N°48 JUIN JUILLET 2010

France Métro : 6,50 € / DOM : 7 €
TOM Surface : 950 XPF / POL. A :1400 XPF
CH : 13,80 CHF / BEL,PORT.CONT : 7,50 €
CAN : 13 $CAD / TUNISIE : 8,80 TND / MAR : 75 MAD

GNU LINUX MAGAZINE / FRANCE

HORS-SÉRIE

Administration et développement sur systèmes UNIX

BONUS / POSTER

La chronologie du projet Debian de 1993 à nos jours en pages centrales !

APT / DPKG

Reconstruisez et personnalisez facilement vos paquets

LVM / CRYPTO

Installez un nouveau disque sur un système chiffré sans réinstaller

SERVICE / FTP

Installez un serveur vsftpd/xinetd avec authentification dédiée

SPÉCIAL DEBIAN

BESOIN D'UN SERVEUR POLYVALENT, RAPIDE ET SUR MESURE ?

debian !

DNS, DHCP, RAID, LVM, CHIFFREMENT, FTP, GESTION DE PAQUETS, ...

DISQUE / RAID

Basculez votre système sur du RAID 1 logiciel

PROJET / PHILO

Comprenez la philosophie du logiciel libre selon Debian

DNS / DHCP

Survivez au crash de vos services DNS et DHCP avec la redondance

BONUS / ARM

Découvrez comment Debian peut tenir sur une machine de 6,6 x 7,2 cm

L 15066 - 48 H - F: 6,50 € - RD

SOMMAIRE

ÉDITO

Combien ?

Oui, combien y a-t-il d'utilisateurs de distributions Debian GNU/Linux ? Difficile de répondre à ce genre de question et souvent, on se contentera de dire : « heu... plein ! ». Je ne suis pas le seul à m'être posé cette question. Anthony Towns, dans un fil de discussion sur l'élection du DPL, se l'est également posée.

Après tout, c'est bien d'être satisfait de sa distribution et conforté régulièrement dans son choix en mettant le doigt sur des points techniquement intéressants ou élégants. Mais insidieusement, la nature humaine aime à se sentir rassurée par le fait que d'autres font les mêmes choix que nous. C'est trompeur, puisque l'histoire a montré à plusieurs reprises que ce n'est pas parce qu'on est nombreux à faire un choix que c'est forcément le meilleur, le plus juste ou le plus intelligent à long terme. Quoi qu'il en soit, me direz-vous... combien ?

PopCon apporte une réponse intéressante, puisqu'il permet de classer les paquets par popularité en inspectant régulièrement les disques. Bien entendu, même si l'activation du service est proposée à l'installation de la distribution, le choix par défaut est de le laisser désactivé. Voici qui élimine pas mal d'installations. Il faut également compter les distributions mises à jour depuis longtemps, qui n'intégraient pas encore la question dans l'installeur Debian. Bref, sur cette base, nous décomptons quelques 13000 installations d'Iceweasel. En comparant cela avec les appels à la page web affichée après mise à jour, on arriverait à plus de 100000 utilisateurs potentiels. Il faut ensuite rapprocher cela du million et demi d'adresses IP enregistrées dans les logs de security.debian.org.

Donc, finalement, on peut fièrement annoncer que... personne ne sait vraiment combien. Il y a tant de machines, tant d'installations différentes, tant de dérivées, tant d'habitudes d'utilisation, ... Peut-être faudrait-il mettre en place un mécanisme d'activation (sic) ?

Tout cela pour en arriver à ce que je voulais dire : je crois que, même s'il n'y avait qu'une poignée d'utilisateurs, j'aimerai et j'utiliserai tout autant Debian GNU/Linux. Et ce, pour une raison toute simple, cette distribution me correspond et apporte une « saveur » à GNU/Linux qui me plaît.

Denis Bodor

Sorcier niveau 18

GNU/Linux Magazine est membre de l'APRIL

Membre April www.april.org

Gnu /Linux Magazine France Hors-série
est édité par Les Editions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 67 10 00 20
Fax : 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com
www.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Wilhelm
Relecture : Véronique Wilhelm
Conception graphique : Kathrin Troeger

Responsable publicité : Tél. : 03 67 10 00 26
Service abonnement : Tél. : 03 67 10 00 20
Impression : VPM Druck Allemagne
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 34 52 34 01
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976
Périodicité : Bimestrielle
Prix de vente : 6,50 €

Une organisation et un mécanisme

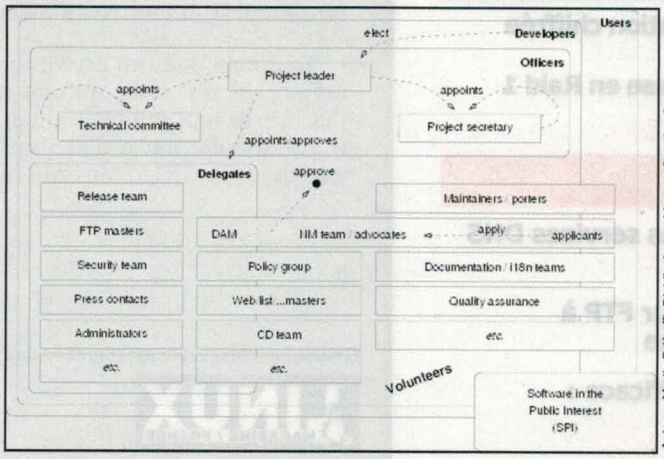
Auteur

■ Aline Hof

Debian est avant tout un vaste projet regroupant un nombre important de bénévoles de nationalités diverses. Ayant opté pour un mode de développement ouvert, c'est aussi une communauté régie par des règles démocratiques. La structure adoptée repose sur plusieurs textes, comme la Constitution Debian, qui permet de définir le rôle de chacun au sein de cet ensemble. L'organisation du projet mise également beaucoup sur l'établissement d'un réseau de confiance ; élément qui s'avère indispensable lorsque les outils de communication principaux entre les développeurs sont des listes de diffusion. En tant que cadre légal du projet, la Software in the Public Interest fait également partie des piliers de cette organisation.

1 LA CONSTITUTION DEBIAN OU LA STRUCTURE DÉCISIONNELLE ADOPTÉE PAR LE PROJET

Gérer un projet vaste, incluant des développeurs provenant des quatre coins du globe, n'est pas chose facile. Voilà pourquoi l'établissement d'une constitution afin d'apporter une forme d'organisation à l'ensemble paraît être indispensable. La première version de la Constitution Debian date de novembre 1998. Elle fut depuis révisée plusieurs fois pour en venir à une version 1.4 datant d'octobre 2007.



Aperçu de la structure décisionnelle au sein du projet Debian.

La Constitution Debian réunit quantité d'informations liées au processus de prise de décision au sein du projet. Elle recense également la structure adoptée au niveau de l'organisation de ce dernier. Plusieurs responsables sont désignés : les développeurs, le responsable de projet, le comité technique, les délégués et le secrétaire du projet. Chacun de ces derniers bénéficie d'un pouvoir de décision en adéquation avec ses fonctions. Leurs prérogatives ainsi que leur mode de nomination sont détaillés au sein du document.

Il est possible pour un même membre de l'organisation d'exercer plusieurs fonctions. Cependant, certaines d'entre elles (chef de projet, président du comité technique et secrétaire du projet) doivent être occupées par des personnes distinctes.

Outre les développeurs et le responsable de projet, sur lesquels nous reviendrons plus en détail, la structure organisationnelle du projet repose en partie sur un comité technique. Les développeurs le contactent lorsqu'ils n'arrivent pas à se mettre d'accord concernant un problème technique. Le comité technique est alors chargé de régler celui-ci. Cet organe regroupe entre quatre et huit développeurs. On trouve, à sa tête, un président, élu par ses membres ; chacun des membres du comité étant automatiquement candidat à cette fonction.

Le secrétaire de projet joue également un rôle important au sein de cette organisation. Il est nommé à la fois par le responsable de projet et le secrétaire courant pour un mandat d'un an. Plusieurs tâches lui sont confiées dans l'exercice de ses fonctions. L'une des plus importantes consiste sans doute à gérer les votes des développeurs lors de résolutions générales ainsi que durant l'élection du responsable de projet. On pourrait aussi lui attribuer le rôle de « porte-parole » de la Constitution, puisque c'est lui que l'on vient trouver lorsqu'un doute subsiste concernant l'interprétation du texte en question.

bien rodés...

2 AU CŒUR DU PROCESSUS DÉCISIONNEL : LES DÉVELOPPEURS ET LE RESPONSABLE DE PROJET

2.1 Le « responsable » Debian, un développeur aux responsabilités limitées

La fonction de « responsable » Debian, instaurée par une résolution générale en août 2007, permettra de réaliser un premier pas en vue d'un investissement plus important au sein du développement du projet. On peut qualifier le responsable Debian de mainteneur. Contrairement au développeur officiel, celui-ci joue un rôle plus limité au sein de l'organisation du projet. Concrètement, il a la possibilité d'envoyer ses paquets dans l'archive Debian selon certaines conditions, mais voit ses prérogatives réduites en matière de prise de décision. Par exemple, il ne participera pas aux élections du responsable de projet.

Comme pour devenir développeur officiel, il s'agira de passer par plusieurs étapes afin que sa candidature arrive au bout du processus (adhésion aux textes fondateurs, signature de sa clé GPG, recommandations de la part de développeurs, etc.).

2.2 Comment devenir développeur officiel ?

En dehors d'apporter des contributions au projet, il est possible de participer de manière plus active au développement de ce dernier en devenant tout simplement développeur officiel. On notera que la fonction de développeur ne renvoie pas à une tâche en particulier, mais peut recouvrir plusieurs fonctions. En dehors de réaliser des tâches purement techniques, comme le maintien d'un paquet ou d'un site, le développeur peut s'occuper de la documentation liée au projet ou réaliser des travaux de traduction.

Debian repose sur une importante communauté de bénévoles. Il est donc important de pouvoir établir un réseau de confiance afin de mener à bien le développement d'un projet d'une telle envergure. Dans cette optique, on peut plus aisément comprendre pourquoi la démarche pour devenir développeur officiel peut paraître assez réglementée, d'autant plus qu'incombent à ce dernier bon nombre de devoirs vis-à-vis du projet.

En plus de la possibilité de faire part de leur avis concernant le développement de la distribution, les développeurs comptent en effet, au nombre de leurs pouvoirs, le fait de nommer mais aussi de révoquer le responsable de projet. S'ils sont un nombre suffisant, il leur sera également possible de faire réviser les documents fondateurs entourant le projet. Par le biais de résolutions, ils pourront dans certains cas outrepasser voire suspendre une décision prise par le responsable de projet et ses délégués.

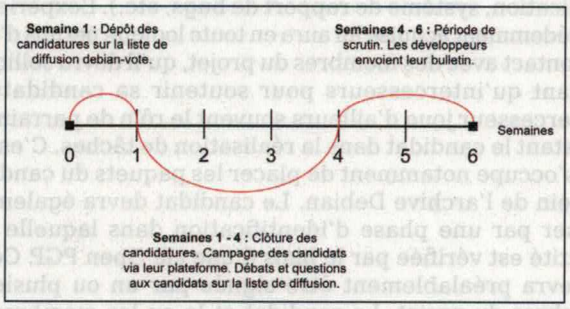
Le processus de candidature pour devenir développeur officiel se divise en plusieurs étapes. Avant même de déposer sa candidature, il est recommandé au candidat d'avoir pris connaissance des documents fondateurs du projet (le Contrat Social Debian ainsi que les Principes du logiciel libre selon Debian). Autant dire qu'il lui est fortement conseillé d'avoir au préalable déjà contribué au projet et d'être familier du champ d'application dans lequel il souhaite s'investir, mais aussi des diverses procédures relatives au développement de la distribution (processus de publication, système de rapport de bugs, etc.). L'expérience précédemment acquise lui aura en toute logique permis d'être en contact avec des membres du projet, qu'il devra solliciter en tant qu'intercesseurs pour soutenir sa candidature. L'intercesseur joue d'ailleurs souvent le rôle de parrain, en assistant le candidat dans la réalisation de tâches. C'est lui qui s'occupe notamment de placer les paquets du candidat au sein de l'archive Debian. Le candidat devra également passer par une phase d'identification dans laquelle son identité est vérifiée par le biais d'une clé Open PGP. Celle-ci devra préalablement être signée par un ou plusieurs membres du projet. Le candidat et le ou les membres en question devront se rencontrer physiquement (des réunions de signature de clés sont organisées). Le candidat devra fournir une pièce d'identité avant de pouvoir voir sa clé signée. Tout au long du processus de candidature, c'est le responsable de candidature, préalablement désigné par le secrétariat, qui prendra en charge le candidat. Il orientera notamment ce dernier vers les tâches qu'il lui sera possible d'effectuer et en profitera pour recueillir des informations sur les compétences du candidat qu'il regroupera au sein d'un rapport final. Ce document passera ensuite aux mains du secrétariat, qui vérifiera la présence de l'ensemble des éléments requis. C'est finalement le responsable des comptes qui aura le dernier mot concernant la candidature. Si celle-ci est acceptée, ce dernier procédera à la création des comptes qui serviront au nouveau développeur.

2.3 À la tête du projet, le Debian Project Leader

Le responsable de projet, ou *Debian Project Leader* (DPL), est le représentant du projet Debian durant toute la durée de son mandat. Il oriente la direction du projet et coordonne celui-ci pendant ce laps de temps. C'est à lui que revient de prendre des décisions lors de situations urgentes. Il lui est possible de proposer des amendements ainsi que des résolutions générales. C'est également lui qui prendra des décisions concernant les fonds utilisés pour les besoins du projet en tenant compte de l'avis des développeurs. Il peut s'entourer de développeurs à qui il pourra confier l'exécution de tâches spécifiques : on parle

alors de délégués du responsable de projet. Les décisions prises par ces derniers ne pourront pas être transgressées par le responsable de projet.

Chaque développeur peut se porter candidat à la fonction de responsable de projet. La période des élections dure en moyenne six semaines. Durant la première, les développeurs intéressés déposent leur candidature. Les trois semaines suivantes sont consacrées à la campagne proprement dite. Chacun d'entre eux défend ce que l'on appelle une plate-forme ou un programme. Ce dernier se présente à peu près de la même manière pour chaque candidat. Vient tout d'abord une première partie, dans laquelle le candidat parle de lui et développe ce qui l'a poussé à poser sa candidature. Il détaille ensuite sur plusieurs points les axes qu'il souhaiterait mettre en avant au sein du projet ainsi que les aspects qu'il compte améliorer. La dernière partie de la plate-forme est consacrée à la critique des programmes proposés par les autres candidats. Les développeurs disposent ensuite de deux semaines pour envoyer leur bulletin de vote.



Déroulement des élections pour la fonction de responsable de projet.

Le projet utilise la méthode Condorcet afin de déterminer l'issue du scrutin. À noter que le même procédé est également appliqué dans le cadre de résolutions générales. Chacun des votants classe les candidats par ordre de préférence, sachant qu'il leur est également possible de recourir à l'option « *Non of the above* » (aucun de ceux qui précèdent). À la fois politologue, philosophe et mathématicien, Condorcet mit en avant le fait que l'issue d'un vote pluraliste ne correspond pas toujours aux avis des votants.

Prenons l'exemple d'une communauté de 50 votants qui doivent choisir entre les candidats Alice, Bernard et Christine :

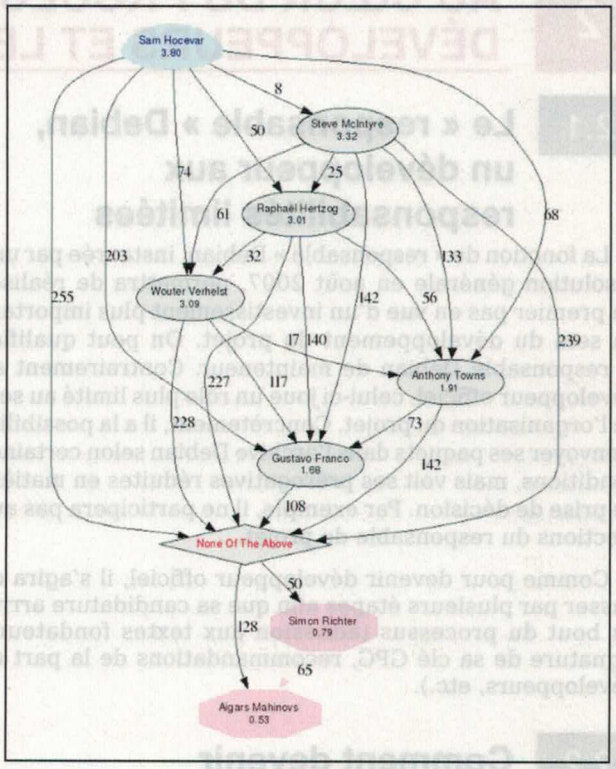
- 22 candidats votent Alice > Christine > Bernard ;
- 14 candidats votent Bernard > Christine > Alice ;
- 12 candidats votent Christine > Bernard > Alice ;
- 2 candidats votent Christine > Alice > Bernard.

Alice arrive ainsi en tête avec 22 voix, Bernard prend la seconde place avec 14 voix et enfin, Christine finit bonne dernière avec 12 voix. Cela nous donne Alice > Bernard > Christine.

Procédons à une comparaison par paires, comme le suggère Condorcet. Les résultats sont alors les suivants :

- 26 préfèrent Bernard > Alice contre 24 pour Alice > Bernard ;
- 36 préfèrent Christine > Bernard contre 14 pour Bernard > Christine ;
- 28 préfèrent Christine > Alice contre 22 pour Alice > Christine.

On obtient finalement Christine > Bernard > Alice, un résultat totalement différent de celui trouvé par le biais d'un vote pluraliste.



Debian a recours à l'ensemble de Schwartz afin de régler les conflits de circularité. Cette méthode utilise un graphe orienté dans lequel chacun des sommets représente un candidat. Ici, le schéma relatif aux élections 2007 du DPL.

Le mandat d'un responsable de projet dure un an, mais il a la possibilité d'être reconduit si l'ancien Debian Project Leader est réélu par ses pairs. Martin Michlmayr ou encore Steve McIntyre furent ainsi à la tête du projet durant deux années consécutives.

2.4 Qu'en est-il de l'actuel Debian Project Leader ?

Après s'être présenté pour la seconde fois, Stefano Zacchiroli est le nouveau responsable de projet depuis le 17 avril 2010. Il succède à Steve McIntyre. Près de 49% des développeurs ont participé à ce dernier scrutin qui eut lieu du 5 mars au 15 avril 2010. Les bulletins de vote qui ont servi à élire le nouveau candidat resteront secrets même après la fin du processus d'élection. Outre Stefano Zacchiroli, trois autres candidats se sont présentés à cette dernière élection : Wouter Verhelst, Charles Plessy et Margarita Manterola.

Développeur Debian depuis 2001, Stephano Zacchiroli a résumé ce qu'il compte apporter au projet en six points précis :

« *I intend to be a present DPL, both in discussions and as the responsible for the project agenda.*

I will provide a stream of DPL activity news, to be frozen and posted monthly to d-d-a.

I will apply mechanisms that make consensus emerge when it exists.

I will push for more gradual and rewarding access paths to Debian.

I will fight strong package ownership when it conflicts with quality.

I will do my best to support, with money and other resources, contributors meetings. »

Il s'agira notamment de ressouder les liens avec la communauté par le biais d'une plus grande transparence dans le travail réalisé par le responsable de projet. En outre, Zacchiroli souhaite également travailler sur l'aspect du site du projet afin de le rendre plus attrayant : « *We all want a sexier website, i.e. a website where people can find what they look for, and which does not make us look like Debian is an operating system of the 1980s.* ». Cette problématique avait déjà été évoquée par Sam Hocevar en 2007 : « *Actuellement, Debian rime avec épicurien, octogénaire et homme des cavernes. Même le site de FreeBSD est plus attirant que le nôtre, comment pouvons-nous espérer attirer des utilisateurs ?* ». On ne peut donc que remarquer que peu de choses ont été réalisées à ce niveau jusque-là. Zacchiroli compte enfin apporter plus de clarté à la structure organisationnelle, notamment pour ce qui concerne les délégués et les différentes équipes de travail. Il ne reste plus qu'à attendre avril 2011 pour voir si le nouveau DPL aura eu le temps de concrétiser toutes ses idées...

3 LA CHARTE DEBIAN, UN « GUIDE » POUR LES DÉVELOPPEURS

En dehors de sa structure organisationnelle, le projet se doit également d'être organisé sur le plan technique. Ian Jackson rédigea pour cela la Charte Debian ou la *Debian Policy* en 1996.

Il s'agit concrètement d'une compilation de règles techniques et de recommandations revenant notamment sur la « construction » de la distribution avec les principes suivis, les différentes sections et niveaux de priorités utilisés dans le développement du projet. Les éléments évoqués permettront aux membres du projet de suivre un certain nombre de standards internes comme externes relatifs au développement de ce dernier. La Charte Debian peut ainsi servir de manuel aux développeurs qui trouveront au sein de ce document bon nombre d'informations relatives aux paquets, aux bibliothèques, à l'OS en particulier.

4 LA SOFTWARE IN THE PUBLIC INTEREST, L'ENTITÉ LÉGALE ENTOURANT LE PROJET

Après avoir réalisé un tour d'horizon de la structure interne adoptée par le projet Debian, il est logique d'évoquer en dernier lieu l'organisation qui sert de cadre à l'ensemble : la *Software in the Public Interest* (SPI). C'est Bruce Perens et Ian Schuessler qui furent à l'origine de la création de SPI afin de donner une entité légale au projet Debian lorsque la *Free Software Foundation* cessa de le soutenir. La SPI est une organisation soutenant le développement et la distribution de logiciels et de matériels libres dont elle encourage l'utilisation de la licence GNU GPL. En dehors de Debian, la SPI supporte bon nombre de projets tels que Drupal, OpenOffice.org, freedesktop.org, PostgreSQL ou encore GNUstep.

Mais pour en revenir au sujet qui nous occupe, il faut savoir que c'est la SPI qui s'occupe de gérer les fonds alloués au projet Debian. En charge de la gestion des fonds nécessaires au développement de la distribution, le responsable de projet joue le rôle de conseiller auprès de la SPI. On notera par ailleurs que bon nombre d'anciens DPL ont occupé ou occupent encore à l'heure actuelle une place au sein de cette association. C'est notamment le cas de Bdale Garbee, qui est l'actuel président de cette entité.

Si l'on souhaite ainsi faire un don au projet, c'est vers la SPI que l'on pourra se tourner. On trouve en Europe deux partenaires à cette organisation : la *Verein zur Förderung Freier Informationen und Software e.V.*, qui se situe en Allemagne, ainsi que l'*Associazione Software Liberon*, en Italie.

Outre la SPI qui lui sert de cadre principal, Debian peut recourir à quelques autres organismes afin de gérer ses capitaux. Une liste publique de ces derniers a été mise en place. Reposant avant tout sur une vaste communauté, on notera enfin que le projet a recours à plusieurs partenaires (Sun Microsystems, Skolelinux, Simtec, Hewlett-Packard, etc.), lui fournissant aussi bien du matériel que des services d'hébergement, par exemple. ■

Références & liens

- Constitution Debian : http://www.debian.org/devel/constitution
- Mainteneur Debian : http://wiki.debian.org/DebianMaintainer
- Le coin du nouveau responsable Debian : http://www.debian.org/devel/join/newmaint#AppMan
- Système de vote utilisé par Debian : http://www.debian.org/vote/
- Les dernières élections du DPL : http://www.debian.org/vote/2010/vote_001
- *Debian Policy* : http://www.debian.org/doc/debian-policy/
- *Software in the Public Interest* : http://www.spi-inc.org/

Auteur : Aline Hof

Debian et le libre : la philosophie

Il est rare de voir de nos jours des distributions GNU/Linux accordant tant d'importance aux principes et convictions entourant leur développement. De par son histoire, Debian fait partie des exceptions. Ainsi, tout utilisateur souhaitant davantage s'impliquer au sein du projet devra au préalable adhérer à un certain nombre de documents « fondateurs ». Au-delà même d'apporter ses compétences à l'édification d'un système, il s'agit d'accepter et de comprendre les idéaux suivis par celui-ci.

1 À L'ORIGINE, LE MANIFESTE DEBIAN

Un peu moins de neuf années après la parution du Manifeste GNU de Richard Stallman, Ian Murdock rédige son propre manifeste lié à la création de la distribution Debian. Dans ce document, sont mis en avant les éléments ayant motivé l'avènement d'un tel projet, ses finalités ainsi que les solutions qu'il tentera d'apporter afin de mettre en avant l'OS Linux.

À une époque où les distributions ne sont encore que peu répandues, Ian met en avant le rôle joué par celles-ci. Elles proposent en effet aux utilisateurs un système clé en main et fonctionnel qui leur permettra de profiter de l'OS Linux. Debian se présente dans cette optique-là comme une distribution « nouvelle génération », car se basant sur un mode de développement ouvert, proche de la philosophie entourant les projets GNU et Linux. Il s'agit en fait de conjuguer les efforts de chacun pour créer un système qui saura se faire une place parmi les solutions commerciales. Un développement ouvert permet de mieux répondre aux attentes des utilisateurs qui pourront eux-mêmes mettre à rétribution leurs compétences afin d'améliorer la distribution.

Auteur : Ilya Schurov. Photo distribuée sous licence Creative Commons Share Alike 2.0.

Ian Murdock explicite, au sein du Manifeste Debian, une partie des motivations qui l'ont poussé à créer la distribution.

Le projet Debian est également l'occasion d'améliorer l'image entourant les distributions Linux. Le « maintien » de la distribution à jour est quelque chose d'important sur lequel bon nombre de projets font défaut. Le travail entourant la réalisation d'un tel projet est quelque chose de fastidieux. Il faut en effet prendre en compte divers éléments pour que le système soit « utilisable » pour le plus grand nombre. Avec son mode de développement, Debian tentera d'apporter une solution à l'ensemble de ces problématiques.

2 LES RELATIONS AVEC LA FREE SOFTWARE FOUNDATION

Au sein du Manifeste Debian, Ian Murdock fait tout particulièrement référence au rôle joué par la *Free Software Foundation* (FSF) en tant que soutien du projet. Pour rappel, la FSF fut fondée par Richard Stallman en 1985 pour encadrer le projet GNU. Suite à une requête de Richard Stallman, Debian se verra attribuer en toute logique la mention GNU/Linux. On parlera dès lors de Debian comme d'une distribution GNU/Linux.

Dans les faits, le support de la FSF n'aura duré qu'un an (de novembre 1994 à novembre 1995). Des désaccords sur le plan technique verront rapidement le jour. Mais c'est surtout la mise à disposition d'un logiciel non libre au sein de la section *contrib* (Netscape, Debian 1.1) qui mettra fin aux relations étroites qu'entretenaient la distribution et l'organisation. La FSF critique en effet le fait que, même si ces derniers ne sont pas directement intégrés au sein du projet, Debian met à disposition des utilisateurs des programmes propriétaires. Elle ne cautionne pas ce choix et ne recommandera plus Debian comme distribution libre.

entourant le projet

3 UN « PACTE » PASSÉ AVEC LA COMMUNAUTÉ DU LIBRE : LE CONTRAT SOCIAL DEBIAN

Bruce Perens reprit les rênes du projet Debian en avril 1996. Après le Manifeste Debian, qui permit d'établir certaines des bases entourant le projet, Perens s'attèle à la rédaction du Contrat Social Debian, forme de « pacte » passé avec la communauté du logiciel libre. La première version du document date de juillet 1997.

Le contrat social énonce un certain nombre de principes liés à la philosophie du projet. Il insiste notamment sur la liberté entourant la distribution et son développement. Le but est de satisfaire les besoins des utilisateurs ainsi que ceux de la communauté et de partager avec ces derniers les travaux réalisés. Les travaux n'étant pas libres sont placés dans des sections particulières. Ils ne font pas partie de la distribution, mais sont néanmoins mis à disposition des utilisateurs.

Concrètement, ce contrat social développe cinq points précis :

- La liberté entourant le projet qui fait que celui-ci ne sera en aucun cas lié à un composant propriétaire.
- L'utilisation de licences libres permettant de partager les travaux réalisés avec la communauté.
- La transparence du projet, puisque les problèmes rencontrés dans le développement de la distribution sont rendus publics via des rapports de bugs.
- L'importance accordée aux utilisateurs dans l'optique de développer une distribution répondant au mieux à leurs attentes.
- Le traitement accordé aux logiciels non libres mis à disposition dans des sections dédiées.

4 « DEBIAN DEMEURERA TOTALEMENT LIBRE » : LES PRINCIPES DU LOGICIEL LIBRE SELON DEBIAN

Le contrat social comprend une partie dédiée aux principes du logiciel libre selon Debian. Ces derniers permettent de nous éclairer sur la notion de « liberté » défendue par le projet. Ces principes sont présentés en dix points :

1- Chaque composant de la distribution peut être réutilisé et redistribué librement et gratuitement. Aucune rétribution n'est nécessaire.

2- Le code source utilisé est rendu public.

3- Les programmes dérivés doivent recourir à la même licence que celle utilisée par le logiciel dans sa version initiale.

4- La distribution de programmes dont le code source a été modifié est autorisée. Ces modifications peuvent être signalées par un nom ou une numérotation de version différente de celle utilisée par l'application dans sa version originale. Il est également possible de diffuser le code source dans sa version initiale accompagné de fichiers de corrections.

5- La licence utilisée ne doit discriminer personne.

6- La licence permet l'utilisation du programme dans n'importe quel champ d'application.

Bruce Perens est l'auteur du Contrat Social Debian et des Principes du logiciel libre selon Debian.

7- La licence doit s'appliquer à tous ceux à qui elle est distribuée.

8- La licence utilisée n'est pas spécifique à Debian. Si l'un de ses composants est utilisé indépendamment du projet, celui-ci bénéficiera des mêmes droits que ceux qui lui étaient accordés au sein de la distribution.

9- L'utilisation d'un logiciel sous licence libre n'implique pas l'obligation de recourir à ce même type de licence pour la distribution d'autres logiciels accompagnant le programme en question.

10- Les licences BSD et GPL sont des exemples de licences dites libres pour le projet.

Bruce Perens fut également l'auteur de ce document qui lui servit de base lors de la rédaction de *l'Open Source Définition*. Il y reprend en effet chacun des dix points énoncés qu'il applique dans ce cas à la description d'une licence libre. Cette définition fut reprise par l'*Open Source Initiative*, organisation dont Perens fut le cofondateur, qui vise à promouvoir l'utilisation de programmes open source. ■

Auteur : Aline Hof

Réclamer l'empaquetage d'une

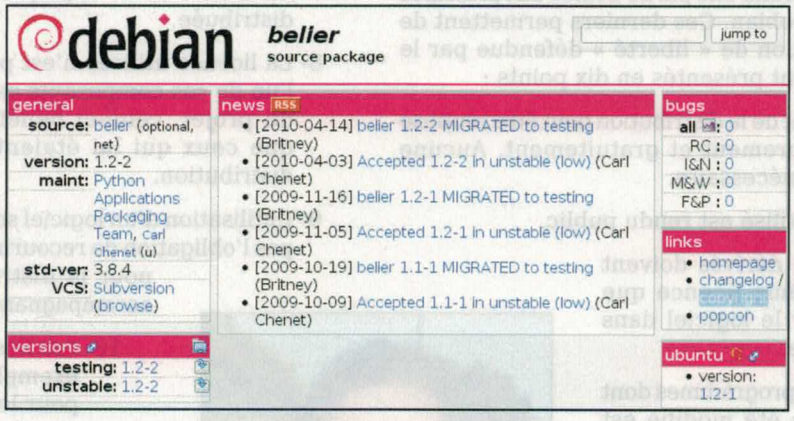
Auteur

- Carl Chenet

Vous êtes utilisateur d'une application qui vous rend bien des services. Vous suivez ce projet depuis quelques versions. Il est très actif et le logiciel qui en résulte peut intéresser un grand nombre de personnes avec les mêmes besoins que vous. Pourtant, ce logiciel n'est pas dans Debian et vous êtes contraint de l'installer à partir des sources du projet, ce qui s'avère gênant lorsque vous mettez à jour l'ensemble de votre système ou simplement l'application en question. Les choses seraient beaucoup plus faciles si cette application était intégrée directement dans Debian. Mais comment faire ? Vous n'avez ni la compétence ni le temps de créer un paquet Debian. La solution est simple : il suffit de demander l'empaquetage de votre application. Suivez le guide.

1 PRÉREQUIS À LA DEMANDE D'EMPAQUETAGE

Tout d'abord, votre logiciel doit être susceptible d'intéresser d'autres personnes que vous, et si possible, un public aussi large que faire se peut. Empaqueter un logiciel et maintenir ce paquet, ce qui signifie mettre à jour le paquet si une nouvelle version sort ou intégrer un patch afin de corriger un problème gênant, va demander beaucoup de temps et de compétences à la personne - le plus souvent un développeur Debian - qui va créer le paquet.



Page du suivi qualité d'un paquet avec le lien vers le fichier copyright

Ensuite, assurez-vous également que ce logiciel n'est pas en fin de vie. Vous pouvez vous en rendre compte simplement en regardant la date de sortie de la dernière version et en jetant un œil à l'activité du dépôt de sources s'il existe. En effet, si des bugs sont découverts, le mainteneur du paquet Debian sera heureux de pouvoir se tourner vers le développeur de l'application. Si celui-ci a abandonné le projet et ne fournit plus de correctifs, il y a fort à parier que ce logiciel sera plus ou moins rapidement dépassé. Il y a donc peu d'intérêt à l'intégrer dans Debian.

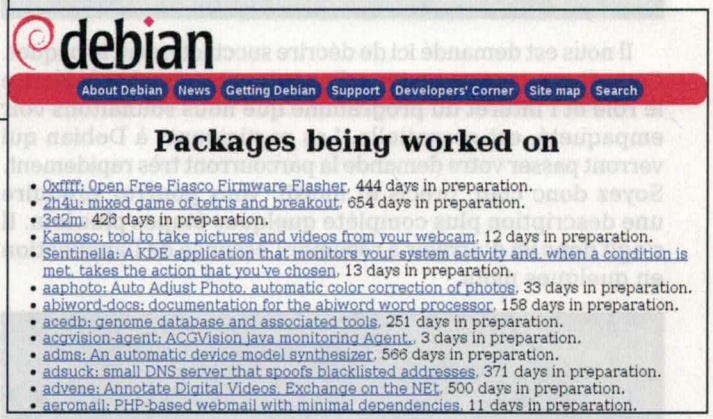
La remarque suivante va paraître triviale, mais assurez-vous également que votre logiciel n'est pas déjà dans Debian. Plusieurs cas possibles : ce logiciel est déjà dans Debian mais pas dans la version de Debian (stable, en test, instable) que vous utilisez au quotidien. Vous pouvez, en effet, être un utilisateur de la version stable et votre logiciel a pu apparaître il y a quelques jours dans la version instable. Autre cas possible : votre logiciel a été intégré dans Debian avec un nom légèrement différent de celui auquel vous pouviez vous attendre, par exemple pour la bibliothèque Python `redis-py`, qui est nommée `python-redis` dans Debian. Si vous soupçonnez un paquet d'empaqueter le logiciel que vous souhaitez proposer, il suffit de consulter son fichier *copyright* accessible via l'adresse http://packages.qa.debian.org/nompaquet, puis en cliquant sur le lien copyright sur la droite. L'adresse du site du projet original est indiquée.

application dans Debian

La page consacrée aux demandes d'empaquetage

Il sera nécessaire également de vérifier qu'une autre personne n'a pas déjà demandé l'empaquetage de l'application. Pour cela, il faudra vous rendre à l'adresse suivante **[1]**. Cette page fournit la liste des paquets que le public souhaite voir intégrés au projet Debian. Il vous faut donc vérifier que le nom de votre application n'y figure pas.

Une dernière vérification reste à effectuer. Il est envisageable que quelqu'un travaille en ce moment-même sur le paquet de cette application, ce qui est possible si cette dernière a récemment suscité un fort intérêt. Pour vérifier, il faut vous rendre à l'adresse suivante **[2]** et chercher dans la liste des paquets en cours de création le nom de votre programme.



La page consacrée aux paquets en cours de réalisation

2 CRÉER UNE ENTRÉE DANS LA LISTE DES PROGRAMMES À INTÉGRER AU PROJET DEBIAN

Nous l'avons vu plus haut, une liste des programmes à intégrer au projet Debian existe **[1]**. Elle est le point de départ de chaque application dans le projet Debian. Nous allons créer cette demande, signalant ainsi à l'univers Debian l'intérêt que peut avoir l'intégration de cette application dans le projet.

Pour effectuer cette demande, nous allons utiliser l'outil **reportbug**. Ce programme permet d'envoyer des rapports de bugs concernant les différents paquets de Debian, mais aussi - et c'est ici l'emploi qui nous intéresse - de générer une demande d'empaquetage. Il faut pour cela donner en argument à la commande **reportbug** un paquet spécial, à savoir le paquet **wnpp** (pour *Work-Needing and Prospective Packages*). Nous procédons de la façon suivante en tapant la commande :

```
$ reportbug --ui=text wnpp
```

Nous demandons ici l'emploi de l'interface en mode texte de **reportbug** et nous indiquons le paquet **wnpp**.

```
*** Welcome to reportbug.  Use ? for help at prompts. ***
Detected character set: UTF-8
Please change your locale if this is incorrect.

Using 'Carl Chenet <chaica@ohmytux.com>' as your from address.
Getting status for wnpp...
Will send report to Debian (per lsb_release).
What sort of request is this? (If none of these things mean anything to
you, or you are trying to report a bug in an existing package, please
press Enter to exit reportbug.)

1 ITP  This is an `Intent To Package'. Please submit a package
description along with copyright and URL in such a report.
2 O    The package has been `Orphaned'. It needs a new maintainer as
soon as possible.
3 RFA  This is a `Request for Adoption'. Due to lack of time, resources,
interest or something similar, the current maintainer is asking for
someone else to maintain this package. They will maintain it in
       the meantime, but perhaps not in the best possible way. In short:
the package needs a new maintainer.
4 RFH  This is a `Request For Help'. The current maintainer wants to
continue to maintain this package, but they needs some help to do this,
because their time is limited or the package is quite big and
       needs several maintainers.
5 RFP  This is a `Request For Package'. You have found an interesting
piece of software and would like someone else to maintain it for Debian.
Please submit a package description along with copyright and
       URL in such a report.

Choose the request type: 5
```

Plusieurs choix vous sont ici proposés. Nous choisissons le 5ème choix, à savoir une requête RFP pour « *Request For Package* » (demande de paquet). Notre but consiste à éveiller l'intérêt d'un participant au projet Debian susceptible de réaliser notre paquet.

```
Please enter the proposed package name: actarus
```

Nous saisissons le nom que devrait avoir le futur paquet. Ce nom correspond en général au nom de l'application pour laquelle un paquet doit être réalisé.

```
Checking status database...
Please briefly describe this package; this should be an appropriate
short description for the eventual package: display an image of Goldorak
```

Il nous est demandé ici de décrire succinctement le paquet. Cette étape, comme toutes celles qui vont s'attacher à décrire le rôle et l'intérêt du programme que nous souhaitons voir empaqueté, est essentielle. Les participants à Debian qui verront passer votre demande la parcourront très rapidement. Soyez donc clair, précis et incisif. Vous pourrez produire une description plus complète quelques étapes plus loin. Il s'agit donc ici d'arriver à résumer l'essence de l'application en quelques mots.

```
Querying Debian BTS for reports on wnpp (source)...
3670 bug reports found:

 Outstanding bugs -- Request for Help; Normal bugs (63 bugs): 2 remain
     1) #248397  RFH: grub2 -- GRand Unified Bootloader
     2) #278442  RFH: athcool -- Enable powersaving mode for Athlon/
Duron processors
     3) #282283  RFH: dpkg -- dselect: a user tool to manage Debian
packages
     4) #313369  RFH: mwavem -- Mwave/ACP modem support software
     5) #332498  RFH: openssl -- Secure Socket Layer (SSL) binary and
related cryptographic tools
     6) #354174  RFH: nas -- The Network Audio System
     7) #354176  RFH: cvs -- Concurrent Versions System
     8) #385614  RFH: loop-aes-utils -- Tools for mounting and
manipulating filesystems
     [...]
(1-61/3670) Is the bug you found listed above [y|N|b|m|r|q|s|f|?]? ?
y - Problem already reported; optionally add extra information.
N - (default) Problem not listed above; possibly check more (skip to
Next page).
b - Open the complete bugs list in a web browser.
m - Get more information about a bug (you can also enter a number
without selecting "m" first).
r - Redisplay the last bugs shown.
q - I'm bored; quit please.
s - Skip remaining problems; file a new report immediately.
f - Filter bug list using a pattern.
? - Display this help.
(1-61/3670) Is the bug you found listed above [y|N|b|m|r|q|s|f|?]? s
```

reportbug vous présente ici tous les rapports relatifs au paquet **wnpp**. Si vous avez correctement effectué les vérifications évoquées plus haut, vous pouvez presser la touche [s] afin d'accéder directement au corps de la demande. Dans le cas contraire, vous avez également la possibilité de rechercher parmi les rapports présentés un qui ferait du votre un doublon. La fonction de filtre accessible grâce à la touche [f] vous permettra de filtrer les titres des rapports à partir d'un motif de recherche.

```
Subject: RFP: actarus -- display an image of Goldorak
Package: wnpp
Severity: wishlist

*** Please type your report below this line ***

* Package name    : actarus
  Version         : x.y.z
  Upstream Author : Name <somebody@example.org>
* URL             : http://www.example.org/
* License         : (GPL, LGPL, BSD, MIT/X, etc.)
  Programming Lang: (C, C++, C#, Perl, Python, etc.)
  Description     : display an image of Goldorak

(Include the long description here.)
```

Votre éditeur de texte s'ouvre et présente les lignes ci-dessus. Nous allons les modifier afin d'arriver à une description réellement représentative de l'application à empaqueter. Les quatre premières lignes ne sont pas à modifier. Le champ **Package name** indique le nom du futur paquet tel que vous l'avez vous-même défini plus tôt. Le champ **Version**, quant à lui, doit donner le numéro de version de l'application. **Upstream Author** annonce l'identité de la personne qui est à l'origine de l'application. Le champ **URL** permettra aux personnes intéressées d'aller consulter le site du projet. Puis il est nécessaire d'indiquer la licence de cette application. Le champ **Programming Lang** permet aux curieux de connaître le langage utilisé pour développer l'application. Enfin, vous reconnaissez la description courte que vous avez écrite un peu plus haut.

