GNU LINUX MAGAZINE / FRANCE

France Métro : 6,20€ - DOM 6,75€ - TOM 950 XPF - BEL : 6,80€ - LUX : 6,80€ - PORT. CONT. : 6,80€ - CH : 12,70CHF - CAN : 11,60$ - MAR : 70DH

L 19275 - 94 - F: 6,20 €

▶MAI ▶2007 ▶NUMÉRO 94

PABX Vidéo avec Asterisk

p. 48

Découvrez les fonctionnalités vidéo d'Asterisk avec une installation complète mettant en œuvre des clients PC, des vidéophones SIP, une caméra IP et des téléphones mobiles 3G via une passerelle RNIS/UMTS.

NOUVEAUTÉS DU NOYAU 2.6.21 p. 06

▶ Toutes les nouveautés passées au crible : virtualisation, clockevents & dyntick, support du matériel, etc.

SHELL ET SAUVEGARDES p. 28

▶ Révélez la puissance du shell au travers d'un cas pratique de solution de sauvegarde et d'analyse de logs.

EXTENSION POUR MYSQL p. 92

▶ Découvrez les possibilités d'extension de MySQL en écrivant votre fonction de reconnaissance phonétique en C.

LOGIN ET TERMINAUX p. 20

▶ Prenez le contrôle du système de login et construisez un environnement de gestion des terminaux virtuels à votre image.

GESTION DE VERSIONS p. 60

▶ Essayez SVK, une alternative et un complément aux systèmes classiques comme CVS, Subversion, Bazaar...

MAINTENANCE EFFICACE p. 44

▶ Facilitez-vous la réinstallation de postes clients et la restauration d'images système en déployant un serveur Partimage.

QT/C++ p. 86

▶ Parallélisez simplement vos codes en utilisant la gestion de threads intégrée dans Qt4.

Embarqué MIPS : La Fonera

▶ Hack et développement avec le routeur FON

Prenez le contrôle de La Fonera et partez à la découverte du firmware GNU/Linux. Développez ensuite vos propres codes et portez des applications sur le routeur.

p. 74

Administration et développement sur systèmes UNIX

V.D.S.

- *Un système dédié complet sans aucune limitation. (Distribution Debian accès root intégral - panel de gestion ultra simplifié).*
- *Des ressources réservées et confortables (Ram, CPU, I/O, Bande Passante).*
- *Un prix bas pour un rapport qualité/prix unique.*

Bande passante incluse de 2 Mbps à 6 Mbps en trafic illimité - Espace disque de 4 à 40 Go - mémoire vive de 64 Mo à 512 Mo

à partir de

9 €

HT/mois

www.sivit.fr

Sivit

Edito

04 ▶ DEBIAN CORNER

04> Debian GNU/Linux 4.0

06 ▶ KERNEL CORNER

06> Nouveautés du Noyau Linux 2.6.21

12 ▶ PEOPLE

12> European Perl Hackathon 2007 – Arnhem

13> Ann Barcomb,au premier hackathon Perl européen

16 ▶ UNIX/USER

16> EDIGÉO : pratiquer l'échange d'information géographique

20> Personnalisation (un peu brutale) du système de login et des terminaux virtuels

28 ▶ SYSADMIN

28> Sauvegarde et monitoring : le shell et votre meilleur atout

44> Installation, configuration et déploiement d'un serveur de gestion d'images : Partimage

48> Voix et image sur IP : Asterisk et la vidéo

60> Gestion de versions avec SVK

72 ▶ HACKS/CODES

72> Perles de Mongueurs (32)

74> Jugamos a la Fonera

81 ▶ DÉVELOPPEMENT

81> Tests unitaires en Smalltalk

86> Qt4 : programmation parallèle

92> Écriture d'une fonction de reconnaissance phonétique pour MySQL

Bienvenue dans ce numéro de mai,

« Microsoft is dead ! » Non, ce n'est pas une nouvelle prophétie d'un rédacteur en chef en manque de sommeil. Il s'agit du titre d'un billet d'humeur de Paul Graham, dino du Lisp et cultivateur de jeunes pousses Web 2.0. Celui-ci détaille la mort déjà effective de la société de Redmond et surtout les causes du décès : Google, AJAX, le haut-débit et Apple. Eh non, le Logiciel libre n'est pas en cause.

