceweasel, le Firefox modifié par Debian. Celui-ci ne peut s'appeler « Firefox », parce que ses modifications n'ont pas été validées par le projet Mozilla. On peut, en effet, changer à souhait les éléments graphiques individuellement ou tout autre élément se trouvant dans /usr/share/nom_de_l_application. Bien entendu, à vous de trouver le juste milieu. Si vous tenez vraiment à personnaliser une application, autant reprendre le paquet source et vous en servir pour créer votre propre version. Cela sera sans doute plus facile à maintenir.

Frédéric Le Roy :: frederic.le.roy@free.fr

INSTALLATION DE DEBIAN VIA UN BOOT RÉSEAU

Dans certains cas, il est impossible d'installer Debian à partir d'un CD-Rom (lecteur CD absent ou en panne). Dans d'autres, on peut vouloir installer Debian sur plusieurs machines à la fois. Pour répondre à cette attente, il est possible de configurer un système Debian afin qu'il permette aux ordinateurs du réseau local de lancer une installation de Debian via un simple boot sur le réseau.

OBJECTIF

Dans cet article, nous allons détailler l'installation et la configuration des applications nécessaires à la mise à disposition d'un serveur d'installation réseau.

Nous disposons d'un réseau local avec un système Debian sur lequel nous allons installer le serveur d'installation réseau. Nous appellerons cet ordinateur le serveur pour la suite de l'article. Le réseau local comporte également plusieurs ordinateurs qui peuvent nécessiter une réinstallation. Nous appellerons ces ordinateurs les clients.

PRINCIPE

Pour booter sur un réseau, il faut utiliser le protocole PXE (Preboot eXecution Environment). Toutes les cartes réseau ne permettent pas ceci, mais la majorité des cartes actuelles le permettent.

Les cartes compatibles avec le protocole PXE contiennent une PROM qui va permettre à la machine de booter sur le réseau. Cette PROM va rechercher sur le réseau un serveur DHCP, qui va pouvoir lui fournir une adresse IP, puis un noyau. Ce noyau va être récupéré via le protocole TFTP.

Il faut donc au minimum deux machines :

- Une machine cliente qui peut booter sur le réseau grâce à sa carte réseau.
- Un serveur qui va fournir l'adresse IP et le noyau au client.

Si le poste client ne peut pas booter sur le réseau grâce à sa carte réseau, il est possible de créer une disquette afin de booter sur le réseau. Ce point sera abordé en fin d'article.

PRÉPARATION DU SERVEUR

Avant de commencer l'installation des services, il faut connaître certaines informations :

- L'adresse IP du serveur
- La version de l'installeur Debian qui sera mise à disposition
- L'architecture des machines clientes

1ÈRE ÉTAPE : LE SERVEUR TFTP

TFTP (Trivial File Transfert Protocol ou en français Protocole simplifié de transfert de fichiers) est un protocole qui permet de transférer simplement des fichiers. C'est un protocole qui ne permet ni de lister les fichiers disponibles, ni de gérer les authentifications. Il faut donc connaître le nom des fichiers que l'on veut récupérer via ce protocole.

! ATTENTION

il est dangereux d'autoriser l'accès au serveur TFTP aux ordinateurs étrangers au réseau local !! Pensez à vérifier que votre firewall bloque le port 69 en udp pour le réseau Internet ou les réseaux sur lesquels vous ne désirez pas autoriser une installation via le réseau.

Il existe plusieurs serveurs TFTP sous Debian. Nous allons utiliser **atftpd**.

Tout d'abord, installons le serveur **atftpd** :

```
# apt-get install atftpd
```

Nous allons choisir de lancer **atftpd** en tant que serveur indépendant et vérifier que le dossier **/tftpboot** (si c'est celui que vous avez choisi lors de la configuration de **atftpd**) existe.

Plus tard, si nous désirerons reconfigurer **atftpd**, nous pourrons lancer la commande suivante :

```
# dpkg-reconfigure atftpd
```

Si le dossier que nous avons spécifié lors de l'installation n'existe pas, nous devons le créer :

```
# mkdir /tftpboot
```

Pour prendre les changements en compte, il faut redémarrer le serveur :

```
# /etc/init.d/atftpd restart
```

2ÈME ÉTAPE : INSTALLATION DU SERVEUR PXE

PXE (Pre-boot eXecution Environment) permet à un poste client de démarrer depuis le réseau un système d'exploitation disponible sur un serveur réseau. Le fichier qui permet de démarrer le système d'exploitation (appelé noyau) est dans la majorité des cas trop gros pour être récupéré par la carte réseau. Dans ce cas, on peut utiliser un bootloader

(PXELinux) qui va télécharger le noyau du système d'exploitation sur un serveur TFTP, puis démarrer dessus.

ATTENTION

le serveur PXE utilise le port 4011, veillez à ce qu'il ne soit pas bloqué pour le réseau local par votre firewall.