Nous modifions donc ces différents champs afin de fournir le plus d'informations pertinentes aux développeurs qui verront passer la demande. Il est encore tout à fait possible d'apporter des modifications aux champs **Package name** et **Description** définis plus tôt. Si vous modifiez le nom du paquet, pensez alors à intervenir sur le nom du paquet de la ligne **Subject** afin que les deux concordent.

```
Subject: RFP: actarus -- display an image of Goldorak
Package: wnpp
Severity: wishlist

*** Please type your report below this line ***

* Package name    : actarus
  Version         : 1.2
  Upstream Author : Marc Robert <marc.robert@fsf.org>
* URL             : http://actarus-project.org
* License         : GPL
  Programming Lang: Python
  Description     : display an image of Goldorak

Actarus displays an image of Goldorak in Ascii art and exits.
You can choose several kind of images, e.g Actarus with Goldorak
or Actarus and his sister Phenicia or Goldorak and Venusia.

Actarus can check an Internet database to retrieve new images
of Goldorak.
```

Il est indispensable de bien réfléchir à la description que vous fournissez. Qu'il s'agisse de la description courte ou des lignes qui suivent (communément appelées description longue pour une requête RFP), elles vont être essentielles pour qu'on s'intéresse à votre annonce. Une mauvaise description est presque toujours synonyme d'échec. Si vous ne réussissez pas à faire s'intéresser un participant régulier

au projet Debian à votre application, le paquet ne sera pas réalisé. Vos deux champs de description représentent votre unique moyen d'assurer la promotion de votre demande d'empaquetage. N'hésitez donc pas à réfléchir longuement pour arriver à un résumé qui fera comprendre immédiatement à tous les lecteurs de l'annonce ce que fait votre application, à quoi elle sert concrètement et ses spécificités.

Nous enregistrons nos modifications et nous quittons l'éditeur. **reportbug** reprend la main afin d'envoyer votre demande vers les serveurs Debian.

```
Spawning sensible-editor...
Report will be sent to "Debian Bug Tracking System" <submit@bugs.debian.org>
Submit this report on wnpp (e to edit) [Y|n|a|c|e|i|l|m|p|q|d|t|s|?]? Y
```

Vous recevrez quelques minutes plus tard un accusé de réception de votre demande par e-mail. Cet e-mail contiendra un numéro de demande qui vous servira à en suivre l'évolution. De plus, votre demande sera ajoutée à la liste des applications à empaqueter. Vous pourrez accéder directement au suivi de votre demande via l'adresse http://bugs.debian.org/numérodemande.

3 LA SUITE DU PROCESSUS

Vous devez maintenant vous armer de patience en attendant qu'un participant au projet Debian s'intéresse à l'application en question et réalise le paquet. Ce processus peut être très long. Debian repose sur des bénévoles, avec un temps libre à consacrer au projet qui n'est pas toujours extensible. Ils ont de plus chacun leur centre d'intérêts, certains s'intéressant au développement, d'autres à l'administration système, etc. Votre demande sera dans tous les cas rapidement lue par un grand nombre de ces personnes. On comprend donc l'importance de fournir une description à même de faire comprendre à ces personnes si elles sont ou non intéressées par votre démarche.

Lorsque vous attendrez de recevoir une réponse à votre demande, ne perdez pas complètement le lien avec votre application. Consultez de temps à autre le suivi de votre demande afin de vérifier si des interventions auxquelles vous pourriez participer n'ont pas lieu.

Si la description de votre projet est percutante et qu'un développeur Debian est emballé, ce dernier va transformer votre demande. Elle va passer du statut de *Request For Package* (RFP) à *Intend To Package* (ITP). Cela signifie que cette personne s'est déclarée responsable de votre demande et qu'elle a commencé à travailler sur le paquet de l'application.

Lorsque le paquet finalement prêt sera envoyé sur le dépôt officiel des paquets du projet Debian, un e-mail vous sera envoyé pour vous avertir de la fin du processus d'empaquetage de votre application. Suite à la réception de cet e-mail, ce paquet devrait être téléchargeable via la commande **aptitude** dans la version instable (*unstable*) de Debian. Si aucun bug majeur n'est détecté sur le paquet, il sera intégré quinze jours plus tard dans la version de test (*testing*) de Debian. Finalement, ce dernier sera inclus à la nouvelle version stable de Debian. Et tout cela grâce à vous et à votre demande d'empaquetage.

POUR CONCLURE

Nous venons de voir que réclamer l'empaquetage d'un logiciel dans Debian n'est pas très compliqué et présente seulement deux difficultés. La première va résider dans les quelques recherches nécessaires à faire avant d'émettre cette demande pour vous assurer qu'elle n'a pas déjà été faite et que le logiciel n'est pas encore disponible dans Debian. La seconde consiste à réaliser une description suffisamment claire, incisive et complète pour qu'un développeur potentiellement intéressé puisse la comprendre en un clin d'œil. Le reste du processus consiste à attendre et répondre aux éventuelles questions qu'on pourrait vous poser au sujet de l'application que vous désirez voir intégrée.

Réclamer l'empaquetage d'un paquet dans Debian n'est pas une action anodine, c'est une participation forte au projet dans son entier. En avertissant les membres du projet de la disponibilité d'un nouveau logiciel intéressant recherché par le public, vous contribuez au maintien de Debian parmi les distributions phares GNU/Linux. ■

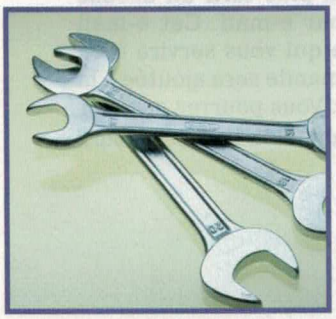
Auteur : Carl Chenet

Carl Chenet, ingénieur système GNU/Linux. Contributeur Debian et développeur du projet Bélier. Membre de l'association Debian France. Également membre de l'association AFPY pour la promotion du langage Python.

Références

- [1] http://www.debian.org/devel/wnpp/requested
- [2] http://www.debian.org/devel/wnpp/being_packaged
- La page web du projet Reportbug : http://packages.debian.org/sid/reportbug
- Le projet Debian : http://www.debian.org

7 utilitaires APT pour mieux



Tout le monde connaît sans doute les commandes apt-get, apt-show et aptitude. Celles-ci permettent toutes sortes d'opérations comme la recherche de paquets, la consultation du descriptif, l'installation binaire ou encore l'installation source. Lorsqu'on cherche sans cesse de nouveaux moyens d'améliorer sa configuration des commandes, ces solutions, bien que complètes, montrent tantôt leurs limites.

Auteur

- Denis Bodor

Ainsi, pour répondre aux besoins des utilisateurs les plus exigeants, certains développeurs ont décidé d'apporter leur touche personnelle à la gestion de paquets. Qu'ils soient développés en Perl, en C, en Ruby ou en Python, des nouveaux outils ont vu le jour et permettent des manipulations fort intéressantes. Voici une petite sélection de quelques utilitaires dont je vous laisse le soin d'évaluer la pertinence en fonction de vos besoins.

1 APT-FILE

Ne vous êtes-vous jamais demandé d'où sortait tel ou tel fichier présent dans votre système ? De quel paquet pouvait bien provenir ce fichier récalcitrant ou cruellement absent ? Sebastien J. Gross a dû vivre cette expérience plus d'une fois, car il est l'auteur de **apt-file**. Pour bien comprendre l'intérêt de l'outil, commençons par détailler les solutions disponibles par défaut.

Lorsqu'on cherche à savoir d'où provient un fichier, c'est par réflexe à **dpkg** qu'on en fera la demande :

```
% which iostat
/usr/bin/iostat

% dpkg -S /usr/bin/iostat
sysstat: /usr/bin/iostat
```

Ou plus élégamment :

```
% dpkg -S `which iostat`
sysstat: /usr/bin/iostat
```

Le paquet souhaité est tout simplement **sysstat**. On notera que **dpkg-query** permet exactement la même chose. En revanche, que se passe-t-il si, par exemple, à la lecture d'une page de manuel, on nous conseille gentiment d'en lire une autre pour **perlreref(1)**. Voici :

```
% man perlreref
No manual entry for perlreref

% dpkg -S perlreref
dpkg : *perlreref* introuvable.
```

Vous venez brutalement de trouver la limite et comprendre par la même occasion à quoi servira **apt-file** :

```
% apt-file search  perlreref
perl-doc: /usr/share/man/man1/perlreref.1.gz
perl-doc: /usr/share/perl/5.10.0/pod/perlreref.pod
perl-doc: /usr/share/perl/5.10.1/pod/perlreref.pod
perl-doc: /usr/share/perl/5.12.0/pod/perlreref.pod
perl-doc-html: /usr/share/doc/perl-doc-html/html/perlreref.html
```

Et voilà, installons le paquet **perl-doc** et consultons la page de manuel tant désirée. **apt-file** n'a pas besoin d'installer un paquet pour en parcourir le contenu, il repose simplement sur le contenu des fichiers **Contents***. Afin de les récupérer avant la première utilisation et les garder à jour, il suffira d'un petit **apt-file update** en tant que super-utilisateur.

Mais **apt-file** va plus loin encore. Puisque les informations sont disponibles, il est également en mesure de vous afficher tous les fichiers contenus dans un paquet avec l'action **show** ou **list** :

```
% apt-file show apt-spy
apt-spy: /etc/apt-spy.conf
apt-spy: /usr/bin/apt-spy
apt-spy: /usr/share/doc/apt-spy/README.Debian
apt-spy: /usr/share/doc/apt-spy/TODO
apt-spy: /usr/share/doc/apt-spy/changelog.Debian.gz
apt-spy: /usr/share/doc/apt-spy/copyright
apt-spy: /usr/share/man/man5/apt-spy.conf.5.gz
apt-spy: /usr/share/man/man8/apt-spy.8.gz
apt-spy: /usr/share/menu/apt-spy
apt-spy: /var/cache/apt-spy/mirrors.txt
```

Remarquez que la recherche avec **apt-file** permettra également l'utilisation de caractères joker façon shell. Ainsi, si vous avec une incertitude sur le nom exact du fichier souhaité, l'outil devrait également vous être d'une certaine utilité.

gérer ses paquets

2 APT-RDEPENDS

Quel paquet nécessite l'installation de tel autre paquet ? Quelle suppression de bibliothèque va provoquer la désinstallation de paquets et combien sont concernés ? Quel paquet était une dépendance qui n'est maintenant plus nécessaire ? Cette dernière question peut trouver une réponse sous la forme d'un outil comme **deborhan** ou **wajig list-orphans**. Ces deux commandes permettant, en effet, de lister tous les paquets qui ne sont plus des dépendances de paquets encore installés :

```
% wajig list-orphans
djvulibre-desktop
libsuitesparse-3.1.0
libbeecrypt6
[...]
libevas-svn-05-engine-x
libzrtpcpp-0.9-0
```

On voit ici une liste de paquets qui pourraient, en principe, être désinstallés. Notez cependant que cette analyse ne se base que sur les dépendances entre paquets. En d'autres termes, si vous avez codé une application dans un coin, reposant sur la bibliothèque Evas, par exemple, la désinstallation pure et simple de la liste complète rendra votre programme défectueux. A utiliser avec sérieux donc et éviter les **aptitude purge `deborphan`** sans réfléchir.

wajig mériterait un article à lui tout seul car il fait bien plus que de lister les paquets qu'on appelle orphelins. Il est ainsi capable, contrairement à **deborphan**, de désinstaller les paquets en question avec la directive **purge-orphans**. La même remarque que précédemment s'applique. Mais il peut également vous apporter bon nombre d'informations et procéder à un grand nombre d'opérations courantes (ou non) dans le système APT : lister la taille des paquets installés, reconfigurer un paquet, convertir un paquet RPM en Debian, afficher les étapes d'un éventuel **dist-upgrade** et même rechercher pour vous un paquet non officiel sur apt-get.org.

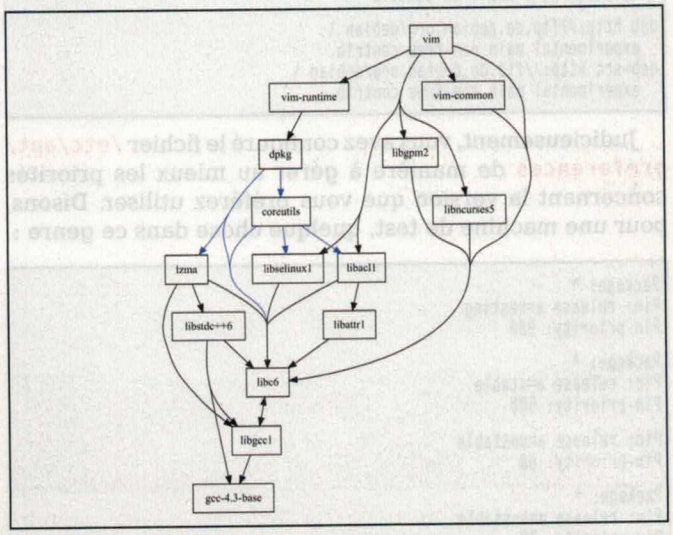
Mais ce qui nous intéresse véritablement ici, c'est de voir littéralement les dépendances entre les paquets. Pour cela, il faudra se tourner vers **apt-rdepends** comme ceci :

```
% apt-rdepends vim-tiny
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
vim-tiny
  Dépend: libc6 (>= 2.7-1)
  Dépend: libncurses5 (>= 5.6+20071006-3)
  Dépend: libselinux1 (>= 2.0.59)
  Dépend: vim-common (= 1:7.1.314-3+lenny2)
libc6
  Dépend: libgcc1
libgcc1
  Dépend: gcc-4.3-base (= 4.3.2-1.1)
  Dépend: libc6 (>= 2.7-1)
gcc-4.3-base
libncurses5
  Dépend: libc6 (>= 2.7-1)
libselinux1
  Dépend: libc6 (>= 2.7-1)
vim-common
  Dépend: libc6 (>= 2.7-1)
```

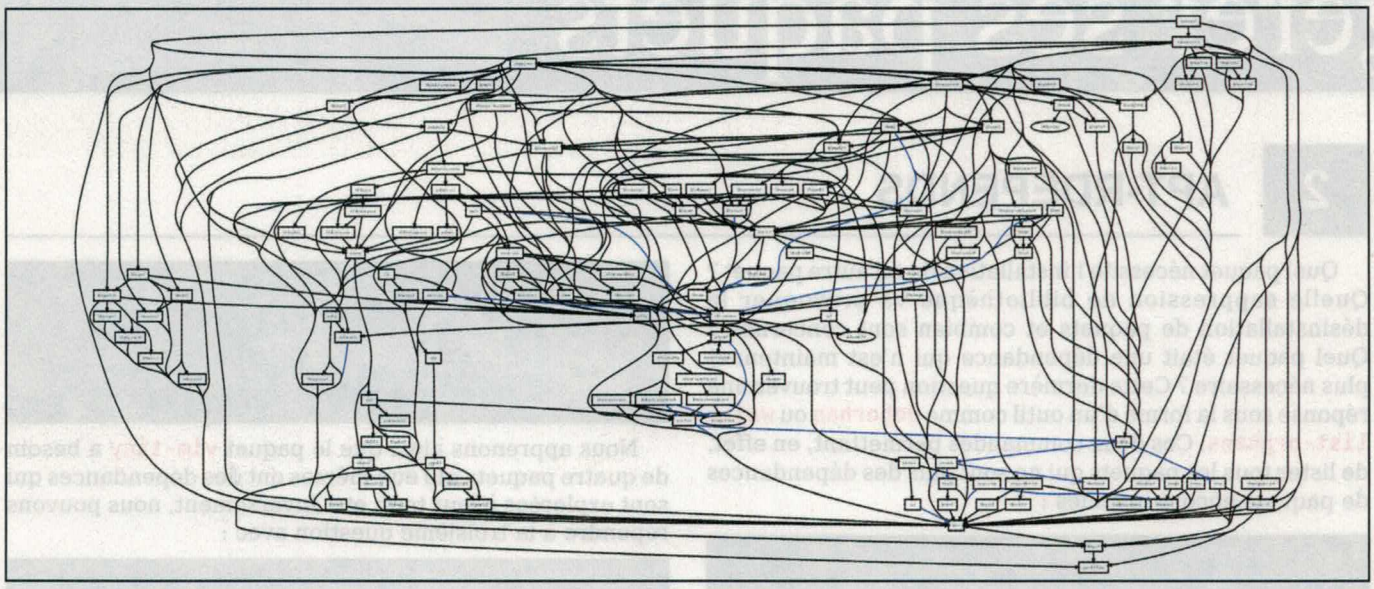
Nous apprenons ainsi que le paquet **vim-tiny** a besoin de quatre paquets qui eux-mêmes ont des dépendances qui sont explorées à leur tour, etc. Inversement, nous pouvons répondre à la troisième question avec :

```
% apt-rdepends -r liblayout-java
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
liblayout-java
  Reverse Dépend: libpentaho-reporting-flow-engine-java (0.9.2-3)
  Reverse Dépend: openoffice.org-report-builder
    (1:1.0.2+OOo2.4.1+dfsg-1+lenny3)
libpentaho-reporting-flow-engine-java
  Reverse Dépend: openoffice.org-report-builder
    (1:1.0.2+OOo2.4.1+dfsg-1+lenny3)
openoffice.org-report-builder
```

Ici, nous apprenons que le paquet **liblayout-java** est une dépendance de quelques autres paquets. Si vous trouvez que ceci n'est pas très visuel, nous pouvons demander une sortie dotty. Il s'agit d'un format structuré destiné à **dot**, un outil de la suite **graphviz**. Nous pouvons ainsi générer automatiquement un graphique hiérarchique représentant les dépendances d'un paquet avec **apt-rdepends -d vim | dot -Tsvg > vimdep.svg**. Nous obtenons ainsi un fichier au format SVG très facile à visualiser. Bien entendu, mieux vaut ne pas s'amuser à imprimer le fichier résultant de ce type de commandes pour le paquet **iceweasel** (du moins pas sur une feuille A4). Mais cela pourra faire un très beau poster...



Dépendances du paquet vim



Dépendances du paquet iceweasel (oui, c'est illisible, c'est juste pour faire peur)

3 APT-SHOW-VERSIONS

Il est probable que, comme de nombreux utilisateurs, vous ayez configuré dans votre **/etc/apt/sources.list** plusieurs sources de paquets, et ce pour plusieurs versions de la distribution. Exemple :

```
deb http://ftp.de.debian.org/debian/ \
  testing main non-free contrib
deb-src http://ftp.de.debian.org/debian/ \
  testing main non-free contrib

deb http://ftp.de.debian.org/debian/ \
  stable main non-free contrib
deb-src http://ftp.de.debian.org/debian/ \
  stable main non-free contrib

deb http://ftp.de.debian.org/debian/ \
  unstable main non-free contrib
deb-src http://ftp.de.debian.org/debian/ \
  unstable main non-free contrib

deb http://ftp.de.debian.org/debian \
  experimental main non-free contrib
deb-src http://ftp.de.debian.org/debian \
  experimental main non-free contrib
```

Judicieusement, vous avez configuré le fichier **/etc/apt/preferences** de manière à gérer au mieux les priorités concernant la version que vous préférez utiliser. Disons, pour une machine de test, quelque chose dans ce genre :

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

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

Pin: release a=unstable
Pin-priority: 80

Package: *
Pin: release a=unstable
Pin-priority: 70
```

Cependant, l'installation d'un paquet est tantôt délicate, vous vous permettez parfois de forcer les choses en spécifiant un numéro de version afin de disposer d'une application ou d'un outil qui n'existe pas encore dans la **testing** actuelle. Mais comment savoir si la version que vous cherchez est dans **testing**, **stable** ou **unstable** sans devoir jouer du **apt-cache show** et du **grep** ? Une réponse simple : **apt-show-version**.

Cet outil liste les versions disponibles des paquets et leur distribution en analysant le fichier état de **dpkg** et les listes d'APT. Il est à même d'afficher ainsi les versions installées et celles disponibles dans chaque distribution. Comme le précise la page de manuel, il « peut s'avérer utile si vous avez mélangé les environnements stable et testing » (oui, tout le monde, ou presque, le fait). Tout comme pour d'autres commandes **apt-***, il est nécessaire de procéder en deux temps. Premièrement, nous initialisons (ou mettons à jour) le cache avec **sudo apt-show-version -i**. Enfin, nous pouvons obtenir nos informations. Exemple :

```
% apt-show-versions -a vim
vim 2:7.2.330-1 install ok installed
vim 1:7.1.314-3+lenny2 stable   ftp.de.debian.org
vim 2:7.2.330-1        testing  ftp.de.debian.org
vim 2:7.2.330-1        unstable ftp.de.debian.org
```

Nous voyons ici que le paquet **vim** est installé en version **2:7.2.330-1**. C'est celle disponible dans **testing** et **unstable**. Les dernières versions de l'outil renseignent également sur le site utilisé. Ainsi, si vous utilisez plusieurs sources, **apt-show-version** vous affichera les informations par site et par distribution. Si vous avez changé les URL des dépôts dans votre **sources.list**, il est possible que les deux sites (ancien et nouveau) figurent dans la sortie de la commande, même après une initialisation. Ceci provient du fait que votre **/var/lib/apt/lists** a conservé un cache pour les précédentes URL. La sortie d'**api-show-version** peut alors devenir déroutante. Pour corriger le problème, utilisez simplement **apt-get –list-cleanup update**.

Mais ce n'est pas tout. Comme l'outil est en mesure de connaître toutes les versions, il est également capable de vous informer sur les mises à jours possibles avec :

```
% apt-show-versions -u
[...]
wmmixer/testing upgradeable from 1.5-6 to 1.5-11
wmsun/testing upgradeable from 1.03-18 to 1.03+1-2
wmtimer/testing upgradeable from 2.92-1 to 2.92-5
wmtz/testing upgradeable from 0.7-7 to 0.7-9
[...]
```

4 APT-LISTBUGS

Vos application, serveur, outils, éditeur préférés semblent avoir un bug et se comportent de manière étrange. Est-ce un problème connu ? Est-ce corrigé dans la version que vous allez installer ? Pour répondre à ces questions, le réflexe est normalement de sortir votre navigateur et de le pointer vers le BTS (*Bug Tracking System*) Debian. Mais qu'en est-il, par exemple, si vous êtes face à une machine serveur, bien entendu, exempte d'interface graphique ? Les plus courageux des lecteurs me diraient sans doute qu'il y a toujours **links** ou **elinks** (et tant pis pour les yeux). Mais il y a aussi **apt-listbugs**. Cet outil sans prétention est destiné à être utilisé avant toute mise à jour sur un système en production. Il permet, en effet, de lister les rapports de bugs pour un paquet donné.

Par défaut, **apt-listbugs** prend en argument la directive **list** suivie d'un nom de paquet. L'utilitaire se connecte alors à **bugs.debian.org** pour consulter la liste des rapports de bugs. Par défaut toujours, il affichera ainsi les bugs ouverts pour la dernière version du paquet avec la « gravité » **critical**, **grave** et **serious**. Vous pouvez cependant afficher d'autres niveaux comme **important**, **normal**, **minor** et **wishlist**, ou simplement tous avec **all**. Exemple :

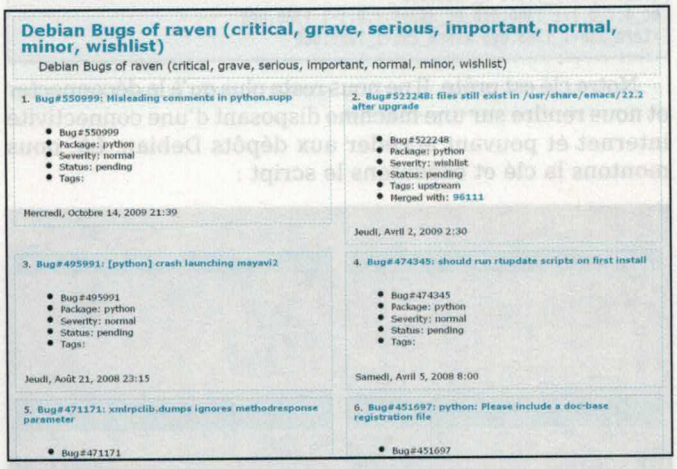
```
% apt-listbugs -s all -S pending list vim
[...]
Bogues de gravité important sur
   vim (2:7.2.330-1 -> ) <non corrigé>
#537299 - vim: potential data loss
    on saturated disk partitions
Bogues de gravité normal sur
   vim (2:7.2.330-1 -> ) <non corrigé>
#413358 - [vim] hangs in konsole
    with ERESTARTSYS and EAGAIN
#264023 - [vim] Numeric keypad keys
    not recognized when $TERM=screen
```

Vous pouvez également filtrer finement les informations sur les paquets en utilisant une expression régulière :

```
% apt-show-versions -r -p "^vim.*i(m|p)"
vim-runtime/testing uptodate 2:7.2.330-1
vim-scripts/testing upgradeable from 7-3 to 20091011
```

```
#566842 - vim: Don't use MODLIBS when
    building with python interpreter
    support and python2.6
[...]
```

Nous affichons ici la liste des bugs encore « ouverts » (*pending*) pour le paquet **vim**. Vous pouvez spécifier autant de noms de paquets que vous le souhaitez à la fin de la commande. Vous pouvez également utiliser **rss** en lieu et place de **list** afin d'obtenir une liste formatée. Vous pourrez ainsi inclure cette commande dans un script lancé régulièrement pour produire un flux RSS avec une simple redirection et un serveur HTTP léger, pour les paquets qui vous semblent importants.



Exemple de sortie RSS d'apt-listbugs affichée avec l'extension Sage de Firefox/Iceweasel

5 APT-ZIP

Tout le monde a Internet, du moins toutes les personnes en charge de serveurs... ou pas. Il existe quelques raisons pour lesquelles il n'est pas possible ou souhaitable de mettre à jour un système Debian GNU/Linux via une connexion FTP ou HTTP. Environnements hostiles, conditions particulières, systèmes embarqués ou paranoïa aiguë du sysadmin sont autant de bonnes raisons. D'autre part, bien que l'installation de base depuis un CD ou un DVD soit possible, la notion de mise à jour reste délicate.

Comment donc alors installer des nouveaux paquets ou les mettre à jour sans connexion internet sur le système visé ? Il suffit de faire appel à **apt-zip**. Ce paquet contient en réalité deux outils. D'une part **apt-zip-list**, destiné à créer une liste de paquets à installer ou mettre à jour et de l'autre **apt-list-inst**, pour l'installation effective. La première peut être utilisée par n'importe quel utilisateur alors que la seconde nécessite des droits administrateur. Ceci pourra, par la même occasion, entrer dans une politique de sécurité et de gestion des mises à jour pour des postes de travail délibérément non connectés au net.

Le principe est le suivant :

- On analyse les besoins de la plate-forme pour déterminer les paquets à installer ou à mettre à jour.

- On produit un script sur un support externe (clé USB) permettant le téléchargement depuis un autre système.
- On exécute le script en question pour charger la clé avec les archives Debian.
- On revient sur le système cible pour installer les paquets depuis le support amovible.

Avec **apt-zip**, ceci prend la forme suivante. Nous choisissons tout d'abord, sur le système non connecté, les paquets que nous devons installer ou mettre à jour avec **apt-zip-list**. Nous avons préparé le système de manière à supporter une clé USB via **/mnt/CLEF** (modification du **/etc/fstab**). Il suffit de spécifier alors le point de montage et les paquets visés (séparés par des virgules) :

```
% apt-zip-list --medium=/mnt/CLEF -p mc,xterm
Mounting /mnt/CLEF
The download size is 2615578 in 3 files.
UnMounting /mnt/CLEF
```

Ceci aura pour effet de créer sur **/mnt/CLEF**, entre autres choses, un fichier **fetch-script-wget-morgane**. Il s'agit, par défaut, d'un script destiné à fonctionner avec les systèmes Unix ou GNU/Linux, même non Debian. Celui-ci se chargera de récupérer nos paquets ainsi que les dépendances associées. On en retrouve ainsi mention à la fin du script :

```
libutempter0_1.1.5-2_i386.deb
mc_4.7.0.1-1_i386.deb mc_3%3a4.7.0.1-1_i386.deb
xterm_256-1_i386.deb xterm_256-1_i386.deb
```

Notre clé est prête. Il ne nous reste plus qu'à la déconnecter et nous rendre sur une machine disposant d'une connectivité internet et pouvant accéder aux dépôts Debian. Là, nous montons la clé et exécutons le script :

```
% cd //media/4A1E-D74D
% ./fetch-script-wget-morgane
Download will be of size: 2615578 in 3 files
2010-04-28 09:49:35 URL:http://ftp.fr.debian.org/
  debian/pool/main/libu/libutempter/
  libutempter0_1.1.5-2_i386.deb [7122/7122]
  -> "libutempter0_1.1.5-2_i386.deb" [1]
2010-04-28 09:49:40 URL:http://ftp.fr.debian.org/
  debian/pool/main/m/mc/
  mc_4.7.0.1-1_i386.deb [2095502/2095502]
  -> "mc_3%3a4.7.0.1-1_i386.deb" [1]
2010-04-28 09:49:41 URL:http://ftp.fr.debian.org/
  debian/pool/main/x/xterm/xterm_256-1_i386.deb
  [512954/512954] -> "xterm_256-1_i386.deb" [1]

% ls
apt-zip.options
fetch-script-wget-morgane
libutempter0_1.1.5-2_i386.deb
mc_3%3a4.7.0.1-1_i386.deb
partial
xterm_256-1_i386.deb
```

Comme vous pouvez le constater, les fichiers sont correctement récupérés et placés à la racine de la clé. Démontons celle-ci et retournons au système de base. Nous pouvons, mais cette fois avec les permissions **root**, lancer la seconde commande, **apt-zip-inst**, en spécifiant le point de montage :

```
% sudo apt-zip-inst --medium=/mnt/CLEF
Password:
Mounting /mnt/CLEF
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants
seront installés :
  libutempter0
Paquets suggérés :
  arj dbview odt2txt catdvi djvulibre-bin
  python-boto python-tz xfonts-cyrillic
Les NOUVEAUX paquets suivants seront installés :
  libutempter0
Les paquets suivants seront mis à jour :
  mc xterm
2 mis à jour, 1 nouvellement installés,
  0 à enlever et 1555 non mis à jour.
Il est nécessaire de prendre 0o/2 616ko
  dans les archives.
Après cette opération, 475ko d'espace disque
  supplémentaires seront utilisés.
Souhaitez-vous continuer [O/n] ?
```

L'outil se charge de tout, paquets demandés et dépendances sont automatiquement installés sans la moindre connexion au net. Le système est à jour.

Une option supplémentaire est disponible, c'est **-M** ou **--method**. Elle permet de spécifier une méthode de téléchargement pour l'hôte qui possède la connectivité nécessaire. Par défaut, la méthode utilisée est **wget**, qui est destinée aux systèmes Unix disposant d'un shell Bash et de la commande **wget**. Une seconde méthode est utilisable, c'est **wget-dos**. Celle-ci est destinée aux systèmes DOS/Windows et utilisera des fichiers **.bat**. Chose admirable, il vous suffira de disposer de **wget.exe** et de l'installer soit pour tout le système Windows, soit le copier en compagnie des scripts **.bat** sur la clé.

Que ce soit pour la méthode Unix ou Windows, il est possible d'opérer à partir d'une clé ou d'un point de montage quelconque. En d'autres termes, si votre machine sans Internet a accès à un disque partagé ou un NAS, il est possible de spécifier à **apt-zip-list** et **apt-zip-inst** l'option **--skip-mount**. Ainsi, le chemin désigné par **--medium** pourra être un simple répertoire. Les combinaisons sont ainsi nombreuses :

- Accès au répertoire partagé depuis la machine sans Internet pour produire les scripts/listes.
- Accès par la machine de téléchargement à un répertoire partagé sur la machine d'installation.
- Copie des fichiers de l'une ou l'autre machine.
- Archivage sous forme de tarball et transfert FTP ou même par mail.
- etc.

On notera qu'**apt-zip** se limite à cela et le lecteur attentif aura sans doute remarqué une petite limitation à contourner. En effet, l'analyse de la configuration et la sélection des paquets et des dépendances seront fonctions du contenu d'un répertoire **/var/lib/apt/lists** à jour. Et pour le mettre à jour avec **apt-get** ou **aptitude**, il faut le net ou jongler avec le contenu du répertoire. C'est là une amélioration possible d'**apt-zip**, qui pourrait alors se charger de procéder à l'action **update** comme il le fait pour l'installation de paquets. Quoi qu'il en soit, chapeau bas, **apt-zip** est incontestablement un petit bijou d'ingéniosité qui devrait en satisfaire plus d'un.

6 APT-SHOW-SOURCE

Avez-vous déjà remarqué que lorsqu'on installe un paquet source (voir article sur la reconstruction de paquets), la version traitée ne correspond pas toujours à celle installée de manière binaire. C'est parfaitement normal, les paquets sources sont gérés indépendamment des paquets binaires par Debian. Vous pouvez néanmoins forcer les choses, comme avec les paquets binaires, en faisant suivre son nom d'un symbole **=** et d'une version.

Ceci n'est ni un problème ni une limitation, bien au contraire. Si la version souhaitée n'existe pas en binaire pour votre distribution, vous pouvez le backporter. En d'autres termes, récupérer les sources et construire un paquet binaire vous-même, que vous installerez dans la foulée. Ainsi, il peut être intéressant de connaître les différences entre versions binaires installées et sans doute à jour, et versions sources disponibles.

C'est le travail d'**apt-show-source**. Exemple :

```
% apt-show-source -p vim-perl

Inst. Package (Version) | Newest Source Package (Version)
-----------------------------------------------------------
vim-perl (1:7.1.293-3)  | vim (2:7.2.330-1)
```

En lançant **apt-show-source** sans argument, vous listerez automatiquement tous les paquets dont la version source est supérieure à la version binaire installée. Notez que l'outil se base sur les informations présentes dans **/var/lib/apt/lists/***. Il convient donc de faire un **aptitude update** avant de l'utiliser.

7 APT-SPY

Mettre à jour et vite ! Voilà bien une phrase que l'on se dit souvent avec Debian. Vite, certes, mais vite comment ? Il existe un nombre très important de miroirs Debian. Trouver le plus rapide est souvent une affaire de chance couplée à un brin de bon sens (les serveurs les plus proches sont souvent les meilleurs). Cependant, rien ne vaut un petit comparatif fait sur mesure. C'est l'objet d'**apt-spy** chargé de vous composer un **sources.list** dépendant des benchs qu'il aura effectués pour vous. Première étape, mettre à jour la liste des miroirs avec :

```
% apt-spy update
Updating...
Grabbing file http://http.us.debian.org/debian/README.mirrors.txt...
Update complete. Exiting.
```

C'est simplement le fichier **README.mirrors.txt** du site officiel, tenu régulièrement à jour, qui sera copié en **/var/cache/apt-spy/mirrors.txt** (et non **/var/lib...** comme le précise la page de manuel). Il ne vous reste plus ensuite qu'à lancer le test. Ici, en spécifiant que seuls les serveurs européens nous intéressent et que nous voulons enregistrer le **sources.list** résultant sous un autre nom (avec **-o**) :

```
% apt-spy -a Europe -o /tmp/lists -d testing

SERVER: ftp.at.debian.org
Benchmarking FTP...
   Downloaded 6977925 bytes in 8.71 seconds
   Download speed: 782.33 kB/sec

SERVER: ftp.be.debian.org
Benchmarking FTP...
   Downloaded 6977925 bytes in 10.86 seconds
   Download speed: 627.63 kB/sec

SERVER: ftp.bg.debian.org
Benchmarking FTP...
   Downloaded 6978658 bytes in 10.99 seconds
   Download speed: 620.29 kB/sec

SERVER: linux.org.by
Benchmarking HTTP...
   Downloaded 2811128 bytes in 15.19 seconds
   Download speed: 180.74 kB/sec
```

Sans trop de surprises, c'est le serveur **ftp.fr.debian.org** qui remporte la victoire. Notez que la distribution spécifiée en argument de l'option **-d** est ajoutée, ainsi que les catégories **contrib** et **non-free**, mais en commentaires :

```
deb ftp://ftp.fr.debian.org/debian/ \
  testing main #contrib non-free
deb-src ftp://ftp.fr.debian.org/debian/ \
  testing main #contrib non-free
deb http://security.debian.org/ \
  testing/updates main
```

Je vais émettre ici un certain nombre de réserves quant à cet outil. Les essais ont provoqué quelques segfault pour le moins inquiétants. Je recommande donc fortement de bien spécifier un fichier de sortie avec **-o** car dans le cas contraire, c'est automatiquement (si vous êtes **root**) le fichier **/etc/apt/sources.list** qui sera modifié. Sachant qu'en cas de problème, voici ce qui pourrait s'y trouver inscrit, écrasant les données précédentes :