Microsoft a raté le virage du Web 2.0 de la diffusion de contenu personnel (façon blog) et la naissance des premières applications Web. Pourquoi ? Sans doute, l'effet mammouth. L'inertie technologique et la position de quasi-monopole. Il est d'ailleurs amusant de rappeler une citation attribuée au fondateur de Microsoft : « Le succès est un mauvais professeur. Il pousse les gens intelligents à croire qu'ils sont infaillibles ».

Je ne vais pas ici passer en revue l'argumentaire développé par Paul Graham (**http://www.paulgraham.com/microsoft.html**). Précisons simplement que tout cela semble logique et parfaitement valide. Cependant, Paul conclut en précisant que Microsoft n'a pas réellement l'intention ou la motivation nécessaire pour reprendre la main. Là, je n'en suis pas certain.

Effectivement, il faudrait que la firme, après Vista (flop annoncé ?), fasse un peu plus que simplement trottiner derrière les nouveaux chefs de file. Mais rien n'est impossible. N'oublions pas IE ! Critiqué de toutes parts et en retard technologique durant des années, le navigateur Windows est toujours présent, toujours utilisé et encore majoritaire dans de nombreux pays. N'importe quel utilisateur d'AWStats (ou autre) peut s'en rendre compte.

Paul Graham compare également Microsoft à IBM, mettant en commun la « non-dangerosité » des deux entités. IBM, pas dangereux ? Vraiment ? J'ai comme un doute. En vérité, Microsoft vieillit et n'arrive pas à injecter vivacité et culture dans ses produits, chose qu'arrivent parfaitement à faire Apple depuis sa « résurrection » ou Google depuis sa création. Vieux ne veut pas dire mort pour autant.

Et GNU/Linux ? Sous le coup des années ou toujours aussi *hype* ? À chacun de juger selon son utilisation, sa distribution et son implication personnelle. Pour moi, GNU/Linux est « dans le coup » et le restera tant qu'il me comblera.

Sur ce, je vous donne rendez-vous au 2 juin pour le prochain numéro.

Denis Bodor

GNU Linux Magazine
est édité par Diamond Editions
B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail :
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 rédation:
Véronique Wilhelm
Conception graphique :
Fabrice Krachenfels
Responsable publicité :
Tél. : 03 88 58 02 08
Service abonnement :
Tél. : 03 88 58 02 08
Relecture :
Dominique Grosse

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 61 72 76 24

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution /N° ISSN : 1291-78 34
Commission Paritaire : 09 08 K78 976
Périodicité : Mensuel
Prix de vente : 6,20 €

WWW.GNULINUXMAG.COM

Denis Bodor

Debian GNU/Linux 4.0

Elle est là ! Juste ici ! Là ! Sous nos yeux émerveillés et tout humides sous le coup d'une émotion intense ! Debian GNU/Linux 4.0, nom de code Etch, est la nouvelle distribution stable du projet Debian.

Après officiellement 21 mois de développement, voici donc venir la nouvelle distribution stable. Les changements sont donc très importants, tant au niveau des versions que des fonctionnalités proposées. On note par exemple que l'installeur (maintenant graphique) prend en charge les systèmes de fichiers chiffrés. Toujours côté sécurité, chiffrement et signature, le système APT de gestion des paquets intègre maintenant la prise en compte d'une signature des listes de paquets et des paquets (architecture SecureApt). Autre changement majeur, UTF-8 est maintenant utilisé par défaut pour les nouvelles installations.

Bien entendu, qui dit « améliorations majeures » dit « processus de mise à jour critique ». Je ne parle, bien entendu, pas du système de gestion des paquets, mais des modifications de configuration apportées par les utilisateurs et administrateurs à leur installation avec la précédente Stable (Sarge). Non seulement la mise à jour via `apt-get dist-upgrade` sera un processus long, mais il demandera une expertise humaine qualifiée.