Installons le serveur PXE et **syslinux**. **Syslinux** est une suite de **botloader** (IsoLinux, PXELinux, ExtLinux, etc.). Dans notre cas, c'est PXELinux qui nous intéresse : il va nous permettre de booter sur le réseau et d'aller télécharger le noyau de l'installeur Debian.

```
# apt-get install pxe syslinux
```

Copions le bootloader PXELinux dans le dossier de atftpd (/tftpboot dans notre cas) :

```
# cp /usr/lib/syslinux/pxelinux.0 /tftpboot/
```

Editons le fichier **/etc/pxe.conf** et vérifions que l'interface réseau indiquée dans la ligne suivante correspond à l'interface réseau du réseau local :

```
interface=eth0
```

Vérifions que l'adresse réseau correspond à l'adresse du serveur (192.168.0.1 dans notre cas) :

```
default_address=192.168.0.1
```

Vérifions aussi que le dossier utilisé par **atftpd** est bien renseigné :

```
tftpdbase=/tftpboot
```

Maintenant, nous allons récupérer l'image de l'installeur Debian à l'une des adresses suivantes (en fonction de votre version de Debian). Nous allons la télécharger dans le dossier /**tmp** :

Architecture	Version	Lien
i386	stable	http://ftp.debian.org/debian/dists/stable/main/installer-i386/current/images/netboot/netboot.tar.gz
i386	testing	http://ftp.debian.org/debian/dists/testing/main/installer-i386/current/images/netboot/netboot.tar.gz
i386	unstable	http://ftp.debian.org/debian/dists/unstable/main/installer-i386/current/images/netboot/netboot.tar.gz
amd64	stable	http://ftp.debian.org/debian/dists/stable/main/installer-amd64/current/images/netboot/netboot.tar.gz
amd64	testing	http://ftp.debian.org/debian/dists/testing/main/installer-amd64/current/images/netboot/netboot.tar.gz
amd64	unstable	http://ftp.debian.org/debian/dists/unstable/main/installer-amd64/current/images/netboot/netboot.tar.gz

Exemple pour la version stable sur une architecture i386 :

```
# cd /tmp
# wget http://ftp.debian.org/debian/dists/stable/main/
installer-i386/current/images/netboot/netboot.tar.gz
```

Décompressons l'archive dans le dossier de **atftpd** (/tftpboot dans notre cas) :

```
# cd /tftpboot
# tar xvzf /tmp/netboot.tar.gz
```

Pour que les changements soient pris en compte, nous allons redémarrer le service PXE :

```
# /etc/init.d/pxe restart
```

3ÈME ÉTAPE : CONFIGURATION DU RÉSEAU

Afin de pouvoir installer un serveur DHCP, nous allons configurer l'interface réseau connectée au réseau local de la manière suivante :

- Carte réseau concernée : eth0
- Adresse IP de notre ordinateur : 192.168.0.1
- Masque de sous réseau : 255.255.255.0

Editons le fichier **/etc/network/interfaces** et modifions la configuration de l'interface eth0 :

```
auto eth0
iface eth0 inet static
        address 192.168.0.1
        netmask 255.255.255.0
        broadcast 192.168.0.255
```

Afin que les changements soient pris en compte, nous allons redémarrer le service réseau :

```
# /etc/init.d/networking restart
```

4ÈME ÉTAPE : INSTALLATIONS DU SERVEUR DHCP

DHCP (Dynamic Host Configuration Protocol) permet à un serveur d'attribuer à des postes clients des configurations réseau de manière transparente pour les utilisateurs des postes clients.

ATTENTION

le serveur DHCP utilise le port 67, veillez à ce qu'il ne soit pas bloqué pour le réseau local par votre firewall.

Il faut commencer par installer le serveur DHCP :

```
# apt-get install dhcpd
```

Editons le fichier /etc/default/dhcp et complétons la ligne suivante pour indiquer les interfaces réseau sur lesquelles le serveur peut attribuer des adresses IP (**eth0** dans notre cas) :

INTERFACES="eth0"

Maintenant, passons à la configuration du fichier principal du serveur DHCP. Editons le fichier **/etc/dhcpd.conf** et ajoutons-y les lignes suivantes. Si le serveur DHCP n'a jamais été utilisé sur le serveur, il est possible de

le vider avant de rajouter les lignes suivantes (pour plus de clarté).

```
# configuration TFTP
subnet 192.168.0.0 netmask 255.255.255.0 {
     range 192.168.0.10 192.168.0.19;
    server-name "server";
    filename "/tftpboot/pxelinux.0";
}
```

Ces lignes vont permettre d'attribuer des adresses IP entre 192.168.0.10 et 192.168.0.19 (nous pouvons donc avoir 10 machines au maximum avec cette configuration).

Redémarrons le service DHCP :

```
# /etc/init.d/dhcp restart
```

C'EST FINI POUR LE SERVEUR

Et voilà, la préparation du serveur est finie. Il ne reste plus qu'à tester le bon fonctionnement sur les postes clients.

PRÉPARATION D'UN CLIENT

Tout d'abord, nous devons vérifier si la carte réseau du client supporte le protocole PXE. Pour le savoir, il suffit d'aller regarder dans la section « Séquence de boot » de votre BIOS pour voir si l'ordinateur est capable de booter via une carte réseau.