```
deb (null) testing main #contrib non-free
deb-src (null) testing main #contrib non-free
deb http://security.debian.org/ testing/updates main
```

Prudence, donc.

CONCLUSION

Nous venons d'opérer un petit tour d'horizon des outils gravitant autour d'**aptitude** et **apt-get**. Tous ne vous seront pas forcément utiles, mais ils complètent en partie des fonctionnalités manquantes ou souhaitables. Nous ne savons pas ce qui sera intégré dans les outils standards des prochaines versions, mais il y a fort à parier que les meilleures idées seront ajoutées tôt ou tard, comme c'était le cas pour les tags qui maintenant s'affichent avec un simple **apt-cache show**. Bien entendu, si une fonctionnalité est absente et que vous l'avez (ou souhaitez) implémentée, rien ne vous en empêche, sauf peut-être un cruel manque de temps. Dans le cas contraire, participez et partagez cela avec les autres utilisateurs de Debian. Cela fera un **atp-*** de plus ! ■

Reconstruire et adapter les paquets

Les développeurs et responsables de paquets Debian font extrêmement bien leur travail. Cependant, il arrive qu'en tant qu'administrateur ou simple utilisateur, les choix faits au moment de la compilation d'une application ou de la construction d'un paquet ne nous conviennent pas. Les exemples ne manquent pas et on est parfois étonné des différences pouvant exister entre les documentations en ligne et les options effectivement disponibles pour un outil en ligne de commandes. Qu'à cela ne tienne, il nous suffit de reconstruire un nouveau paquet.

Les options ou fonctionnalités manquantes ne représentent pas la seule motivation à disposer de ses propres versions de certains paquets. En effet, lorsqu'on cherche l'optimisation maximale, la compilation peut également être un point important dans la recherche de performances. Les options de compilation, dont les niveaux d'optimisation offerts par GCC, sont un exemple très basique. Certains paquets et outils permettent également d'activer ou non le support SMP. Alors qu'aujourd'hui il n'est presque plus possible d'acquérir une machine sans processeur double ou quadruple cœur, il serait dommage de se passer de ce type de fonctionnalités.

Attention cependant à bien évaluer le gain et le résultat qu'il est possible d'obtenir. Passer un temps important à chercher et activer la moindre option de compilation, avec tous les problèmes que cela peut déclencher, vaut-il les quelques cycles d'horloge que vous gagnerez au final ? Bien entendu, à cette question, je ne peux répondre à votre place. Dans le cadre de cet article, ce qui nous intéresse avant tout est d'étudier la manière de reconstruire un paquet dans un but précis, et bien plus important : utiliser des versions binaires de ses outils et applications préférés, sans pour autant détruire le système de gestion de paquets et la structure de la distribution.

Gardez bien cela en tête et faites-en la fondation de votre politique de gestion du système : ne jamais (plus) installer « à la hache » une application dans votre GNU/Linux. C'est contraignant, certes, mais vous verrez qu'au final, vous y gagnerez en productivité et en cohérence. Oui, cela veut aussi dire que les applications propriétaires ou d'autres choses « clé en main » s'installant chaotiquement dans `/opt` ou `/usr/local` sont à prohiber ou à convertir proprement. Je pense en particulier aux kits de développement (SDK) de certains systèmes embarqués, par exemple, qui, une fois désarchivés, vous proposent tout bonnement de lancer un `./install.sh`. Le résultat est souvent catastrophique et on s'en aperçoit bien malheureusement au moment où l'on cherche à désinstaller la chose en question (non, il n'y a quasiment jamais de `./uninstall.sh`).

Le fait de reconstruire un paquet présente également un autre avantage. Il arrive parfois qu'un paquet au format Debian n'existe pas pour votre distribution. Soit il n'est présent que pour une version plus récente de la votre, soit il provient d'une distribution dérivée de Debian comme Ubuntu. Dans le premier cas, il est possible qu'une version backportée existe. Quelqu'un, sans doute un développeur Debian, a créé une version du paquet pour une édition plus ancienne de la distribution et l'a rendue disponible. C'est le cas, par exemple, du paquet `libapache-mod-security` backporté de *Etch* vers *Lenny* et disponible via http://www.backports.org/debian.

Dans le second cas, il suffira parfois de récupérer les éléments sources du paquet (voir plus loin dans l'article), d'y faire quelques changements et de construire un paquet binaire pour le rendre disponible pour sa distribution et l'installer localement.

1 DPKG-BUILDPACKAGE, CLASSIQUE ET EFFICACE

La reconstruction de paquets n'est pas un processus très complexe d'un point de vue de l'administrateur ou de l'utilisateur. Il n'en va, bien entendu, pas de même en ce qui concerne les mécanismes internes de génération et les scripts utilisés. Mais ici, ce n'est (presque) pas notre problème.

Si votre installation Debian GNU/Linux est correctement configurée, vous trouverez dans votre fichier `/etc/apt/sources.list` (ou les fichiers présents dans `/etc/apt/sources.list.d`) des lignes définissant des sources pour les paquets binaires, mais aussi des sources `deb-src`.

en fonction de ses besoins

Ces dernières permettent au système APT de déterminer l'emplacement des différents composants d'un paquet source. Un paquet source en tant que tel n'a pas d'existence réelle, il est composé de :

- 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. C'est tout simplement le *tarball* tel qu'il existe sur les sites FTP officiels. La politique de Debian à ce niveau n'est pas de modifier directement les sources, si nécessaire, d'un outil ou d'une application, mais de reposer sur une version donnée, originale, et de patcher les fichiers ensuite.
- 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). Ce patch ajoutera également tous les fichiers nécessaires au processus de construction du paquet, stockés dans un répertoire **debian**. En d'autres termes, ce patch magique transforme les sources officielles en sources Debian.
- 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 (champ **Build-Depends:**) ou encore les différentes sommes de contrôle permettant de vérifier qu'on travaille effectivement sur le bon tarball en version amont. C'est également ce fichier qui indiquera le nom du responsable du paquet, la version concernée, la *homepage* du projet amont, ou encore le type d'architecture matérielle visée. Ce fichier est signé avec GnuPG.

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

Nous allons ici utiliser un exemple concret pour illustrer nos propos. Le but est de construire, à partir des sources, le paquet contenant **bc** (la calculatrice en ligne de commandes). **bc** n'est pas une application très riche en termes de dépendances, mais elle constitue un exemple relativement simple pour une démonstration rapide. Ce qui va être détaillé par la suite est également valable pour reconstruire de plus grosses applications.

Compiler une application avec un système Debian nécessite deux éléments :

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

Note : vérifier la signature d'un fichier

Le fichier de description **.dsc** est signé par le responsable du paquet pour en assurer l'authenticité. Les outils de construction de paquets vérifient ce genre de choses, mais il est toujours intéressant de savoir comment procéder manuellement. Ainsi, si nous prenons le fichier **.dsc** du paquet correspondant à l'outil de calcul **bc**, nous procédons comme suit :

```
% gpg --verify  bc_1.06.95-2.dsc
gpg: Signature faite le jeu. 15 oct. 2009 19:22:38
  CEST avec la clé RSA ID B4FBDAA5
gpg: Impossible de vérifier la signature: clé publique non trouvée

% gpg --recv-keys B4FBDAA5
gpg: requête de la clé B4FBDAA5 du serveur hkp wwwkeys.eu.pgp.net
gpg: clé B4FBDAA5: clé publique
   " John G. Hasler <jhasler@newsguy.com> " importée
gpg: 3 marginale(s) nécessaires,
  1 complète(s) nécessaires, modèle
de confiance classic
gpg: profondeur: 0  valide:   1  signé:   4
confiance: 0-. 0g. 0n. 0m. 0f. 1u
gpg: profondeur: 1  valide:   4  signé:  36
confiance: 3-. 1g. 0n. 0m. 0f. 0u
gpg:        Quantité totale traitée: 1
gpg:                     importée: 1  (RSA: 1)

% gpg --verify  bc_1.06.95-2.dsc
gpg: Signature faite le jeu. 15 oct. 2009 19:22:38
  CEST avec la clé RSA ID B4FBDAA5
gpg: Bonne signature de " John G. Hasler <jhasler@newsguy.com> "
gpg:                 alias " John G. Hasler <john@dhh.gt.org> "
gpg:                 alias " John Hasler <jhasler@newsguy.com> "
gpg:                 alias " John G. Hasler <jhasler@debian.org> "
gpg: ATTENTION: Cette clé n'est pas certifiée avec une signature de confiance !
gpg:          Rien ne dit que la signature appartient à son propriétaire.
Empreinte de clé principale:  4274595ED5466A250D8E847D73401E35B4FBDAA5
```

La clé publique du développeur Debian n'étant pas dans notre trousseau, nous la récupérons sur un serveur de clé (www.keys.eu.pgp.net en l'occurrence) et nous procédons à la vérification qui réussit. Le fichier est bien signé par la clé publique, mais nous n'avons aucun chemin nous assurant qu'elle appartient effectivement à John G. Hasler.

La première étape consiste donc à installer le méta-paquet **build-essential**. Il s'agit d'un paquet ne contenant pas grand-chose, mais dont l'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**). Ce sont les dépendances de **build-essential** qui provoquent l'installation du reste des paquets. C'est un raccourci pour une installation rapide des éléments vitaux dont nous avons besoin.

La seconde étape consiste à installer les dépendances nécessaires à la reconstruction du paquet binaire qui nous

intéresse. Tout comme pour les dépendances entre paquets binaires, 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
Package: bc
Binary: bc, dc
Version: 1.06.95-2
Priority: standard
Section: math
Maintainer: John G. Hasler <jhasler@debian.org>
Build-Depends: bison, debhelper (>= 7), file, flex,
  libreadline-dev | libreadline6-dev | libreadline5-dev,
  texi2html, texinfo
Architecture: any
[...]
```

Nous avons ici plusieurs informations importantes qui se trouvent également dans le fichier **.dsc** du paquet source. Nous apprenons par exemple que le paquet source **bc** produira deux paquets binaires **bc** et **dc**. Nous avons la version de l'outil et le nom du responsable du paquet. Enfin, **Build-Depends:** nous indique plusieurs choses. Nous avons besoin, pour construire **bc** et **dc** de :

- **bison** ;
- **debhelper** en version supérieure ou égale à 7) ;
- **file** et **flex** ;
- **libreadline-dev** ou **libreadline6-dev** ou **libreadline5-dev** (n'importe lequel de ces paquets fera l'affaire) ;
- et enfin **texi2html** et **texinfo**.

Pour installer automatiquement ces paquets ou, du moins, les paquets qui ne le sont pas encore, il suffit d'utiliser **apt-get build-dep** suivi du nom du paquet que nous voulons construire. Enfin, il ne nous reste plus qu'à télécharger 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
Lecture des informations d'état... Fait
Nécessité de prendre 373ko dans les sources.
Réception de : 1 http://ftp.fr.debian.org
  testing/main bc 1.06.95-2 (dsc) [1 659B]
Réception de : 2 http://ftp.fr.debian.org
  testing/main bc 1.06.95-2 (tar) [361kB]
Réception de : 3 http://ftp.fr.debian.org
  testing/main bc 1.06.95-2 (diff) [11,1kB]
373ko réceptionnés en 0s (494ko/s)
dpkg-source: info: extraction de bc dans bc-1.06.95
dpkg-source: info: extraction de bc_1.06.95.orig.tar.gz
dpkg-source: info: mise en place de bc_1.06.95-2.diff.gz
dpkg-source: info: upstream files that have been modified:
bc-1.06.95/bc/main.c
bc-1.06.95/bc/y.tab.h
bc-1.06.95/doc/bc.info
bc-1.06.95/doc/bc.texi
bc-1.06.95/doc/dc.1
bc-1.06.95/doc/dc.info
bc-1.06.95/doc/dc.texi
```

Si vous obtenez un message d'erreur vous signalant que la signature de la description (**.dsc**) ne peut être vérifiée, c'est tout simplement qu'il vous manque la signature du développeur Debian en question ou une partie du chemin pour la vérifier. Habituellement, et si votre système est parfaitement à jour, le problème provient sans doute de l'absence du paquet **debian-keyring**. Ce paquet contient quelques 30 Mo de données. Il s'agit de toutes les clés publiques des développeurs et mainteneurs Debian.

apt-get source nous informe sur les actions qu'il mène :

- récupération des trois fameux éléments constituant un paquet source ;
- décompression des archives ;
- application du patch.

Dans les plus récentes versions de la commande, la liste des fichiers amonts modifiés est affichée. Ici, il s'agit principalement de la documentation info et man, mais également de deux fichiers sources. Ceci est très courant. Debian, en effet, apporte des corrections très régulièrement aux sources officielles. C'est d'ailleurs ce qui explique que le navigateur Firefox s'appelle Iceweasel dans la distribution, puisque le terme « Firefox » est déposé et ne peut désigner qu'un logiciel dont les sources sont validées par la fondation. Comprenez bien là qu'il ne s'agit pas d'un problème de licence ou de logiciel libre, mais de celle découlant de l'utilisation d'une marque déposée. Rien de plus.

Remarquez que **apt-get source** fait un certain nombre de manipulations à votre place. S'il vous arrivait d'endommager le contenu du répertoire des sources Debian, il ne sera pas nécessaire de refaire appel à **apt-get**. Vous pourrez tout simplement supprimer le répertoire en question, puis utiliser :

```
% dpkg-source -x bc_1.06.95-2.dsc
dpkg-source: info: extraction
  de bc dans bc-1.06.95
dpkg-source: info: extraction
   de bc_1.06.95.orig.tar.gz
dpkg-source: info: mise en place
  de bc_1.06.95-2.diff.gz
dpkg-source: info: upstream files
  that have been modified:
bc-1.06.95/bc/main.c
bc-1.06.95/bc/y.tab.h
bc-1.06.95/doc/bc.info
bc-1.06.95/doc/bc.texi
bc-1.06.95/doc/dc.1
bc-1.06.95/doc/dc.info
bc-1.06.95/doc/dc.texi
```

Ceci est également valable dans le cas où vous disposez des fichiers par un autre moyen qu'Internet ou s'il s'agit d'un paquet non officiel. En récupérant les trois éléments (les fichiers **.diff.gz** et **.dsc** d'une part et le tarball amont de l'autre) depuis différentes sources, vous pouvez obtenir le même résultat qu'avec **apt-get source**.

Voici l'état de notre répertoire de travail :

```
% ls -F
bc-1.06.95/
bc_1.06.95-2.diff.gz
bc_1.06.95-2.dsc
bc_1.06.95.orig.tar.gz
```

La construction à proprement parler se fera via la commande **dpkg-buildpackage**. Il suffit d'entrer dans le répertoire des sources et d'utiliser :

```
% cd bc-1.06.95/
% dpkg-buildpackage -b
dpkg-buildpackage : définir CFLAGS
  à la valeur par défaut : -g -O2
dpkg-buildpackage : définir CPPFLAGS
  à la valeur par défaut :
dpkg-buildpackage : définir LDFLAGS
  à la valeur par défaut :
dpkg-buildpackage : définir FFLAGS
  à la valeur par défaut : -g -O2
dpkg-buildpackage : définir CXXFLAGS
  à la valeur par défaut : -g -O2
dpkg-buildpackage: paquet source bc
dpkg-buildpackage: version source 1.06.95-2
dpkg-buildpackage: source changé par
  John G. Hasler <jhasler@debian.org>
dpkg-buildpackage: architecture hôte i386
fakeroot debian/rules clean
dh_testdir
dh_testroot
rm -f build-stamp install-stamp
[ ! -f Makefile ] || /usr/bin/make distclean
dh_clean    config.log confdefs.h doc/dc.html
debian/rules build
dh_testdir
./configure --prefix=/usr --with-readline
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
[...]
```

L'option **-b** nous permet de limiter le fonctionnement de la commande à la seule construction d'un paquet binaire dans le répertoire parent. Ici, nous retrouvons **bc_1.06.95-2_i386.deb** et **dc_1.06.95-2_i386.deb** à cet endroit. En l'absence de l'option **-b**, nous produirons également le fichier **.dsc**, un tarball source et un fichier **.changes** indiquant les informations générales sur le paquet et les derniers changements effectués. La reconstruction de ces fichiers dans le répertoire parent permettra à d'autres personnes de recréer le répertoire source Debian tel qu'il est sur votre disque dur tout en conservant les principes de Debian de séparation entre sources amont et modifications Debian.

En fin de processus, nous avons :

```
[...]
dh_installdeb -pdc
dh_shlibdeps -pdc
dh_gencontrol -pdc
dh_md5sums -pdc
dh_builddeb -pdc
dpkg-deb : construction du paquet " dc "
   dans " ../dc_1.06.95-2_i386.deb ".
dpkg-genchanges -b >../bc_1.06.95-2_i386.changes
dpkg-genchanges: envoi d'un binaire - aucune inclusion de code source
signfile bc_1.06.95-2_i386.changes
gpg: " John G. Hasler <jhasler@debian.org> " a été ignoré:
  la clé secrète n'est pas disponible
gpg: [stdin]: clearsign failed:
  la clé secrète n'est pas disponible

dpkg-buildpackage: envoi d'un binaire seulement
  (aucune inclusion de code source)
dpkg-buildpackage: avertissement:
  Échec de signature du fichier .changes
```

Divers outils propres à la distribution Debian sont utilisés pour vérifier l'état des données, nettoyer les sources ou encore configurer les documentations. Ensuite, la signature

Note : Fonctionnement de LD_PRELOAD

La variable **LD_PRELOAD** demande au *dynamic linker* (alias le *dld*) de charger une bibliothèque partagée pour intercepter des appels de fonctions. En d'autres termes, si nous avons un programme qui utilise la fonction **getpid()**, par exemple, nous pouvons écrire une bibliothèque partagée comme ceci :

```
#include <sys/syscall.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

pid_t getpid(void) {
  printf("Hello, world!\n");
  return syscall(SYS_getpid);
}
```

Nous affichons ici un message avec **printf()**, puis nous retournons le résultat de l'appel système correspondant. En chargeant notre bibliothèque, nous interposons notre **getpid** entre le programme et la libc où est implémenté le **getpid** original. Nous écrivons un simple programme exécutant à la fois directement l'appel système puis la fonction de la bibliothèque C :

```
#include  <syscall.h>
#include  <unistd.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <sys/types.h>

int main(int argc, char **argv) {
  long ID1, ID2;

  /* appel systeme */
  ID1 = syscall(SYS_getpid);
  printf ("syscall(SYS_getpid)=%ld\n", ID1);

  /* appel libc */
  ID2 = getpid();
  printf ("getpid()=%ld\n", ID2);

  return(EXIT_SUCCESS);
}
```

Après compilation des deux éléments, nous pouvons tester :

```
% gcc -Wall -fPIC -shared -o getpid.so getpid.c
% gcc -Wall programme.c -o programme

% ./programme
syscall(SYS_getpid)=17911
getpid()=17911

% LD_PRELOAD=./getpid.so ./programme
syscall(SYS_getpid)=17477
Hello, world!
getpid()=17477
```

Le message **Hello, world!** est inséré juste avant la ligne retournant le numéro de processus. Notre interception fonctionne parfaitement. Bien entendu, il en va de même avec n'importe quel programme auquel nous n'avons pas touché. Pour preuve, un **echo $$** dans un shell Bash lancé pour l'occasion provoquera exactement le même comportement :

```
% LD_PRELOAD=./getpid.so bash -c 'echo $$'
Hello, world!
16347
```

des fichiers doit avoir lieu. Comme nous n'avons absolument rien modifié dans la configuration du paquet source, une tentative de signature aura lieu sous l'identité du responsable du paquet. Sans surprise, comme vous n'êtes pas John G. Hasler, la clé privée permettant l'opération n'est pas disponible. Il en résultera, toutefois, la production des paquets binaires.

Chose invisible ici, **dpkg-buildpackage** a utilisé par défaut un outil spécifique qu'il est possible de préciser via une option **-r**. L'argument est généralement **fakeroot** (**-rfakeroot** étant utilisé d'office). **fakeroot** permet de construire le paquet sans être super-utilisateur (**root**). En effet, le paquet binaire contient une arborescence qui sera copiée à la racine du système de fichiers lors de l'installation. Il faut donc, si nécessaire, pouvoir donner aux fichiers de cette arborescence les privilèges et « appartenances » à **root**. C'est précisément ce que permet de faire **fakeroot** de manière simulée. Notez bien que **fakeroot** n'a rien à voir avec un **chroot**, c'est de l'utilisateur UID 0 dont on parle ici.

fakeroot utilise la variable d'environnement **LD_PRELOAD** permettant ainsi de forcer le chargement de **libfakeroot.so** et d'intercepter les appels aux fonctions qui nous intéressent. L'idée est ici de modifier le comportement de commandes comme **getuid**, **chown**, **chmod**, ou encore **stat** afin de créer un environnement factice. Utiliser **sudo** peut également être une solution fonctionnelle, ou encore **fakeroot-ng**, une évolution du **fakeroot** original utilisant **ptrace** et l'interception de l'appel système **open** en lieu et place de la technique reposant sur **LD_PRELOAD**. Le principal avantage est de pouvoir fonctionner avec des binaires compilés statiquement et n'utilisant donc pas d'appel système au travers le la glibc.

Comme dit plus haut, 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 ayant l'extension **.deb** :

```
% sudo dpkg -i ./bc_1.06.95-2_i386.deb
(Lecture de la base de données...
  184405 fichiers et répertoires déjà installés.)
Préparation du remplacement de bc 1.06.94-1
  (en utilisant ./bc_1.06.95-2_i386.deb) ...
Dépaquetage de la mise à jour de bc ...
Paramétrage de bc (1.06.95-2) ...
Traitement des actions différées
  (" triggers ") pour " install-info "...
Traitement des actions différées
  (" triggers ") pour " man-db "...
```

2 SIMPLIFIONS AVEC APT-SRC

La reconstruction d'un paquet binaire à partir des sources Debian via la méthode qui vient d'être décrite n'a rien de bien complexe. En revanche, elle peut vite devenir lassante. Pire encore, si vous prenez l'habitude de recompiler des paquets, il est également probable que vous risquiez de jouer du **apt-get source** un peu partout dans votre arborescence. En conséquence, vous allez vous retrouver avec des sources un peu partout ou, dans le meilleur des cas, avec un répertoire plein de fichiers **.deb**, **.dsc** et **.changes**.

Pour vous 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 pas forcément très organisé ou soigneux dans son rangement. Comme vous l'aurez compris en lisant ce qui vient d'être détaillé, l'installation des sources d'un paquet n'est pas dépendante d'une arborescence standardisée comme pour les paquets binaires qui utilisent **/var/cache/apt** et **/var/lib/apt**.

Note

Certaines actions de l'outil **apt-src** nécessitent le passage en **root**. L'utilitaire prend cela automatiquement en charge via **sudo** et vous demandera votre mot de passe utilisateur. Bien entendu, ceci à condition que l'utilisateur en question soit effectivement dans la liste des *sudoers*.

apt-src automatise toutes les étapes décrites plus haut. La commande **apt-src install bc**, utilisée après un **apt-src update**, donnera les mêmes résultats qu'un **apt-get source bc**, installant dans le répertoire courant une arborescence de sources patchées prêtes à être compilées. La principale différence dans ce cas tient dans le fait que **apt-src** arrive à se souvenir de ce qu'il fait. Ainsi, en nous trouvant au fin fond de l'arborescence d'un disque, les sources peuvent être retrouvées rapidement :

```
% apt-src list
i   bc  1.06.95-2  /mnt/mega10/DEB/S/bc-1.06.95
```

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 celle des paquets **-dev** nécessaires à sa construction, ou encore des outils de compilation adéquats.

Après utilisation de l'action **install**, nous pouvons utiliser la commande **apt-src build bc** pour obtenir nos deux paquets **bc** et **dc** que nous pourrons installer dans la foulée. Là encore, l'utilitaire fonctionne de manière indépendante de l'emplacement où l'on se trouve. Ainsi, en sautant dans **/tmp** et en utilisant cette commande, nous obtenons :

```
% apt-src build bc
I: Building in /mnt/mega10/DEB/S/bc-1.06.95 ..
dpkg-buildpackage : définir CFLAGS
  à la valeur par défaut : -g -O2
dpkg-buildpackage : définir CPPFLAGS
  à la valeur par défaut :
```

```
dpkg-buildpackage : définir LDFLAGS
  à la valeur par défaut :
dpkg-buildpackage : définir FFLAGS
  à la valeur par défaut : -g -O2
dpkg-buildpackage : définir CXXFLAGS
  à la valeur par défaut : -g -O2
dpkg-buildpackage: paquet source bc
dpkg-buildpackage: version source 1.06.95-2
dpkg-buildpackage: source changé par
  John G. Hasler <jhasler@debian.org>
dpkg-buildpackage: architecture hôte i386
[...]
dpkg-buildpackage: envoi d'un binaire seulement
  (aucune inclusion de code source)
I: Successfully built in /mnt/mega10/DEB/S/bc-1.06.95
```

On retrouve bien alors nos deux **.deb** dans **/mnt/mega10/DEB/S/** comme on peut le supposer. Il est également possible d'utiliser l'option **-i** (**apt-src -i build**) permettant d'installer automatiquement le ou les paquets construits. Si, comme dans le cas des sources de **bc**, cette option est utilisée avec des paquets sources produisant plusieurs paquets binaires, on provoquera l'installation de tous les paquets binaires générés (ici **bc**, mais aussi **dc**).

Vous l'aurez compris, le principal avantage ce cet outil est le suivi des sources installées. On peut ainsi les lister, les mettre à jour, les nettoyer (**make clean**) ou en récupérer la version ou le chemin (**location**). Bien entendu, il est également possible de supprimer automatiquement une arborescence source avec un **apt-src remove** suivi du nom du paquet. Notez que tous les fichiers seront supprimés et non pas seulement les éléments sources. Ceci signifie que les paquets binaires construits disparaîtront en même temps que le reste.

Enfin, on notera la possibilité de renseigner **apt-src** sur la présence d'une arborescence installée indépendamment avec l'action **import**. **apt-src** devient rapidement plus agréable à utiliser que le couple **apt-get source** et **dpkg-buildpackage**. Cependant, si vous êtes un accro de la reconstruction, il y a peut-être mieux...

3 TOUJOURS PLUS SIMPLE ET PLUS AUTOMATIQUE : APT-BUILD

Alors que **apt-src** permet toujours « 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**. Mieux vaut alors prévoir d'avoir suffisamment d'espace si **/var** est placé sur un système de fichiers spécifique. Notez toutefois que 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. Vous pouvez reconfigurer ensuite le paquet à souhait avec un **sudo dpkg-reconfigure apt-build**.

Ainsi, vous serez amené à préciser (ou laisser les valeurs par 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 **/etc/apt/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/**. Vous pouvez donc répondre négativement pour ensuite placer la ligne dans le fichier qui vous conviendra.
- Les options et arguments à utiliser avec le compilateur 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.

Ici, notre architecture système repose sur un Intel QuadCore Q6600. Le système de configuration nous propose donc **prescott**. Comme le précise la fenêtre (**dialog**) qui se présente à l'écran, si votre architecture n'est pas listée, vous devrez en choisir une quelconque, puis éditer le fichier **/etc/apt/apt-build.conf** manuellement et faire un rapport de bug de sévérité *wishlist*.

Le fichier de configuration reflétera vos choix comme ceci :

```
build-dir = /var/cache/apt-build/build
repository-dir = /var/cache/apt-build/repository
Olevel = -O2
mtune = -mtune=prescott
options = " -march=prescott -O2 -pipe -fomit-frame-pointer"
make_options = " "
```

Le choix **prescott** correspond en réalité à l'option **-mtune** passée à **gcc** et permet 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.

Dixit la documentation de 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.

Nous avons également, dans le fichier, 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 jeux d'instructions. SSSE3, par exemple, est spécifique aux Intel Xeon et Intel Core 2. 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 par défaut.

L'important est de bien comprendre la différence entre ces deux options :

- **-march** : permet de produire un binaire uniquement pour ce type de processeur.
- **-mtune** : permet d'optimiser le binaire pour un modèle de processeur d'une architecture donnée. Le binaire pourra toujours s'exécuter sur un i486.
- **-mcpu** : obsolète pour les architectures x86.

On peut ainsi 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 Atlhon XP (**-mtune=athlon-xp**). Il en résultera un binaire optimisé pour l'Atlhon 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**.

Si vous doutez de la pertinence de ces options ou encore de l'intérêt de recompiler vos binaires, voici un petit exemple reprenant le programme **hello world** utilisé précédemment. Nous utiliserons ici l'option **-v** permettant d'en apprendre un peu plus sur ce que fait **gcc**. Commençons par compiler notre simplissime programme sans rien spécifier :

```
% gcc -v -Wall programme.c -o programme
[...]
Thread model: posix
gcc version 4.4.3 20100108 (prerelease) (Debian 4.4.2-9)
COLLECT_GCC_OPTIONS='-v' '-Wall' '-o' 'programme'
'-mtune=generic' '-march=i486'
/usr/lib/gcc/i486-linux-gnu/4.4.3/cc1 -quiet -v
programme.c -quiet -dumpbase programme.c
-mtune=generic -march=i486 -auxbase programme
-Wall -version -o /tmp/ccU9pq4d.s
[...]
```

Par défaut, nous n'avons tout simplement pas d'optimisation et une compilation pour i486. Voyons maintenant ce que dit **gcc** en lui passant **native** comme valeur de **-march**, ce qui a pour effet de le laisser détecter le processeur en présence :

```
% gcc -v -Wall programme.c -o programme -march=native
[...]
Thread model: posix
gcc version 4.4.3 20100108 (prerelease) (Debian 4.4.2-9)
COLLECT_GCC_OPTIONS='-v' '-Wall' '-o' 'programme'
/usr/lib/gcc/i486-linux-gnu/4.4.3/cc1 -quiet -v
programme.c -march=core2 -mcx16 -msahf
--param l1-cache-size=32 --param l1-cache-line-size=64
--param l2-cache-size=4096 -mtune=core2 -quiet
-dumpbase programme.c -auxbase programme -Wall
-version -o /tmp/ccyr8A0e.s
[...]
```

Nous avons ici un **-march=core2** et un **-mtune** identique. Comprenez-vous alors qu'il devient très intéressant de préciser ces options ? **native** est généralement un bon choix. Cependant, pour connaître les bonnes options, commencez par déterminer avec exactitude le modèle de processeur de votre machine avec :

```
% cat /proc/cpuinfo
[...]
processor  : 0
vendor_id  : GenuineIntel
cpu family : 6
model      : 15
model name : Intel(R) Core(TM)2 Quad CPU Q6600  @ 2.40GHz
stepping   : 11
cpu MHz    : 2304.022
cache size : 4096 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, comme Gentoo. De là, on tire la ligne d'options idéale pour **gcc**.

Votre **apt-build** est maintenant 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
Lecture des informations d'état... Fait
[...]
-----> Updating package lists <-----
[...]
3 538o réceptionnés en 4s (750o/s)
Lecture des listes de paquets... Fait
-----> Downloading source bc (1.06.95-2) <-----
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
[...]
373ko réceptionnés en 0s (432ko/s)
dpkg-source: info: extraction de bc dans bc-1.06.95
dpkg-source: info: extraction de bc_1.06.95.orig.tar.gz
dpkg-source: info: mise en place de bc_1.06.95-2.diff.gz
dpkg-source: info: upstream files that have been modified:
bc-1.06.95/bc/main.c
bc-1.06.95/bc/y.tab.h
bc-1.06.95/doc/bc.info
bc-1.06.95/doc/bc.texi
bc-1.06.95/doc/dc.1
bc-1.06.95/doc/dc.info
bc-1.06.95/doc/dc.texi
-----> Building bc <-----
dpkg-buildpackage : définir CFLAGS à la valeur par défaut : -g -O2
dpkg-buildpackage : définir CPPFLAGS à la valeur par défaut :
dpkg-buildpackage : définir LDFLAGS à la valeur par défaut :
dpkg-buildpackage : définir FFLAGS à la valeur par défaut : -g -O2
dpkg-buildpackage : définir CXXFLAGS à la valeur par défaut : -g -O2
```

```
dpkg-buildpackage: paquet source bc
dpkg-buildpackage: version source 1.06.95-2
dpkg-buildpackage: source changé par root <root@morgane.ed-diamond.com>
dpkg-buildpackage: architecture hôte i386
debian/rules clean
dh_testdir
dh_testroot
[...]
dpkg-genchanges: envoi d'un binaire - aucune inclusion de code source
dpkg-buildpackage: envoi d'un binaire seulement (aucune inclusion de
code source)
----> Cleaning up object files <-----
Cleaning in directory .
dh_testdir
dh_testroot
[...]
-----> Moving packages to repository <-----
-----> Building repository <-----
-----> Updating package lists <-----
Atteint http://ftp.fr.debian.org testing Release.gpg
[...]
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
[...]
```

En une seule commande, les sources sont téléchargées et compilées 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. 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. On voit même quelques mentions de **définir CFLAGS à la valeur par défaut**. Comme le précise le **README.Debian**, c'est parfaitement normal. Les lignes que vous voyez sont les commandes utilisées par **make**, mais un *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.

Nous en arrivons maintenant 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-build** a une option pour vous : **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** pour lesquels on ne dispose pas des sources. Mais ceux-ci, de toute façon, vous n'en faites pas usage, n'est-ce pas ?

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 et ne le sera sans doute jamais.

CONCLUSION

Nous venons de faire le tour de trois solutions permettant de reconstruire des paquets à partir des sources Debian. A vous de choisir celle qui vous conviendra le mieux en fonction de vos besoins. Soit vous souhaitez apporter des modifications dans un paquet et produire une version personnalisée, soit vous cherchez les optimisations possibles. Personnellement, je n'ai que peu d'affinité avec **apt-build** car c'est un système complet un peu trop proche de ce que propose **aptitude**. En effet, la frontière n'est pas vraiment claire et un **apt-build remove** suivi d'un nom de paquet vous supprimera le paquet binaire de votre système. C'est une question de goûts personnels, mais je préfère largement un outil qui s'en tient strictement à ce qu'il doit faire et à ce que son nom suggère.

Inversement, je me sais un tantinet laxiste avec l'organisation de mes répertoires et me suis déjà retrouvé plus d'une fois en train de jouer avec **find** pour retrouver où j'avais bien pu mettre les sources d'un paquet Debian que j'avais reconstruit quelques jours auparavant.

Ainsi, mon choix se portera tout naturellement sur **apt-src** puisqu'il m'offre suffisamment de liberté tout en automatisant et en organisant les sources comme il se doit.

Vous aurez peut-être (sans doute ?) une vision différente de la reconstruction de paquets. Mais au final, la seule chose importante revient à ne plus jamais utiliser un **make install** sauvage et chaotique. N'est-il pas ? ■

Créez votre paquet Debian

Debian, comme presque toutes les distributions, utilise un système de gestion de paquets. Celui de cette distribution, APT, est éprouvé de longue date et ne cesse d'être amélioré au fil du temps. Afin de ne pas corrompre le travail de ce dernier, il est indispensable de ne pas 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.

La suite de cet article n'a pas la prétention de replacer les guides officiels et surtout les conseils que peuvent donner des développeurs Debian parrains pour devenir responsable de paquet(s). L'objectif est bien plus modeste, puisque nous vous proposons simplement d'apprendre comment intégrer proprement une application, un outil ou une bibliothèque dans votre distribution Debian. Cet article pourra sans doute servir de point de départ pour qui souhaitera en apprendre davantage et se lancer de manière un peu plus officielle dans la prise en charge de paquets. Responsable de paquet est presque un métier, ou du moins une tâche importante qu'il ne faudra pas négliger. Comme le démontre un autre article dans le présent hors-série, le projet Debian est très organisé et structuré. Il requiert un minimum d'engagement et un sens des responsabilités. Ne lisez pas ce qui suit pour ensuite vous lancer dans la tâche tête baissée sans réfléchir. Un certain nombre de paquets Debian attendent déjà leur futur responsable. Visitez la page « Paquets en souffrance et paquets souhaités » sur le site du projet pour en connaître la liste (http://www.debian.org/devel/wnpp/). Rappelez-vous que c'est d'un engagement vis-à-vis d'autres personnes qui, comme vous, souhaitent faire avancer les choses et améliorer leur distribution préférée.

Cette précision apportée, nous pouvons maintenant faire nos premiers pas.

1 LA PRÉHISTOIRE

Sans système de gestion de paquets, ou en d'autres termes, avec une distribution tenant plus de LFS (*Linux From Scratch*) que de Debian, Ubuntu, Fedora ou Mandriva, on aura l'habitude d'installer ses paquets ainsi :

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

Si on est un peu ordonné, on s'avisera alors d'utiliser l'option `--prefix=/usr` afin de placer les éléments copiés par le `make install` dans `/usr/bin`, `/usr/share` et `/usr/lib` plutôt que dans `/usr/local/*`. Le choix d'une installation dans `/usr` ou `/usr/local` est quelque chose qui pourrait être discuté pendant longtemps. Légitimement, `/usr/local` est dédié aux installations locales d'applications. Dans la réalité, les chemins de recherche par défaut incluent rarement ce répertoire. Pire encore, s'il s'agit d'une distribution utilisant un système de gestion de paquets, vous risquez ni plus ni moins de vous retrouver avec des versions différentes des mêmes éléments dans `/usr` et dans `/usr/local`. Ce n'est pas tout, les scripts d'autoconfiguration reposent sur le test d'un environnement local à un instant donné et ce, de manière indépendante de la gestion des paquets. Autrement dit, si vous retirez un paquet contenant une bibliothèque dont votre application a besoin, vous n'en serez pas averti. Sauf, bien entendu, lorsqu'il s'agira de l'exécuter ou la recompiler.

Sur bien des points, le triptyque `./configure && make && make install` est le pire choix lorsqu'on veut installer un logiciel non empaqueté. Préférez alors des environnements chrootés, des machines virtuelles ou encore, plus simplement, une machine dédiée pour ce type de tests. Bien entendu, ce qui est vrai avec `autoconf` l'est également pour d'autres systèmes de configuration de sources comme `cmake`.

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. Oui, il existe des solutions pour contourner cette tâche, comme des outils de traçage d'installation gardant en mémoire ce que fera un `make install`. Ces solutions ne règlent cependant qu'une partie du problème. N'essayez pas de tricher. Pour faire stable, il faut faire propre et c'est tout !

2 NOTRE BASE DE TRAVAIL : NANO-INTRODUCTION À AUTOCONF

Nous partirons du principe ici que la majorité des applications et outils utilisent un système d'autoconfiguration des sources. En effet, rares sont aujourd'hui les sources contenant un simple **Makefile**. Le plus courant et le plus critiqué des systèmes de ce type est sans nul doute le couple **automake**/**autoconf**. Comme nous avons l'intention de garder une certaine simplicité dans nos explications, notre exemple de programme à empaqueter sera un utilitaire créé sur mesure pour l'occasion. C'est également une excuse pour apporter un petit plus à l'article en montrant un exemple de construction de sources utilisant **automake**/**autoconf**.

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 vides). Les sources de notre outil sont placées dans un sous-répertoire **src**. Nous avons là trois fichiers :

- **main.c**

```
/* 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**

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

#include "afficher.h"

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

- et **afficher.h**

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

int afficher(char *str);

#endif
```

Rien de bien fantastique, vous l'avez compris, notre outil est aussi petit qu'inutile. Il s'agit d'un simple exemple de type **Hello World**. 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'
src/Makefile.am: installing `./depcomp'
Makefile.am: installing `./INSTALL'
Makefile.am: installing `./COPYING'

% ls configure
configure
```

Remarquez les options **-a** et **-c** utilisées avec **automake**, permettant 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 les 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 *amont* (voir article précédent). 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
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking whether make sets $(MAKE)... (cached) 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
==========================================


% make
Making all in src
make[1]: entrant dans le répertoire
  " /mnt/six2/SRC/autoconf/HS48/baseaa/src "
if gcc -DPACKAGE_NAME=\"bonjour\" -DPACKAGE_TARNAME=\"bonjour\"
-DPACKAGE_VERSION=\"0.1\" -DPACKAGE_STRING=\"bonjour\ 0.1\"
-DPACKAGE_BUGREPORT=\"lefinnois@lefinnois.net\"
-DPACKAGE_URL=\"\" -DPACKAGE=\"bonjour\"
-DVERSION=\"0.1\" -I. -I.
-g -O2 -MT main.o -MD -MP -MF ".deps/main.Tpo" -c
-o main.o main.c; \
        then mv -f ".deps/main.Tpo" ".deps/main.Po";
        else rm -f ".deps/main.Tpo"; exit 1; fi
if gcc -DPACKAGE_NAME=\"bonjour\" -DPACKAGE_TARNAME=\"bonjour\"
-DPACKAGE_VERSION=\"0.1\" -DPACKAGE_STRING=\"bonjour\ 0.1\"
-DPACKAGE_BUGREPORT=\"lefinnois@lefinnois.net\"
-DPACKAGE_URL=\"\" -DPACKAGE=\"bonjour\"
-DVERSION=\"0.1\" -I. -I.     -g -O2
-MT afficher.o -MD -MP -MF ".deps/afficher.Tpo"
-c -o afficher.o afficher.c; \
        then mv -f ".deps/afficher.Tpo" ".deps/afficher.Po";
        else rm -f ".deps/afficher.Tpo"; exit 1; fi
gcc  -g -O2   -o bonjour  main.o afficher.o
make[1]: quittant le répertoire
  " /mnt/six2/SRC/autoconf/HS48/baseaa/src "
make[1]: entrant dans le répertoire
  " /mnt/six2/SRC/autoconf/HS48/baseaa "
make[1]: Rien à faire pour " all-am ".
make[1]: quittant le répertoire
  " /mnt/six2/SRC/autoconf/HS48/baseaa "

% 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 spécifiée 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 pas y avoir de problème, **make distcheck** étant précisément dédié à cela.

3 EMPAQUETONS !

Nous avons maintenant un exemple de sources amont parfaitement maîtrisables 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 le paquet **build-essential** qui installera un compilateur C et C++, les en-têtes de la bibliothèque C standard, la commande **make** et le paquet **dpkg-dev**. Ce dernier est celui contenant les outils de fabrication de paquets Debian, en plus de provoquer l'installation d'autres éléments sous la forme de paquets, dont les **binutils**, **cpio** 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 regroupent un lot de scripts très utiles, **fakeroot**, que vous connaissez déjà (voir article précédent) ou encore **lintian**, le testeur de paquets. Une installation complète et fonctionnelle de

GnuPG est également souhaitée afin de signer le **.dsc** du paquet source.