Les changements dans la prise en charge du matériel, par exemple, découlant du passage du noyau Linux 2.4.x au 2.6.x imposent une attention toute particulière. `devfs` n'existe plus et est remplacé par `udev`. Les anciens scripts `hotplug` sont remplacés par le démon du même nom. Ou encore, `pcmcia-cs` laisse la place à `pcmciautils`.

Même si beaucoup d'utilisateurs étaient déjà « passés à Testing », les serveurs et autres machines en production utilisent en grande majorité la distribution stable. Là, les problèmes concernent davantage les paquets d'applications serveur. Un grand nombre de mises à jour majeures ou de nouvelles intégrations sont présentes :

- Apache 2.2
- MySQL 5.0.32
- PHP 5.2.0
- Postfix 2.3.6
- PostgreSQL 8.1
- Asterisk 1.2

Mieux vaut dont expérimenter avant mise à jour. L'installation d'une distribution Sarge sur une machine de test, puis sa configuration à l'identique offre une bonne solution. S'en suivra alors la mise à jour via APT et la reconfiguration avec prise de notes. Des tests supplémentaires après mise à jour confirmeront la bonne marche des choses. Reste ensuite à documenter la procédure et les astuces mises en œuvre et à procéder à la mise à jour du ou des serveurs en production (un lundi de préférence).

Parmi les paquets importants retirés de la distribution, on trouve `webmin`, `eGroupWare` et `libc5`. Côté « disparition » toujours, l'architecture Intel 386 n'est plus supportée. Le plus petit processeur de la famille Intel est donc le 486 et compatible. On notera que ceci règle officiellement un certain nombre de problèmes, puisque certains paquets binaires ne fonctionnaient déjà plus sans l'émulation 486 du noyau (qui n'existe d'ailleurs plus pour des raisons de sécurité). C'était le cas, par exemple, de `busybox`. Les architectures supportées sont maintenant au nombre de 11 : Sparc, Alpha, PowerPC, Intel IA-32 (i386) et IA-64 (ia64), HPPA-RISC, MIPS, ARM, S/390 (s390), AMD64 et Intel EM64T. Ces deux dernières sont des nouveautés introduites avec la version 4.0.

Enfin, pour en finir avec la partie technique de Debian GNU/Linux 4.0, précisons que la distribution intègre X.org 7.1 supportant le *rendering* accéléré indirect avec AIGLX et les autres fonctionnalités nécessaires au fonctionnement de Compiz qui est également inclus sous la forme de paquets `compiz-*`. Debian GNU/Linux 4.0 marque donc peut-être un tournant vers une distribution plus « sexy ».

Ceci nous permet d'enchaîner sur un autre évènement important : le changement de leader Debian (DPL). C'est à présent Samuel Hocevar (dit « Sam ») qui est en charge de ce poste et qui devrait, on l'espère, faire bouger les choses. On notera que ses réflexions vont dans le bon sens : « Rendre Debian à nouveau sexy », « Même le site web FreeBSD est plus sexy que le nôtre », etc.

Il est vrai que la page principale du projet Debian a une touche très 90's et que le site de la FSF, longtemps resté très sobre (simpliste ?), est un bel exemple de « redesign » intelligent et moderne. Nous verrons bien comment cela se passera dans un proche avenir. Nul doute qu'un certain nombre de développeurs Debian vont « souffler » un peu avant de repartir de plus bel. Et puis, un Français à la tête de Debian, ça ne peut être qu'une bonne chose... ou pas :)

Denis Bodor,

db@ed-diamond.com
lefinnois@lefinnois.net

Éric Lacombe, Matthieu Barthélemy

Nouveautés du Noyau Linux 2.6.21

C'est déjà la dixième édition de notre rubrique sur le noyau. Nous la fêtons en vous proposant de découvrir les nouveautés qu'apporte la version 2.6.21 du noyau Linux. Parmi celles-ci, la virtualisation y prend une bonne place, mais d'autres bonnes nouvelles nous attendent. Ainsi, une première étape vers une architecture de noyau tickless a été franchie avec l'intégration du patch dyntick. L'écriture de pilotes est maintenant facilitée par l'intégration du patch devres, lequel propose une infrastructure de gestion des ressources et des erreurs. Aussi, bien d'autres fonctionnalités et mises à jour prennent place dans cette nouvelle mouture et la lecture de ces lignes vous en apprendra davantage.