PREMIER CAS : LE CLIENT PEUT BOOTER SUR LA CARTE RÉSEAU

Configurez votre BIOS de façon à ce que votre client boote via le réseau. Pour certains ordinateurs, le boot réseau est optionnel via l'appui sur une touche F12 au démarrage (un message indiquant cette possibilité est alors affiché). Pour les autres, il va falloir configurer le BIOS pour que l'ordinateur boote sur la carte réseau au démarrage.

SECOND CAS : LE CLIENT NE PEUT PAS BOOTER SUR LA CARTE RÉSEAU

Si le BIOS ne permet pas le boot sur la carte réseau, il est possible de booter grâce à une disquette. Pour cela, il est possible de récupérer des images sur le site http://rom-o-matic.net. Il suffit de spécifier le type de carte réseau que vous avez et le type d'image que vous désirez récupérer (« Floppy bootable ROM image » dans le cas d'une disquette).

Récupérez l'image qui correspond à votre carte réseau sur le site, puis mettez-la sur la disquette :

```
$ cat nomDeLImage.zdsk > /dev/fd0
```

Il ne vous reste qu'à rebooter l'ordinateur, en ayant pris soin de configurer le BIOS pour qu'il démarre sur le lecteur disquette.

LE BOOT SUR LE RÉSEAU

Maintenant que le poste client est prêt à booter sur le réseau, redémarrez-le. Vous devez voir l'installeur Debian s'afficher sur votre écran.

CONCLUSION

Nous avons vu dans cet article qu'il est possible de booter un ordinateur sur un noyau hébergé sur une autre machine. Dans notre cas, nous avons utilisé cette possibilité afin de booter sur l'installeur Debian. Il est toutefois possible d'utiliser cette fonctionnalité de manière différente : on peut par exemple réutiliser de vieilles machines en les transformant en terminaux (boot via PXE en terminal en utilisant la partition racine disponible sur un partage NFS). Le Hors Série n°18 sur le recyclage de vieilles machines illustre ce dernier cas.

RÉFÉRENCES

Spécification du protocole PXE
http://dev.brantleyonline.com/files/pxespec.pdf

PXE Linux
http://syslinux.zytor.com/pxe.php

TFTP sur Wikipedia
http://fr.wikipedia.org/wiki/TFTP

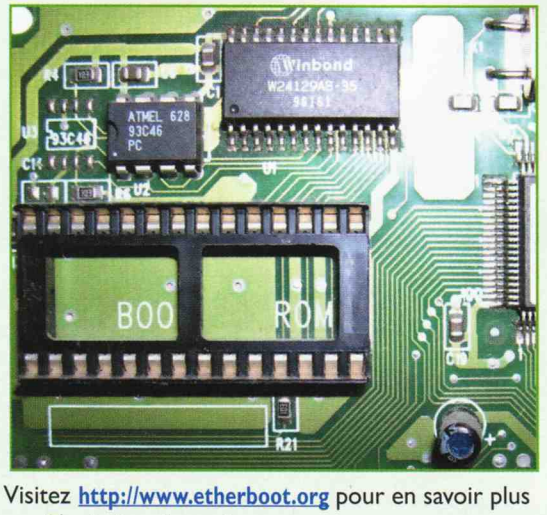
DHCP sur Wikipedia
http://fr.wikipedia.org/wiki/DHCP

Images pour booter sur le réseau
http://rom-o-matic.net

gPXE

Avant l'arrivée massive du standard PXE, les machines devant démarrer via le réseau utilisaient Etherboot. Le principe était similaire, à la différence qu'Etherboot était prévu pour fonctionner dans une ROM. Ce composant de mémoire morte appelé « BootROM » était inséré sur un support spécifique de l'adaptateur réseau.

Etherboot a fait l'objet d'un article dans GLMF 3 de février 1999. Aujourd'hui, Etherboot est dépassé et les développeurs du projet restructurent leur code afin de respecter le standard PXE. Le projet se nomme « gPXE » et les objectifs sont identiques à ceux d'Etherboot : développer un code pouvant être placé dans un BIOS Flash, en ROM/EPROM/EEPROM, sur une disquette ou un CD-Rom.



Visitez http://www.etherboot.org pour en savoir plus sur l'évolution du projet.

CENTRALISEZ LES MISES À JOUR

Si, comme moi, vous avez plus qu'une seule machine fonctionnant avec Debian GNU/Linux, sans doute, avez-vous remarqué un certain gâchis de ressource. Je parle, bien entendu, du fait de télécharger les paquets à installer ou mettre à jour sur chaque machine.

Il suffit d'installer un proxy, me direz-vous. Certes, c'est une solution efficace, mais la gestion du cache n'est pas spécifique au téléchargement de paquets. De plus, la configuration d'un serveur mandataire (en bon français) comme Squid n'est pas aussi simple que la solution que je vous propose maintenant : le paquet `apt-proxy`.