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 :

```
% mkdir DEB
% cd DEB
% cp /quelque/part/bonjour-0.1.tar.gz .
% tar xfzv bonjour-0.1.tar.gz
% cd bonjour-0.1

% dh_make -e lefinnois@lefinnois.net \
-f ../bonjour-0.1.tar.gz -c gpl2
Type of package: single binary, indep binary,
    multiple binary, library, kernel module,
    kernel patch or cdbs?
[s/i/m/l/k/n/b] s

Maintainer name : Denis Bodor
Email-Address   : lefinnois@lefinnois.net
Date            : Sat, 24 Apr 2010 17:48:45 +0200
Package Name    : bonjour
Version         : 0.1
License         : gpl2
Using dpatch    : no
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. Cet outil est extrêmement bien conçu mais un peu intrusif. Vous remarquerez qu'ici, seule l'adresse mail a été précisée, pourtant mon nom apparaît comme par enchantement. **dh_make** a récupéré mon login et parcouru les ressources à sa disposition (**/etc/passwd**, LDAP, NIS, YP, etc.) pour retrouver les informations manquantes.

Note : bug

A la date où sont faites ces manipulations, un bug est présent dans **dh_make**. Un message d'erreur précisant **Unable to make debian/source subdirectory: Aucun fichier ou dossier de ce type** s'affiche et le processus s'arrête. Il y a, en effet, un problème ligne 115 des sources en Perl (**/usr/bin/dh_make**) et il faut remplacer **mkdir 'debian/source'** par **mkdir 'source'**. Ce bug a déjà été rapporté (#576523) et la correction ne devrait pas tarder à arriver, avec une nouvelle version du paquet.

dh_make a procédé à un certain nombre de manipulations. Premièrement, il a copié notre tarball source en **bonjour_0.1.orig.tar.gz** dans le répertoire parent, respectant ainsi les conventions et règles en usage pour les paquets sources. Il 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.

Il n'est possible de « débianiser » des sources amont qu'une seule fois. Si vous faites 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'autre choix que de tout recommencer ou tenter de corriger le problème manuellement.

Les options utilisées ici permettent de préciser la licence de nos sources (**-c**) avec un choix possible entre GPL, Apache, BSD, etc. **-f** demande à **dh_make** de ne pas recréer une archive source, mais de simplement copier le tarball que nous spécifions sous un autre nom.

La partie sur l'utilisation d'**automake**/**autoconf** en début d'article n'était ni innocente, ni inutile. Comme l'indiquent les dernières lignes retournées par la commande, **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**, consiste à éditer les fichiers dans **debian/**.

3.1 Le fichier 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, mais nous devons le compléter :

```
Source: bonjour
Section: unknown
Priority: extra
Maintainer: Denis Bodor <lefinnois@lefinnois.net>
Build-Depends: debhelper (>= 7.0.50), autotools-dev
Standards-Version: 3.8.4
Homepage: <insert the upstream URL, if relevant>
#Vcs-Git: git://git.debian.org/collab-maint/bonjour.git
#Vcs-Browser: http://git.debian.org/?p=collab-maint/bonjour.
git;a=summary

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

La première ligne indique le nom du paquet source. Elle est automatiquement complétée, tout comme la valeur pour **Maintainer** (le responsable du paquet), **Standards-Version** (qui est la version de la charte Debian), **Package** (le nom du paquet binaire), **Priority** (l'importance du paquet, **extra** est parfait dans la plupart des cas qui nous concernent) 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 donnée. Il faut alors le spécifier manuellement. Ce sera le cas, par exemple, d'outils spécifiques à une famille de machines. Un utilitaire détectant par exemple les fonctionnalités SSE2 d'un processeur ne fonctionnera forcément pas sur architecture ARMel ou MIPS.

`Section` détermine la section et sous-section dans lesquelles placer le paquet. En tant que responsable de paquet, même amateur, vous devez vous plier à la règle, même s'il est vrai que les notions de section et sous-section ne sont plus vraiment d'usage actuellement d'un point de vue utilisateur. 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 logiciels 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 la sous-section `utils` (sous-entendu `main/utils` donc).

Note : J'adore Vim

Si comme moi, vous êtes utilisateur de l'éditeur Vim, vous aurez l'agréable surprise de constater qu'il va vous aider dans votre tâche de personnalisation des fichiers dans `debian/`. La colorisation automatique qu'il propose vous permettra ainsi de faire moins d'erreurs. Exemple typique : pour le champ `Section`, la mention d'une section inexistante provoquera automatiquement une mise en forme du texte en gris clair sur fond rouge. Difficile de faire plus explicite. Par la suite, il sera sans doute judicieux d'utiliser la commande `set noet ts=8 sw=8` car nous travaillerons sur des fichiers de type `makefile` s'accommodant peu des conversions de tabulation en groupes d'espace.

`Description` permet de donner la description courte (60 caractères maxi et sur une ligne). C'est celle qui apparaît, par exemple, avec `apt-cache search`, `aptitude 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 devrait 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.

Si vous avez l'habitude de consulter les descriptions de paquets Debian, vous remarquerez que cela n'est pas toujours respecté par certains responsables de paquets.

Nous utiliserons donc ici :

```
Description: The classic helloworld program in French
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, `${shlibs:Depends}`, par exemple, est une variable qui sera complétée lors de la première construction et analyse de votre paquet. Un script spécifique analysera le binaire (avec `ldd` entre autres) et déterminera les noms des paquets contenant les bibliothèques dynamiques utilisées. Il en va de même pour `${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 plugins sous la forme de bibliothèques partagées.

Note : Plusieurs types de dépendances

`Depends` permet de définir les dépendances de votre paquet. Celles-là même qui s'installeront lorsqu'on utilisera `aptitude install`. Mais il existe aussi d'autres manières de lier votre paquet avec d'autres, de manière mois radicale que `Depends`. Ainsi, nous avons les champs `Recommends` pour spécifier les paquets qui ne sont pas vraiment indispensables, mais qui sont normalement utilisés avec le logiciel et `Suggests` pour les paquets qui contiennent des données ou des applications qui fonctionnent parfaitement avec le logiciel. La nuance avec `Recommends` est subtile, ici, il faut comprendre que c'est un conseil alors que `Recommends` indique des paquets souhaitables mais dont l'absence n'empêchera pas votre application de fonctionner (elle pourrait simplement devenir inutile ou en être fortement handicapée).

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. L'exemple typique est celui d'un logiciel fournissant un service réseau. Il ne peut y avoir deux serveurs DHCP sur une même machine. `Provides` est en rapport avec la notion de paquets virtuels. Le paquet `lighttpd`, par exemple, fournit le paquet virtuel `httpd`. Le paquet `nginx` fait de même. Le simple fait d'installer l'un de ces paquets satisfait les dépendances liées à `httpd`. L'installation du paquet `nginx` provoquera la désinstallation de `lighttpd`. La désinstallation de `nginx`, ensuite, cassera une dépendance vis-à-vis de tous les paquets ayant besoin de `httpd` et on provoquera ainsi leur désinstallation (après confirmation de l'utilisateur).

Enfin, `Build-Depends` informe sur les dépendances de construction du paquet. C'est cette liste qui sera utilisée par `apt-get build-dep` (voir article précédent). 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 :

```
bash (>= 4.1-1 ), binutils (>= 2.20-6 ), coreutils (>= 7.2-1 ),
libacl1 (>= 2.2.41-1 ), libattr1 (>= 1 ), libc6 (>= 2.10.1-5 ),
libc6-dev (>= 2.10.1-5 ), libc6-i686 (>= 2.10.1-5 ),
libgmp3c2 (>= 2 ), libmpfr1ldbl (>= 2.3.1.dfsg.1-2 ),
libncurses5 (>= 5.7+20090803-2 ), libselinux1 (>= 2.0.85-4 ),
locales (>= 2.10.1-5 ), zlib1g (>= 1 )
```

La plupart de ces paquets ne nécessitent pas de mention particulière puisqu'ils seront déjà installés avec **debhelper** ou via des dépendances de **build-essential**. 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 de développement correspondant. Les deux méthodes peuvent être utilisées de concert, mais comprenez bien que c'est bien vous et uniquement vous qui devrez finalement tirer les conclusions qui s'imposent et définir les dépendances de construction. Vous l'avez peut-être remarqué, ni la méthode **strace**, ni **objdump** ne nous informe de la nécessité de disposer des **autotools-dev**, par exemple. Inversement, dans le traitement du **/tmp/log**, bon nombre de paquets ne seront pas à mentionner, comme **coreutils** ou **binutils**, déjà nécessaires à la compilation/construction de paquets en tant que tel. Notre ligne sera ici **Build-Depends: debhelper (>= 5), autotools-dev, libc6-dev (>= 2.3.6.ds1-4)**.

3.2 Le fichier rules

Le fichier **debian/rules** est un **Makefile** utilisé par **dpkg-buildpackage** pour créer le paquet. On y trouvait auparavant 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 du **debhelpher** (scripts **dh_***).

Avant la version 7 du **debhelper**, les fichiers **rules** étaient encore relativement conséquents et nécessitaient une grande attention de la part du responsable de paquets, et ce, y compris pour les outils les plus simples à compiler. Les efforts de simplification et d'automatisation nous ont conduits à un stade où le fichier **rules** pour notre code se limite à ceci :

```
#!/usr/bin/make -f
%:
        dh $@
```

Deux paquets Debian permettent de se rendre compte aisément de l'avancée de **debhelper**. Nous avons d'une part **hello**, le classique **hello world** GNU, et d'autre part **hello-debhelper**. En installant les paquets sources des deux versions, on ne peut que constater la simplification :

```
% wc -c hello-2.5/debian/rules
2174 hello-2.5/debian/rules
% wc -c hello-debhelper-2.5/debian/rules
783 hello-debhelper-2.5/debian/rules
```

Une version plus « construite » du fichier **rules** pourrait être celle-ci :

```
#!/usr/bin/make -f
build:
        dh build

clean:
        dh clean

binary-arch:
        dh binary-arch

binary-indep:
        dh binary-indep

binary: build binary-arch binary-indep
```

On retrouve alors les cibles habituelles pour la construction de paquets. Il existe une quantité très importante de scripts **dh_*** que **dh** utilise de manière séquentielle. La commande **dh** que nous utilisons dans notre fichier **rules** prend en argument un nom de séquence parmi **build**, **clean**, **install**, **binary-arch**, **binary-indep** et **binary**. Ce sont là les différentes cibles d'un fichier **rules**. Les commandes **dh_*** lancées par chaque séquence sont :

- **build:** : **dh_auto_configure**, **dh_auto_build**, **dh_auto_test**.
- **clean:** : **dh_testdir**, **dh_testroot**, **dh_auto_clean** et **dh_clean**.
- **install:** : Tout ce qui est lancé pour **build**, plus **dh_installdirs**, **dh_install**, **dh_auto_install**.
- **binary:** : Tout ce qui est lancé pour **install** et toutes les commandes **debhelper** de la cible **binary**.
- **binary-arch:** : Identique à **binary**, mais les commandes sont lancées avec l'option **-s** (construit les paquets dépendants de l'architecture).
- **binary-indep:** : Identique à **binary**, mais ne lance pas **dh_strip**, **dh_shlibdeps** ou **dh_makeshlibs**. Lance les commandes avec l'option **-i** pour construire les paquets indépendants de l'architecture.

Les scripts **dh_*** sont suffisamment « intelligents » pour fonctionner seuls. Ainsi, si nous prenons l'exemple de **dh_auto_configure**, il est capable de configurer les sources qu'il s'agisse d'une utilisation d'**autoconf**/**automake** ou d'un autre processus de configuration automatique, comme un **Makefile.PL**.

Pour un paquet comme le notre, la création/modification du fichier **rules** n'est pas un problème. Sachez cependant que les scripts **debhelper** savent parfaitement s'adapter à des besoins très spécifiques. Voici un exemple. Imaginons que pour la construction de notre binaire et la version de l'outil à empaqueter, nous devions limiter ses fonctionnalités ou les étendre. Dans la plupart des cas, on procédera en utilisant des options spécifiques du script **configure** (type **--enable-framebuffer**, par exemple). Dans notre **rules**, nous pourrons alors utiliser une configuration spécifique pour la cible **build** :

```
build:
    dh build --before configure
    ./configure --enable-framebuffer
    dh build --after configure
```

La cible **build** lance normalement simplement **dh build**, mais nous utilisons tout d'abord l'option **--before configure** pour traiter tout ce qui se trouve avant la configuration dans la séquence **build**. Nous lançons manuellement **./configure**, puis nous utilisons l'option **--after configure** pour traiter la suite de la séquence. Pour quelques astuces d'utilisation supplémentaires, consultez http://kitenet.net/~joey/blog/entry/cdbs_killer___40__design_phase__41__.

Le nombre de combinaisons possibles dans un fichier **rules** est très important. Si vous souhaitez en apprendre davantage et optimiser le contenu (pour supprimer, par exemple, les appels inutiles à des scripts **dh_***), la meilleure solution est de récupérer un grand nombre de paquets sources (à commencer par **hello-debhelper**) et d'étudier les fichiers **rules** qui s'y trouvent.

3.3 Autres fichiers

Le fichier **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 qu'il faudra ensuite modifier et nettoyer :

```
bonjour (0.1-1) unstable; urgency=low

  * Initial release

-- Denis Bodor <lefinnois@lefinnois.net>  Sat, 24 Apr 2010 17:48:45 +0200
```

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. Ceux-là resteront en place même après une désinstallation du paquet. C'est l'opération de purge qui supprimera le paquet avec ses fichiers de configuration.

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**. Ici, **dh_make** a déjà spécifié les fichiers **NEWS** et **README**.

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

compat contient un simple numéro, c'est un indicateur de compatibilité avec **debhelper** permettant de s'assurer du bon fonctionnement de la construction avec d'autres systèmes. Ici, un **7** indique la toute nouvelle version 7 du **debhelper**.

copyright est également déjà renseigné mais doit être personnalisé. Comme nous avons spécifié une licence GPLv2, nous avons le choix entre un texte précisant la version de la licence sans autre possibilité ou la version choisie et n'importe quelle licence en version supérieure : **version 2 of the License, or (at your option) any later version**. C'est un sujet à controverse et tantôt, selon la législation du pays concerné, un problème : celui de la mise sous licence via un texte qui n'existe pas encore.

4 LE CAS CDBS

CDBS, pour *Common Debian Build System*, est une autre méthode de création de paquets sources. Sa création a été initiée suite à une constatation évidente. Le fichier **rules** est, de loin, l'élément le plus délicat à configurer dans un paquet source. Même avec l'assistance de **debhelper**, ceci représente le principal point de difficulté rencontré par les nouveaux responsables de paquet. De plus, la maintenance des paquets sources est rendue difficile lorsqu'un changement majeur a lieu. Ainsi, par le passé, la variable **DEB_BUILD_OPTIONS** s'est vue modifiée en échangeant **debug** au bénéfice de **noopt**. S'en est suivie la nécessité de revoir bon nombre de fichiers **rules** pour les mettre à jour. Enfin, nous l'avons déjà dit, les scripts **debhelper** sont souvent lancés sans nécessité et alourdissent le processus de construction et parfois de maintenance des sources. Face à cet ensemble de problèmes, CDBS se veut être une solution plus élégante.

CDBS se propose ainsi de clarifier et simplifier le fichier **rules** en utilisant un ensemble de fichiers **Makefile** placés dans **/usr/share/cdbs**. Vous pouvez générer un squelette de paquet source reposant sur CDBS avec **dh_make**. Il suffit de spécifier un type **b** lorsqu'on vous le demande. Dans notre cas, le fichier **rules** ressemblerait alors à ceci :

```
#!/usr/bin/make -f

include /usr/share/cdbs/1/rules/debhelper.mk
include /usr/share/cdbs/1/class/autotools.mk
```

Notez bien que CDBS n'élimine pas **debhelper**, mais l'étend tout en restant indépendant. Paradoxalement, CDBS et **debhelper** ne sont pas deux projets directement concurrents, mais plutôt deux manières de voir la façon de simplifier les choses, même si les utilisateurs des deux « clans » semblent signifier dans leur documentation que la méthode « d'en face » n'est pas vraiment utile.

Pour plus d'informations sur CDBS, consultez **http://build-common.alioth.debian.org/cdbs-doc.html**.

5 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 distribuable. Assurez-vous d'être dans le répertoire des sources amont et utilisez **dpkg-buildpackage**. Si vous avez lu l'article sur la reconstruction de paquets, cette commande vous est maintenant familière. Notez cependant l'absence de l'option **-b** limitant le processus de construction au paquet binaire.

```
% dpkg-buildpackage
dpkg-buildpackage: paquet source bonjour
dpkg-buildpackage: version source 0.1-1
dpkg-buildpackage: source changé par
  Denis Bodor <lefinnois@lefinnois.net>
dpkg-buildpackage: architecture hôte i386
fakeroot debian/rules clean
dh clean
   dh_testdir
   dh_auto_clean
   dh_clean
dpkg-source -b bonjour-0.1
dpkg-source: info: utilisation du
  format source " 3.0 (quilt) "
dpkg-source: info: construction de bonjour
  à partir de ./bonjour_0.1.orig.tar.gz
dpkg-source: info: construction de bonjour
  dans bonjour_0.1-1.debian.tar.gz
dpkg-source: info: construction de bonjour
  dans bonjour_0.1-1.dsc
[...]
```

Dès le lancement, les scripts vont commencer par construire le paquet source en utilisant le tarball amont. On remarque la première cible exécutée, qui est **debian/rules clean**.

```
[...]
debian/rules build
dh build
   dh_testdir
   dh_auto_configure
checking for a BSD-compatible install... /usr/bin/install -c
[...]
```

Le script **configure** est ensuite lancé en utilisant les paramètres permettant l'installation dans une arborescence correspondant aux besoins de la distribution. C'est le début de la cible **debian/rules build**. Suit, directement après, la compilation en elle-même :

```
[...]
config.status: executing depfiles commands
   dh_auto_build
make[1]: entrant dans le répertoire
" /mnt/six2/SRC/autoconf/HS48/DEB/bonjour-0.1 "
Making all in src
make[2]: entrant dans le répertoire
" /mnt/six2/SRC/autoconf/HS48/DEB/bonjour-0.1/src "
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 -O3 -c main.c
[...]
```

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

```
[...]
fakeroot debian/rules binary
dh build
dh binary-arch
   dh_testroot -a
   dh_prep -a
   dh_installdirs -a
   dh_auto_install -a
[...]
   dh_install -a
   dh_installdocs -a
   dh_installchangelogs -a
   dh_installexamples -a
   dh_installman -a
   dh_installcatalogs -a
   dh_installcron -a
   dh_installdebconf -a
   dh_installemacsen -a
   dh_installifupdown -a
   dh_installinfo -a
   dh_pysupport -a
   dh_installinit -a
   dh_installmenu -a
   dh_installmime -a
   dh_installmodules -a
   dh_installlogcheck -a
   dh_installlogrotate -a
   dh_installpam -a
   dh_installppp -a
   dh_installudev -a
   dh_installwm -a
   dh_installxfonts -a
  dh_bugfiles -a
   dh_lintian -a
   dh_gconf -a
   dh_icons -a
   dh_perl -a
   dh_usrlocal -a
```

```
  dh_link -a
  dh_compress -a
  dh_fixperms -a
  dh_strip -a
  dh_makeshlibs -a
  dh_shlibdeps -a
  dh_installdeb -a
  dh_gencontrol -a
  dh_md5sums -a
  dh_builddeb -a
[...]
```

Comme vous pouvez le voir, énormément de scripts sont appelés pour rien (**dh_gconf**, **dh_installmime** ou encore **dh_installifupdown**). L'utilisation de CDBS donne un résultat un peu moins dense, mais certains scripts sont encore lancés pour rien (**dh_installinfo**, **dh_installpam**, ou **dh_installdebconf**). Aucune automatisation ne pourra rendre les choses à la fois simples et efficaces. Si vous voulez vraiment que la construction soit totalement optimisée, il faudra vous passer de **dh** et appeler les scripts vous-même. Plus loin encore, vous pouvez en revenir à l'ancienne méthode consistant à créer le **Makefile rule** de toutes pièces en implémentant manuellement chaque cible. Mais là, vous troquerez optimisation contre complexité.

Enfin, il est temps de construire le fichier **.deb**, puis de signer les fichiers importants, à commencer par **bonjour_0.1-1.dsc** :

```
[...]
dpkg-deb : construction du paquet " bonjour " dans " ../bonjour_0.1-1_
i386.deb ".
dh binary-indep
signfile bonjour_0.1-1.dsc
gpg: signatures créées jusqu'ici: 15

Entrez le PIN
[sigs faites: 15]

dpkg-genchanges  >../bonjour_0.1-1_i386.changes
dpkg-genchanges: inclusion du code source original dans l'envoi
(" upload ")
```

```
signfile bonjour_0.1-1_i386.changes
gpg: signatures créées jusqu'ici: 16

Entrez le PIN
[sigs faites: 16]

dpkg-buildpackage: envoi complet (inclusion du code source d'origine)
```

Si une version utilisable de GnuPG est détectée, une clé secrète est recherchée pour l'utilisateur spécifié dans le fichier **debian/control**. Si la clé est trouvée, la phrase de passe est demandée pour signer le fichier **.dsc** puis le fichier **.changes**. Ici, la sortie est un peu particulière puisqu'on fait usage d'une carte OpenPGPv1 où se trouve la sous-clé de l'utilisateur (moi). Ce n'est donc pas une phrase de passe qui est demandée, mais le code PIN de la carte.

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.debian.tar.gz** : le contenu du répertoire **debian/** à inclure dans les sources. Anciennement, un fichier **bonjour_0.1-1.diff.gz** aurait été créé, un patch à appliquer aux sources amont pour obtenir les sources Debian. Aujourd'hui, il s'agit d'un tarball qui sera pris en charge par **dpkg-source**. En cas de changement dans un fichier source amont, un sous-répertoire **patches** en plus de **debian** sera créé dans l'archive. Un fichier dans un format spécifique y sera ajouté afin de patcher les sources officielles.
- **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.

CONCLUSION

A présent, vous devez être capable de faire vos premiers pas dans la création de paquets Debian. Si vous souhaitez aller plus loin et ne pas garder votre travail pour vous, 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 le trouverez sur http://www.debian.org/doc/manuals/maint-guide/. Là encore, 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 et tomber. 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.

Debian GNU/Linux est de loin la distribution la plus avancée dans bien des domaines. Mais aussi organisé et rodé que soit le projet, il a toujours besoin de contributions. Debian 6.0 se profile doucement à l'horizon avec un « freeze » prévu pour fin mai ou début juin. Le responsable de publication, Adam Barratt, a annoncé dernièrement que la situation de *squeeze* n'était pas aussi bonne qu'espérée, mais que la prochaine version stable devrait voir le jour d'ici quelques mois. Debian est un projet vivant sur lequel reposent de plus en plus d'utilisateurs (et je compte ici, en toute logique, également ceux d'Ubuntu). Si vous avez un peu de temps pour « renvoyer l'ascenseur », n'hésitez pas. ■

Linux et Debian : des histoires de noyaux

Le titre précise « Linux » et non « GNU/Linux » car c'est bien du noyau dont nous allons parler et non du système dans son ensemble. Le noyau n'est pas un élément logiciel comme les autres. Il s'agit du fondement de tout le système. Pour autant, sa gestion, son installation et sa mise à jour doivent être aussi simples que pour n'importe quel autre élément, qu'il s'agisse d'un environnement complet ou d'un simple outil en ligne de commandes. Voyons comment Debian règle ce délicat dilemme.

Avant de commencer, il est utile de préciser ce qui peut bien motiver un utilisateur ou un administrateur système à toucher au noyau. Actuellement, la mise à jour d'une distribution Debian ou dérivée est quelque chose qui se passe généralement sans encombre. Ceci sera tout aussi vrai avec les applications et outils de l'espace utilisateur que pour le noyau ou ses modules. Résultat, on se demande pourquoi on voudrait bien changer quoi que ce soit à ce niveau.

Plus le temps passe, plus l'installation et la configuration d'un système GNU/Linux sont automatisées. Les interfaces réseaux sont détectées et configurées, les clés USB de toutes sortes automatiquement prises en charge et l'adaptateur graphique ne nécessite même plus de configuration spécifique pour afficher un bureau 3D bling-bling. Nous sommes loin de l'époque où la prise en charge d'une carte Ethernet ou d'un contrôleur audio nécessitait une recompilation du noyau. L'étape fameuse indispensable pour passer du stade « débutant » au « padawan linux », où bon nombre d'utilisateurs se sont cassés les incisives, paraît aujourd'hui désuète. Qui recompile encore son noyau de nos jours ?

Voici quelques réponses possibles :

- Les administrateurs système, tout d'abord, qui, fort judicieusement, considèrent que l'utilisation d'un noyau modulaire n'est rien de moins qu'un appel à installer un rootkit LKM. Sans possibilité de charger des modules, on limite les actions d'un éventuel pirate souhaitant s'installer durablement ou se garder une porte d'entrée dissimulée dans le système.
- Les personnes souhaitant faire fonctionner un matériel récalcitrant ou nécessitant l'installation d'un support dans le noyau qui n'est pas encore pris en charge dans les sources officiels ou dans le noyau fourni par leur distribution.
- Les développeurs ou utilisateurs éclairés voulant disposer d'une version récente du noyau afin de découvrir et profiter rapidement des dernières avancées.
- Les concepteurs de systèmes embarqués pour qui chaque kilooctet de mémoire compte, ainsi que chaque cycle CPU. Habituellement certes, ceux-ci ne profiteront pas des solutions mises en place par une distribution spécifique mais, comme le montre l'article en bonus en fin de magazine, les choses changent et Debian, par exemple, trouve sa place dans le monde de l'embarqué.
- Les utilisateurs souhaitant comprendre le fonctionnement du système dans son ensemble ou voulant apporter leur petite touche personnelle un peu partout.

Vous l'avez compris, recompiler un noyau, des modules, ou en modifier les sources sont autant de manipulations toujours d'actualité. 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 sa mise à jour, se fait très simplement en utilisant `aptitude`. Mais encore faut-il choisir la bonne version pour votre machine. Le projet met à disposition plusieurs variantes compilées et empaquetées du noyau Linux pour chaque version stable de ce dernier.

Ainsi, pour la version 2.6.32, on trouve sept différents paquets :

```
linux-image-2.6.32-4-486
  - Linux 2.6.32 for old PCs
linux-image-2.6.32-4-686-bigmem
  - Linux 2.6.32 for PCs with 4GB+ RAM
linux-image-2.6.32-4-686
  - Linux 2.6.32 for modern PCs
linux-image-2.6.32-4-amd64
   - Linux 2.6.32 for 64-bit PCs
linux-image-2.6.32-4-openvz-686
  - Linux 2.6.32 for modern PCs, OpenVZ support
linux-image-2.6.32-4-vserver-686-bigmem
  - Linux 2.6.32 for PCs with 4GB+ RAM, Linux-VServer support
linux-image-2.6.32-4-vserver-686
  - Linux 2.6.32 for modern PCs, Linux-VServer support
```

Ces paquets contiennent tous un noyau Linux. Ils se distinguent par le choix de la plate-forme dans la configuration des sources ainsi que par certaines autres options sélectionnées ou patchs appliqués. Le choix est relativement simple si vous connaissez votre matériel. Les paquets ***openvz*** et ***vserver*** sont des versions spéciales du noyau, conçues pour fonctionner dans un environnement de paravirtualisation de type OpenVZ ou Linux-Vserver. Pour d'autres versions du noyau, il existe également des versions Xen.

Note :

Les anciens paquets ***smp*** destinés aux architectures multiprocesseurs (SMP pour *Symmetric MultiProcessing*) n'existent plus. Toutes les versions binaires du noyau supportent à présent le SMP. Une différenciation était faite pendant un temps 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.*** n'est pas un paquet tout à fait comme les autres. Lors de son installation, il va générer, entre autres choses, 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-2.6.32-4-686_2.6.32-11_i386.deb**, par exemple, vous constaterez que le fichier **/boot/initrd.img-2.6.32-4-686** n'est pas présent dans l'archive. Celui-ci sera généré lors de l'utilisation d'**aptitude install** :

```
Running depmod.
Running update-initramfs.
update-initramfs: Generating /boot/initrd.img-2.6.32-4-686
Running update-grub
```

update-initramfs est appelé pour créer le RAMdisk initial (voir article « Comprendre le boot d'un système Linux » dans *GNU/Linux Magazine* 127 actuellement en kiosque). 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, de chiffrement de disque ou LVM2 (voir article dans le présent numéro), il est impératif que les scripts présents dans le RAMdisk configurent et activent le RAID, les disques et/ou les volumes avant le montage et le passage sur le système de fichiers racine. Si vous changez votre configuration d'un système standard vers du RAID logiciel, vers LVM ou vers des disques LUKS, vous devrez utiliser **update-initramfs** afin de prendre les changements en compte pour le prochain démarrage.

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 dépend de la version du noyau et de sa révision. Ceci est défini lors de la compilation du noyau et de la construction du paquet (voir plus loin).

Enfin, bien entendu, il faut configurer les *bootloaders* chargés de copier l'image du noyau et l'image initramfs en mémoire. En cas de changement, il faut ajouter ou supprimer des entrées dans la configuration. Là encore, c'est le système de gestion de paquets qui s'occupera de la manipulation.

1 RECOMPILATION D'UN NOYAU DEBIAN

Nous l'avons dit précédemment, les occasions de reconstruire un noyau existent même si elles sont relativement rares ou spécifiques à certains besoins. 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 cela :

- les sources du noyau **linux-source-2.6.*** ;
- le paquet **kernel-package** fournissant, entre autres, le script de construction du paquet binaire **make-kpkg** ;
- et les fichiers d'en-têtes de la **libncurses**, **libncurses5-dev** permettant la configuration du noyau via l'interface **menuconfig** (ncurses/Dialog).

Notez que ce dernier point n'est pas absolument nécessaire puisque le noyau dispose, par défaut, d'une interface de configuration en ligne de commandes par question/réponse (**make config**). Vous conviendrez cependant que ce n'est pas là un exemple d'ergonomie et que l'installation d'un petit paquet supplémentaire n'est pas cher payé. Enfin, si vous souhaitez quelque chose de plus convivial à la sauce GTK+ (**make gconfig**) ou Qt (**make xconfig**), il vous faudra installer les paquets de développement des *toolkits* en question.

Chronologie Debian

Août 1993

Ian Murdock fonde le 16 août le projet Debian. Le nom de la distribution est formé à partir de son prénom et de celui de sa future ex-femme, Debra. L'objectif était de réunir au sein d'une même distribution les « meilleurs » logiciels libres.

Sortie de **Debian 0.01**.

1

Janvier 1994

Publication du Manifeste Debian. Document dans lequel Ian Murdock explique les raisons pour lesquelles le projet fut créé et insiste sur le fait que Debian est avant tout une distribution ouverte.

Debian 0.91 : correction de bugs mineurs concernant le gestionnaire de paquets dpkg.

2

Novembre 1994

Debian est soutenu par la FSF durant une année. C'est suite à l'une des requêtes de l'organisation que le projet se voit affublé de la mention GNU/Linux. On parlera dès lors de Debian comme d'une distribution GNU/Linux. Des désaccords verront cependant le jour concernant la structure et le développement de la distribution.

3

Mars 1995

Debian 093R5 : qualifiée de première version « moderne » de la distribution. Intègre pleinement le gestionnaire de paquets dpkg.

4

Novembre 1995

Debian 093R6 : Dernière mise à jour à utiliser le format *assembler output*. Apparition de dselect, interface en mode texte assurant la gestion des paquets.

Mise en place du premier serveur master.debian.org.

5

Juin 1996

Debian 1.1 (*Buzz*) : Première version utilisant un nom de code tiré des personnages du film *Toy Story* et recourant au format de fichier ELF (*Executable and Linking Format*). Debian 1.0 ne fut jamais publié, car un CD-Rom contenant la version en cours de développement fut accidentellement distribué.

6

Décembre 1996

Debian 1.2 (*Rex*) : Rex est le fruit du travail de plus d'une centaine de développeurs. Plus besoin de réinstaller votre système, il suffit de le mettre à jour pour profiter des dernières nouveautés. On peut désormais recourir aux paquets RPM et Slackware en les convertissant via un programme dédié (alien). Le système de paquets peut désormais récupérer les logiciels depuis un site FTP.

7

Juin 1997

Debian 1.3 (*Bo*) : La première mise à jour disponible sur un CD-Rom officiel. Elle regroupe près de 1000 paquets et 200 développeurs.

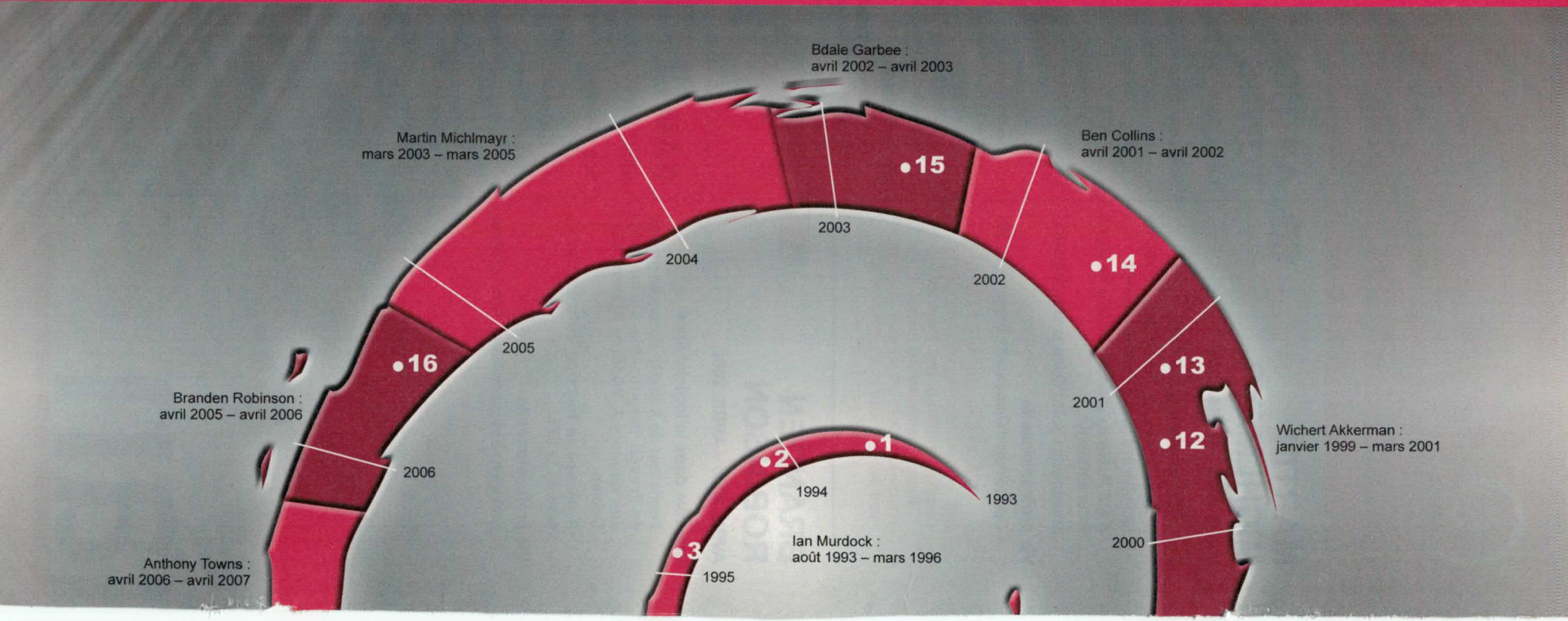
Naissance de la SPI (*Software in the Public Interest*) afin d'apporter une entité légale au projet.

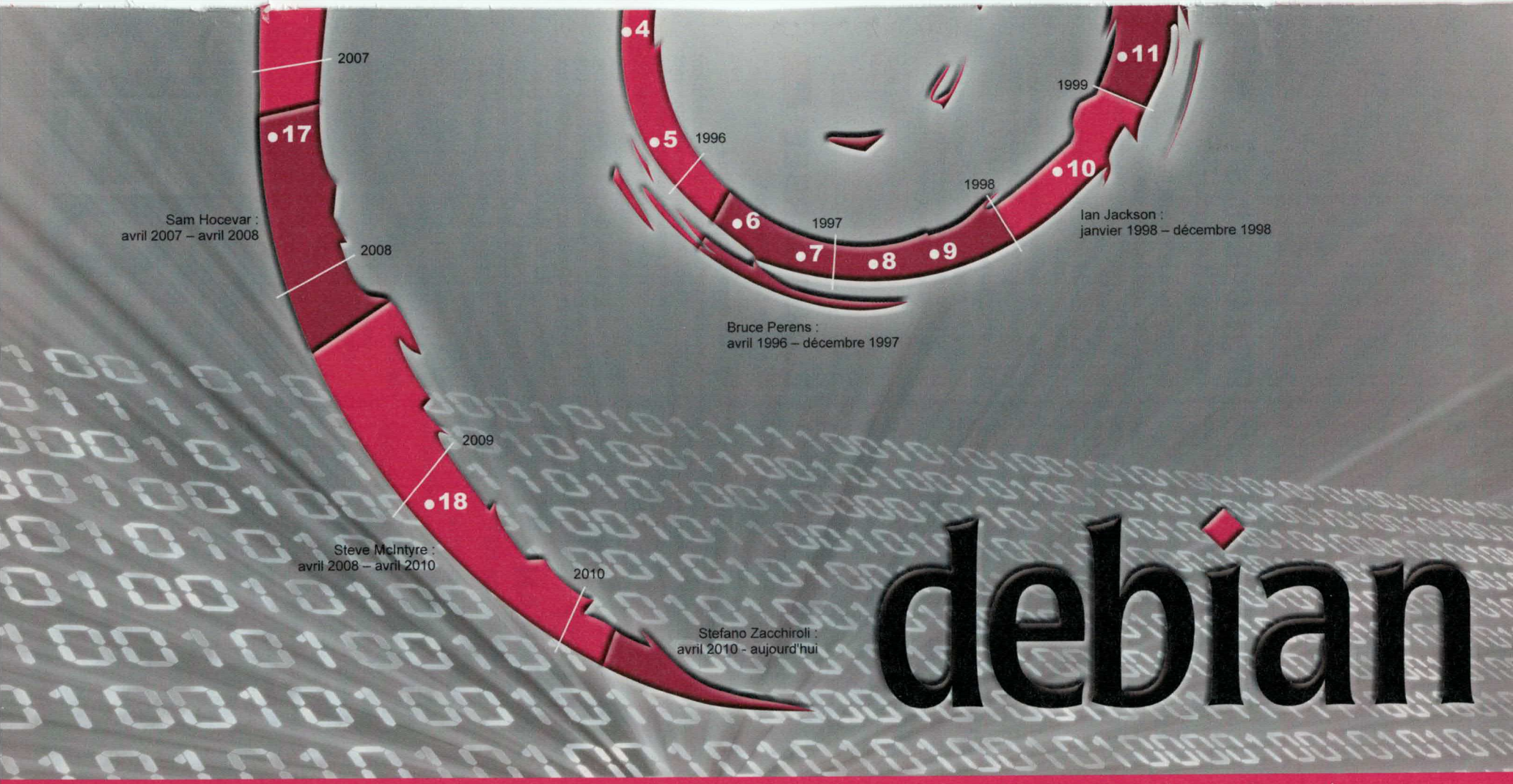
8

Juillet 1997

Parution du **Contrat Social** Debian dans sa version 1.0.

9





Juillet 1998

Debian 2.0 (*Hamm*) : Support de plusieurs architectures et utilisation de la version 2.0 de glibc, libc6.

10

Mars 1999

Debian 2.1 (*Slink*) : Intégration du système APT facilitant la gestion des paquets. Quatre architectures sont désormais supportées.

11

Août 2000

Debian 2.2 (*Potato*) : Un système d'installation amélioré. Utilisation de PAM (*Pluggable Authentication Modules*) au sein de la plupart des utilitaires. La distribution regroupe près de 4000 paquets et supporte deux nouvelles architectures.

12

Décembre 2000

Introduction de la version **Testing** qui permet de réduire le temps de gel et d'accueillir les paquets matures avant que ces derniers ne passent en *Stable*. Elle trouve ainsi sa place entre *Stable* et *Unstable*.

13

Juillet 2001

Première **Debconf** réunissant les développeurs de la distribution. L'événement est reconduit chaque année.

14

Juillet 2002

Debian 3.0 (*Woody*) : Dix architectures sont supportées. L'accent a été mis sur la sécurité (signature numérique des paquets, administration de pare-feu, etc.). Cette version propose pour la première fois l'environnement de bureau KDE.

15

Juin 2005

Debian 3.1 (*Sarge*) : La distribution passe à 9000 paquets. Une suite bureautique complète (OpenOffice.org) est intégrée. Aptitude est proposé comme alternative à dselect.

16

Avril 2007

Debian 4.0 (*Etch*) : Cette mise à jour inclut le support de l'architecture AMD64. Le gestionnaire de fenêtres avancé Compiz est pris en charge. Firefox et Thunderbird ont été respectivement renommés en Iceweasel et Icedove. Trois environnements de bureau sont proposés (GNOME 2.14, KDE 3.5.5 et Xfce 4.4).

17

Février 2009

Debian 5.0 (*Lenny*) : Onze architectures sont supportées par la distribution qui regroupe plus de 23000 paquets. LXDE fait partie des environnements pris en charge. On notera également le support du système de fichiers NTFS.

18

Debian Project Leader

Avec une communauté regroupant près de 1000 développeurs, le responsable de projet ou Debian Project Leader fait figure de représentant officiel du projet. Élu par ses pairs pour un mandat d'une année, c'est lui qui orientera le développement de la distribution tout au long de cette période. Depuis la fondation de la distribution en 1993, douze développeurs se sont succédés à cette fonction.

IAN MURDOCK

AOÛT 1993 – MARS 1996

Son rôle dans le logiciel libre

- Fondateur du projet Debian.
- L'un des membres fondateurs de l'*Open Source Initiative (OSI)*, une organisation encourageant le mouvement open source, la promotion de logiciels non propriétaires et souhaitant faire prendre conscience au public de ces perspectives dans une optique éducative.
- L'un des fondateurs de Linux International, une association mondiale regroupant des utilisateurs encourageant l'utilisation de logiciels libres et open source.
- *Chief Technology Officer* au sein du groupe de travail *Free Standard Groups*, en charge de la promotion de standards open source.
- Développeur de dpkg, le gestionnaire de paquets à la base de Debian.
- À l'origine de Progeny Debian, une distribution commerciale qui fournissait des plates-formes personnalisées.

Vie professionnelle

- Vice-Président des plates-formes émergentes chez Sun Microsystems.

« Originally, many people claimed that the open development of the Linux kernel was an invitation to chaos and disaster, yet Linux is not a disaster. Neither is Debian, for a good reason. »

BRUCE PERENS

AVRIL 1996 – DÉCEMBRE 1997

Son rôle dans le logiciel libre

- A rédigé le Contrat Social Debian ainsi que les Principes du logiciel libre selon Debian.
- Auteur de l'*Open Source Definition* conçue à partir des principes du logiciel libre selon Debian. Cette dernière est au cœur de l'*Open Source Initiative*.
- Co-fondateur d'Open Source Initiative avec Eric Raymond en 1998.
- L'un des fondateurs de la *Software in the Public Interest* servant de cadre légal au projet Debian.
- Fondateur de UserLinux, un projet de distribution « professionnelle » dérivée de Debian, qui vit le jour en 2003 et fut stoppé en 2006.
- Développeur de BusyBox, surnommé le « couteau suisse », car il fournit un lot d'utilitaires UNIX pour des systèmes embarqués.

Vie professionnelle

- CEO (*Chief Executive Officer*) chez Kiloboot, une start-up qui proposera des produits open source sous double licence.

« It's not about ego. Really. If it was ego, I'd be telling you about my awesome... oh, never mind. »

IAN JACKSON

JANVIER 1998 – DÉCEMBRE 1998

Son rôle dans le logiciel libre

- Fut Vice-Président puis Président de la SPI.
- Contribua à la rédaction de la Linux FAQ.
- A développé authbind, un utilitaire réseau.
- A créé une version modifiée du jeu en 3D Spaceships.

Vie professionnelle

- Travaille au développement de XenSource, Xen étant une plate-forme de virtualisation.

« I'm sorry if the following sounds combative and excessively personal, but that's my general style. »

WICHERT AKKERMAN

JANVIER 1999 – MARS 2001

Son rôle dans le logiciel libre

- A occupé la fonction de secrétaire au sein de la SPI.
- A développé Zope, un serveur d'applications web en Python.
- Participa au développement de Plone, un système de gestion de contenu.

Vie professionnelle

- *Release Manager* chez Plone Foundation, une organisation entourant le développement du projet Plone et proposant des services de support et d'assistance.

« It is simple to make things. It is hard to make things simple. »

BEN COLLINS

AVRIL 2001 – AVRIL 2002

Son rôle dans le logiciel libre

- A été administrateur de groupe Unix au sein de l'un des centres de recherche de la NASA.
- A occupé le poste de *Kernel Engineer* chez Canonical Ltd.
- Fut CEO chez SwissDisk Inc., une société offrant des services de stockage en ligne.

Vie professionnelle

- *Kernel Programmer* chez Bluecherry LLC, qui propose notamment des applications de vidéosurveillance sous Linux.

« If I had saved all the messages I was going to send, and decided not to, I would have to get a new harddrive :) »

BDALE GARBEE

AVRIL 2002 – AVRIL 2003

Son rôle dans le logiciel libre

- Membre du CA (Conseil d'Administration) d'*Open Media Now*, une association visant notamment à proposer une infrastructure multimédia ouverte et à promouvoir des solutions multimédias multiplates-formes.
- Siège au CA de *The Linux Foundation*, un consortium dont les objectifs sont de promouvoir, protéger et standardiser Linux.
- Membre du CA de *CE Linux Forum*, une communauté internationale de développeurs dont le but est de trouver des solutions afin d'améliorer l'embarqué sous Linux.
- Occupe la fonction de Président au sein de la Software in the Public Interest.

Vie professionnelle

- *Open Source & Linux Chief Technologist* chez HP.

« With the kinds of phenomenal growth rates we're seeing, Linux and the whole of open source is one of the brighter stars in the IT sky right now. »

deb

Liens

Le responsable de projet Debian : **http://www.debian.org/devel/leader**

Programme du développeur français Sam Hocevar : **http://www.debian.org/vote/2007/platforms/sho.fr.html**

Programme du dernier *Debian Project Leader* : **http://www.debian.org/vote/2010/platforms/zack**

Élections 2010 du DPL : **http://www.debian.org/vote/2010/vote_001**

MARTIN MICHLMAYR

MARS 2003 – MARS 2005

Son rôle dans le logiciel libre

- A joué le rôle de conseiller auprès de la Software in the Public Interest.
- A été *Publicity director* au sein de Linux International.
- Membre du CA de l'Open Source Initiative.

Vie professionnelle

- *Open Source Community Expert* chez HP.

« There's one thing I'd like to add, something we regularly do in programming, but what we sometimes seem to forget in communication: Be liberal in what you accept, and conservative in what you send. »

BRANDEN ROBINSON

AVRIL 2005 – AVRIL 2006

Son rôle dans le logiciel libre

- Fondateur de *X Strike Force*, un groupe de travail responsable du maintien des paquets X Window System pour Debian.
- A occupé la fonction de trésorier au sein de la SPI.

Vie professionnelle

- *Software Engineer* chez Cisco Systems.

« Hmm, I always thought my best moments were frivolous one-liners. »

ANTHONY TOWNS

AVRIL 2006 – AVRIL 2007

Son rôle dans le logiciel libre

- A contribué au développement de *Debian Bug Tracking System*.
- Fut à l'origine de l'intégration de la version *testing* au sein du développement de Debian.
- A occupé les fonctions de trésorier et de secrétaire au sein de Linux Australia, une organisation regroupant l'ensemble des LUG d'Australie.
- Membre actif de HUMBUG (*Home Unix Machine Brisbane User Group*), un groupe d'utilisateurs Unix.

Vie professionnelle

- Fondateur de *Thinkers Garden*, un « bac à sable » d'idées en tous genres.
- Consultant indépendant pour Erisian Consulting.

« Ah, what I love about Debian is just how _rewarding_ it is to contribute. »

SAM HOCEVAR

AVRIL 2007 – AVRIL 2008

Son rôle dans le logiciel libre

- A contribué à Wikimédia France et Wikipédia.
- Développeur au sein du projet VideoLAN.
- Auteur de la seconde version de WTFPL (*Do What The Fuck You Want To Public License*), une licence libre très permissive.

Vie professionnelle

- Ingénieur R&D senior au sein de DONTNOD Entertainment, studio de développement de jeux vidéos.

« Dear Debian,
Roses are red,
Violets are purple,
It is today time I said
I candidate for DPL! »

STEVE MCINTYRE

AVRIL 2008 – AVRIL 2010

Son rôle dans le logiciel libre

- Membre de *Electronic Frontier Foundation*, une organisation internationale défendant la liberté d'expression sur le Web.
- Occupe la fonction de trésorier au sein de l'association *Debian UK Society*.
- A développé JTE (*Jigdo Template Export*), un outil facilitant le téléchargement de fichiers importants sur le Web.

Vie professionnelle

- *Staff Software Engineer* chez ARM.

« One thing I will commit to (right now) is to encourage people to ignore (or even better, castigate) nay-sayers who have nothing more to contribute to Debian than poisonous tabloid-style rhetoric and negativity. »

STEFANO ZACCHIROLI

AVRIL 2010 - AUJOURD'HUI

Son rôle dans le logiciel libre

- Participe au projet MANCOOSI (*MANagement the COmplexity of the Open Source Infrastructure*) visant à améliorer la mise à jour de logiciels libres.
- Co-rédacteur de la politique Debian entourant le langage de programmation Objective Caml.
- A été membre du projet MoWGLI (*Mathematics On the Web: Get it by Logic and Interfaces*), favorisant la publication de documents mathématiques sur le Web.

Vie professionnelle

- Chercheur au laboratoire Preuves, Programmes et Systèmes de l'Université Paris Diderot dans le cadre du projet MANCOOSI.

« Oh gosh, I didn't realize how embarrassing can be to send the "nominate myself" post until now :) »

ian

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 grâce à **fakeroot** ou **sudo**. Nous allons ici utiliser directement l'identité **root**. En effet, la logique veut qu'une telle manipulation soit 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** serait de se prémunir contre d'éventuelles erreurs de manipulations. Ce qui ne doit de toute façon pas arriver lorsque le super-utilisateur reconstruit un noyau (hé oui). De plus, nous travaillerons dans le répertoire **/usr/src** qui n'est pas autorisé en écriture pour le commun des mortels. Vous pouvez cependant choisir une autre méthode consistant à désarchiver les sources du noyau dans un sous-répertoire de votre **$HOME** et utiliser un passage en **root** ponctuellement. Ceci ne sera pas traité ici.

Commencez par installer les trois paquets cités plus haut (ici, **linux-source-2.6.32**) puis placez-vous dans le répertoire **/usr/src** où vous trouverez **linux-source-2.6.32.tar.bz2**. Contrairement à ce que le nom du fichier laisse penser, il ne s'agit pas des sources vanilla (source d'origine) du noyau Linux, mais d'une version spécifique à Debian. Celle-ci respecte les DFSG (*Debian Free Software Guidelines* ou Principes du logiciel libre selon Debian) et a donc été modifiée en conséquence. Notez que ces modifications sont principalement des correctifs de sécurité et des ajouts validés par les développeurs du noyau, mais non encore disponibles.

Décompressez et désarchivez ces sources avec **tar xfvj** directement dans **/usr/src**. Vous obtiendrez ainsi une arborescence dans **linux-source-2.6.32**. Placez-vous ensuite 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. En d'autres termes, nous partons du principe que votre système fonctionne déjà sur un 2.6.32 (**2.6.32-4-686** en l'occurrence) installé depuis les dépôts officiels Debian. Nous souhaitons conserver ce noyau et donc, avoir deux versions différentes du 2.6.32 en cas de problème. Vous remarquerez, par exemple, que les modules installés par un paquet officiel sont placés dans **/lib/modules/2.6.32-4-686** qui correspond au nom du paquet **linux-image***. Le suffixe **-4-686** est ajouté par le responsable du paquet Debian précisément pour permettre aux utilisateurs de construire facilement leur noyau sans conflit et, bien entendu, pour distinguer les différentes préférences de construction. Ici, nous avons un noyau 2.6.32 en révision 4, compilé pour architecture 686.

Notre première manipulation sera absolument didactique puisque nous allons nous contenter de 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-2.6.32-4-686** en **.config** dans le répertoire des sources du noyau (répertoire courant donc), puis 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. Un coup d'œil au fichier **README** inclus dans **kernel-package** et on constatera qu'on nous recommande d'utiliser **make menuconfig**. En toute logique, si vous recompilez un noyau, c'est pour apporter des modifications à sa configuration et donc procéder manuellement. L'étape que nous opérons là est un simple raccourci pour partir, non pas sur une base vierge, mais sur la configuration par défaut de Debian.

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 à la construction et la prise en charge du RAMdisk d'initialisation lors de l'installation du paquet (le fameux **update-initramfs**). Si, dans le cas présent, vous oubliez cette option, le système sera très certainement incapable de démarrer. Le support IDE/SATA/SCSI est, par défaut, compilé sous forme de modules et c'est le bootloader GRUB Legacy ou GRUB2 qui charge le fichier image depuis le disque. Après un délai dépendant à la fois de la rapidité de votre ou vos processeurs et de la mémoire disponible, vous obtiendrez le fichier **linux-image-2.6.32_2.6.32-10.00.Custom_i386.deb** dans **/usr/src** (le répertoire parent de là où est lancée la construction, exactement comme pour un paquet classique). C'est là, bien sûr, le paquet Debian correspondant à votre noyau fraîchement compilé que vous pourrez installer avec **dpkg -i**.

Après installation de ce paquet tout frais, vous disposez de deux noyaux 2.6.32. Le premier installé depuis le dépôt Debian et compilé par le responsable officiel (**2.6.32-4.686**) et le votre :

```
% dpkg -l "linux-image-2.6.32*"
linux-image-2.6.32
  2.6.32-10.00.Custom
  Linux kernel binary image for version 2.6.32
linux-image-2.6.32-4-686
  2.6.32-11
  Linux 2.6.32 for modern PCs
```

Nous allons maintenant construire un troisième noyau, mais cette fois dans une version personnalisée. Avant toutes choses, une première étape consiste à nettoyer les sources et les informations données résultant dans l'utilisation du précédent **make-kpkg** avec un simple **make-kpkg clean** dans le répertoire **/usr/src/linux-source-2.6.32**. Vous pouvez alors utiliser deux options très importantes de **make-kpkg** :

- **–revision** est destinée au système de paquets. Par défaut, la révision utilisée est **$(version)-10.00.Custom**. C'est le numéro de version du noyau plus une chaîne de caractères personnalisée. 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 d'une version déjà

Note : pas d'initramfs ?

Votre installation du nouveau noyau n'a pas provoqué la génération de l'image du RAMdisk initial ? Jetez un œil dans les scripts de post-installation placés dans **/etc/kernel/postinst.d**. Il est probable que vous y trouviez un fichier **initramfs-tools**. Ce dernier, selon la version du paquet **initramfs-tools**, peut inclure la condition suivante :

```
# kernel-package passes an extra arg;
# hack to not run under kernel-package
[ -z "$2" ] || exit 0
```

Ce script, cependant, n'est pas compatible à l'heure actuelle avec la structure mise en place par Debian, selon la version de la distribution que vous utilisez (Bug#523735). En effet, comme les arguments passés en paramètres sont **2.6.32** et **/boot/vmlinuz-2.6.32**, le script **update-initramfs** n'est jamais lancé :

```
# we're good - create initramfs.
# update runs do_bootloader
update-initramfs -c -t -k "$1"
```

L'une des solutions proposées dans le fil de discussion découlant du rapport de bug est de remplacer ce script par celui présent dans **/usr/share/kernel-package/examples/etc/kernel/postinst.d/initramfs**, bien plus complet. Ceci rejoint les recommandations faites dans les **README.gz** de **kernel-package** :

```
Also, make sure you have configured /etc/kernel-img.conf
and also /etc/kernel/*.d by dropping in the correct
scripts there. (symlink creation, initramfs generation,
running boot loaders, etc)
```

Vous pouvez localement contourner le problème en utilisant directement **update-initramfs -c -t -k 2.6.32**. Utilisez l'option **-v** pour vous assurer que les modules utilisés dans la future image soient bien ceux de **/lib/modules/2.6.32** et non d'un autre noyau.

installée. Ce numéro de révision doit être composé uniquement de caractères alphanumériques et des caractères **~**, **+** et **.**. De plus, il doit impérativement comporter un chiffre.

- **--append-to-version** vous permet d'influer sur la version du noyau (par opposition à sa « révision ») et donc sur les chemins 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 une valeur qui devrait être un argument de l'option précédente, vous allez créer un conflit avec le paquet noyau déjà installé. Ils pourraient ainsi partager, en effet, tous deux le même répertoire de modules, ce qui n'est ni souhaitable ni fonctionnel. Ce n'est pas tout, le même nom sera utilisé pour l'image initramfs et pour le fichier du noyau dans **/boot**. Clairement, vous irez au devant de gros problèmes en confondant les deux options. Ce numéro de version ne peut contenir que des caractères alphanumériques minuscules et les caractères **~**, **-**, **.** et **+**.

Relancez donc la configuration du noyau et apportez vos modifications, comme la désactivation des fonctionnalités qui ne vous servent pas et ne vous serviront jamais (PCMCIA, ISDN, FireWire, etc.). Relancez ensuite la construction avec :

```
% make-kpkg clean && make-kpkg \
--append-to-version=-hs48-1-686 \
--initrd --revision=1.0.lef kernel_image
[...]
répertoire " /usr/src/linux-source-2.6.32 "
make[1]: quittant le répertoire
  " /usr/src/linux-source-2.6.32 "
echo done > debian/stamp/conf/minimal_debian
exec debian/rules  DEBIAN_REVISION=1.0.lef
  APPEND_TO_VERSION=-hs48-1-686
  INITRD=YES  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-2.6.32-hs48-1-686_1.0.lef_i386.deb**. Installez-le comme à l'habitude puis observez la sortie de la commande **dpkg -l** :

```
linux-image-2.6.32
  2.6.32-10.00.Custom
  Linux kernel binary image for version 2.6.32
linux-image-2.6.32-4-686
  2.6.32-11
  Linux 2.6.32 for modern PCs
linux-image-2.6.32-hs48-1-686
  1.0.lef
  Linux kernel binary image for version
    2.6.32-hs48-1-686
```

Nous retrouvons nos trois noyaux 2.6.32. Il en va de même pour le contenu de **/lib/modules** :

```
2.6.32/
2.6.32-4-686/
2.6.32-hs48-1-686/
```

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 intégrant le support de tout votre matériel pour un serveur et ensuite utiliser **make-kpkg** en oubliant l'option **--initrd**. Pensez simplement à bien structurer vos numéros de version et de révision si vous souhaitez tester plusieurs configurations pour un même noyau.

2 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 noyau Linux, ni dans les sources installées par Debian dans **/usr/src**. C'est le cas, par exemple, du support pour les capteurs LIRC (**lirc-modules-source**), des webcams (**gspca-source**), certains systèmes de fichiers (**squashfs-source** ou **openafs-modules-source**) ou encore des modems (**sl-modem-source**). Les raisons de la non-inclusion dans les sources du noyau peuvent être variées :

- code trop « sale » ;
- problèmes de compatibilité avec Linux ou un responsable d'une branche du noyau ;
- problèmes liés aux licences ou à l'inclusion d'objets binaires propriétaires (blob) comme des *firmwares* ;
- ou encore développement délibéré en marge du projet Linux.

Quoi qu'il en soit, Debian fournit quelques-uns de ces modules en version source et tantôt binaire. Dans ce dernier cas, 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 versions 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 car installé dans le « mauvais » **/lib/modules**. Il faut également comprendre qu'il est plus facile pour Debian de fournir un paquet source pour le module que des paquets binaires qu'il faut alors décliner dans toutes les versions du noyau possibles.

Note : oups, trop tard

Nous allions ici nous servir du module permettant le support de SquashFS (**squashfs-source**). Ce système de fichiers compressé et en lecture seule est utilisé pour de nombreux live CD comme ceux de Debian, Gentoo, Ubuntu, Fedora, ... Il est également utilisé dans les firmwares OpenWRT et DD-WRT sur certains systèmes embarqués. L'utilisation d'un système de fichiers SquashFS est généralement complété par un autre système de fichiers de type union, comme UnionFS ou aufs. Ceci permet d'obtenir une base de système en lecture seule avec enregistrement des modifications par ailleurs. Le tout étant alors transparent pour l'utilisateur qui pense avoir affaire à un système de fichiers en lecture/écriture tout à fait classique. Malheureusement (ou pas), ce système de fichiers est pris en charge maintenant directement par le noyau 2.6.32 tel que disponible pour Debian GNU/Linux. Ce qui nous mène à mettre le doigt sur un point important. Le paquet **squashfs-source** étant toujours disponible (mémo personnel : penser à faire le ménage dans **/etc/apt/***), il n'est cependant pas utile. On notera par ailleurs que sa construction échouera. Si vous rencontrez ce type de problème, voyez large et ne focusez pas uniquement sur le paquet et les erreurs de compilation. Il est possible que d'autres paquets (dont celui du noyau) aient évolué et le rendent obsolète.

Notre exemple de construction de module se basera sur le support SMAPI via le module **tp_smapi**. Celui-ci, spécifique aux ordinateurs IBM/Lenovo de type ThinkPad, permet de rendre accessibles via SYSFS un certain nombre d'informations matérielles. L'objectif principal des développeurs de **tp_smapi** est de « remonter » des informations sur les batteries du portable et, accessoirement, d'améliorer le support HDAPS de protection des disques durs en cas de chute (plus d'info sur http://tpctl.sourceforge.net).

Le choix de ce support en guise d'exemple présente un avantage pédagogique. Le paquet source de ce module se décline, en effet, en plusieurs versions qui implémentent plusieurs méthodes de construction différentes, que nous allons voir respectivement.

2.1 La méthode du barbare : make-kpkg

La première méthode, classique, consiste en l'installation du paquet **tp-smapi-modules**. Ceci aura pour effet d'installer une archive dans **/usr/src**, qu'il vous suffira de décompresser :

```
% tar xfjv tp-smapi.tar.bz2
modules/
modules/tp-smapi/
modules/tp-smapi/CHANGES
modules/tp-smapi/README
modules/tp-smapi/hdaps.c
modules/tp-smapi/thinkpad_ec.h
modules/tp-smapi/debian/
modules/tp-smapi/debian/compat
modules/tp-smapi/debian/README.Debian
modules/tp-smapi/debian/copyright
modules/tp-smapi/debian/rules
modules/tp-smapi/debian/sysfs.conf
modules/tp-smapi/debian/control
modules/tp-smapi/debian/control.modules.in
modules/tp-smapi/debian/changelog
modules/tp-smapi/Makefile
modules/tp-smapi/tp_smapi.c
modules/tp-smapi/thinkpad_ec.c
```

L'opération va alors créer un répertoire **modules** si celui-ci n'existe pas encore et y placer les sources de notre module. Entrez alors dans **/usr/src/linux-source-2.6.32** et utilisez :

```
% make-kpkg --append-to-version=-hs48-1-686 \
--revision=1.0.lef modules_image
[...]
exec debian/rules
 DEBIAN_REVISION=1.0.lef
  APPEND_TO_VERSION=-hs48-1-686
  modules_image
[...]
make[1]: quittant le répertoire
  " /usr/src/modules/tp-smapi "
Module /usr/src/modules/tp-smapi
  processed fine
```

Notez que nous devons réutiliser **--append-to-version** et **--revision**. Une fois la compilation terminée, vous trouverez un fichier **tp-smapi-modules-2.6.32-hs48-1-686_0.40-7+1.0.lef_i386.deb** dans **/usr/src**. Il s'installera, comme précédemment, avec **dpkg -i**. On retrouvera les modules compilés dans **/lib/modules/2.6.32-hs48-1-686/extra/**. Enfin, avec **dpkg -l**, on listera bien notre nom de paquet **tp-smapi-modules-2.6.32-hs48-1-686** et sa version **0.40-7+1.0.lef**, le tout correspondant avec nos arguments passés précédemment.

Notez bien que, 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 déjà installé, vous devez utiliser l'option **--append-to-version=** suivie d'une partie de la valeur retournée par **uname -r** (**-4-686**, 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 miser sur une configuration homogène et qui ne laisse pas planer de doute sur la responsabilité d'un développeur Debian (via un rapport de bug, par exemple).

2.2 La méthode du ranger : module-assistant

Si vous avez trouvé que la méthode d'installation précédente était somme toute relativement simple, vous allez être surpris. En effet, même si elle s'inscrit dans la suite logique de la compilation d'un nouveau noyau, presque plus personne ne l'utilise.

Le **module-assistant** est un outil en ligne de commandes écrit par Eduard Bloch. Il a pour but de simplifier la construction de paquets pour les modules présents sous la forme de sources. Le principal avantage qu'il offre est d'automatiser une grande partie des traitements et de limiter les dépendances nécessaires. Ainsi, pour un certain nombre de modules, les sources du noyau ne sont même pas nécessaires. La simple présence d'un paquet **linux-headers-2.6.*** correspondant à votre noyau peut être suffisante. Comme le précise la page de manuel, le **module-assistant** nécessite, en effet, un certain nombre de prérequis. Toutefois, il est conçu pour l'utilisateur « standard » et est en mesure de simplifier également ces dépendances (« Si vous ne comprenez rien au texte ci-dessus, exécutez m-a prepare[...] », dixit la page de manuel de **module-assistant**).

Dans le cas de notre module pour ThinkPad, après installation du paquet **module-assistant** (et nettoyage des répertoires suite aux opérations précédentes), nous n'avons qu'à faire ceci :

```
% m-a prepare tp-smapi
Récupération des sources du noyau
  de la version : 2.6.32-4-686
En-têtes du noyau disponibles dans
  /lib/modules/2.6.32-4-686/build
Création du lien symbolique...
apt-get install build-essential
[...]
```

```
% m-a -t a-i tp-smapi
[...]
unpack
Extracting the package tarball,
  /usr/src/tp-smapi.tar.bz2, please wait...
"/usr/share/modass/overrides/tp-smapi-source"
   build KVERS=2.6.32-4-686
   KSRC=/lib/modules/2.6.32-4-686/build
   KDREV=2.6.32-11 kdist_image
dh_testdir
[...]
  CC [M]  /usr/src/modules/tp-smapi/thinkpad_ec.o
  CC [M]  /usr/src/modules/tp-smapi/tp_smapi.o
  CC [M]  /usr/src/modules/tp-smapi/hdaps.o
  Building modules, stage 2.
  MODPOST 3 modules
  CC      /usr/src/modules/tp-smapi/hdaps.mod.o
  LD [M]  /usr/src/modules/tp-smapi/hdaps.ko
  CC      /usr/src/modules/tp-smapi/thinkpad_ec.mod.o
  LD [M]  /usr/src/modules/tp-smapi/thinkpad_ec.ko
  CC      /usr/src/modules/tp-smapi/tp_smapi.mod.o
  LD [M]  /usr/src/modules/tp-smapi/tp_smapi.ko
[...]
dpkg -Ei /usr/src/tp-smapi-modules-2.6.32-4-686
_0.40-7+2.6.32-11_i386.deb
Sélection du paquet tp-smapi-modules-2.6.32-4-686
  précédemment désélectionné.
Dépaquetage de tp-smapi-modules-2.6.32-4-686
   (à partir de .../tp-smapi-modules-2.6.32-4-686
_0.40-7+2.6.32-11_i386.deb) ...
Paramétrage de tp-smapi-modules-2.6.32-4-686
(0.40-7+2.6.32-11) ...
```

C'est tout ! Votre module a été :

- décompressé ;
- configuré ;
- compilé ;
- testé ;
- empaqueté ;
- et installé.

La ligne de commandes **m-a -t a-i tp-smapi** est délibérément trompeuse. On aurait tout aussi bien pu utiliser ici **module-assistant auto-install tp-smapi**. L'option **-t** est une préférence personnelle, elle permet de conserver une sortie en mode texte brut et non, comme c'est le cas par défaut, d'utiliser une interface texte colorée reposant sur **dialog**.

module-assistant est un outil auquel on prend rapidement goût et les responsables des paquets de modules pour le noyau ne s'y trompent pas. Bien souvent, un simple coup d'œil dans le **README.Debian** installé par un de ces paquets sources et vous trouverez mention de **m-a**. Sachez également que le **module-assistant** est capable de prendre en argument un nom de module à récupérer/installer et même de lister les modules disponibles, installés et compilés. Jetez un œil à la page de manuel ou lancez simplement **m-a** sans argument pour obtenir une interface texte à menus que vous pourrez explorer à loisir.

2.3 La méthode de l'elfe : DKMS

Voilà, vous voici décidé. Le **module-assistant** est simple d'utilisation. C'est la voie que vous prendrez ? Attendez ! Vous n'êtes pas au bout de vos surprises. Certains développeurs trouvent que cette solution est encore trop contraignante pour les utilisateurs et proposent une alternative. Il s'agit du *Dynamic Kernel Module Support Framework*, un framework automatisant l'installation de modules. Je rappelle ici qu'il n'existe que deux solutions pour une distribution de modules : soit il existe une version pour votre noyau et vous l'installez, soit vous recompilez. Et pourquoi pas les deux en une seule commande ?

C'est ce que se proposent de faire DKMS et les paquets compatibles. Ils sont certes encore peu nombreux, mais notre module pour ThinkPad est dans la liste : **tp-smapi-dkms**. Pour obtenir un module pour votre système, il suffit d'un **aptitude install** :

```
Les NOUVEAUX paquets suivants
vont être installés :
dkms{a} tp-smapi-dkms
[...]
Paramétrage de dkms (2.1.1.2-2) ...
Paramétrage de tp-smapi-dkms (0.40-7) ...

Creating symlink
/var/lib/dkms/tp-smapi/0.40/source ->
   /usr/src/tp-smapi-0.40

DKMS: add Completed.
Kernel preparation unnecessary for
  this kernel.  Skipping...

Building module:
cleaning build area....
make KERNELRELEASE=2.6.32-4-686 -C
 /lib/modules/2.6.32-4-686/build
  M=/var/lib/dkms/tp-smapi/0.40/build.....
cleaning build area....

DKMS: build Completed.
[...]
tp_smapi.ko:
Running module version sanity check.
- Original module
   - No original module exists
     within this kernel
- Installation
   - Installing to /lib/modules/2.6.32-4-686/updates/dkms/
[...]
depmod.......

DKMS: install Completed.
[...]
```

Fini ! Le framework s'occupe de tout, de la détection de la version à l'installation du module, en passant par la vérification des dépendances de compilation et la compilation elle-même. Il n'y a même pas de nouveau paquet dans la liste, **tp-smapi-dkms** se suffisant à lui-même. Mieux encore, contrairement à ce que propose le **module-assistant**, la mise à jour se fera automatiquement en même temps que celle du paquet. Vous n'avez tout simplement plus rien à faire.

Si vous êtes curieux de connaître le fonctionnement interne, rien de plus simple. L'installation du paquet source de **tp-smapi-dkms** (**apt-get source**) installera le paquet source de **tp-smapi**. C'est en effet les mêmes sources Debian qui créeront les deux paquets (avec et sans DKMS). Vous aurez alors tout loisir d'inspecter le fonctionnement du fichier **debian/rules**.

2.4 Friandise, juste pour montrer que ça marche

Après toutes ces manipulations, nous méritons bien une petite récompense. Notre module **tp-smapi** fait parfaitement son travail et nous apprend plein de choses sur la batterie présente dans mon vieux X31 :

```
% cd /sys/devices/platform/smapi/BAT0
% cat model
IBM-X23
% cat barcoding
1Z75M48TR40
% cat manufacture_date
2007-11-15
% cat first_use_date
2009-11-06
% cat cycle_count
31
% cat chemistry
LION
```

Idéal pour vérifier la provenance, la qualité et l'état d'une batterie achetée sur un site d'enchères en ligne ou livrée avec un lappy d'occasion. Non ?

CONCLUSION

Comme précisé en début d'article, la recompilation du noyau, mais également de tous paquets binaires, doit être motivée. 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 d'une qualité exemplaire.

Si vous comptez vous pencher davantage sur la construction de noyaux et passer du temps à expérimenter les différents 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 bugs. Vos résultats, vos problèmes et vos solutions peuvent être utiles, tout comme le temps que vous consacrerez à la tâche. ■

Mise à jour d'une installation

Le chiffrement de disques via dm-crypt ou loop-aes est proposé dans la procédure d'installation classique de Debian. Chiffrer son disque dur, qu'il s'agisse de celui où réside le système ou d'une unité annexe, présente bien des avantages. Le couple LVM + chiffrement fait ainsi véritablement des merveilles. Mais qu'en est-il lorsqu'il s'agit d'ajouter un nouveau support de stockage ?

Auteur

- Denis Bodor

Le chiffrement de disques est généralement utilisé pour les machines nomades comme les laptops. En effet, il peut être relativement difficile de penser qu'on puisse voler une machine de bureau. De la même manière, les unités de disque USB ou FireWire nécessitent également instinctivement ce genre de protection. Ces deux utilisations classiques du chiffrement limitent les problématiques d'évolution. On ne se posera, en effet, pas vraiment de question lorsqu'il s'agira d'accéder à un peu plus d'espace. La solution consistera, en toute logique, en un changement de disque suivi d'une copie.

Malheureusement, sous-estimer la ténacité et la témérité d'éventuels voleurs est une erreur qu'on ne commet qu'une fois. Ajoutons à cela un brin de paranoïa tout à fait naturel chez le sysadmin et se fera alors immédiatement sentir le besoin de chiffrer les disques d'une machine de développement, de bureau ou personnelle. Je parle ici de machines dites *desktop* (des trucs lourds). Un cambriolage dans une maison ou un appartement, où la hifi et les couverts en argent hérités de grand-mère disparaissent, n'est somme toute qu'une affaire d'assurance. Les données personnelles, en fonction de ses activités, peuvent avoir une bien plus grande valeur pour qui saura en faire un usage illicite.

1 PLANTONS LE DÉCOR

Très judicieusement, vous avez donc installé votre distribution préférée avec les options de chiffrement et de gestion de volumes (LVM). Vous avez donc ainsi divisé votre seul et unique disque comme ceci :

```
% cat /proc/partitions
major minor  #blocks  name
   8     0   39062500 sda
   8     1    1951866 sda1
   8     2   37110150 sda2
 254     0   37109122 dm-0
 254     1    3706880 dm-1
 254     2    2146304 dm-2
 254     3   31252480 dm-3
```

Nous sommes ici en présence d'un disque **sda** comprenant deux partitions. La première, **sda1**, contient un système de fichiers ext3 monté en **/boot** afin que le *bootloader* puisse charger le noyau et l'initramfs pour ensuite déchiffrer le contenu de **sda2** et retrouver les volumes LVM.

Le déchiffrement du contenu de **sda2** repose sur le contenu de **/etc/crypttab** :

```
sda2_crypt /dev/sda2 none luks
```

L'ouverture du périphérique via LUKS, après saisie correcte d'une phrase de passe, donne naissance à un nouveau périphérique bloc **/dev/mapper/sda2_crypt** (et **/dev/dm-0**). C'est précisément ce périphérique (254:0) qui sera alors intégré au groupe de volumes (VG) utilisé par le système :

```
% vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "lefbase" using metadata type lvm2

% vgs
  VG      #PV #LV #SN Attr   VSize  VFree
  lefbase   1   3   0 wz--n- 35,39G    0
```

Nous avons ici un groupe **lefbase** dont nous pouvons voir la composition :

```
% pvscan
  PV /dev/dm-0   VG lefbase   lvm2 [35,39 GB / 0    free]
  Total: 1 [35,39 GB] / in use: 1 [35,39 GB] / in no VG: 0 [0   ]

% pvs
  PV         VG      Fmt  Attr PSize  PFree
  /dev/dm-0  lefbase lvm2 a-   35,39G    0
```

Cet affichage des volumes physiques (PV) en présence montre effectivement l'utilisation de **/dev/dm-0** dans le

chiffrée

groupe de volumes (VG) **lefbase**. Nous pouvons pousser plus loin en affichant les volumes logiques (LV) qui se partagent ce groupe :

```
% lvscan
  ACTIVE            '/dev/lefbase/home' [3,54 GB] inherit
  ACTIVE            '/dev/lefbase/swap' [2,05 GB] inherit
  ACTIVE            '/dev/lefbase/racine' [29,80 GB] inherit

% lvs
  LV     VG      Attr   LSize  Origin Snap%  Move Log Copy%  Convert
  home   lefbase -wi-ao  3,54G
  racine lefbase -wi-ao 29,80G
  swap   lefbase -wi-ao  2,05G
```

La commande **lvscan** nous montre ainsi trois volumes logiques : **home**, **racine** et **swap**, respectivement utilisés pour le *filesystem* monté dans **/home**, à la racine du système et utilisé comme espace de swap. En y regardant de plus près, on remarque que le contenu de **/dev/lefbase**, notre groupe de volumes, contient des liens symboliques vers les entrées du *device mapper* **/dev/mapper**.

```
% ls -l /dev/lefbase
total 0
lrwxrwxrwx 1 root root 24 avr 19 16:40 home -> /dev/mapper/lefbase-home
lrwxrwxrwx 1 root root 26 avr 19 16:40 racine -> /dev/mapper/lefbase-racine
lrwxrwxrwx 1 root root 24 avr 19 16:40 swap -> /dev/mapper/lefbase-swap
```

C'est d'ailleurs ces entrées qui sont utilisées comme périphériques contenant les filesystems montés par le système :

```
</code> % mount | grep mapper /dev/mapper/lefbase-racine on / type ext3
(rw,errors=remount-ro) /dev/mapper/lefbase-home on /home type ext3 (rw)
</code>
```

Enfin, on remarquera que le device mapper met également en place des entrées dans **/dev/** :

```
% ls -l /dev/mapper
total 0
crw-rw---- 1 root root  10, 60 avr 19 16:40 control
brw-rw---- 1 root disk 254,  1 avr 19 16:40 lefbase-home
brw-rw---- 1 root disk 254,  3 avr 19 16:40 lefbase-racine
brw-rw---- 1 root disk 254,  2 avr 19 16:40 lefbase-swap
brw-rw---- 1 root disk 254,  0 avr 19 16:40 sda2_crypt

% ls -l /dev/dm-*
brw-rw---- 1 root disk 254, 0 avr 19 16:40 /dev/dm-0
brw-rw---- 1 root disk 254, 1 avr 19 16:40 /dev/dm-1
brw-rw---- 1 root disk 254, 2 avr 19 16:40 /dev/dm-2
brw-rw---- 1 root disk 254, 3 avr 19 16:40 /dev/dm-3
```

On notera l'utilisation de majeurs/mineurs identiques pour les pseudo-fichiers de **/dev/** et **/dev/mapper**.

2 LE PROBLÈME

Incidieusement, le problème qui se pose à nous a déjà été affiché dans les résultats des commandes présentées ci-avant. Regardez bien. Nous avons un système de fichiers utilisé comme **/home** de seulement 3.5 Go et une racine (**/usr** et **/var** inclus) de 30 Go. Un choix qui pouvait s'avérer judicieux au moment de l'installation du système mais qui, à présent, limite grandement l'utilisation de la machine. Bien entendu, ici, l'exemple est construit de toutes pièces, mais illustre parfaitement l'intérêt de LVM sur n'importe quel type d'installation.

En temps normal, la correction du problème reviendrait simplement à ajouter un disque, le partitionner, intégrer le volume physique dans un groupe et étendre les volumes logiques de son choix. Mais ici, nous travaillons avec plusieurs « couches » de *device mapping*. Ce n'est pas directement **sda2** qui est utilisé dans le groupe de volumes, mais le périphérique bloc issu de l'interprétation (et le déchiffrement) par la couche cryptographique LUKS/dm-crypt.

L'opération sera donc sensiblement plus délicate.

3 EN AVANT !

Nous venons d'ajouter un disque dans notre configuration matérielle. Celui-ci apparaît sous l'ID **ata-Maxtor_6Y120L0_Y341YEGE** et l'entrée **/dev/sdb**. Nous nous empressons donc de créer une partition sur ce disque. Ici, à des fins de tests et pour accélérer les manipulations, nous n'utiliserons pas la totalité du disque de 120 Go, mais uniquement une petite partie. Bien entendu, il convient de créer une partition couvrant l'ensemble du disque dans le cas d'une utilisation de l'espace dans son ensemble.

Une fois cette partition **/dev/sdb1** créée, il nous reste à transformer ses données en données chiffrées avec la commande **cryptsetup**, comme nous le ferions pour un disque externe, par exemple :

```
% cryptsetup --verbose --verify-passphrase luksFormat /dev/sdb1

WARNING!
========
This will overwrite data on /dev/sdb1 irrevocably.
Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase: ******
Verify passphrase: ******
Command successful.
```

Cette opération détruira toutes les données en présence et, dès lors, le disque (ou plutôt la partition) n'est plus utilisable en tant que tel. Nous pouvons maintenant ouvrir le périphérique en question avec :

```
% cryptsetup luksOpen /dev/sdb1 sdb1_crypt
Enter LUKS passphrase:
key slot 0 unlocked.
Command successful.
```

La commande prend en argument le nom utilisé pour le mapping des données déchiffrées. On retrouve alors ce nom sous la forme d'une nouvelle entrée dans **/dev/mapper** ainsi que sous **/dev/dm-4**. Afin d'automatiser l'ouverture du périphérique LUKS au démarrage, nous ajoutons une ligne à notre **/etc/crypttab** pour le transformer en ceci :

```
sda2_crypt /dev/sda2 none luks
sdb1_crypt /dev/sdb1 none luks
```

A ce stade, il est possible de vous assurer de la bonne marche des choses et du bon fonctionnement de la configuration en testant via un redémarrage effectif du système. Ceci n'est pas nécessaire, mais il arrive malheureusement trop souvent qu'une configuration soit parfaitement fonctionnelle en l'état, mais ne « survive » pas à un (re)démarrage faute de configuration adéquate. Prenez le temps de regarder où vous mettez les pieds. Mais avant de rebooter le système, il paraît logique de prendre en compte les changements effectués dans la configuration au niveau de votre initramfs. On supposera, en effet, lors de la première phase de boot, que le système sera incapable de savoir comment utiliser **sdb1** si le fichier **/etc/crypttab** est uniquement stocké sur le disque qui est chiffré (justement).

```
% update-initramfs -u
update-initramfs: Generating /boot/initrd.img-2.6.26-2-686
```

Suite au redémarrage et après avoir saisi les deux phrases de passe, votre système doit présenter un **/dev/mapper/sdb1_crypt**. Le contenu en version non chiffrée est disponible via cette entrée. C'est donc celle-ci qui sera utilisée pour intégrer notre unique groupe de volumes LVM. Cependant, si vous avez été attentif à la progression du démarrage du système, quelque chose vous a sans doute mis la puce à l'oreille car...