Comme son nom l'indique, ce serveur mandataire est spécifiquement dédié au système APT et particulièrement facile à mettre en œuvre. Nous prendrons en guise d'exemple une configuration à deux postes. Le premier, `raven`, est en fonctionnement 24h/24. Le second, `lithium`, est une machine de test habituellement dédiée aux expérimentations les plus folles. `apt-proxy` sera donc, naturellement, installé sur `raven` avec un simple `apt-get install`.

Dans la foulée, nous nous penchons sur le fichier de configuration `/etc/apt-proxy/apt-proxy-v2.conf`. On peut ici préciser quelques éléments importants :

`address=192.168.0.1` est l'adresse de l'interface sur laquelle on attend les requêtes.

`port=9999` est le port IP pour l'attente de connexion.

`min_refresh_delay=1h` est le délai minimum avant une tentative de mise à jour.

`complete_clientless_downloads = 1` permet de poursuivre les téléchargements, même si plus aucun client n'est connecté.

`cache_dir = /var/cache/apt-proxy` est le répertoire de cache où seront stockés les paquets et les listes.

`;http_proxy = host:port` permet de spécifier un serveur mandataire (oui, `apt-proxy` est un serveur proxy qui peut utiliser un autre serveur proxy générique). Remarquez que cette option est commentée et le restera.

`;bandwidth_limit = 100000` permet de limiter la bande passante pour ne pas nuire au fonctionnement du système où est installé `apt-proxy`. En phase de test, cette option est désactivée.

`cleanup_freq = 1d`, nous vérifions l'âge des éléments du cache, une fois par jour.

`max_age = 120d`, tout ce qui n'a pas été « accédé » depuis 120 jours est effacé.

`max_versions = 3` permet de définir le nombre maximum de versions d'un paquet à conserver dans le cache.

Suit alors une série de sections désignées entre crochets (`[]`). Il s'agit des *backends* à utiliser, vos sources de paquets. On distingue ainsi `[debian]` (branche principale), `[security]` (mise à jour de sécurité) et quelques autres dont `[ubuntu]` (sans commentaire). La directive `backends =` permet de lister, dans l'ordre de préférence, les serveurs à utiliser. Choisissez quelque chose de localement proche en premier. Vous pouvez également ajouter des sections en fonction de ce qui se trouve actuellement dans vos `sources.list`. En fin de configuration, n'oubliez pas de redémarrer le serveur avec `/etc/init.d/apt-proxy restart`.

NOTE

Remarquez qu'il faut préciser l'URL complète de chaque serveur. **http://ftp.fr.debian.org/**, par exemple, n'est pas suffisant. Il faut préciser **http://ftp.fr.debian.org/debian**, afin que le proxy puisse trouver le sous-répertoire `dists` et les différentes distributions. Ceci ne va pas sans poser de problème avec les dépôts ne respectant pas les conventions de nommage.

C'est tout. Il ne vous reste plus qu'à vous connecter sur `lithium` et à modifier votre `sources.list` en prenant en compte le serveur mandataire :

```
deb     http://raven:9999/debian stable main contrib non-free
deb-src http://raven:9999/debian stable main contrib non-free

deb     http://raven:9999/debian testing main contrib non-free
deb-src http://raven:9999/debian testing main contrib non-free

deb     http://raven:9999/debian unstable main contrib non-free
deb-src http://raven:9999/debian unstable main contrib non-free
```

Il suffit ensuite de mettre à jour tout cela avec un `apt-get update`.

`apt-proxy` est très intéressant pour un parc de machines Debian qu'il faut mettre à jour rapidement. Cependant, comme précisé en note, il est fort probable que vous rencontriez des problèmes avec les serveurs non officiels. Pour vous aider, utilisez les options de débogage du serveur `apt-proxy` et `apt-show-versions` avant toute manipulation définitive.

INSTALLER DEBIAN À PARTIR D'UN SYSTÈME EXISTANT

Il existe plusieurs manières d'installer un système Debian, que ce soit la manière classique à partir d'un CD-Rom, l'installation via le réseau ou encore l'installation via debootstrap. C'est cette dernière méthode que nous allons développer ici. L'utilitaire debootstrap permet d'installer une version de Debian dans un dossier quelconque du disque dur. Les usages sont multiples : installation d'une application ou d'un service dans un environnement chrooté, installation sur disque dur d'une version de Debian.

OBJECTIF DE LA MANIPULATION

SITUATION INITIALE

Nous disposons d'un ordinateur avec un système Debian installé. Le disque dur sur lequel est installé le système commence à montrer des signes de faiblesse. Il va donc falloir réinstaller le système sur un nouveau disque dur. Malheureusement pour nous, la machine ne dispose pas de lecteur de CD-Rom. De plus, le réseau local ne comprend aucune autre machine Debian, et il est donc impossible de prévoir un serveur d'installation via un boot réseau.

Voici les éléments importants à connaître de la machine à réinstaller :

- Processeur AMD Barton : architecture i386, type k7 ;
- Mémoire vive : 1 Go ;
- Disque dur principal : `/dev/hda` ;
- Nom de la machine : Ptit-Barton.

Nous avons à notre disposition un disque dur en bon état offrant une capacité de 80 Go. Ce disque dur va être installé dans la machine en tant que Maître Secondaire. Il sera donc reconnu en tant que `/dev/hdc` sous Debian. Ce disque dur a déjà été utilisé et contient des partitions d'un ancien système. Nous supposerons que les données importantes présentes sont déjà sauvegardées.

DÉTAILS DE LA PROCÉDURE

Tout d'abord, nous allons préparer le nouveau disque dur à recevoir le nouveau système. Ensuite, nous installerons le nouveau système : installation du système de base, configuration du minimum requis pour pouvoir démarrer le nouveau système, installation d'un noyau et pour finir installation et configuration d'un *bootloader* (Grub en l'occurrence).

NORMES DE LECTURE

Dans la suite de l'article, les commandes précédées d'un dièse **#** seront à exécuter en tant que super utilisateur (*root*). Les commandes précédées d'un dollar `$` seront à exécuter en tant qu'utilisateur normal.

PRÉPARATION DU NOUVEAU DISQUE DUR

STRUCTURE DU NOUVEAU DISQUE

Tout d'abord, nous allons préparer le nouveau disque dur à recevoir l'installation. Nous allons créer trois partitions sur le disque :

- La partition racine `/`. Nous utiliserons dans cet exemple le système de fichier ext2.
- La swap.
- Une partition étendue, que l'on va laisser vide. Elle servira suivant les besoins ultérieurs.

NOTE

Une installation classique d'un système Debian utilise plusieurs partitions pour le système. Généralement, on utilise des partitions spécifiques pour `/tmp`, `/usr`, `/var`, `/boot` et `/home`. Dans notre cas, nous créons simplement une partition racine afin de centrer l'article sur l'utilisation de **Debootstrap** et non sur la gestion des partitions d'un environnement GNU/Linux.

La machine sert de serveur web, FTP et subversion. Elle n'a pas de serveur X. Les données mises à disposition seront stockées sur la partition étendue (ce point ne sera pas abordé dans cet article). Une taille de 4 Go est donc suffisante pour la partition racine.

Auparavant, on conseillait d'avoir une swap d'une taille double à la taille de la mémoire vive. Ce calcul n'est plus d'actualité, mais nous allons tout de même créer une partition de swap de 2 Go afin de configurer dans le futur un système d'hibernation « *suspend to disk* ».

LES UTILITAIRES NÉCESSAIRES

Pour préparer le disque dur, nous utiliserons les outils suivants :

- **parted** : il va nous permettre de consulter la table des partitions et de la modifier.
- **mkfs.ext2** : il va nous permettre de créer le système de fichier ext2 sur une partition.

- **mkswap** : il va nous permettre de créer la partition de swap.

L'utilitaire parted n'est pas installé de base sous Debian. Voici la commande à exécuter pour installer Parted :

```
# apt-get install parted
```

Parted peut être utilisé en ligne de commande ou en interpréteur de commande. Pour l'utiliser en tant qu'interpréteur, il suffit de lancer la commande parted en lui donnant en paramètre le disque dur que nous allons manipuler. Le prompt de la commande parted est (parted) :

```
# parted /dev/hdc
GNU Parted 1.7.1
Using /dev/hdc
Welcome to GNU Parted! Type 'help' to view a list of
commands.
(parted)
```

AFFICHAGE DE L'ÉTAT INITIAL DU DISQUE DUR

La commande parted <périphérique> print permet d'afficher la table des partitions. Voyons donc comment est constitué notre disque dur :

```
# parted /dev/hdc print

Disk /dev/hdc: 80,0GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  Flags
 1      32,3kB  7000MB  7000MB  primary   ext3         boot
 2      7000MB  80,0GB  73,0GB  extended
 5      7000MB  7386MB  387MB   logical   linux-swap
 6      7386MB  80,0GB  72,6GB  logical   ext3
```

VIDONS LE DISQUE DUR

Pour supprimer une partition avec parted, il faut utiliser la commande rm et lui donner en paramètre le numéro de la partition à supprimer. La suppression de la partition étendue (ici la deuxième) implique la suppression des partitions logiques (dans notre cas, la cinquième et la sixième).

Supprimons les partitions :

```
(parted) rm 1
(parted) rm 2
```

Vérifions que toutes les partitions ont bien été supprimées :

```
(parted) print

Disk /dev/hdc: 80,0GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start  End  Size  Type  File system  Flags
```

CRÉATION DES NOUVELLES PARTITIONS

Maintenant que le disque est vide, nous allons pouvoir créer nos partitions. Nous allons les créer sans système de fichier. Une fois que les partitions seront définies, nous leur affecterons un système de fichier.

La commande qui permet de créer les partitions est : mkpart <type de partition> <position de début> <position de fin>. Les types de partition possibles sont primary (partition primaire), extended (partition étendue) et logical (partition logique).