Note : partition ou *raw device* ?

Utiliser un disque entier sans aucune partition est possible aussi bien pour un système de fichiers que pour une unité chiffrée ou un volume physique. C'est cependant une très mauvaise idée car, dans ce cas, rien ne permet de différencier un disque utilisé d'un disque totalement vierge et ce, aussi bien pour un système GNU/Linux que pour un Windows ou un *BSD.

Dans le cas de LVM, la présence des métadonnées sur le disque n'est pas suffisante pour témoigner de l'utilisation du disque pour un outil autre que ceux de LVM. **fdisk**, par exemple, considérera le disque comme exempt de données et vous pourriez le partitionner à tort.

Enfin, notez qu'un type **8e Linux LVM** existe et permet de désigner les partitions dédiées à LVM. Chose qui n'est cependant pas applicable ici, puisque notre LVM est monté sur des mapping et non sur des disques physiques.

4 BAAM !

C'est un peu comme paf le chien ou flipflap la girafe. Encore un pas et c'est la catastrophe, le « baam initramfs ! ». En effet, ce premier redémarrage n'était pas innocent. Il nous permet de voir comment fonctionnent les scripts présents dans l'initramfs et les différents fichiers de configuration utilisés.

La partie inquiétante du démarrage est la manière dont se succèdent les demandes de phrases de passe. Nous avons, dans un premier temps, la tentative d'ouverture de **sda2** et, plus tard, celle pour **sdb1**. En effet, la première est gérée par les scripts de l'initramfs tandis que la seconde a lieu alors que le système a basculé sur le système de fichiers racine LVM+Crypto. Voyez-vous le problème ?

Si nous ajoutons, dans l'état, le disque **sdb1_crypt** dans le groupe de volumes **lefbase**, ce dernier ne pourra être utilisé au démarrage du système car incomplet. Notre système ne démarrera donc tout simplement pas. Il faut donc être prudent et regarder de près comment fonctionne l'ensemble de l'image initramfs.

On constate alors qu'un fichier spécifique est utilisé pour « débloquer » l'accès aux périphériques chiffrés, c'est **conf/conf.d/cryptroot**. Celui-ci peut être analysé, comme l'ensemble de l'initramfs, en décompressant et désarchivant l'image avec :

```
% mkdir /tmp/i
% cd /tmp/i
% gunzip -c /boot/initrd.img-2.6.26-2-686 | cpio -i
33741 blocks
% cat conf/conf.d/cryptroot
target=sda2_crypt,source=/dev/sda2,key=none,lvm=lefbase-racine
target=sda2_crypt,source=/dev/sda2,key=none,lvm=lefbase-swap
```

On constate alors clairement que le nouveau disque n'est pas pris en compte. Le lecteur curieux pourra se dire que le fichier de configuration en question, généré par le *hook* **/usr/share/initramfs-tools/hooks/cryptroot**, analysera correctement la configuration en fonction de l'utilisation des volumes LVM. En réalité, ce n'est qu'à moitié vrai et plusieurs solutions s'offrent à nous pour contourner le problème.

Quoi qu'il en soit, dans le cas qui nous intéresse ici, ajouter tout l'espace disponible dans le volume logique **home** n'est pas possible automatiquement. Le script composant le **cryptroot** utilise l'action **deps** de la commande **dmsetup** sur le système de fichiers racine. Il en ressort que si ledit système de fichiers n'utilise pas un espace stocké dans le volume physique **sdb1_crypt**, il n'y a pas de dépendance directe. Pour autant, l'absence d'un volume physique, du fait de la « non-ouverture » du périphérique LUKS **sdb1**, bloquera le démarrage car le groupe de volumes sera incomplet.

Deux solutions sont alors envisageables :

- Nous pouvons manuellement ajouter une ligne **target=sdb1_crypt,source=/dev/sdb1,key=none,lvm=lefbase-racine** dans le fichier **cryptroot** et générer une nouvelle image initramfs avec **find | cpio -H newc -o | gzip -9 > /chemin/image**. Un nouveau problème apparaît alors, dans le fait que **update-initramfs -u** refusera de fonctionner. Forcer la commande à fonctionner ne servira à rien, puisque nous obtiendrons à nouveau un **cryptroot** incomplet.
- Nous pouvons délibérément provoquer une dépendance entre le device mapping **racine** et le nouveau volume physique en étendant ce volume logique de quelques mégaoctets (ou pourcent) que nous prendrons sur le volume physique en question. Dès lors, comme le système de fichiers racine dépendra de **sda2_crypt** et **sdb1_crypt**, le fichier **cryptroot** sera composé en conséquence et provoquera le déchiffrement des deux disques LUKS.

Bien entendu, une troisième solution consisterait à revoir le script/hook pour mettre en œuvre une autre méthode de détection de la configuration, plus intrusive que l'utilisation des éléments rapportés par **dmsetup** et reposant sur les outils LVM, par exemple, en complément de la lecture des **fstab** et **crypttab**. Ceci sort du cadre du présent article.

5 CONTOURNONS LE PROBLÈME

La solution retenue ici est donc la seconde. Nous allons forcer la détection en utilisant une partie de l'espace mis à disposition par le nouveau volume physique pour le volume logique **racine** et le reste pour **home**. Rappelons que notre **sdb1** est maintenant « ouvert » et accessible par **sdb1_crypt**. Dans un premier temps, nous devons ajouter le descripteur de volume en début de partition avec :

```
% pvcreate /dev/mapper/sdb1_crypt
  Physical volume "/dev/mapper/sdb1_crypt"
  successfully created
```

Ceci aura pour effet de faire du périphérique un volume physique à proprement parler. En effet, nous ne l'avons pas mentionné plus tôt, mais il n'existe pas de configuration stockée sous forme de fichier pour l'architecture LVM. Tout est stocké directement dans les descripteurs de volume. Seule la configuration générale de LVM dispose d'un fichier de configuration et de métadonnées présentes dans **/etc/vlm**.

La seconde étape consistera à étendre notre groupe de volumes (VG) **lefbase** en y ajoutant le nouveau volume physique :

```
% vgextend lefbase /dev/mapper/sdb1_crypt
  Volume group "lefbase" successfully extended
```

A ce stade, il ne faut surtout pas redémarrer le système car le volume physique, bien qu'ajouté dans le groupe, n'est absolument pas utilisé. En conséquence, la commande **dmsetup** affichera ceci :

```
% dmsetup deps
sda2_crypt: 1 dependencies      : (8, 2)
lefbase-racine: 1 dependencies  : (254, 0)
sdb1_crypt: 1 dependencies      : (8, 17)
lefbase-home: 1 dependencies    : (254, 0)
lefbase-swap: 1 dependencies    : (254, 1)
```

La génération d'un initramfs à ce moment ne prendrait pas en compte le périphérique supplémentaire. Ainsi, la partition LUKS ne serait pas déchiffrée et le groupe de volumes sera incomplet.

Nous nous empressons donc de répartir cet espace dans nos deux volumes logiques avec :

```
% lvextend -l +50%FREE /dev/lefbase/racine
  Extending logical volume racine to 34,80 GB
% lvextend -l +100%FREE /dev/lefbase/home
  Extending logical volume home to 7,86 GB
```

La syntaxe de **lvextend** peut paraître obscure au premier regard. Il est en effet possible de spécifier la manière d'étendre un volume :

- Soit par rapport à une proportion relative (option **-l**) à la taille d'un groupe de volumes (**%VG**), à la taille actuelle du volume logique (**%LV**), à l'espace disponible sur un volume physique (**%PV**) ou par rapport à l'espace disponible sur le groupe de volumes, comme ici (**%FREE**).
- Soit en fournissant une donnée numérique (**+L**) en kilo, méga, téra, péta ou exa octets.

Ici, la procédure est simple. La première commande attribue au volume logique **racine** 50% de l'espace disponible et la seconde 100% (des autres 50% encore libres) au volume **home**. Un fois ces deux commandes utilisées, nous obtenons bien le résultat attendu :

```
% lvdisplay
  --- Logical volume ---
  LV Name                /dev/lefbase/home
  VG Name                lefbase
  LV UUID                IVRLRu-52wu-OdzZ-GLmh-SH8a-febe-21JECI
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                7,86 GB
  Current LE             2011
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           254:2

  --- Logical volume ---
  LV Name                /dev/lefbase/swap
  VG Name                lefbase
  LV UUID                6s02m3-0bbg-0Q90-oSmo-RjØT-Q4mm-uMxGlP
  LV Write Access        read/write
  LV Status              available
  # open                 2
```

```
  LV Size                2,05 GB
  Current LE             524
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           254:3

  --- Logical volume ---
  LV Name                /dev/lefbase/racine
  VG Name                lefbase
  LV UUID                xwqG9D-jzjx-0VQD-bz9P-YoUG-wd25-CDLimW
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                34,80 GB
  Current LE             8910
  Segments               3
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           254:4
```

La commande **lvdisplay** est un peu plus bavarde que la simple **lvs**. Celle-ci est également déclinée en **vgdisplay** et **pvdisplay**, respectivement pour les groupes de volumes et les volumes physiques. Ici, notre **home** est bien passé de 3,54 Go à 7,86 Go. Mais ce n'est pas le cas pour le système de fichiers qui s'y trouve.

Note : De la bienveillance de la doc

Attention, là, il convient de préciser que, même si on se croit plus malin que la documentation, ce n'est généralement pas le cas. Lorsqu'on vous dit qu'il est impératif de démonter le système de fichiers avant d'utiliser **resize2fs** sur un périphérique bloc, ce n'est pas sans raison. Votre humble serviteur a cru être plus malin que la documentation et s'en est mordu les doigts en perdant plus d'une heure à regarder défiler les messages d'erreur et de correction d'un **fsck -y**.

Ici, nous avons eu la présence d'esprit de dédier un système de fichiers pour notre **/home**. Ceci présente l'avantage de permettre la déconnexion et la reconnexion sur le compte **root** afin de faire un simple **umount /home**. Nous pouvons ensuite utiliser **e2fsck -f**, indispensable à l'opération de redimensionnement d'un système de fichiers ext2 ou ext3 (**resize2fs** refusera de s'exécuter dans le cas contraire) :

```
% e2fsck -f /dev/mapper/lefbase-home
e2fsck 1.41.3 (12-Oct-2008)
Passe 1 : vérification des i-noeuds, des blocs et des tailles
Passe 2 : vérification de la structure des répertoires
Passe 3 : vérification de la connectivité des répertoires
Passe 4 : vérification des compteurs de référence
Passe 5 : vérification de l'information du sommaire de groupe
/dev/mapper/lefbase-home : 1786/232000 fichiers (0.2% non contigus),
34574/950272 blocs

% resize2fs /dev/mapper/lefbase-home
resize2fs 1.41.3 (12-Oct-2008)
Resizing the filesystem on /dev/mapper/lefbase-home to 2059264 (4k)
blocks.
Le système de fichiers /dev/mapper/lefbase-home a maintenant une taille
de 2059264 blocs.

% mount /home

# df -h
[...]
/dev/mapper/lefbase-home
                      7,8G   80M  7,3G   2% /home
```

Comme vous pouvez le voir, l'opération est aisée et après remontage de **/home**, **df**, nous présente bien l'espace disponible étendu, comme espéré. Il est maintenant temps de procéder à la mise à jour de l'initramfs, comme précédemment. De la même manière, avant de redémarrer le système, car ce sera nécessaire, on fera preuve d'un peu de prévoyance en s'assurant du contenu du **cryptroot** qui se trouve dans l'image, à coups de **gunzip** et de **cpio**. **sdb1_crypt** doit y être mentionné, tout comme **sda2_crypt**. Ceci fait, procédez à un premier redémarrage. Le changement sera explicite puisque les deux phrases de passe vous seront demandées l'une après l'autre, sans pause ou exécution d'autres scripts.

Mais nous n'en avons pas fini...

6 ET MON ROOTFS ALORS ?

N'oubliez pas, nous avons également ajouté un peu d'espace à notre système de fichiers racine. Espace qui, pour l'instant, est correctement affiché avec **lvs**, mais n'est aucunement reporté dans le résultat d'un **df -h**. Ici, nous ne pouvons pas nous amuser à démonter le système de fichiers puisqu'il est utilisé dès le démarrage. Il ne nous reste donc plus qu'une solution, démarrer sur un live CD contenant toutes les commandes utiles. Comme l'univers est bien fait, un live CD parfaitement adéquat se trouve offert avec une publication sœur, qui n'est autre que le dernier *GNU/Linux Pratique* 59. Il s'agit de SystemRescueCd, ou plus exactement d'une version légèrement modifiée (avec les locales françaises) de la dernière version de la distribution.

Voilà l'occasion de découvrir quelques commandes et outils LVM que nous n'avons pas encore traités. L'objectif, ici, est de démarrer sur un système « Live » pour déchiffrer manuellement les deux partitions LUKS, recomposer l'architecture LVM dans son ensemble, pour enfin redimensionner le système de fichiers racine comme il se doit. Voici les différentes étapes de la procédure :

- Ouverture de **sda2** et déchiffrement des données LUKS :

```
sysresccd /~ % cryptsetup luksOpen /dev/sda2 sda2_crypt
Enter LUKS passphrase: ******
key slot 0 unlocked.
Command successful.
```

- Ouverture de **sdb1** et déchiffrement des données LUKS :

```
sysresccd /~ % cryptsetup luksOpen /dev/sdb1 sdb1_crypt
Enter LUKS passphrase:
key slot 0 unlocked.
Command successful.
```

- Vérification de la disponibilité des périphériques déchiffrés et découverte des volumes physiques :

```
sysresccd /~ % dmsetup ls
sda2_crypt      (253, 0)
sdb1_crypt      (253, 1)

sysresccd /~ % ls /dev/mapper
control  sda2_crypt  sdb1_crypt

sysresccd /~ % pvscan
  PV /dev/block/253:0   VG lefbase   lvm2 [35.39 GiB / 0    free]
  PV /dev/block/253:1   VG lefbase   lvm2 [9.32 GiB / 0    free]
  Total: 2 [44.71 GiB] / in use: 2 [44.71 GiB] / in no VG: 0 [0   ]
```

- Activation des volumes logiques par activation de tous les volumes logiques d'un groupe (il est également possible d'activer individuellement chaque volume logique avec **lvchange**) :

```
sysresccd /~ % vgchange -a y lefbase
  3 logical volume(s) in volume group "lefbase" now active

% lvscan
  ACTIVE   '/dev/lefbase/home' [7.86 GiB] inherit
  ACTIVE   '/dev/lefbase/swap' [2.05 GiB] inherit
  ACTIVE   '/dev/lefbase/racine' [34.80 GiB] inherit
```

- Vérification du bon fonctionnement de l'ensemble et de l'espace disponible après montage :

```
sysresccd /~ % mkdir /mnt/racine
sysresccd /~ % mount /dev/mapper/lefbase-racine /mnt/racine
sysresccd /~ % df -h /mnt/racine
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/lefbase-racine
                       30G  769M   28G   3% /mnt/racine
sysresccd /~ % umount /mnt/racine
```

- Analyse du système de fichiers avec **fcsk** et redimensionnement :

```
sysresccd /~ % e2fsck -f /dev/mapper/lefbase-racine
e2fsck 1.41.10 (10-Feb-2009)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/mapper/lefbase-racine: 25273/1954064 files
  (1.0% non-contiguous), 319313/7813120 blocks

% resize2fs /dev/mapper/lefbase-racine
sysresccd /~ % resize2fs /dev/mapper/lefbase-racine
```

```
resize2fs 1.41.10 (10-Feb-2009)
Resizing the filesystem on
  /dev/mapper/lefbase-racine to 9123840 (4k) blocks.
The filesystem on /dev/mapper/lefbase-racine
  is now 9123840 blocks long.
```

- Vérification du succès de l'opération (nous sommes effectivement passés de 30 Go à 35 Go) :

```
sysresccd /~ % mount /dev/mapper/lefbase-racine /mnt/racine
sysresccd /~ % df -h /mnt/racine
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/lefbase-racine
                       35G  773M   32G   3% /mnt/racine
```

- Démontage du système de fichiers, désactivation des volumes logiques, fermeture des périphériques LUKS et redémarrage :

```
sysresccd /~ % umount /mnt/racine
sysresccd /~ % vgchange -a n lefbase
  0 logical volume(s) in volume group "lefbase" now active
sysresccd /~ % lvscan
  inactive  '/dev/lefbase/home' [7.86 GiB] inherit
  inactive  '/dev/lefbase/swap' [2.05 GiB] inherit
  inactive  '/dev/lefbase/racine' [34.80 GiB] inherit
sysresccd /~ % cryptsetup luksClose sdb1_crypt
sysresccd /~ % cryptsetup luksClose sda2_crypt
sysresccd /~ % pvscan
  No matching physical volumes found
sysresccd /~ % reboot
```

7 QUELQUES ASTUCES EN PLUS

J'ajouterai deux petites choses en guise de conclusion de cet article. Premièrement, le fait de disposer d'une telle installation (avec deux disques chiffrés) ne gène en rien l'utilisation d'un *rescue* CD bien conçu. Ainsi, SystemRescueCd s'en sort honorablement, mais le CD d'installation NetInst de Debian GNU/Linux également. En effet, en utilisant le menu avancé sur l'écran de boot syslinux, il est possible de choisir un mode rescue. Là, après démarrage, on vous proposera de rentrer les phrases de passe des deux partitions avant de vous demander quel système de fichiers utiliser en guise de racine pour un shell de secours. Bien entendu, une fois les disques LUKS déchiffrés, les éléments LVM sont automatiquement récupérés et utilisés. Vous pouvez donc monter **/dev/mapper/lefbase-racine** et corriger d'éventuels problèmes de configuration, comme avec une installation « normale ».

Le second ajout à cet article porte sur le chiffrement et les phrases de passe LUKS. Sachez que celles-ci ne sont ni uniques, ni invariables. Comprenez par là qu'il est non seulement possible de changer la phrase de l'une ou l'autre unité, mais qu'il est également possible de préciser plusieurs phrases de passe valides (jusqu'à 8).

Pour afficher les slots disponibles, il vous suffira d'utiliser la commande :

```
% cryptsetup luksDump /dev/sda2
[...]
Key Slot 0: ENABLED
Iterations: 141348
Salt:    9738ee075ec60a2f7015ed31681587d1
         677ac932921719068bab8bde3ef2fe78
Key material offset: 8
AF stripes: 4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

Sept emplacements ou slots peuvent encore être utilisés. Pour ajouter une clé (une phrase de passe), utilisez simplement **cryptsetup luksAddKey /dev/sda2**, entrez une des phrases enregistrées, puis deux fois la nouvelle phrase. Une nouvelle exécution de **cryptsetup luksDump** devrait vous afficher **Key Slot 1: ENABLED**.

Pour supprimer une phrase de passe, on utilisera :

```
% cryptsetup luksRemoveKey /dev/sda2
Enter LUKS passphrase to be deleted: ******
key slot 0 selected for deletion.
Enter any remaining LUKS passphrase: ****
key slot 0 verified.
Command successful.
```

Notez que l'action **luksDelKey**, prenant en dernier argument un numéro de slot, fonctionne toujours mais doit être considérée comme périmée. L'action **luksRemoveKey** ne prend pas d'argument autre que le périphérique concerné. C'est la saisie de la phrase à effacer qui désigne implicitement le slot et la commande est confirmée par la saisie d'une des phrases encore disponibles.

Nous voici arrivés au terme de cet article qui, je l'espère, a montré non seulement la bonne intégration du support du chiffrement LUKS et de LVM dans Debian, mais aussi la souplesse de cette solution combinée. Bien entendu, il ne s'agit là que d'une manière de voir les choses. Rien ne vous empêche, en effet, de revoir votre copie et installer un système qui sera plus adapté (ou plus tordu). Jongler avec les volumes et les device mappings n'est qu'une affaire de logique et, si le cœur vous en dit, vous pourriez tout aussi bien composer vos groupes avec des volumes physiques non chiffrés pour ensuite uniquement chiffrer certains volumes logiques. Pire, vous pouvez également utiliser plusieurs couches de chiffrement, bien que cela ne présente pas grand intérêt.

Enfin, l'utilisation de RAID logiciel n'a pas été traitée ici, mais selon les moyens à votre disposition, ceci peut être mis en œuvre. La bête noire du chiffrement de disque reste, et restera je pense encore longtemps, la défaillance de tout ou partie des secteurs d'un disque. L'idée est ici de composer un RAID logiciel avec des unités de disques standards (non chiffrées), de chiffrer les disques RAID pour ensuite ajouter une couche de LVM. Les possibilités ne manquent pas... ■

Objectif sécurité : On passe

Souvent, quand on monte une machine pour soi, que ce soit un serveur ou une station de travail, on installe son Linux sur le disque disponible et on se dit qu'on fera des backups au fil de l'eau, sur le streamer installé sur le serveur ou sur un disque USB. Après le premier crash disc, on en monte un autre, on réinstalle son système et, si tout va bien, on ne perd qu'une journée à refaire les 3254 configurations qui vont bien.

Auteur

- Cédric Pellerin

Au bout du deuxième ou du troisième crash, la moutarde nous monte au nez et on se prend à rêver d'avoir un second disque en miroir dans sa machine. Cependant, le système est déjà réinstallé, on a déjà refait cette $£%%*@% de conf pour la énième fois et on ne veut surtout pas tout recommencer.

Heureusement, il existe une méthode pour nous sortir de ce mauvais pas, sans perdre aucune donnée. Nous allons vous la décrire plus loin.

1 RAPPEL : LE RAID, QU'EST-CE QUE C'EST ?

Raid signifie chez nous *Redundant Array of Inexpensive (or Independant) Disks*. Ce concept a été défini en 1987 (Linux n'existait pas encore, mais les dinosaures n'existaient déjà plus :-p) à l'Université de Californie, à Berkeley. Il consiste en l'assemblage de plusieurs disques physiques afin d'obtenir une configuration sécurisée et/ou permettant une accélération des accès disques par un mécanisme de *striping*, c'est-à-dire en répartissant les accès sur plusieurs disques.

Les termes clés du raid sont :

- *Striping* : entrelacement des données sur plusieurs disques. On peut aussi parler de « volumes agrégés par bande » pour lever de la geekette dans les grands salons :-)
- *Mirroring* : Ecriture simultanée sur plusieurs disques, comme si chacun était le miroir des autres.
- *Error correction* : Somme de contrôle des données qui permet de reconstruire ces dernières si l'un des volumes de striping est mort.

Afin de savoir de quoi on parle, plusieurs « niveaux » de raid ont été définis au cours du temps. Les principaux sont les suivants :

- Raid 0 : Striping sur plusieurs disques. On accélère les accès disques, mais la sécurité n'est pas renforcée, au contraire. La perte d'un disque entraîne la perte de tout le *stack*.
- Raid 1 : Mirroring. Les données sont écrites en parallèle sur plusieurs disques. Chaque disque contient exactement les mêmes données que les autres, la sécurité est donc maximale, mais le coût aussi.
- Raid 5 : *Striping with distributed parity*. Ce niveau de raid nécessite au moins trois disques. Le principe consiste à éclater les données sur 2 disques (pas toujours les mêmes) et à écrire une somme de contrôle sur le troisième. Cette solution permet la perte d'un disque sans perte de données, tout en accélérant l'écriture grâce au striping. La perte d'espace est inversement proportionnelle au nombre de disques dans le stack. Elle est au maximum de 33% avec un stack de trois disques.
- Raid 6 : Comme en raid 5, avec deux disques pour la parité. On peut donc perdre deux disques du stack sans perdre les données.
- Raid 1+0 ou raid 10 : C'est en fait un striping de stacks raid 1. Ce système nécessite un minimum de quatre disques (deux stacks raid 1). On combine sécurité maximum et accélération au prix d'un coût très élevé (le double d'un raid 1).
- Raid 0+1 ou raid 01 : C'est... un miroir de raid 0. La différence entre le raid 10 et le raid 01, me direz-vous ? Ben heu, c'est comme la différence entre schtroumpf

en Raid 1

vert et vert schtroumpf. Certains universitaires à lunettes vous démontreront, chiffres à l'appui, le net avantage de l'un sur l'autre, mais pas moi.

Je vous renvoie à l'excellente page de Wikipédia (http://en.wikipedia.org/wiki/RAID) si vous souhaitez approfondir le sujet.

2 ETAT DES LIEUX

Afin de préciser les choses, je vous propose de partir du schéma suivant :

- **/dev/sda** : disque d'origine ;
- **/dev/sdb** : le disque neuf que l'on ajoute.

Nous supposons **/dev/sda** partitionné comme suit :

- **/dev/sda1** : **/boot** (environ 200 Mo) ;
- **/dev/sda2** : **/** ;
- **/dev/sda3** : le swap.

Une fois les stacks raid créés, nous désirons aboutir à :

- **/dev/md0** : **/boot** ;
- **/dev/md1** : **/** ;
- **/dev/md2** : le swap.

La distribution utilisée est une Debian stable (Debian 5.0.4 *Lenny* à la date d'écriture de l'article). La suite des opérations devrait fonctionner sans problème avec votre distribution préférée, mais je ne peux le garantir à 100%.

Certains me diront tout de go, quel intérêt de mettre le *swap* en raid ? Bonne question ; la réponse est simple à comprendre si vous avez des tiroirs *hotplug* comme on en trouve de plus en plus depuis l'avènement du SATA **[1]** : **sda** ou **sdb** lâche, vous pouvez le remplacer à chaud sans arrêter la machine, même si vous utilisez le swap. Dans le cas contraire, vous êtes obligé d'arrêter l'ordinateur ou de prier très fort que vos données ne sont pas swappées sur le disque que vous enlevez...

Pour en revenir à nos moutons, la procédure que je vais décrire ici est assez simple, mais il faut faire très attention à bien la suivre, sous peine de perdre tout ou partie de vos données. Je ne vous répéterai jamais assez : BACKUP, BACKUP, BACKUP. En tout état de cause, ni *GNU/Linux Magazine France*, ni les Editions Diamond, ni moi, ni mon chat, ni mes poissons rouges ne pourront être tenus pour responsable de toute perte de données lors de l'exécution de cette procédure. Le seul tribunal compétent en cas de litige est celui d'alpha du centaure, na !

3 LET'S GO

La première chose à faire est de se procurer un disque dur aussi proche que possible du premier en termes de taille et de géométrie (nombre de têtes, de clusters et de secteurs par cluster). Si cela n'est pas possible, il faut se rabattre sur un disque au moins aussi gros que l'original, pour des raisons que je vous laisse deviner :-)

Une fois le deuxième disque monté correctement dans l'ordinateur, on boot tout ce qu'il y a de plus normalement.

Votre machine a redémarré avec son deuxième disque tout beau tout neuf, il faut donc commencer par le partitionner. Avec les x86, un utilitaire sympathique nommé **sfdisk** évite de faire le boulot à la main.

```
# sfdisk -d /dev/sda | sfdisk /dev/sdb
```

Un petit tour sous **fdisk** pour vérifier que tout va bien et on en profite pour passer toutes les partitions de **/dev/sdb** en type « *linux raid autodetect* », c'est-à-dire en type 0xFD.

L'étape suivante consiste à installer ce qui peut nous manquer côté raid :

```
# apt-get install mdadm
# modprobe raid1
```

Ensuite, par mesure de précaution si le nouveau disque n'est pas neuf, on remet à zéro toute trace d'une précédente utilisation potentielle en raid :

```
# for i in "/dev/sdb1 /dev/sdb2 /dev/sdb3"; do mdadm --zero-superblock $i; done
```

On peut créer nos stacks raid, mais en utilisant uniquement **/dev/sdb**. **mdadm** nous permet ce genre de manœuvre qui ferait frémir à juste titre toute carte raid normalement constituée grâce à l'usage du mot-clé « missing » :

```
# mdadm --create /dev/md0 --level=1 --raid-disks=2 missing /dev/sdb1
# mdadm --create /dev/md1 --level=1 --raid-disks=2 missing /dev/sdb2
# mdadm --create /dev/md2 --level=1 --raid-disks=2 missing /dev/sdb3
```

Si **mdadm** ne vous a pas craché des insultes, c'est que tout va bien, ce que nous vérifions tout de suite avec un

```
# cat /proc/mdstat
```

qui devrait vous montrer 3 stacks raid 1 avec un disque *down* (_) et un *up* (**U**).

Maintenant, il nous faut créer les *filesystems* de la manière la plus classique. Pour les besoins de la démonstration, je suis parti sur de l'ext3, mais si vous voulez mettre du reiserfs ou du xfs, libre à vous. L'important est de mettre le même filesystem que sur les partitions correspondantes sur **/dev/sda**.

```
# mkfs.ext3 /dev/md0
# mkfs.ext3 /dev/md1
# mkswap /dev/md2
```

L'utilitaire **mdadm** stocke ses données de base dans le fichier **/etc/mdadm/mdadm.conf** qu'il nous faut compléter maintenant que les stacks sont actifs. Cependant, il faut conserver l'original lorsque nous adjoindrons **sda** à la pile.

```
# cp /etc/mdadm/mdadm.conf /etc/mdadm/mdadm.conf_orig
```

Pour éviter de se taper à la main des UUID complexes, la formule magique est :

```
# mdadm --examine --scan >> /etc/mdadm/mdadm.conf
```

qui va gentiment compléter **mdadm.conf** avec notre configuration actuelle.

Une fois ce raid un poil bancal monté, il va falloir copier les données de **/dev/sda** vers **/dev/sdb**, ce que nous faisons facilement :

```
# mkdir /mnt/md0
# mkdir /mnt/md1
# mount /dev/md1 /mnt/md1
# mount /dev/md0 /mnt/md0

# cp -dpRx / /mnt/md1
# cd /boot
# cp -dpRx . /mnt/md0
```

Explication très rapide : nous montons **/dev/md0** (futur **/boot**) et **/dev/md1** (futur **/**) dans 2 répertoires séparés, puis nous copions en mode archive les données depuis **/boot** et **/**. Pour ceux qui ne connaîtraient pas, cela fonctionne, car l'option **-x** de **cp** lui interdit de changer de filesystem en cours de copie, c'est pour cela que **/boot** n'est pas recopié lors du premier **cp**.

Il reste maintenant à aller modifier le **fstab** du raid en éditant **/mnt/md1/etc/fstab** et en remplaçant respectivement **/dev/sda1** par **/dev/md0**, **/dev/sda2** par **/dev/md1** et **/dev/sda3** par **/dev/md2**. Le fichier devrait ressembler à ça :

```
proc            /proc           proc    defaults          0       0
/dev/md1        /               ext3    errors=remount-ro 0       1
/dev/md0        /boot           ext3    defaults          0       2
/dev/md2        none            swap    sw                0       0
/dev/hdc        /media/cdrom0   udf,iso9660 user,noauto     0       0
```

Une fois ceci fait, il nous faut aller expliquer certaines choses à Grub **[2]** en modifiant **/mnt/md0/grub/menu.lst** :

- ajouter la commande **fallback 1** après la commande **default 0** ;
- recopier le bloc de *boot* du kernel sous le bloc d'origine en remplaçant **(hd0,0)** par **(hd1,0)** et **root=/dev/sda2** par **root=/dev/md1**.

En gros, ça doit donner quelque chose comme ça :

```
title           Debian GNU/Linux, kernel 2.6.26-2-686
root            (hd0,0)
kernel          /vmlinuz-2.6.26-2-686 root=/dev/sda2 ro quiet
initrd          /initrd.img-2.6.26-2-686

title           Debian GNU/Linux, kernel 2.6.26-2-686 (disc 2)
root            (hd1,0)
kernel          /vmlinuz-2.6.26-2-686 root=/dev/md1 ro quiet
initrd          /initrd.img-2.6.26-2-686
```

On met à jour notre initramfs, si on en a un :

```
# update-initramfs -u
```

Et on finit la première partie en installant Grub sur **/dev/sdb** :

- S'il n'y est pas, ajoutez **(hd1) /dev/sdb** dans **/boot/grub/device.map**.
- Lancez.

```
# grub-install /dev/sdb
```

ou le faites-le à la main :

```
# grub
grub> root (hd1,0)
grub> setup (hd1)
grub> quit
```

Et si tout s'est bien passé, on y va franchement :

```
# reboot
```

Lors du *reboot*, on choisit l'option numéro 2 du menu de Grub afin de booter sur notre stack raid incomplet mais fonctionnel.

Normalement, tout devrait se passer comme au reboot précédent, mises à part les 2 lignes du kernel vous indiquant qu'il a bien trouvé les stacks raid. Il ne nous reste plus qu'à terminer l'opération en modifiant **/dev/sda** qui n'est pas actif, puisque nous venons de booter sur **/dev/md1** et **/dev/md0**, c'est-à-dire **/dev/sdb2** et **/dev/sdb1**.

On commence par passer les partitions de **/dev/sda** en « linux raid autodetect », qui sont toujours de type 0xFD, via un brave :

```
# fdisk /dev/sda
```

Contrairement à certaines idées reçues, une utilisation responsable et intelligente de **fdisk** ne détruit absolument pas les données.

Une fois ceci fait, on ajoute enfin sa deuxième patte à notre raid bancal :

```
# mdadm --add /dev/md0 /dev/sda1
# mdadm --add /dev/md1 /dev/sda2
# mdadm --add /dev/md2 /dev/sda3
```

puis on attend la fin de la synchronisation, chose que l'on peut suivre avec :

```
# watch cat /proc/mdstat
```

Il nous reste à mettre un **mdadm.conf** à jour via la commande magique déjà connue :

```
# cp /etc/mdadm/mdadm.conf_orig /etc/mdadm/mdadm.conf
# mdadm --examine --scan >> /etc/mdadm/mdadm.conf
```

Afin de s'autoriser un maximum de sécurité au boot, il faut modifier le **menu.lst** de Grub afin qu'il ressemble à ceci :

```
title          Debian GNU/Linux, kernel 2.6.26-2-686
root           (hd0,0)
kernel         /vmlinuz-2.6.26-2-686 root=/dev/md1 ro quiet
initrd         /initrd.img-2.6.26-2-686

title          Debian GNU/Linux, kernel 2.6.26-2-686 (disc 2)
root           (hd1,0)
kernel         /vmlinuz-2.6.26-2-686 root=/dev/md1 ro quiet
initrd         /initrd.img-2.6.26-2-686

title          Debian GNU/Linux, kernel 2.6.26-2-686 fallback
root           (hd0,0)
kernel         /vmlinuz-2.6.26-2-686 root=/dev/sda2 ro quiet
initrd         /initrd.img-2.6.26-2-686

title          Debian GNU/Linux, kernel 2.6.26-2-686 fallback
root           (hd1,0)
kernel         /vmlinuz-2.6.26-2-686 root=/dev/sdb2 ro quiet
initrd         /initrd.img-2.6.26-2-686
```

Là, nous pouvons booter sur **/dev/sda** ou **/dev/sdb** avec le filesystem en raid (les deux premières options) ou passer en *fallback* sur les disques sans le raid (attention au **fstab** qui va râler) avec les deux dernières.

Pour finir, on remet à jour l'initramfs avec :

```
# update-initramfs -u
```

Et on reboot en croisant les doigts. Normalement, tout se passe bien et vous avez une belle machine enfin sécurisée. Vous pouvez vérifier en débranchant un disque ou l'autre (mais pas les deux ;-)) et en rebootant. Vous devriez pouvoir démarrer sans problème, seul un **cat /proc/mdstat** vous apprendra qu'il y a un souci.

Attention, une fois le disque rebranché, il faut demander à **mdadm** de le remettre dans le stack, il ne le fera pas tout seul :

```
# mdadm --manage /dev/md0 --add /dev/sda1
# mdadm --manage /dev/md1 --add /dev/sda2
# mdadm --manage /dev/md2 --add /dev/sda3
```

Voilà une solution possible pour avoir enfin une sécurité digne de ce nom. Pour information, je n'ai réellement testé cette procédure que sur un environnement x86. Il est certainement possible de faire pareil sur du Sparc ou du PA-Risc, mais je ne sais pas trop comment silo ou palo vont se comporter... Ca fera peut-être l'objet d'une prochaine mise à jour :-)

Pour ceux d'entre vous qui voudraient tester sans risque dans une VM, voici quelques conseils pour VirtualBox :

- Monter sa VM normalement avec un disque dur virtuel de taille **fixe**.
- Installer sa distribution préférée dessus.
- Cloner le disque dur avec :

```
VBoxManage clonehd <disque_original.vdi> <disque_clone.vdi>
```

- L'utilisation de **sfdisk** n'est pas conseillée puisque les partitions existent déjà. En revanche, il est indispensable de formater les stacks raid, sinon on se retrouve avec des UUID dupliqués (à vérifier avec la commande **blkid**).

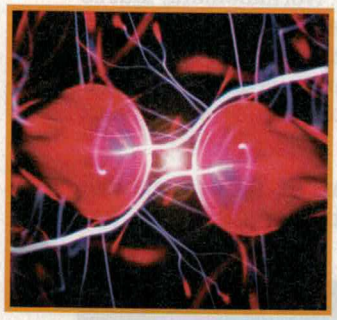
Si certains d'entre vous ont maintenant envie d'étendre cette procédure pour faire du raid 5 ou 10, je vous laisse faire... ■

Références

- **[1]** Certains ont même essayé de faire du *hotplug* en PATA avec les fameux tiroirs totalement passifs que l'on trouvait partout à l'époque. Quelques rares ont eu de la chance, les autres ont eu la joie de contempler un contrôleur IDE et/ou un disque dur cramé. A éviter, sauf pour s'amuser et tester sa chance :-)
- **[2]** ATTENTION, nous sommes ici en Grub legacy et non en Grub 2. Les correspondances ne devraient pas être trop compliquées à trouver dans **/boot/grub/grub.cfg** pour ceux d'entre vous qui ont migré.

Auteur : Cédric Pellerin

Survivre au crash de ses services



Auteur

- Cédric Pellerin

Parmi les services réseau indispensables se trouvent les serveurs DNS et DHCP. Je ne parle pas ici de ceux mis à votre disposition par votre FAI, mais de ceux que vous avez montés chez vous afin de faire fonctionner correctement votre réseau local. Certes, madame Michu, avec ses deux PC qui se battent en duel, peut se contenter du DHCP de sa trucbox et des DNS de son fournisseur. Mais pour nous qui avons une, voire plusieurs dizaine(s) de machines physiques ou virtuelles en fonctionnement permanent à la maison (au grand désespoir de notre chère moitié la plupart du temps), nous qui participons, bien malgré nous, à faire la fortune de notre fournisseur d'électricité, nous qui aimons maîtriser à 100% notre plan d'adressage IP, nous ne pouvons nous satisfaire de ce 192.168.0.0/24 étriqué et de ces DNS qui répondent quand ils n'ont rien de mieux à faire.

Pour cela, nous avons mis en place depuis longtemps une machine sur laquelle tournent, entre autres, un serveur DNS (Bind 9 le plus souvent, mais aussi parfois DJBDNS ou autres pour ceux qui aiment l'exotisme ;-)) et un serveur DHCP. Tout va pour le mieux dans le meilleur des mondes jusqu'au jour où le serveur, pourtant chouchouté, décide de tomber en panne, par exemple en grillant son alimentation. Là, c'est la catastrophe. « Papa, j'ai plus accès au Net, j'étais en plein raid avec ma guilde », « Chéri, le site des 4 belges ne répond plus, au secours ! ». Et là, on se dit « si seulement j'avais prévu une redondance ». La haute disponibilité, c'est comme les backups, c'est quand on en a besoin qu'on s'aperçoit qu'on les a oubliés.

Pourtant, c'est tellement simple, enfin une fois qu'on s'est farci quelques heures de surf sur le Net, qu'on a déblayé les HOWTO qui nous expliquent comment faire un système redondé avec SSO et DNS sur LDAP, le DHCP stocké en base MySQL et le tout sécurisé par un serveur Radius monté sur une plate-forme Heartbeat. « Mais je veux juste un système maître/esclave, moi ! », dit le sysadmin au bord de la crise de nerfs après 3 boîtes d'aspirine...

Ne cherchez plus, la solution est extrêmement simple et je vous la fournis gracieusement, petits veinards.

1 LE SERVEUR DNS

Avec toutes mes excuses auprès des défenseurs des droits des minorités, je ne parlerai ici que de Bind dans sa version 9. Je suppose que vous avez déjà un serveur fonctionnel, sinon, il vous suffit de vous reporter à l'article idoine dans ce même numéro.

Le réseau considéré est constitué de la façon suivante :

- plan d'adressage en 172.17.0.0/24 ;
- serveur existant en 172.17.0.1, appelé ci-après « serveur maître » ou « maître » tout court ;
- serveur esclave que nous ajoutons en 172.17.0.2 ;
- une station de travail en 172.17.0.10.

La première étape consiste à informer notre serveur maître de l'existence de son esclave. Pour ce faire, il suffit d'ajouter la directive suivante dans le fichier **named.conf** :