Tout d'abord, créons la partition racine / et vérifions le résultat obtenu :

```
(parted) mkpart primary 0Go 4Go
(parted) print
Disk /dev/hdc: 80,0GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Number  Start   End     Size    Type     File system  Flags
 1      32,3kB  3997MB  3997MB  primary
```

Maintenant, créons la partition de swap :

```
(parted) mkpart primary  4Go 6Go
(parted) print
Disk /dev/hdc: 80,0GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Number  Start   End     Size    Type     File system  Flags
 1      32,3kB  3997MB  3997MB  primary
 2      3997MB  5996MB  1999MB  primary
```

Et finissons enfin par la partition étendue, que nous laisserons vide pour le moment :

```
(parted) mkpart extended 6Go 80Go
(parted) print
Disk /dev/hdc: 80,0GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Number  Start   End     Size    Type      File system  Flags
 1      32,3kB  3997MB  3997MB  primary
 2      3997MB  5996MB  1999MB  primary
 3      5996MB  80,0GB  74,0GB  extended               lba
```

Maintenant que les partitions sont prêtes, nous pouvons quitter l'utilitaire Parted :

```
(parted) quit
```

CRÉATION DES SYSTÈMES DE FICHIERS

Les partitions sont presque prêtes. Il ne leur manque plus qu'un système de fichier.

Pour formater une partition avec un système de fichier ext2, nous allons utiliser la commande mkfs.ext2. L'option -L de mkfs.ext2 permet de donner un label à la partition (il s'agit d'un paramètre facultatif).

```
# mkfs.ext2 -L / /dev/hdc1
Vérifions que le système de fichier ext2 a été prix en
compte :
# parted /dev/hdc print

Disk /dev/hdc: 80,0GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  Flags
 1      32,3kB  3997MB  3997MB  primary   ext2
 2      3997MB  5996MB  1999MB  primary
 3      5996MB  80,0GB  74,0GB  extended               lba
```

Pour créer une partition de swap, nous allons utiliser la commande `mkswap`. De la même manière que pour créer le système de fichier ext2, nous pouvons aussi affecter un label à la partition.

```
# mkswap -L swap /dev/hdc2
Vérifions que le système de fichier a été prix en compte :
# parted /dev/hdc print

Disk /dev/hdc: 80,0GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  Flags
 1      32,3kB  3997MB  3997MB  primary   ext2
 2      3997MB  5996MB  1999MB  primary   linux-swap
 3      5996MB  80,0GB  74,0GB  extended               lba
```

INSTALLATION DU NOUVEAU SYSTÈME

Maintenant que le disque est prêt, nous allons installer le nouveau système sur le nouveau disque.

MONTER LA NOUVELLE PARTITION RACINE

Tout d'abord, il faut créer un dossier pour monter la partition racine de notre nouveau système :

```
# mkdir /mnt/debinstall
```

Montons maintenons la nouvelle partition racine :

```
# mount -t ext2 /dev/hdc1 /mnt/debinstall/
```

Si vous avez choisi d'installer votre système sur plusieurs partitions, c'est à ce stade de la manipulation qu'il va falloir agir. Il faut créer les dossiers concernés et y monter les partitions associées.

Examinons l'exemple suivant : nous avons choisi de mettre `/usr` et `/tmp` dans des partitions distinctes. La partition `/usr` est `/dev/hdc5` et elle est au format ext2. La partition `/tmp` est `/dev/hdc6` et elle est au format ext2.

Créons les dossiers pour ces partitions :

```
# mkdir /mnt/debinstall/usr
# mkdir /mnt/debinstall/tmp
```

Maintenant, montons ces 2 partitions :

```
# mount -t ext2 /dev/hd5 /mnt/debinstall/usr
# mount -t ext2 /dev/hd6 /mnt/debinstall/tmp
```

INSTALLATION DES UTILITAIRES

Debootstrap est l'outil officiel de Debian pour installer un système de base. Il permet d'installer dans un dossier le minimum requis pour faire tourner une distribution Debian.

Afin de créer le nouveau système, nous allons devoir installer le paquet `debootstrap`. Debootstrap ne dépend que de `/bin/bash`. Toutefois, il utilise les utilitaires **wget** et **ar**. Il faut donc penser à les installer sur votre système.

```
# apt-get install debootstrap wget ar
```

Il est possible d'installer Debootstrap que des distributions autres que Debian. Un paquet au format `rpm` est disponible à cette adresse : http://people.debian.org/~blade/install/debootstrap/.

UTILISATION DE DEBOOTSTRAP

Pour utiliser Debootstrap, nous avons besoin de plusieurs informations :

- L'architecture de notre matériel : Intel x86 (i386), PowerPC (powerpc), Intel Itanium (ia64), AMD64 (amd64), etc.
- Le nom de la version de Debian que nous désirons installer (sarge, etch, sid).
- L'adresse du miroir où les paquets vont être téléchargés. Les adresses des différents miroirs sont disponibles à l'adresse suivante : http://www.debian.org/mirror/list.

Dans notre cas, nous avons une architecture i386. Nous désirons installer une Debian Etch et nous allons choisir un serveur français.

Lançons l'installation de notre nouvelle Debian dans le dossier `/mnt/debinstall` :

```
# /usr/sbin/debootstrap --arch i386 etch /mnt/debinstall
http://ftp.fr.debian.org/debian
```

L'installation doit se finir par la ligne suivante :

```
I: Base system installed successfully.
```

Vérifions ce qu'a donné l'installation :

```
# ls -l /mnt/debinstall/
bin boot dev etc home initrd lib lost+found media mnt opt
proc root sbin srv sys tmp usr var
```

Nous nous retrouvons donc avec une arborescence Debian classique.

CHANGEONS DE RACINE

Maintenant que le système est installé, nous allons nous *chrooter* dans ce système, c'est-à-dire indiquer au système que notre racine ne sera plus `/` mais `/mnt/debinstall` (qui sera alors vu comme `/`). Cette manipulation a pour objectif de permettre l'utilisation de commandes dans le système

chrooté sans que les fichiers ou paramètres du système initial soit pris en compte.

Afin de vérifier le bon fonctionnement de la commande `chroot`, créons un fichier de test dans le dossier `/mnt/debinstall` :

```
# touch /mnt/debinstall/flag-nouveau-system
```

Vérifions qu'il n'existe pas sur le système actuel :

```
# ls /flag.nouveau.systeme
ls: /flag.nouveau.systeme: Aucun fichier ou répertoire de ce type
```

Chrootons-nous :

```
# chroot /mnt/debinstall/ /bin/bash
```

Vérifions que notre fichier de test est maintenant à la racine :

```
# ls /flag.nouveau.systeme
/flag.nouveau.systeme
#
```

Ce petit exemple montre bien que nous avons changé de racine. Pour sortir du chroot, il suffit de taper `exit`.

DÉFINITION DES POINTS DE MONTAGE

Pour que notre système puisse démarrer, il faut déjà qu'il sache quelles partitions monter. Par défaut, Debootstrap crée un fichier `/etc/fstab` vide. Il va donc falloir le remplir, au minimum avec notre partition racine, notre swap et le dossier `/proc`.

Fichier `/etc/fstab` :

```
# file system   mount point   type    options     dump pass
/dev/hdc1       /             ext2    defaults    0    1
/dev/hdc2       none          swap    sw          0    0
proc            /proc         proc    defaults    0    0
```

Si vous avez choisi de répartir l'installation sur plusieurs partitions, complétez le fichier en conséquence. Si on reprend l'exemple précédent (pour rappel `/usr` dans `/dev/hdc5` en ext2 et `/tmp` dans `/dev/hd6` en ext2), il faut ajouter les lignes suivantes :

```
/dev/hdc5     /usr       ext2    defaults          0    0
/dev/hdc6     /usr       ext2    defaults          0    0
```

Maintenant, il faut monter le dossier `/proc`. Ce dossier est un pseudo système de fichier qui contient des informations sur le système et les processus.

```
# mount -t proc proc /proc
```

Vérifions si le dossier `/dev` contient les périphériques hdc* :

```
# ls /dev/hdc*
```

S'il ne contient pas les périphériques hdc*, nous allons créer `hdc` et `hdc1` (notre disque et la partition racine). Ils seront nécessaires pour la configuration du bootloader (Grub).

```
# mknod /dev/hdc b 22 0
# mknod /dev/hdc1 b 22 1
```

Si vous avez choisi de répartir l'installation sur plusieurs partitions, il faut créer également les périphériques des partitions supplémentaires. La commande pour créer les périphériques est `mknod /dev/hdc[n] b 22 [n]`. `[n]` représente le numéro de la partition. Dans le cas de l'exemple précédent, il faut lancer les deux commandes suivantes :

```
# mknod /dev/hdc5 b 22 5
# mknod /dev/hdc6 b 22 6
```

CONFIGURATION DES LOCALES ET DU CLAVIER

Afin de ne pas être embêté avec un clavier qwerty lors du démarrage sur le nouveau système, nous allons configurer les locales, puis la configuration du clavier :

```
# apt-get install locales
# dpkg-reconfigure locales
```

Choisissons les locales suivantes : `fr_FR ISO-8859-1`, `fr_FR.UTF-8 UTF-8` et `fr_FR@euro ISO-8859-15`

Maintenant, passons à la configuration du clavier :

```
# apt-get install console-data
# dpkg-reconfigure console-data
```

AFFECTER UN MOT DE PASSE À L'UTILISATEUR ROOT

Afin de pouvoir se connecter en tant que super utilisateur au démarrage, nous allons lui affecter un mot de passe :

```
# passwd
```

INSTALLER UN NOYAU

Pour que le nouveau système puisse démarrer, il faut installer un noyau. Le noyau que nous allons installer n'est pas forcément le même que celui qui tourne actuellement sur le système.

Dans notre cas, installer une image de noyau déjà toute prête peut poser problème lors de la génération des fichiers nécessaires au boot. En effet, la version de noyau en mémoire ne correspond pas forcément à une image de noyau via Apt.