```
allow-transfer {
      127.0.0.1;
      172.17.0.2; # Adresse du slave
};
```

qui lui explique qu'il a le droit de transférer les zones vers le serveur 172.17.0.2.

Ensuite, nous montons un *bind* de façon parfaitement conventionnelle sur l'esclave et nous recopions le fichier `named.conf` du maître. Puis nous y apportons les modifications suivantes :

- Toutes les zones internes sont à passer de « master » à « slave ».
- Pour toutes les zones que nous voulons voir répliquées, nous ajoutons les deux directives suivantes :

```
masters { 172.17.0.1; }; # Adresse du master
allow-notify { 172.17.0.1; };
```

Dans le but de vérifier que tout va bien, nous recopions aussi les fichiers de zone du *master* sur le *slave*, puis nous lançons le bind slave et normalement, il devrait démarrer sans soucis.

Sur notre station de travail, nous modifions le `resolv.conf` afin de n'avoir plus que le slave en serveur DNS :

```
nameserver 172.17.0.2
```

Nous tentons de pinger un nom du LAN puis, si tout va bien, un nom du nain Ternet. En théorie, nous sortons et nous surfons sans problème.

Afin de permettre à Bind de modifier lui-même ses fichiers en fonction des instructions du master, il faut lui donner les droits en écriture sur ses fichiers de configuration dans `/etc/bind` :

```
chgrp bind /etc/bind/*
chmod g+w /etc/bind/*
chmod g+w /etc/bind
```

Maintenant nous retournons sur le master, nous incrémentons le numéro de série dans `named.conf` (ne jamais oublier de le faire à chaque modification) et nous relançons Bind :

```
/etc/init.d/bind9 reload
```

En consultant les logs, nous devons nous apercevoir que le master a envoyé ses modifications au slave. Pour valider, nous allons juste modifier une entrée dans un fichier de zone du LAN (ajoutez une station, par exemple). Ensuite, nous incrémentons le numéro de série, puis nous relançons Bind. En consultant les fichiers de zone sur le slave, nous devons remarquer qu'ils reflètent exactement ceux du master, du moins du point de vue de Bind. En effet, toute notre belle ordonnance dans les fichiers de zone a disparu, au profit d'un tri alphabétique d'un goût plus que moyen. M'enfin, s'il est content comme ça, on va pas se plaindre, hein ?

Pour finir, il nous reste juste à ajouter ce deuxième serveur DNS dans la configuration de notre DHCP afin que toutes nos stations en profitent. Une fois cela fait, on relance le réseau sur les stations et on arrête violemment le DNS master. Si tout continue de rouler, et ça devrait être le cas, c'est gagné !

Au fait, à propos du serveur DHCP, si on en profitait pour les mettre aussi en maître/esclave ?

2 LE SERVEUR DHCP

Comme vous avez pu le constater, mettre un serveur Bind en haute disponibilité est l'affaire de quelques minutes. Pour le DHCP, ce n'est pas plus long, ni plus complexe. Afin de respecter les us et coutumes, j'ai privilégié le serveur DHCP 3 classique.

Comme pour le serveur DNS, je présuppose que vous disposez d'un serveur DHCP fonctionnel. Si tel n'était pas le cas, merci de vous reporter à l'article susmentionné. Il vous faut, bien évidemment, installer un serveur DHCP sur la deuxième machine. Un simple `apt-get install dhcp3-server` suffira.

Il existe deux méthodes pour avoir de la haute disponibilité pour un serveur DHCP, la bonne et la moins bonne. La première qui vient à l'esprit est de se dire qu'il suffit d'en monter un deuxième avec les mêmes IP fixes et les plages d'IP variables non recouvrantes. Ensuite, la vitesse de propagation du réseau et la réactivité des serveurs feront le reste. Cette méthode, que l'on pourrait qualifier de « bourrin », a le mérite de fonctionner (enfin, souvent), mais elle reste assez aléatoire et vous n'êtes pas à l'abri d'un conflit d'adresses IP pas facile à déverminer si vous faites la moindre erreur de configuration.

La bonne méthode consiste à... lire le manuel (si, si, le RTFM fonctionne parfois) et à installer un serveur « peer » avec les options prévues pour. Vous remarquerez que dans le cas du DHCP, on ne parle plus de maître et d'esclave, mais de pair (*peer*). C'est moins violent, nettement plus politiquement correct, mais ça marche aussi bien. Cette opération se fait en deux temps : premièrement, informer le serveur primaire qu'il a un secondaire à sa botte ; deuxièmement, informer le secondaire qu'un pair est dans le coin.

On part du principe que vous montez les serveurs DHCP sur les mêmes machines que les DNS. L'adressage réseau ne change donc pas. Pour rappel, le primaire est en 172.17.0.1 et le secondaire en 172.17.0.2.

2.1 Le serveur primaire

Les lignes à ajouter dans son fichier de configuration (`/etc/dhcp3/dhcpd.conf` sur une Debian) sont les suivantes :

```
failover peer "monlanamoi" {
        primary;                                # C'est moi le chef
        address 172.17.0.1;                     # Mon IP à moi
        port 647;
        peer address 172.17.0.2;    # Adresse du pair
        peer port 647;
        max-response-delay 60;
        max-unacked-updates 10;
        mclt 3600;
        split 128;                              # Laisser à 128, uniquement
sur le primaire
        load balance max seconds 3;
}
```

Il convient aussi d'ajouter la ligne :

```
failover peer "monlanamoi";
```

dans les *pools* d'adresses que l'on veut voir dupliqués. Pour votre mise en production, il est conseillé de trouver un nom moins godiche que « monlanamoi » :-p

2.2 Le serveur secondaire

Dans un premier temps, recopiez la configuration du serveur primaire et modifiez juste le bloc `failover` :

```
failover peer "monlanamoi" {
        secondary;                              # Moi esclave soumis
(enfin, presque...)
        address 172.17.0.2;
        port 647;
        peer address 172.17.0.1;        # J'ai un pair à cette adresse
        peer port 647;
        max-response-delay 60;
        max-unacked-updates 10;
        mclt 3600;
        load balance max seconds 3;
}
```

Il suffit ensuite de redémarrer le primaire, de démarrer le secondaire, de débrancher le câble réseau du primaire et de vérifier que tout continue de fonctionner, même si on rafraîchit l'adresse IP de la station. Ensuite, libre à vous de jongler avec les câbles branchés/débranchés pour valider que tout fonctionne. Vous pouvez aussi suivre les effets de vos activités barbares dans le *syslog* des deux serveurs...

CONCLUSION

Comme vous avez pu le constater, il suffit d'une deuxième machine et de quelques lignes de configuration en plus pour se retrouver à l'abri de tout, sauf d'une panne de courant généralisée. Mais là, sauf à avoir des bougies qui ne s'allument qu'en SNMP, on a plus trop besoin d'un accès au réseau tant qu'EDF n'a pas réparé... ■

Auteur : Cédric Pellerin

Installation d'un serveur FTP à authentification séparée

Le protocole FTP a souvent une triste réputation, tant par le fait de sa sécurité limitée que par le mécanisme consistant à utiliser plusieurs ports rendant sa mise en œuvre problématique avec une bonne politique de pare-feu. Néanmoins, à défaut d'autre chose, tout aussi standard et largement utilisé, il arrive qu'on n'ait pas vraiment le choix. Ainsi, quitte à installer un serveur FTP, autant le faire correctement.

Les serveurs FTP ne manquent pas dans la distribution Debian GNU/Linux. **wzdftpd**, **ftpd**, **muddleftpd** ou encore **vsftpd** sont quelques exemples de paquets fournissant ce type de services. Ici, c'est **vsftpd** qui sera utilisé, mais avec une configuration un peu spécifique :

- Nous ne voulons pas de serveur sous la forme de *daemon* mais une exécution via le super-serveur **inetd** pour contrôler plus finement le filtrage et la sécurité.
- Nous ne souhaitons pas reposer sur la gestion d'identité du système mais sur un fichier d'utilisateurs bien distinct.
- Nous souhaitons que le dépôt FTP soit la racine d'un serveur HTTP et donc que les fichiers et répertoires appartiennent à l'utilisateur faisant fonctionner ledit serveur.

C'est un cas relativement classique d'hébergement de services. Nous partons, en effet, du principe que la machine est un serveur web mis à disposition pour des personnes gérant leurs fichiers via FTP. Ceci implique le fait qu'il pourra s'agir d'utilisateurs non Unix et qu'il est hors de question de leur fournir un accès au shell distant. C'est le type de mise en œuvre qu'on pourrait trouver chez n'importe quel hébergeur de pages personnelles.

1 INSTALLATION DU SERVEUR FTP

Classiquement, l'installation du serveur passera par un simple **aptitude install vsftpd**. Cette installation provoquera également l'installation ou la mise à jour de la libSSL. Le daemon est, en effet, également capable de faire du sFTP (*FTP over SSL*) afin de chiffrer les informations de *login/password* mais également les données qui transitent. Nous laisserons cela de côté dans le présent article, en nous intéressant uniquement à la configuration décrite plus haut.

Dès l'installation, on peut constater que le serveur est immédiatement lancé et écoute sur toutes les interfaces. Voilà qui n'est pas très plaisant. Nous nous empressons donc d'arrêter le service avec **/etc/init.d/vsftpd stop**. Nous pouvons alors nous pencher sur la configuration placée dans **/etc/vsftpd.conf**. Voici la configuration par défaut débarrassée des commentaires :

```
% cat /etc/vsftpd.conf | egrep -v '^$|^#'
listen=YES
anonymous_enable=YES
dirmessage_enable=YES
use_localtime=YES
xferlog_enable=YES
connect_from_port_20=YES
secure_chroot_dir=/var/run/vsftpd/empty
pam_service_name=vsftpd
rsa_cert_file=/etc/ssl/private/vsftpd.pem
```

La première ligne implique le fonctionnement en mode daemon, un simple **listen=NO** réglera ce premier problème. Nous transformons la configuration en ceci :

```
listen=NO
anonymous_enable=NO
local_enable=YES
virtual_use_local_privs=YES
write_enable=YES
connect_from_port_20=YES
pam_service_name=vsftpd
guest_enable=YES
guest_username=www-data
user_sub_token=$USER
local_root=/var/www
chroot_local_user=YES
hide_ids=YES
xferlog_enable=YES
```

Nous avons donc :

- `listen`, qui nous permet de spécifier ou non le mode daemon. Implicitement, un `NO` ici indiquera une utilisation via `inetd`/`xinetd`.
- `anonymous_enable` nous permet d'activer ou non (autoriser) les connexions en mode *anonymous*, consistant généralement pour l'utilisateur à saisir un login `anonymous` et son adresse mail en guise de mot de passe.
- `local_enable` permet les connexions des logins locaux. Attention, ne vous y trompez pas, ceci reste dépendant de votre configuration PAM (que nous allons traiter par après). Si votre serveur FTP accepte autre chose que des connexions anonymes, vous devez obligatoirement mettre `YES` ici.
- `virtual_use_local_privs` indique que les utilisateurs virtuels (ceux qui n'existent pas dans le système) possèdent les même privilèges que les utilisateurs locaux. Par défaut, les utilisateurs virtuels disposent des mêmes droits que les utilisateurs anonymes.
- `write_enable` permet d'activer les commandes FTP susceptibles de modifier le système de fichiers. Si vous souhaitez offrir un accès en écriture à certains utilisateurs, vous devez mettre `YES` ici.
- `connect_from_port_20` active la prise en charge des connexions sur le port ftp-data. On choisira `YES` ici afin de garantir le bon fonctionnement avec des clients FTP ne laissant pas le choix à leurs utilisateurs.
- `pam_service_name` nous permet d'identifier le service dans la configuration PAM (voir plus loin). `vsftpd` est un choix tout à fait logique.
- `guest_enable` : définie comme `YES`, cette option nous permet de classifier toutes les connexions non anonymes comme étant des invités. En d'autres termes, nous pouvons remapper les logins des utilisateurs sous une identité précise définie par le paramètre qui suit. Ici, nous mettons en place un serveur FTP à destination des personnes en charge du contenu d'un serveur web. Il est donc de bon ton de remapper ces identités vers `www-data`, l'identité habituellement utilisée sur un système Debian GNU/Linux pour les serveurs HTTP comme Lighttpd, Nginx ou encore Apache.
- `guest_username` permet de spécifier l'identité à utiliser pour les utilisateurs invités.
- `user_sub_token` : cette option ne nous est pas d'une grande utilité ici, mais est cependant précisée puisque nous traitons indirectement des utilisateurs virtuels. Elle permet de générer automatiquement un répertoire personnel pour ces utilisateurs en fonction du contenu de `guest_username`. Le chemin est ainsi composé à partir du répertoire personnel du véritable utilisateur. Si le chemin est, par exemple, `/var/www/homes/$USER`, nous pouvons spécifier ici `$USER` et chaque connecté verra alors son répertoire personnel composé de `/var/www/homes/` plus son login. Notez que ceci fonctionne également si la chaîne spécifiée pour `local_root` contient la variable indiquée pour `user_sub_token`.
- `local_root` est notre répertoire racine du serveur FTP. Le serveur, après connexion d'un utilisateur non anonyme, tentera de changer de répertoire vers celui-ci.
- `hide_ids` nous permet d'ajouter un peu d'anonymisation. Cette option permet de montrer systématiquement les répertoires et fichiers comme appartenant à un utilisateur `ftp:ftp` et non comme appartenant à l'identité sous laquelle s'exécute effectivement le serveur ou à qui appartiennent effectivement les fichiers.
- `xferlog_enable` : comme nos utilisateurs n'ont pour tâche que de gérer le contenu d'un ou plusieurs serveurs, mieux vaut être prudent et garder une trace de leurs activités. Ceci évitera le classique et pathétique « c'est pas moi, je sais pas, j'ai rien fait » (on ne discute généralement pas avec le contenu d'un log). Exemple :

```
CONNECT: Client "192.168.12.6"
[lab] OK LOGIN: Client "192.168.12.6"
[lab] OK RENAME: Client "192.168.12.6",
   "/fae.php /cms/fae_old.php"
[lab] OK UPLOAD: Client "192.168.12.6",
   "/pdf/fae.php", 14604 bytes, 64.96Kbyte/sec
[lab] OK DELETE: Client "192.168.12.6",
   "/pdf/fae.php"
[lab] OK RENAME: Client "192.168.12.6",
   "/pdf/fae_old.php /pdf/fae.php"
```

Une fois cette nouvelle configuration en place, nous ne pouvons plus démarrer le service via le script d'init :

```
% /etc/init.d/vsftpd start
/etc/vsftpd.conf:
   listen disabled - service will not start
```

C'est parfait et précisément ce que nous espérions.

2 MISE EN PLACE DE L'AUTHENTIFICATION

Notre service n'est pas encore fonctionnel et avant de le rendre disponible, nous allons ajouter les éléments permettant la gestion des utilisateurs virtuels. Le serveur reposera sur PAM, le système *Pluggable Authentication Modules* (modules d'authentification enfichables, hum hum). PAM est une invention de Sun Microsystems, mais est largement diffusé dans tous les systèmes Unix en logiciel libre. Il est ainsi supporté depuis plusieurs années avec les architectures Solaris, Linux, FreeBSD, NetBSD, etc.

L'avantage de PAM est de proposer une modularité très importante du service d'authentification. Partant de la constatation que les services et d'autres éléments nécessitant

d'identifier les utilisateurs et de s'assurer de ces informations étaient une fonctionnalité pouvant prendre bien des formes, PAM choisit de se présenter comme un service.

Ainsi, il joue le rôle d'agent permettant aux outils de fonctionner sans prendre en charge directement les tâches d'authentification. Un client PAM demandera simplement à l'infrastructure d'authentifier un utilisateur et aura en retour une information sur le succès ou l'échec de l'opération. Il devient donc possible de définir par service quelles règles doivent être mises en place. Mieux encore, on peut modifier ces règles sans avoir à toucher à la configuration du service demandeur.

Un simple coup d'œil dans **/etc/pam.d** vous donnera une idée de l'étendue de l'utilisation de PAM sur votre système. On retrouve ainsi des services de connexion distants comme SSH ou Samba, mais également des services locaux comme **login**, **su**, **sudo** ou encore GDM.

PAM repose également, côté authentification, sur cet aspect modulaire. Il existe donc plusieurs méthodes pour connaître l'identité d'un utilisateur, s'assurer qu'elle soit valide et éventuellement limiter le champ d'application.

Cet article n'est pas un cours sur PAM, mais nous allons cependant donner un exemple générique avant de traiter du cas du serveur FTP à mettre en place. Prenons, par exemple, le service **login** permettant aux utilisateurs de se connecter localement via un TTY :

```
auth       requisite  pam_securetty.so
auth       requisite  pam_nologin.so
session    required   pam_env.so readenv=1
session    required   pam_env.so readenv=1 envfile=/etc/default/locale
@include common-auth
auth       optional   pam_group.so
session    required   pam_limits.so
session    optional   pam_lastlog.so
session    optional   pam_motd.so
session    optional   pam_mail.so standard
@include common-account
@include common-session
@include common-password
```

PAM utilise 4 mécanismes différents :

- **account** vérifie si le compte demandé est disponible et valide ;
- **auth** assure l'authentification réelle ;
- **password** permet de mettre à jour l'authentification ;
- **session** assure la mise en place et la fermeture de session.

La modularité de PAM repose sur des bibliothèques dynamiques (***.so**), ce sont des plugins ou modules. Chaque module met ou non à disposition les différents mécanismes listés ci-avant. Ne vous étonnez donc pas de retrouver les mêmes noms de fichiers un peu partout. Ainsi, pour notre **login**, nous avons :

- La vérification du compte reposant sur **pam_unix.so**, **pam_deny.so** et **pam_permit.so**. **pam_unix** met en place le mécanisme de vérification de compte classique pour un système Unix consistant à vérifier les fichiers, comme **/etc/passwd**. Les deux autres plugins permettent respectivement de refuser la validation et de l'accepter, sans condition. Il sont présents dans **common-account** comme solution de repli si **pam_unix** échoue.
- L'authentification à proprement parler utilise les mêmes plugins, plus **pam_securetty.so**, **pam_nologin.so** et de manière optionnelle **pam_nologin.so**. Le premier permet de refuser l'authentification si l'utilisateur ne fait pas usage d'un terminal considéré comme sûr (fichier **/etc/securetty**), le second bloque les authentifications si **/etc/nologin** existe. C'est une méthode pour suspendre les opérations de connexions. Le dernier n'est pas nécessaire au succès de l'authentification, mais permet de s'authentifier avec un mot de passe pour le groupe (fichier **/etc/security/group.conf**).
- La gestion de la session utilise les trois classiques modules, plus **pam_env.so**, **pam_limits.so**, **pam_lastlog.so**, **pam_motd.so**, **pam_mail.so** et **pam_ck_connector.so** permettant respectivement de gérer les variables d'environnement, limiter l'accès aux ressources, afficher la dernière connexion, afficher le message du jour et renseigner sur la disponibilité de nouveaux messages/mails.
- La mise à jour utilise simplement les trois modules par défaut.

Notez la possibilité d'inclusion avec **@include** exactement à la manière de ce que proposent les directives des langages de programmation comme C. Ceci présente l'avantage, comme c'est le cas ici, de n'avoir qu'une seule occurrence du triptyque **pam_unix**/**pam_deny**/**pam_permit** pour tous les services. Le point important d'un fichier de service comme **login** est la judicieuse utilisation des directives :

- **required** : indique qu'un échec pour le module conduit à un échec de PAM.
- **requisite** : idem à la précédente directive, mais stoppe immédiatement le mécanisme PAM sans traiter les autres modules.
- **sufficient** : indique qu'un succès pour ce module est suffisant et qu'il n'est pas nécessaire de traiter les suivants.
- **optional** : permet, en toute logique, de préciser que ce module est optionnel. S'il s'agit du seul module utilisé pour un service, il déterminera le succès ou l'échec de PAM.

Pour notre serveur FTP, nous avons une configuration PAM installée par défaut avec le paquet **vsftpd** :

```
auth    required   pam_listfile.so \
  item=user sense=deny \
  file=/etc/ftpusers onerr=succeed
@include common-account
@include common-session
@include common-auth
auth    required        pam_shells.so
```

Comme vous pouvez le constater, ce n'est absolument pas ce que nous désirons, puisque les vérifications portent sur l'existence d'un véritable compte utilisateur sur la machine, le contenu du fichier **/etc/ftpusers** et sur la disponibilité d'un shell valide pour l'utilisateur. Avant de nous attacher à la modification de ce fichier, nous devons ajouter un module nous permettant de, tout simplement, reposer sur le contenu d'un fichier de type **htpasswd**. Ce module n'est pas livré par défaut avec PAM et nous devons installer un nouveau paquet : **libpam-pwdfile**.

Nous pouvons ensuite utiliser :

```
auth    required pam_pwdfile.so pwdfile /etc/vsftpd.passwd
account required pam_permit.so
```

Pour créer le fichier **/etc/vsftpd.passwd**, il nous suffira d'utiliser la commande **htpasswd** exactement comme pour créer une authentification pour un serveur HTTP. Cette commande est disponible via l'installation du paquet **apache2-utils**.

3 LANCEMENT VIA INETD

Tout est en place pour le lancement du serveur. Il ne nous reste qu'à configurer notre super-serveur **inetd**. Nous allons préférer ici **xinetd**, dont la configuration est bien plus agréable. Après installation du paquet du même nom, nous nous plaçons dans **/etc/xinetd.d** et ajoutons un fichier **vsftpd** contenant :

```
# vsftpd is the secure FTP server.
service ftp
{
disable         = no
socket_type     = stream
wait            = no
user            = root
server          = /usr/sbin/vsftpd
per_source      = 5
instances       = 200
only_from       = 192.168.12.0/24
banner_fail     = /etc/vsftpd.busy_banner
log_on_success += PID HOST DURATION
log_on_failure += HOST
}
```

Ce fichier détermine la manière dont doit être lancé le serveur **/usr/sbin/vsftpd** et sous quelles conditions :

- **disable** : active ou non le service.
- **socket_type** : détermine le type de *socket* pour la communication parmi **stream**, **dgram**, **raw** ou **seqpacket** pour respectivement définir des binaires traitant un flux de données, des datagrammes, de l'IP ou une suite séquentielle et cohérente de datagrammes.
- **wait** : permet de déterminer la façon de traiter de multiples connexions. Si le service est monothreadé, il faut utiliser **yes**, et **xinetd** lancera le binaire une fois et n'acceptera pas d'autre connexion. Si la valeur est à **no**, le serveur est capable de gérer plusieurs *threads* et donc de recevoir plusieurs connexions.
- **user** : l'identité sous laquelle doit être lancé le binaire.
- **server** : le chemin vers le binaire serveur, ici **vsftpd**.
- **per_source** : permet de spécifier le nombre maximum d'instances par adresse source. Une valeur ou **UNLIMITED** peuvent être utilisés et on calibrera judicieusement ce paramètre en fonction de ses besoins entre supporter les connexions nécessaires et limiter les possibilités d'attaques de type DoS.
- **instances** : idem que le paramètre précédent, mais toutes sources confondues. Là encore, **UNLIMITED** pourra être utilisé, mais c'est généralement une mauvaise idée.
- **only_from** : ceci nous permet de limiter la source des connexions en fonction de leur adresse. Ici, nous avons mis en place un VPN et ne souhaitons pas que n'importe quelle machine puisse accéder au service. Nous limitons donc l'accès au réseau 192.168.12.*.
- **banner_fail** : prend en argument un fichier dont le contenu sera envoyé à la machine distante pour l'informer de l'échec de la connexion. Un simple **echo "Serveur non disponible" > /etc/vsftpd.busy_banner** vous permettra de créer un tel fichier. A vous de voir si vous souhaitez être plus ou moins poli.
- **log_on_success** : permet de spécifier les informations à placer dans les journaux systèmes en cas de connexion réussie. Ici, respectivement, le PID du processus, l'hôte distant et la durée de la session.
- **log_on_failure** : idem au paramètre précédent, mais pour les connexions échouées.

Une fois ces éléments correctement enregistrés dans le fichier, il ne vous restera plus qu'à relancer le super-serveur ou en recharger la configuration via un **/etc/init.d/xinetd reload**. Dès lors, votre nouveau serveur FTP devrait être parfaitement fonctionnel.

CONCLUSION

Nous venons de voir brièvement les possibilités offertes par différents biais : PAM, **vsftpd** et **xinet.d**. Nous n'avons pas couvert ici 10% des possibilités de chaque élément et nous vous invitons à expérimenter autour de cette base pour, par exemple, ajouter plus de restrictions ou encore utiliser SSL pour chiffrer les transactions. ■

Un contrôle parental efficace : Projet Cohorte

Auteur

- Cédric Pellerin

Pour n'importe lequel d'entre nous qui baignons dans l'informatique toute la journée, nous relevons nos mails, nous surfons sur le Web sans même y penser. Pour nous, Internet est ce qu'il est réellement, c'est-à-dire un immense réseau de serveurs dont nous comprenons la fonction et les modes opératoires ; pour un nombre non négligeable de nos contemporains, il s'agit surtout d'une énorme boîte de Pandore dont ils ont très peur de soulever le couvercle… Force nous est d'admettre que le risque est loin d'être nul pour eux de se retrouver la cible de nombreuses attaques, voire même d'y participer à l'insu de leur plein gré.

Afin de les aider à éviter cela au maximum, nous avons élaboré le projet décrit ci-après. L'exposé technique de ce projet se déroulera sur plusieurs articles plus ou moins successifs (NDLR : dans le magazine mensuel), dont la somme aboutira à la réalisation d'une passerelle bien pleine et, si possible, bien faite.

Note : Pourquoi ?

Fréquentant relativement fréquemment des personnes qui utilisent l'informatique, presque contraints et forcés, je me suis aperçu que, comme très souvent, leur « inaptitude » à la manipulation des ordinateurs vient surtout d'un manque criant de formation initiale. Pour eux, leur PC est une boîte noire qui oscille entre le magique et le miraculeux et ils contemplent Internet comme l'équipage de l'Andromeda contemple un trou noir : c'est à la fois fascinant et terrifiant.

Afin de faire passer cette étape cruciale, il est important de leur montrer plusieurs choses :

- Non, l'ordinateur de mord pas.
- Non, ça ne va pas exploser s'il se trompe.
- L'ordinateur ne fait qu'exécuter les ordres qu'on lui donne. Si ça ne « marche pas », c'est que l'utilisateur n'a pas su lui dire ce qu'il voulait (enfin, la plupart du temps).
- Ne pas hésiter à faire des analogies avec ce qu'ils maîtrisent. Le défunt Minitel est un bon vecteur de traduction (www.sncf.fr vs 3615 SNCF, par exemple).
- Leur expliquer qu'il leur appartient de sécuriser leur connexion à Internet, mais qu'aucun monstre vert et gluant ne sortira jamais de l'écran pour manger leur disque dur.

Une fois rassurés, il faut leur apprendre une chose primordiale qui consiste à lire ce que l'écran affiche et à éviter de deviner et de cliquer n'importe où sans comprendre.

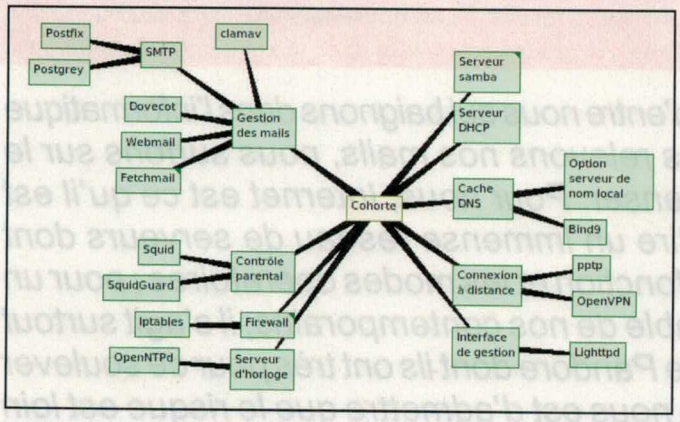
Une bonne explication de ce qui se passe sur Internet, de ce qu'est réellement un spam, un virus, un cheval de Troie, etc., peut les responsabiliser et leur faire comprendre ce qu'ils font. Combien d'entre eux mettent leur antivirus à jour uniquement parce que « la machine le demande » ? Utiliser un ordinateur, c'est comme conduire une voiture, si on part du principe que l'autre est un fou dangereux et qu'il va faire une bêtise, on a presque jamais d'accident. Une fois qu'on connaît l'autre, on peut lui laisser un peu plus d'espace et moins s'en méfier. Sur Internet, c'est pareil. Un mail dont on ne connaît pas la provenance, on l'ouvre avec des pincettes et surtout on ne clique pas sur la pièce jointe si on n'est pas sûr à 100% de ce qu'on fait.

Avec quelques conseils de bon sens et une rapide formation, on devrait commencer à limiter la casse due à 90% du temps à l'ignorance des utilisateurs.

1 POURQUOI L'AVOIR APPELÉ « COHORTE » ?

Dans l'armée romaine, une cohorte était une unité relativement petite de 300 hommes (480 à 800 après la réforme de Caius Marius vers -106) à la base de la stratégie guerrière de la république romaine. Dans ce même ordre d'idée, une machine cohorte a pour vocation de servir de brique de base dans une stratégie de qualité de l'accès à Internet et de défense de la liberté de nos utilisateurs non techniques.

2 PRÉSENTATION GÉNÉRALE



Comme indiqué dans le *mind map* ci-dessus, on peut répertorier plusieurs services importants que nous aimerions voir mis en place :

- Le contrôle parental grâce à Squid et à SquidGuard, qui fera l'objet du présent article.
- La gestion des mails à « géométrie variable » avec anti-spams, antivirus et webmail, dont nous détaillerons les arcanes dans la deuxième partie.
- Un serveur DHCP et un serveur DNS afin d'être autonome sur notre réseau local.
- Un système d'accès à distance via un VPN basé soit sur OpenVPN pour les Unices de tous poils, soit sur un PPTP pour les amateurs de simple ou double vitrage.
- Un serveur HTTP permettant de faire fonctionner le webmail et l'interface de configuration.
- Un firewall.
- Un serveur NTP (tant qu'on a les mains dedans...).
- Un serveur samba dont la présence peut sembler iconoclaste voire incongrue, mais qui nous permet de mettre à la disposition de nos amis windowsiens tout ce dont ils ont besoin en matière de logiciels libres – Ave les mises à jour, la maison ne recule devant aucun sacrifice – comme Firefox et quelques extensions type Adblock, Thunderbird, OpenOffice, etc. Ce service permettra aux gens de garder à jour leurs logiciels sans aller fouiller partout sur Internet.

3 LE CONTRÔLE PARENTAL

Terme dévoyé depuis les années 2000 par une série de « logiciels », dont le contournement est à la portée d'un crustacé arthritique et fraîchement trépané, qui n'ont servi au mieux qu'à alourdir encore une pauvre machine déjà bien surchargée et au pire, à fournir certains adolescents en URL croustillantes :)

Il est clair qu'une protection totale contre l'ensemble des sites porno/facho/trucphobes relève de la folie douce. Il faut être capable de confondre OpenOffice avec un firewall pour rêver de ça...

Nous n'essayerons donc pas de bloquer un ado bien décidé à aller parfaire son éducation, nous allons juste lui rendre la tâche la plus compliquée possible.

Pour atteindre ce but philanthropique, nous aurons besoin de deux comparses, j'ai nommé Squid et SquidGuard. Leur installation se passe sans douleur sous Debian grâce à un simple :

```
% aptitude install squid squiguard
```

à condition d'être en **sid/unstable**, **squidguard** n'étant pas packagé dans **squeeze**. Pour ce faire, il suffit d'ajouter la ligne suivante dans votre **/etc/apt/sources.list** :

```
deb ftp://ftp.fr.debian.org/debian/ sid main non-free contrib
```

Puis de faire un :

```
% aptitude update
```

Pour les utilisateurs d'Ubuntu, il suffit d'avoir les *repositories* Universe activés.

4 CONFIGURER SQUID

Une fois cette installation haletante achevée, nous pouvons passer à la configuration. Il ne faut pas perdre de vue notre objectif qui est de faire fonctionner SquidGuard le plus simplement possible. En aucun cas nous n'aborderons la configuration de Squid pour servir de proxy à aMsn ou autres. C'est d'ailleurs devenu un problème récurrent sur le net : comment trouver une configuration simple pour un logiciel donné ? On trouve tout ce qu'on veut pour laver le linge avec un Samba connecté à un LDAP via un VPN, mais pour dénicher une configuration de base pour démarrer, il n'y a plus rien... Bref, revenons à nos moutons. La configuration proposée est volontairement simple (qui a dit simpliste ?) mais elle reste parfaitement compatible avec une installation plus poussée de Squid.

Pour les besoins de l'article, nous supposons que votre réseau est en 192.168.0.0/24. N'oubliez pas de modifier la configuration ci-dessous en fonction de vos paramètres réels.

```
http_port 3128
cache_mem 64 MB
cache_dir ufs /var/cache/squid 1024
acl lan src 192.168.0.0/255.255.255.0 # Modify if needed
acl all src 0.0.0.0/0.0.0.0
http_access allow lan
visible_hostname Proxy
```

Ce fichier explique à Squid qu'il lui faut :

- Ecouter sur le port 3128.
- Allouer 64 Mo de cache en RAM.
- Allouer 1 Go de cache sur le disque.
- Créer une ACL (*Access Control List*) nommée **lan** qui regroupe tout le LAN.
- Créer une ACL nommée **all** qui regroupe tout le monde (Squid en a besoin).
- Autoriser l'accès en HTTP pour le LAN via l'ACL qui va bien.
- Enfin, présenter un *hostname* qui ne soit pas le nom de notre serveur.

Nous relançons Squid :

```
% /etc/init.d/squid restart
```

afin d'être sûr, puis nous testons son bon fonctionnement en ajoutant les paramètres qui vont bien dans notre navigateur internet.

Jusque-là, rien n'a changé vu de l'extérieur, mis à part un accès plus rapide aux pages en cache, mais ce n'est pas le but.

5 PARAMÉTRAGE DE SQUIDGUARD

Toute l'analyse niveau 7 va être faite par un comparse assez peu connu de Squid, j'ai nommé SquidGuard. Ce logiciel a pour but de récupérer l'URL ou l'IP demandée, regarder si elle est référencée dans une liste, la catégoriser et vérifier si la machine qui en fait la demande a le droit d'y accéder. Si oui, il laisse passer, sinon il renvoie sur une URL prédéfinie censée expliquer à l'utilisateur qu'il peut aller se faire voir chez les Vénusiens (pas toujours les mêmes qui s'y collent, ils ont déjà assez de soucis avec leur économie comme ça).

Il va donc falloir commencer par récupérer une liste de sites « sensibles ». Une archive est disponible sur le net à l'adresse suivante : **http://squidguard.mesd.k12.or.us/blacklists.tgz**.

Un simple **wget** nous permet de la rapatrier. Il suffit ensuite de la décompresser dans le répertoire prévu à cet effet :

```
% cd /var/lib/squidguard/db
% tar xvzf blacklists.tgz
```

Une fois cette étape trépidante terminée, nous pouvons nous attaquer au fichier de config de Squidguard : **/etc/squid/squidGuard.conf**. La syntaxe est relativement simple, elle est basée sur une série de déclarations suivies des ACL qui vont bien.

Commençons par les déclarations génériques :

```
dbhome /var/lib/squidguard/db/blacklists
logdir /var/log/squid

#
# TIME RULES:
# abbrev for weekdays:
# s = sun, m = mon, t =tue, w = wed, h = thu, f = fri, a = sat

time workhours {
        weekly mtwhf 08:00 - 16:30
        date *-*-01  08:00 - 16:30
}
```

Nous continuons avec la définition des sources, c'est-à-dire les machines et/ou les utilisateurs qui vont chercher à se connecter via Squid :

```
#src admin {
#       ip              1.2.3.4 1.2.3.5
#       user            root foo bar
#       within          workhours
#}

#src foo-clients {
#       ip              172.16.2.32-172.16.2.100 172.16.2.100
172.16.2.200
#}

#src bar-clients {
#       ip              172.16.4.0/26
#}
```

Pour le moment, nous n'en définissons aucun, cela signifie que tout le LAN est concerné. Après les sources, nous définissons les « destinations » :

```
dest good {
}
dest local {
}
dest ads {
        domainlist      ads/domains
        urllist         ads/urls
}
dest aggressive {
        domainlist      aggressive/domains
        urllist         aggressive/urls
}
dest audio-video {
        domainlist      audio-video/domains
        urllist         audio-video/urls
}
dest drugs {
        domainlist      drugs/domains
        urllist         drugs/urls
}
dest gambling {
        domainlist      gambling/domains
        urllist         gambling/urls
}
dest hacking {
        domainlist      hacking/domains
        urllist         hacking/urls
}
dest mail {
        domainlist      mail/domains
}
dest porn {
        domainlist      porn/domains
        urllist         porn/urls
}
dest proxy {
        domainlist      proxy/domains
        urllist         proxy/urls
}
dest redirector {
        domainlist      redirector/domains
        urllist         redirector/urls
}
dest spyware {
        domainlist      spyware/domains
        urllist         spyware/urls
}
```

```
dest suspect {
        domainlist      suspect/domains
        urllist         suspect/urls
}
dest violence {
        domainlist      violence/domains
        urllist         violence/urls
}
dest warez{
        domainlist      warez/domains
        urllist         warez/urls
}
```

Là, nous avons découpé soigneusement en fonction des listes noires préalablement installées. Enfin, nous pouvons mettre tout ce petit monde en marche avec les fameuses ACL :

```
acl {
#       admin {
#               pass     any
#       }

#       foo-clients within workhours {
#               pass     good !in-addr !adult any
#       } else {
#               pass any
#       }

#       bar-clients {
#               pass    local none
#       }

        default {
                pass !ads !aggressive !audio-video !drugs !gambling !hacking
!mail  !porn  !proxy  !redirector  !spyware  !suspect  !violence  !warez  all
                redirect        http://localhost/block.html

        }
}
```

Comme vous pouvez le constater, une multitude de possibilités de filtrage s'offrent à nous, et encore, nous n'en voyons là qu'une petite partie. Pour ceux que ça intéresse, un petit tour sur le net, voire dans le code source de Squidguard, ne sera pas inutile, la documentation officielle est en effet assez indigente.

La traduction de la seule ACL active est plutôt simple à comprendre. Elle peut se traduire par : Par défaut, tu laisses passer tout ce qui n'est pas dans la « destination » **ads**, ni dans la destination « agressive » ni.... ni dans la « destination » **warez**. Si c'est dans l'une de celles-ci, tu rediriges sur la page locale nommée **block.html**.

Cette page est à votre entière discrétion. Un simple message suffira pour tester :

```
<html>
<head>
<title>Blocked by squidGuard</title>
</head>
<body>
<center><h2>You are not allowed to watch this website</h2>
<p>You may go and troll on <a href='http://linuxfr.org'>LinuxFr :-p</a></p>
</center>
</body>
</html>
```

Placez simplement ce code dans **/var/www/block.html**.

Dans le cas présent, j'ai utilisé le serveur HTTP interne (un bon vieux Lighttpd) pour renvoyer la page de blocage. Nous verrons son installation et sa configuration dans un article ultérieur mais, sans dévoiler des secrets d'état, je peux déjà vous révéler qu'un :

```
% aptitude install lighttpd
```

fait 99% du travail.

Il nous reste encore à régler quelques permissions :

```
% chown proxy:proxy -R /var/lib/squidguard/db/*
% find /var/lib/squidguard/db -type f | xargs chmod 644
% find /var/lib/squidguard/db -type d | xargs chmod 755
</code<

puis demander à squidGuard de compiler ses règles avec la commande suivante :

<code>
sudo -u proxy squidGuard -C all
```

Nous finissons par expliquer à Squid qu'il doit jouer avec son petit camarade en ajoutant la ligne suivante à la fin du fichier **/etc/squid/squid.conf** :