Afin de pallier ce problème, nous allons installer les sources du noyau et le compiler nous même, en activant les options pour générer les fichiers nécessaires au boot. Nous allons utiliser la méthode Debian de compilation de noyau : elle va produire un paquet Debian, qu'il nous faudra installer.

Tout d'abord, il faut installer quelques paquets qui sont nécessaires à la compilation et à la génération d'un paquet Debian :

```
# apt-get install kernel-package fakeroot libncurses5-dev
libc6-dev libc-dev gcc make initramfs-tools yaird
```

Maintenant, installons les sources :

```
# apt-get install linux-source-2.6.18
```

Allons dans le dossier des sources et décompressons les sources :

```
# cd /usr/src
# tar xvjf linux-source-2.6.18.tar.bz2
# cd linux-source-2.6.18
```

Lançons l'outil de configuration du noyau :

```
# make mrproper
# make menuconfig
```

Dans notre cas, nous allons laisser les options par défaut. Sortons de l'utilitaire en sauvant les modifications (même si nous n'avons réalisé aucune modification, ceci permet de créer le fichier de configuration).

Passons maintenant à la phase de création du paquet Debian du noyau :

```
# fakeroot make-kpkg --initrd --append-to-version -custom
--revision 1 kernel-image
```

Le paquet du noyau est généré dans le dossier /usr/src. Nous pouvons maintenant l'installer :

```
# cd /usr/src
# dpkg -i linux-image-2.6.18-custom_1_i386.deb
```

SYSTÈME D'AMORÇAGE – BOOTLOADER

Maintenant, il reste à configurer le système d'amorçage, appelé aussi « bootloader ». Nous allons utiliser Grub.

Tout d'abord, installons Grub :

```
# apt-get install grub
```

Installons Grub sur le nouveau disque :

```
# grub-install /dev/hdc1
```

La liste des disques durs configurés dans le fichier /boot/grub/device.map est affichée à la fin de la commande. Au besoin, modifiez cette liste.

```
(hd0)   /dev/hdc
```

Et pour finir, générons le fichier de configuration de Grub :

```
# update-grub
```

Le fichier /boot/grub/menu.lst est maintenant généré. C'est lui qui indique la liste des noyaux disponibles.

REDÉMARRAGE ET DÉCOUVERTE DU NOUVEAU SYSTÈME

Il suffit de redémarrer l'ordinateur et d'indiquer au BIOS de booter sur le nouveau disque dur. Il reste toutefois à réinstaller les applications et services que vous désirez.

CONCLUSION

Votre nouveau système est installé et n'attend plus que vous pour le configurer comme n'importe quel autre système après une installation via CD-Rom.

Debootstrap peut toutefois être utilisé avec d'autres objectifs. Il peut servir à isoler un service (serveur web, FTP, etc.) en le chrootant : toute compromission de ce serveur n'affecterait pas le système hôte du système chrooté. Une autre utilisation possible est l'installation d'applications dans un système différent du système hôte : on peut par exemple installer une application non disponible sur un système 64 bits dans un environnement 32 bits afin de pouvoir l'utiliser.

RÉFÉRENCES

Installer Debian GNU/Linux à partir d'un système Unix/Linux : http://www.debian.org/releases/stable/i386/apcs04.html.fr

Informations sur le dossier /proc (en anglais) : http://www.comptechdoc.org/os/linux/howlinuxworks/linux_hlproc.html

QUICK TIP : INSTALLER DEBIAN GNU/LINUX A DISTANCE

debootstrap permet, vous l'avez compris, d'installer rapidement un système minimal sur un point de montage ou dans un répertoire destiné à servir de racine à un système « chrooté ». Il est toutefois possible de pousser plus loin. C'est ce que propose Erik Jacobson dans une excellente documentation détaillant l'installation d'un système Debian GNU/Linux via SSH.

L'astuce consiste à préparer une partition avec debootstrap sur le système distant (merci les partitions swap de plus de 500 Mo). On copie ensuite les fichiers de configuration réutilisables avant d'entrer dans un chroot sur cette nouvelle installation qu'on configure alors à souhait.

On prépare ensuite un noyau supportant l'adaptateur réseau et les autres périphériques vitaux, puis on installe le serveur OpenSSH ou un démon Telnet. Dernière étape, on configure le système original de manière à ce qu'il démarre le nouveau noyau. Il est temps de démonter la partition cible et de *rebooter* sur le nouveau système.

Si tout se passe bien, on peut ensuite se connecter sur le nouveau système et faire le ménage dans les partitions d'origine. On peut ensuite copier le système minimal sur le système de fichiers nettoyé et reconfigurer le bootloader. Le second reboot est définitif et vous permet de personnaliser le système comme n'importe quelle distribution Debian.

Le document d'Erik Jacobson :

http://www.underhanded.org/papers/debian-conversion/remotedeb.html

Denis Bodor :: db@ed-diamond.com :: lefinnois@lefinnois.net

GNU LINUX PRATIQUE

En KIOSQUE

NUMERO 39