```
redirect_program /usr/bin/squidGuard
```

Enfin, nous relançons Squid, soit avec un bon vieux **/etc/init.d/squid restart**, soit avec une méthode plus « douce » : **squid -k reconfigure** pour être certain que tout est bien en place et nous pouvons tester avec une URL « chaude » du genre **www.rotten.com**, ou toute autre que vous connaissez certainement par cœur.

Si tout va bien, vous devez voir apparaître votre page de blocage, sinon reprenez les deux configurations calmement et vérifiez pas à pas. N'oubliez pas de vider le cache de votre navigateur entre chaque essai. Je vous conseille même de le supprimer complètement, Squid prenant le relais avec brio.

6 UTILISATION DE L'AUTHENTIFICATION

Nous avons vu dans le chapitre précédent qu'il est possible de définir des sources pour SquidGuard. Si vous voulez sélectionner les autorisations en fonction des adresses IP de vos machines, il vous suffit d'ajouter une ou plusieurs sources du genre :

```
src foo-clients {
      ip            192.0.2.32-192.168.0.100 192.168.0.105 192.168.0.200
}

src bar-clients {
      ip              192.168.4.0/26
}
```

et les ACL correspondantes du style :

```
foo-clients within workhours {
        pass     !porn !warez any
} else {
        pass any
}
```

Ce qui signifie que les clients dont l'IP est listée dans la source **foo-clients** ont le droit d'aller partout sauf sur les sites de warez et de porn pendant les heures de travail. Le reste du temps, ils font ce qu'ils veulent :)

Cette configuration est très bien pour ceux qui ont des machines dédiées, mais que faire en cas de machine partagée ? La seule réponse possible est de forcer les

utilisateurs à s'authentifier. Pour cela, il existe plusieurs méthodes supportées par Squid, mais la plus simple reste celle dénommée NCSA. Elle consiste à suivre les étapes suivantes :

- Créer les comptes avec **htpasswd**.
- Modifier la configuration de Squid.
- Ajouter quelques ACL dans **squidGuard.conf**.

Créer les comptes avec **htpasswd** : il faut commencer par récupérer **htpasswd** avec un **aptitude install apache2-utils**, puis créer le premier utilisateur avec la commande :

```
% htpasswd -c /etc/squid/passwd admin
```

et les suivants avec :

```
% htpasswd /etc/squid/passwd toto
```

Modifier la configuration de Squid : il faut ajouter quelques lignes dans le fichier **/etc/squid/squid.conf** :

```
auth_param basic program /usr/lib/squid/ncsa_auth /etc/squid/passwd
auth_param basic children 5
auth_param basic realm Veuillez vous connecter SVP
auth_param basic credentialsttl 1 hours
auth_param basic casesensitive off

acl ncsa_users proxy_auth REQUIRED
http_access allow ncsa_users
```

Nous pouvons relancer Squid avec :

```
% squid -k reconfigure
```

Ajouter des ACL dans **squidGuard.conf** : maintenant que vous manipulez les ACL comme des pros, je vais juste me contenter d'un exemple simple :

```
src root {
        user admin
}

src others {
        user toto
}

acl {
        root {
                pass all
        }
        others {
                pass !drugs !gambling !hacking !mail !porn !proxy
!redirector !spyware !suspect !violence all
                redirect        http://localhost/block1.php?clientaddr=%a
&clientuser=%i&clientgroup=%s&url=%u
        }
        default {
                pass !ads !aggressive !audio-video !drugs !gambling !hacking
!mail !porn !proxy !redirector !spyware !suspect !violence !warez all
                redirect        http://localhost/block1.php?clientaddr=%
a&clientuser=%i&clientgroup=%s&url=%u
        }
}
```

En résumé, nous laissons passer les utilisateurs de la source **root** (c'est-à-dire **admin**) et nous filtrons les autres. Vous remarquerez aussi que l'URL de redirection s'est enrichie de plusieurs paramètres. Ceux-ci permettent de personnaliser votre page de blocage en passant en GET l'adresse du client, son user, son groupe et l'URL qu'il espérait atteindre. Vous pouvez bien entendu utiliser tout ça dans un petit développement en PHP, Python, C ou Seaside pour affiner les permissions en fonction de plein d'événements extérieurs (notamment la présence ou l'absence d'une clé RFID, extension qui verra peut-être le jour prochainement ; vous serez les premiers informés...).

Une autre possibilité consiste à utiliser PAM pour l'authentification. Pour cela, il faut remplacer dans **squid.conf** la ligne :

```
auth_param basic program /usr/lib/squid/ncsa_auth /etc/squid/passwd
```

par :

```
auth_param basic program /usr/lib/squid/pam_auth
```

Il faut ensuite créer le fichier **/etc/pam.d/squid** avec les deux lignes suivantes :

```
auth     required /lib/security/pam_unix.so
account  required /lib/security/pam_unix.so
```

et créer les users sans shell :

```
% useradd toto -s /sbin/nologin
```

Pour debugger tout ça, une bonne méthode consiste à remplacer :

```
redirect_program /usr/bin/squidGuard
```

par :

```
redirect_program /usr/bin/tee -a /tmp/redirector.log
```

dans **squid.conf**.

N'oubliez pas de relancer Squid après chaque modification !

CONCLUSION (PROVISOIRE ?)

Voilà ce qu'en peu de mots l'on peut dire d'une installation de base de Squid et SquidGuard dans le but de créer son propre système de contrôle parental. Il est bien évident que ce système est loin d'être incontournable, mais il a l'avantage d'être gratuit, simple et surtout maîtrisé à 100% par vous-même. Il vous est loisible d'ajouter ou d'enlever des URL des listes noires, d'affiner le filtrage en fonction de vos besoins... Bref, vous avez la mainmise sur l'outil.

Le contournement le plus simple consiste à supprimer le support du proxy dans le navigateur. C'est bien essayé, petit, mais t'as pas vu la gentille petite règle **iptables** que j'ai mise sur le routeur :

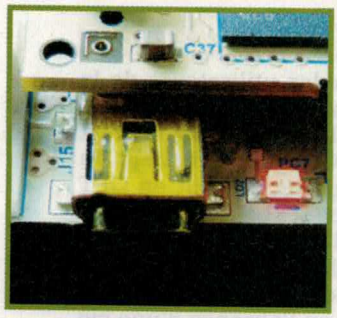
```
%iptables -A -tnat -APREROUTING -ieth0 -ptcp -dport80 -jREDIRECT -to-port3128
```

Celle-ci ne fonctionne que si vous mettez Squid sur votre routeur, s'il s'agit d'une machine séparée, il faudra penser à la laisser accéder au net, sinon vous risquez de boucler un brin... Je vous laisse trouver la solution par vous-même, interrogation écrite demain.

Bien évidemment, cet article ne constitue pas du tout une documentation exhaustive sur Squid, ni même sur SquidGuard. Une multitude d'informations existent sur la Toile, plus ou moins complètes et plus ou moins à jour, attention à bien faire le tri. ■

Auteur : Cédric Pellerin

Exploration de la nouvelle



Nous vous avons déjà parlé, par le passé, des cartes ACME. Le précédent modèle, construit autour du processeur ETRAX 100LX d'Axis aujourd'hui en fin de vie, est maintenant relégué dans l'ombre par la nouvelle Fox G20 utilisant un AT91SAM9G20, un processeur ARM9 cadencé à 400 Mhz. Mais les nouveautés ne se limitent pas à cela.

Dire que l'ancienne carte LX832, et implicitement la Fox LX et la Fox « classique », sont de l'histoire ancienne est aller un peu vite en besogne. La LX832 (8 Mo de Flash et 32 de RAM) succéda à la LX (4 Mo de Flash et 16 de RAM) qui est une évolution de la Fox (4/16 également, mais en MCM pour *Multi Chip Module*). De toutes ces cartes, la seule encore en production est la LX832, mais le fabricant recommande fortement l'utilisation de la nouvelle Fox G20 pour les nouveaux projets. Personnellement, je dispose d'une carte Fox de toute première génération, qui me sert pour différents projets et reposant sur OpenWRT. Il est évident, toutefois, que même si elle reste parfaitement adaptée, sa grande sœur, pour quelques 30 euros de plus, offrira plus de fonctionnalités. Le tableau 1 présente un comparatif des deux générations de cartes Fox.

	FOX Board LX832	FOX Board G20
SDRAM	32 Mo 16 bit 50 MHz	**64 Mo** 32 bit 133 MHz
Dataflash	Non	8 Mo
Norflash	8 Mo	Non
microSD	Non	**Max 8 Go**
CPU	Axis ETRAX 100LX	Atmel AT91SAM9G20
CPU Clock	100 MHz	**400 Mhz**
Real Time Clock	Non	**Oui**
Batterie RTC	Non	**Oui**
USB Host	2 x USB 1.1	2 x USB 2.0
USB Client	Non	Oui
Port Console	Oui	Oui
Consommation	220 mA @ 5V	**60 mA** @ 5V
Canaux A/N	Non	4 x 10bit
PWM	Non	Oui
I2C(TWI)	Soft	**Hard**
SPI	Soft	**Hard**
LCD	Non	Connecteur 4D Systems oLED
Interface CCTV	Non	ITU-R BT. 601/656
Interface JTAG	Non	Oui
UART	2	**6**

Comme vous pouvez le voir, la G20 dispose d'une puissance de calcul bien supérieure ainsi que d'un emplacement pour carte microSD. Celle-ci est destinée à accueillir le système d'exploitation (Gentoo ou Debian) et la carte ne fonctionnera pas sans ce support, malgré la présence de 8 Mo de Flash. La présence d'une RTC est une grande évolution par rapport aux modèles précédents avec lesquels il était nécessaire de faire appel à un montage extérieur alimenté par une pile lithium type CR2032. Le port destiné aux afficheurs LCD ou oLED de 4D Systems n'est autre qu'un port série disposant de ligne d'alimentation 5V et de masse. La consommation a été grandement réduite, même s'il ne s'agit là que d'une indication concernant la carte elle-même (et non les périphériques USB ou autres ajoutés et auto-alimentés). Une grande partie des E/S ont également subi un gros changement avec de l'i2c et la SPI gérées matériellement. ACME a choisi de conserver une compatibilité mécanique. La Fox G20 s'intégrera en lieu et place des précédentes cartes assez facilement. Autre changement notable, la carte se compose en réalité d'un module AT91SAM9G20 de 4 cm de côté enfiché sur la carte d'accueil, permettant ainsi de réduire l'encombrement (et le coût) pour les projets spécifiques.

Attention

Contrairement aux précédentes cartes ACME Fox, la G20 n'est PAS tolérante au 5V ! Vous devez travailler en 3.3V. Dans le cas contraire, vous allez endommager les ports en entrée, et ce irrémédiablement. Lisez la documentation, certaines lignes GPIO sont en 1.8V. Et ces remarques s'appliquent également aux différents ports série.

Il faut bien comprendre que les philosophies de développement et d'architecture sont bien différentes d'un modèle à l'autre. La mémoire Flash NOR a disparu au bénéfice d'une mémoire de stockage NAND. Il ne s'agit plus ici de faire fonctionner le système en Flash, mais d'y placer les *bootloaders* pour un démarrage sur microSD (qui malheureusement est NAND également, comme tout le monde le sait). Ceci, qui peut être vu comme un défaut architectural, est compensé par la puissance de l'ensemble et la possibilité d'utiliser un système GNU/Linux presque « normal », comme une distribution Debian pour Armel. Il est clair cependant qu'on s'éloigne du concept strict de système embarqué et de ses contraintes d'un point de vue logiciel (un `fsck` au démarrage après interruption brutale est plutôt surprenant sur ce type de plate-forme).

carte ACME Fox G20

Note : Mémoire Flash NOR ET NAND

Lorsqu'on parle de Flash NOR ou NAND, on fait référence à la technologie utilisée par la mémoire. La Flash NOR est la plus ancienne, plus lente et moins dense, mais elle permet un accès aléatoire aux données et donc de faire de l'XiP (*eXecute in Place*). La mémoire Flash NAND est plus récente et permet un stockage plus important. Plus économique, elle est utilisée pour le stockage des données, mais n'est pas adaptée pour les programmes. De plus, l'intégrité n'est pas garantie à 100% et les fabricants ajoutent en général un système de gestion/correction d'erreurs, comme pour les disques durs. Retenez simplement que NOR = plus cher / pour les programmes et NAND = moins cher / pour le stockage.

1 RECONSTRUCTION DU SYSTÈME

1.1 Prérequis

Comme pour tous systèmes embarqués, il n'est pas concevable, même si les cartes intègrent des ressources (CPU et mémoire) de plus en plus conséquentes, de compiler sur la plate-forme elle-même. On optera donc pour la cross-compilation, tout à fait classiquement. ACME a bien préparé le terrain, contrairement à d'autres fabricants, puisque l'ensemble des outils sont disponibles et reposent sur l'utilisation de distributions populaires et adaptées.

La carte est vendue en kit intégrant une carte microSD avec, au choix, une distribution Debian ou Gentoo. Pour notre part, vous vous en serez douté, nous avons opté pour Debian. De ce fait, l'ensemble du processus de cross-construction du système embarqué se fera sur un hôte fonctionnant également sous Debian.

Il vous faudra ajouter une ligne à votre **sources.list** :

```
deb http://www.emdebian.org/debian/ lenny main
```

La présence de l'USB host permet toutes les extravagances, stockage, son, Wifi, ...

Après un **aptitude update**, installez immédiatement **emdebian-archive-keyring** et procédez à un nouvel **update**. L'installation des outils adéquate se résumera donc par la mise en œuvre de l'outil **aptitude** avec les paquets suivants :

```
% sudo aptitude install libc6-armel-cross libc6-dev-armel-cross \
binutils-arm-linux-gnueabi gcc-4.3-arm-linux-gnueabi \
g++-4.3-arm-linux-gnueabi uboot-mkimage apt-cross \
dpkg-cross libncurses5-dev linux-libc-dev-armel-cross
```

Ceux-ci nous permettront à la fois de compiler un noyau 2.6.32 pour la carte, mais également de construire le système de fichiers racine. Nous pourrons également construire via les outils **dpkg*** des paquets pour nos propres applications.

1.2 Le noyau

ACME Systems a déjà fait énormément de travail pour vous. En effet, un noyau Linux parfaitement fonctionnel est déjà disponible pour la carte G20. Il est bien entendu possible de recompiler un noyau à partir des sources vanilla 2.6.32.2, à condition de leur appliquer un patch spécifique disponible sur le Wiki ACME System.

Pour la construction de l'ensemble, on commence donc par récupérer et décompresser les sources du noyau en question. Puis on applique le patch adéquat :

```
% cd kkpart
% mkdir G20
% cd G20

% wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.32.2.tar.
bz2
% tar xfjv linux-2.6.32.2.tar.bz2

% wget http://foxg20.acmesystems.it/lib/exe/fetch.php?\
media=debian:linux-2.6.32.2-foxg20-patches-v5.txt
% mv fetch.php\?media\=debian\:linux-2.6.32.2-foxg20-patches-v5.txt \
linux-2.6.32.2-foxg20-patches-v5.txt

% cd linux-2.6.32.2
% patch -p1 < ../linux-2.6.32.2-foxg20-patches-v5.txt
patching file arch/arm/Kconfig
patching file arch/arm/mach-at91/at91sam9260_devices.c
```

```
patching file arch/arm/mach-at91/board-foxg20.c
patching file arch/arm/mach-at91/Kconfig
patching file arch/arm/mach-at91/Makefile
patching file drivers/mmc/host/at91_mci.c
patching file foxg20.config-2.6.32.2-COMPLEX-v3
patching file foxg20.config-2.6.32.2-SIMPLE
patching file foxg20-script
patching file makefile
patching file README.FOX
```

Nous disposons maintenant de sources adaptées. Plusieurs fichiers ont été ajoutés, dont un fichier de configuration du noyau en version légère (**foxg20.config-2.6.32.2-SIMPLE**), une configuration plus complète et plus lourde (**foxg20.config-2.6.32.2-COMPLEX-v3**) et un **README.FOX** très concis. A ce stade, si vous avez acquis une carte comprenant une microSD sous Debian, il est de bon ton d'éventuellement récupérer une copie de la configuration du noyau actuel. Pour cela, connectez-vous à la carte, chargez le module **configs**, puis récupérez le fichier **/proc/config.gz** via le réseau ou une clé USB. Un **diff** vous montrera uniquement de très légères différences entre cette configuration et celle stockée dans **foxg20.config-2.6.32.2-COMPLEX-v3** (quelque options de debugage tout au plus).

Nous allons maintenant construire le noyau en nous basant sur l'une des configurations disponibles :

```
% cp foxg20.config-2.6.32.2-COMPLEX-v3 .config
% make menuconfig
% make uimageGzip
```

Après de longues minutes de travail intensif du processeur, la commande devrait aboutir sur un message comme :

```
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
make[2]: quittant le répertoire " /mnt/DEVEL/G20/linux-2.6.32.2 "
make[1]: quittant le répertoire " /mnt/DEVEL/G20/linux-2.6.32.2 "
gzip -v9 < arch/arm/boot/Image > Image.gz
51.7%
/bin/bash scripts/mkuboot.sh \
               -A arm \
               -O linux \
               -T kernel \
               -C gzip \
               -a 0x20008000 \
               -e 0x20008000 \
               -n $(pwd | sed 's,.*/,,;s/^/FOX-/;') \
               -d Image.gz ./uImage
Image Name:   FOX-linux-2.6.32.2
Created:      Tue Feb 23 16:02:46 2010
Image Type:   ARM Linux Kernel Image (gzip compressed)
Data Size:    1792203 Bytes = 1750.20 kB = 1.71 MB
Load Address: 0x20008000
Entry Point:  0x20008000

Your new kernel is in: ./uImage
```

Nous obtenons ainsi un fichier **uImage**, qui est une image du noyau directement utilisable avec U-Boot. Avant de nous pencher sur son installation, il convient de compiler l'ensemble des modules pouvant l'accompagner :

```
% make modules
% make modules_install
```

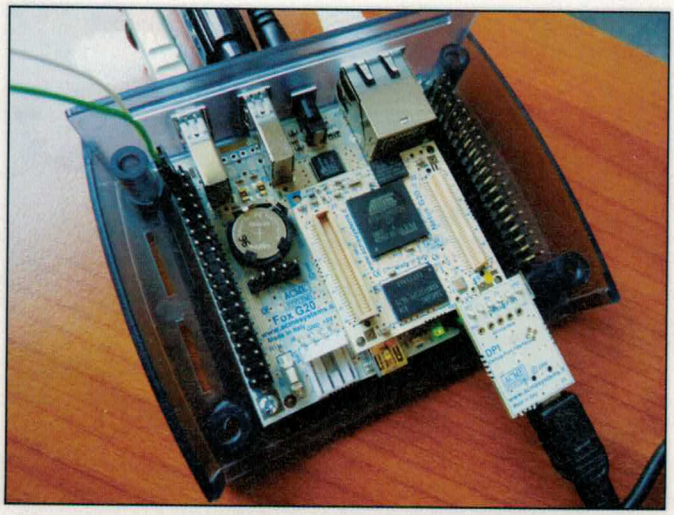
Ne vous inquiétez pas pour cette dernière commande. Vous n'allez pas installer les modules dans le système hôte, mais dans un sous-répertoire **FoxModules** dans les sources du noyau. Ceci provient du fait que le patch précédemment appliqué modifie les **Makefile** et automatise ainsi bon nombre de choses à votre place.

L'installation du nouveau noyau sur le système embarqué est relativement simple, puisqu'il vous suffit de monter la première partition contenant le système de fichiers FAT16 et d'y copier l'image du noyau. Attention, le fichier en question doit s'appeler **uimage** sur le support et non **uImage**. Ceci peut être fait en retirant la microSD de la carte ACME et en l'installant sur le système hôte ou directement sur le système embarqué lui-même, en montant **/dev/mmcblk0p1** dans un répertoire quelconque et en écrasant le fichier **uimage** précédent.

```
# mount /dev/mmcblk0p1 /mnt
# cp /tmp/uImage /mnt/uimage
# sync
# umount /mnt
```

Le script de démarrage destiné à U-Boot (un peu comme le **menu.lst** de GRUB legacy) est également « compilé » lors de la fabrication de l'image du noyau via la cible **foxg20-script.bin**. Celui-ci utilise le fichier **foxg20-script** avec l'outil **mkimage** pour produire une version binaire du script. Il suffira alors de la copier en compagnie de l'image du noyau pour le mettre à jour. Notez qu'il vous est parfaitement possible de changer le contenu de **foxg20-script** pour personnaliser le script. Cependant, il sera sans doute préférable de copier le fichier original pour manuellement produire un autre fichier binaire avec :

```
% mkimage -C none -T script -n 'FOXG20 Boot Script' \
  -d foxg20-scriptLEF foxg20-script.my.bin
Image Name:   FOXG20 Boot Script
Created:      Thu Apr 29 10:38:02 2010
Image Type:   PowerPC Linux Script (uncompressed)
Data Size:    247 Bytes = 0.24 kB = 0.00 MB
Load Address: 0x00000000
Entry Point:  0x00000000
Contents:
   Image 0:      239 Bytes =    0 kB = 0 MB
```



La carte G20 bordée de connecteurs au pas de 2.54 permettant de facilement développer une carte fille.

Note : En cas de problème

La carte G20 est équipée de 8 Mo de mémoire flash NAND contenant, entre autres choses, un bootloader de stage 1 et le bootloader stage 2 U-Boot. En cas d'échec du démarrage du système, vous pouvez accéder au shell U-Boot peu de temps après la mise sous tension via la console série à 115200 bps. U-Boot est un environnement très complet capable d'analyser à la fois la mémoire, la Flash et le contenu de la microSD. De plus, en cas d'impossibilité de charger, décompresser ou exécuter le noyau, vous disposez d'un certain nombre de méthodes alternatives dont BOOTP/TFTP, DHCP/TFTP, RARP/TFTP et même NFS.

Ainsi, un message comme celui-ci ne vous mènera pas dans une impasse :

```
## Booting kernel from Legacy Image at 20000000 ...
   Image Name:   FOX-linux-2.6.32.2
   Image Type:   ARM Linux Kernel Image (gzip compressed)
   Data Size:    1792203 Bytes =  1.7 MB
   Load Address: 20008000
   Entry Point:  20008000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ...
Error: inflate() returned -3
GUNZIP: uncompress, out-of-mem or
overwrite error - must RESET board to recover
resetting ...
```

On commencera par vérifier les variables d'environnement en place avec :

```
foxg20> printenv
ethaddr=00:04:25:AE:00:88
bootcmd=mmc init; fatload mmc 0 0x23000000
   foxg20-script.bin; source 0x23000000
stdin=serial
stdout=serial
stderr=serial
ethact=macb0
Environment size: 160/16892 bytes
```

Nous constatons qu'un script de démarrage est défini pour un accès à la microSD et un chargement du script **foxg20-script.bin** depuis un système de fichiers FAT16 à l'adresse **0×23000000**, pour finalement passer la main au script en question.

Pour nous assurer du fonctionnement de l'ensemble, nous commençons par initialiser l'accès à la microSD :

```
foxg20> mmc init
mmc: setting clock 150000 Hz, block size 512
mmc: clock 150000 too low; setting CLKDIV to 255
Manufacturer ID:       1A
OEM/Application ID:    5051
Product name:          SD
Product Revision:      1.0
Product Serial Number: 51957
Manufacturing Date:    09/08
SD Card detected (RCA 45928)
CSD data: 32002f00 23c1a5f5 ffb6dedd 000aa7fd
CSD structure version:   1.0
MMC System Spec version: 0
Card command classes:    5f5
Read block length:       1024
Supports partial reads
Write block length:      1024
Does not support partial writes
Supports group WP:       32
Card capacity:           1995440128 bytes
File format:             0/0
Write protection:
mmc: Using 3145728 cycles data timeout (DTOR=0x73)
mmc: setting clock 5000000 Hz, block size 512
.mmc1 is available
```

Nous vérifions ensuite la première partition :

```
foxg20> fatinfo mmc 0
..Interface:  MMC
  Device 0: Vendor: Man 1a5051 Snr 0000caf5 Rev: 1 0 Prod: SD
            Type: Hard Disk
            Capacity: 951.5 MB = 0.9 GB (1948672 x 512)
.Partition 1: Filesystem: FAT16 "kernel      "
```

Puis listons son contenu :

```
foxg20> fatls mmc 0
.......
  1792267   uimage
      231   foxg20-script.bin
2 file(s), 0 dir(s)
```

De ce côté, tout semble donc fonctionner parfaitement. Le problème est ailleurs. En effet, c'est lors de la décompression du noyau que le problème survient. Il s'agit tout simplement d'un problème d'adresse de chargement de l'image du noyau. Nous pouvons ainsi, dans un premier temps, tester ce même nouveau noyau en procédant à son chargement manuel depuis un serveur TFTP. Pour cela, nous commençons par installer un serveur TFTP (ici sur 192.168.0.1) et y placer le fichier **uImage** obtenu précédemment. Nous revenons ensuite vers U-Boot pour faire le test.

Nous définissons deux nouvelles variables, respectivement pour préciser l'adresse du serveur TFTP et le nom de l'image à récupérer :

```
foxg20> set serverip 192.168.0.1
foxg20> set bootfile uImage
```

Puis, prenant en compte le fait qu'un serveur DHCP ait bien été mis en place pour attribuer une adresse à la carte ACME, nous utilisons la commande U-Boot **dhcp** pour obtenir cette dernière et charger le fichier image en mémoire :

```
foxg20> dhcp
macb0: link up, 100Mbps full-duplex (lpa: 0x45e1)
BOOTP broadcast 1
DHCP client bound to address 192.168.0.170
Using macb0 device
TFTP from server 192.168.0.1;
  our IP address is 192.168.0.170
Filename 'uImage'.
Load address: 0x22000000
Loading: #################################
         ##################
done
Bytes transferred = 1795376 (1b6530 hex)
```

Nous spécifions ensuite les arguments à passer au noyau via la définition d'une variable **bootargs** contenant **'mem=64M console=ttyS0,115200 noinitrd root=/dev/mmcblk0p2 rw rootwait init=/sbin/init'**. Enfin, nous provoquons l'exécution avec **bootm**, en précisant bien l'adresse à utiliser (**0×22000000**).

Pour les tests, vous pourrez interrompre le démarrage pour accéder à U-Boot et changer la variable **bootcmd** afin de faire pointer la configuration vers le nouveau fichier script binaire. Une fois satisfait, vous pourrez soit renommer votre fichier en **foxg20-script.bin** sur la partition VFAT de la microSD ou, plus simplement, enregistrer la variable avec un **saveenv** dans U-Boot.

En ce qui concerne les modules, ceux-ci sont installés sur le système de fichiers ext2 (le reste de la microSD, **/dev/mmcblk0p2**). Ici, la méthode la plus simple consiste à faire appel à un outil comme **rsync** (bien qu'un simple **scp** fera également l'affaire) depuis l'hôte de cross-compilation :

```
% rsync -avc FoxModules/lib/. root@192.168.0.170:/lib/.
modules/2.6.32.2/kernel/sound/usb/
modules/2.6.32.2/kernel/sound/usb/caiaq/

sent 2012385 bytes  received 77968 bytes  76012.84 bytes/sec
total size is 21668139  speedup is 10.37
```

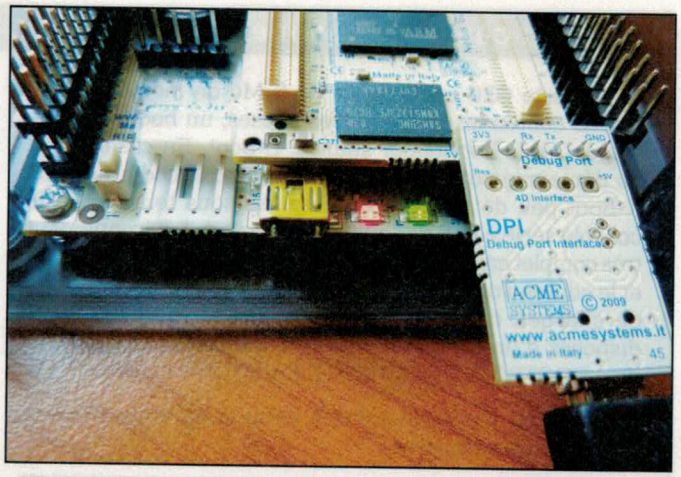
On n'oubliera pas d'actualiser le fichier **modules.dep** via un petit **depmod -a** sur le système embarqué juste après la copie. Ne reste ensuite plus qu'à redémarrer en croisant les doigts si l'on a vraiment joué avec la configuration du noyau.

1.3 Rootfs

Le système de fichiers racine de la carte G20 est, par défaut, la seconde partition de la carte microSD. Il existe des alternatives permettant, selon votre configuration d'U-Boot, de démarrer avec un rootfs sur une clé USB Mass Storage ou, éventuellement, sur la flash NAND intégrée. Notez cependant que cette dernière possibilité n'a pas été explorée avec succès dans nos expérimentations, la flash en question, Atmel AT45DB642 en SPI, ne semblant pas être détectée par le noyau par défaut tel que livré par ACME Systems. Ceci semble provenir de la conception même de la carte. En effet, les mêmes lignes du processeur sont utilisées pour le bus SPI0 et l'accès à la MMC. Il est impossible d'accéder à la fois aux deux bus en même temps. En d'autres termes, si vous activez la MMC pour booter sur la carte SD, vous n'avez pas accès à la flash intégrée, et inversement. Ceci est, par ailleurs, pris en charge dans les sources patchées du noyau (**arch/arm/mach-at91/board-foxg20.c**) avec des choses comme :

```
/* * SPI devices. */ static struct spi_board_info foxg20_spi_devices[] =
{ #if !defined(CONFIG_MMC_AT91)
{
  .modalias       = "mtd_dataflash",
  .chip_select    = 1,
  .max_speed_hz   = 15 * 1000 * 1000,
  .bus_num        = 0,
},
#endif };
```

Il reste néanmoins possible d'utiliser uniquement la flash interne de la carte G20 aussi bien pour U-Boot que pour le noyau et le rootfs, avec un stockage des données sur clé USB, par exemple (voire du rootfs lui-même). On notera d'ailleurs que les partitions MTD sont déjà prévues dans ce sens.



Le connecteur USB device permet à Linux via le support USB gadget de faire passer la carte G20 pour un périphérique USB de son choix.

```
foxg20> flinfo
DataFlash:AT45DB642
Nb pages:   8192
Page Size:   1056
Size= 8650752 bytes
Logical address: 0xD0000000
Area 0: D0000000 to D00041FF (RO) Bootstrap
Area 1: D0004200 to D00083FF      Environment
Area 2: D0008400 to D0041FFF (RO) U-Boot
Area 3: D0042000 to D0251FFF      Kernel
Area 4: D0252000 to D083FFFF      FS
```

Revenons-en à l'installation par défaut. La carte MicroSD doit être partitionnée de manière à disposer d'une partition VFAT (FAT16) pour le noyau et le script de configuration U-Boot ainsi que d'une partition Linux. Ceci fait, une fois les systèmes de fichiers initialisés avec **mkfs.dos** et **mke2fs**, vous pourrez les monter sur le PC hôte.

Pour obtenir un système identique (ou plus à jour) que celui installé par défaut sur la carte livrée, il vous suffira de récupérer le fichier **http://foxg20.acmesystems.it/download/microsd_20100413/rootfs.tar.bz2**, puis de désarchiver le tarball directement sur le système de fichiers monté :

```
% sudo tar xvjpSf rootfs.tar.bz2 -C /media/rootfs
```

Notez qu'il est impératif de procéder à la manipulation en tant qu'utilisateur **root** afin que tous les fichiers et répertoires appartiennent effectivement à l'utilisateur UID 0. N'oubliez pas également de copier les éléments du noyau sur le système de fichiers VFAT. Ceci fait, quittez le point de montage, démontez le système de fichiers et placez la carte dans l'emplacement de la carte G20. Il n'y a rien d'autre à configurer puisque U-Boot est déjà paramétré pour chercher son script et le noyau sur la partition VFAT.

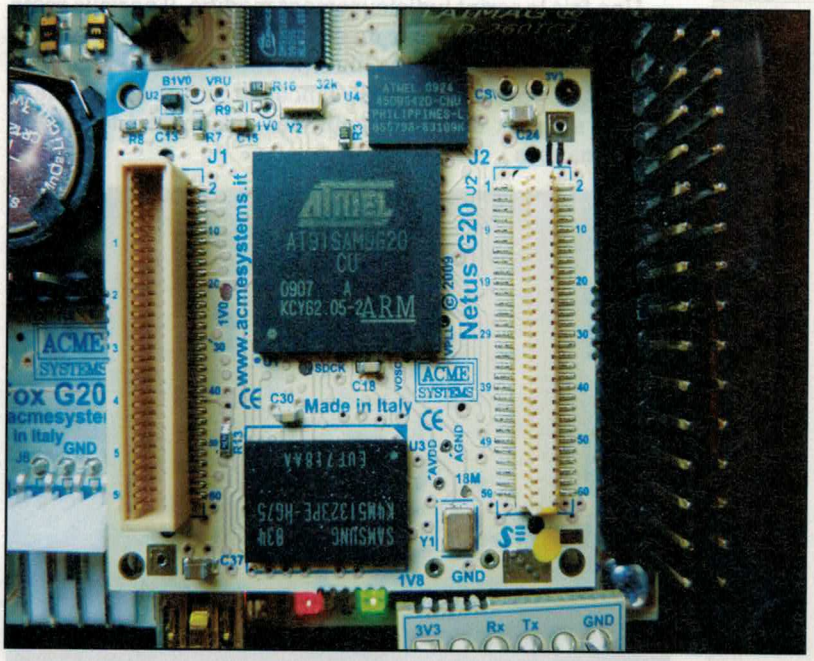
Il est également possible de construire, de toutes pièces, un système Debian en partant d'une base minimaliste et en installant ensuite les paquets qui vous intéressent. Il sera également possible de cross-compiler les paquets avec les outils **dpkg-cross** et **apt-cross**. ACME met à disposition un script **gen_root3.sh** se chargeant d'une grande partie des opérations en reposant sur **debootstrap**.

En tête du script shell en question, il est possible de personnaliser un certain nombre d'éléments de configuration :

```
# L'url où récupérer les fichiers Debian
# Choisissez de préférence un miroir proche
# de vous (ou testé avec apt-spy
url=http://ftp.fr.debian.org/debian
# Le dépôt classique pour les mises à jour
# de sécurité. Ne pas changer, sauf si
# vous maintenez votre propre miroir
secure_url=http://security.debian.org/

# Paramétrage de l'architecture et de la
# distribution à utiliser. Normalement,
# vous n'avez rien à toucher ici.
arch=armel
root=$arch-root
dist=lenny

# Cette variable DOIT IMPERATIVEMENT
# correspondre à la version du noyau que
# vous allez utiliser sur la carte G20
kernel_version=2.6.31.5

# Eléments de configuration du réseau
# pour le futur système. Ces informations
# seront placées dans le
# /etc/network/interface
# du système

#
# eth peut prendre les valeurs
#    static pour un IP fixe
#    dhcp pour une configuration DHCP
#
# Dans le premier cas, il est nécessaire
# de spécifier les différents éléments
# de la configuration.
#
# eth peut prendre les valeurs
#    static pour un IP fixe
#    dhcp pour une configuration DHCP
#
# Dans le premier cas, il est nécessaire
# de spécifier les différents éléments
# de la configuration.
eth=static
address=192.168.1.90
netmask=255.255.255.0
gateway=192.168.1.1
#eth=dhcp

# Le nom d'hôte de la cible
newhostname=debarm

# Liste des paquets supplémentaires
# à installer
# Ne placez pas ici l'ensemble des paquets
# que vous souhaitez installer sur le système
# le premier correspondra au premier reboot
# dans une phase d'installation classique
# Beaucoup de paquets nécessitent un système
# complet et configuré pour pouvoir
# s'installer. Ces paquets ne pourront pas
# être configurés par le système durant
# ce premier démarrage
extra_debs=openssh-server

# Contenu du fichier /etc/fstab du
# système cible. Prenez garde à bien
# définir les bons pseudo-fichiers
# de /dev ainsi que les points de
# montage correspondants
root_partition=/dev/mmcblk0p2
root_fstype=ext2
root_fsargs=noatime
```

Une fois le script judicieusement modifié, il ne vous reste plus qu'à vous placer sur un système de fichiers disposant de suffisamment d'espace libre (> 150 Mo) et de le lancer. Il n'est pas recommandé de lancer le script directement sur un support amovible pour des raisons évidentes de performances :

```
% sudo ./gen_root3.sh
Password:
Running debootstrap to retrieve the basic
  system for arch armel
I: Retrieving Release
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Found additional base dependencies:
libedit2 libkeyutils1 libkrb53
libvolume-id0 openssh-blacklist
openssh-client
I: Extracting util-linux...
[...]
I: Extracting zlib1g...
Creating dev/console
mknod: `dev/console': Le fichier existe
Creating etc/fstab
Creating etc/hosts
Creating etc/hostname
Creating etc/inittab to run stage2 automatically
Creating etc/timezone
Creating etc/localtime
Creating etc/network/interfaces
Creating etc/inittab.full
Creating stage2 script - run on first boot of target

If you are networking booting then make sure you
have armel-root set as the root filestore
area and boot the target.

If you are booting using an SD card then
clean the SD and copy the contents of armel-root
to the root partition, usually the 1st, on the SD.
  e.g.:
    mke2fs /dev/SOMEDISKn
    mount /dev/SOMEDISKn /MOUNTPOINT/debarm
    cp -rp armel-root/. /MOUNTPOINT/debarm
    sync
    sync
    umount /MOUNTPOINT/debarm

When you boot the target you will probably see a
lot of kernel messages then stage2 will
automatically be started.

This will unpack and install all the
required packages. Once completed it
will reboot into multi-user Debian with networking.

You can login as root with no password.

I STRONGLY advise that your first steps are to
set a root password and then to run:

apt-get update
apt-get install locales
dpkg-reconfigure locales
```



Le cœur de la carte, le module NetusG20 de 4x4 cm avec son processeur Atmel AT91SAMG20 à 400 Mhz.

Le message en fin de procédure résume parfaitement les points importants :

- Il faut copier le contenu du répertoire **armel-root** sur le support amovible.
- Le premier démarrage terminera l'installation des paquets.
- Le premier login se fera en **root** et sans mot de passe.
- Les deux premières choses à faire après login sont de définir un mot de passe et mettre à jour le système avec **apt-get**.

Nous ajouterons ici qu'il n'est pas superflu de jeter un œil aux différents fichiers de configuration comme **/etc/fstab** ou **/etc/network/interface**. De plus, n'oubliez pas de copier les modules noyaux dans le **/lib/modules** depuis le répertoire **FoxModules** des sources du noyau.

Notez que cette procédure est tout aussi valide pour une microSD que pour une clé USB. La seule différence résidera dans la désignation du système de fichiers racine, qui sera alors **/dev/sda1** en lieu et place de **/dev/mmcblk0p2**. Ceci aussi bien pour le **/etc/fstab** que pour la configuration **bootargs** de U-Boot.

Lors du premier démarrage, vous devez voir s'afficher les informations concernant **/stage2**, la seconde phase d'installation de Debian GNU/Linux :

```
INIT: Entering runlevel: 1
Starting /stage2
About to configure and install
  all required packages.
  This may take a while
I: Installing core packages...
INIT: version 2.86 reloading
I: Unpacking required packages...
I: Unpacking libacl1...
I: Unpacking libattr1...
[...]
Cleaning /dev/.udev for udev
Installing full inittab
Creating /etc/apt/sources.list
Creating empty /lib/modules/2.6.32.2/modules.dep

rebooting into Debian
(Ignore any error about not
  being able to set the clock)
```

Comme le précise le message, cela « take vraiment un while » puisque nous sommes sur un système embarqué. Voilà qui change des messages du genre qui s'affichent en général brièvement sur un Core2Duo. Comme vous pouvez le voir, le second stage d'installation se termine par un redémarrage. Là, il faudra être vigilant si vous avez démarré via U-Boot. Par exemple, avec un noyau en TFTP et un rootfs USB. En effet, le processus d'installation Debian n'a que faire des paramètres ponctuels utilisés et le redémarrage se fera ainsi avec les paramètres par défaut.

Une fois rebooté, la carte G20 dispose d'un système tout neuf, prêt à servir.

CONCLUSION ET PERSPECTIVES

L'espace disponible pour un simple article de présentation est malheureusement trop réduit pour couvrir les très nombreuses fonctionnalités de cette carte. Nous reviendrons sans doute sur le sujet avec une utilisation concrète de la plate-forme dans le cadre d'une projet (sans doute) domotique.

Il est vrai que la disponibilité des ressources de calcul permettent un certain nombre de largesses. L'utilisation d'une distribution Debian presque classique, avec son florilège d'outils, fait souvent oublier qu'on travaille sur un système qui tient sur une platine de moins de 7 cm de côté. Cependant, selon l'usage auquel est destinée la carte G20, on préférera « rentrer dans le rang » en remplaçant tout bonnement Debian par OpenWrt (version dite Crux pour la carte G20). Comme dit en début d'article, le lancement d'un **fsck** est quelque chose de très troublant, tout comme la taille du rootfs et divers autres points. La carte G20 n'est pas parfaite, loin de là, le problème de l'utilisation conjointe de la flash SPI et de la carte microSD est assez révélateur. ■