[ACTUALITÉ] LE NOYAU 2.6.21

Éric Lacombe – tuxiko@free.fr, eric.lacombe@security-labs.org

La virtualisation

La virtualisation est un thème très actif dans le noyau Linux. Plusieurs solutions sont développées. Nous passons de la virtualisation à base d'isolateurs (*containers*) fondée sur un seul noyau avec différents espaces de nommage, à la virtualisation complète où les systèmes invités ont l'impression de s'exécuter directement sur le matériel. Entre ces deux mondes, nous trouvons la paravirtualisation. Dans cette approche, le système invité coopère avec l'hyperviseur afin que ce dernier puisse anticiper ses actions et ainsi améliorer les performances d'exécution.

Les containers

Concernant l'approche par containers, c'est au tour du système de fichiers Sysfs d'être modifié afin de supporter le principe des *shadow directories* : chaque répertoire peut être décliné en différentes versions tout en conservant le même nom. Ainsi, chaque espace de nommage (créé pour le besoin des applications profitant des containers) dispose de ses propres ressources lesquelles conservent le même nom. Ainsi, quelle que soit l'application qui est cloisonnée dans son espace de nommage, sa vision du système est toujours cohérente (l'interface réseau, par exemple, conservera le même nom au travers de Sysfs).

VMI

Le noyau 2.6.20 (cf. KC 91) intègre une interface standard, `paravirt_ops`, définissant l'ensemble des opérations (bas niveau comme l'activation des interruptions, etc.) que doit fournir (via leur implémentation) un hyperviseur supportant la paravirtualisation. C'est ensuite le noyau invité qui utilise ces opérations afin de communiquer avec l'hyperviseur sur le système hôte. Ainsi, n'importe qu'elle version du noyau Linux à partir de la version 2.6.20 peut tourner sur n'importe quel hyperviseur compatible avec la structure `paravirt_ops`. Il reste cependant nécessaire de recompiler le noyau invité afin d'intégrer les opérations définies par l'hyperviseur à chaque changement de celui-ci.

Au début de l'année 2006, Zachary Amsden, de la société VMware, a proposé le mécanisme VMI (*Virtual Machine Interface*) offrant un rôle similaire à `paravirt_ops` (défini que plus tard) avec des caractéristiques intéressantes. Le concept de base est qu'un hyperviseur compatible VMI fournit à un OS invité (également compatible VMI) une ROM VMI (c.-à-d. un fichier binaire) contenant la définition de toutes les opérations bas niveau pour que l'OS invité puisse interagir avec l'hyperviseur. Ainsi, l'OS invité n'a pas besoin d'être recompilé lors d'un changement d'hyperviseur.

VMI a été créé afin de résoudre différents problèmes. Parmi ceux-là, nous en citons trois ayant une importance notable. En premier lieu, un noyau compatible VMI peut s'exécuter directement sans modification sur le matériel et cela sans perte de performances ; ou bien sur tous les hyperviseurs compatibles VMI. Ensuite, l'interface fournit un moyen pour demander à l'hyperviseur quelles sont les capacités matérielles disponibles, d'aide à la virtualisation. Ainsi, un OS VMI-fié ne nécessite pas de modifications pour s'exécuter, quelle que soit la technologie matérielle de virtualisation déployée (présente et future). Finalement, un OS VMI-fié profite également des pilotes de périphériques dont disposent l'hyperviseur sur le système hôte.

Après un passage par la branche `-mm`, VMI a été intégré dans cette version 2.6.21 de Linux. Son implémentation a cependant été modifiée. VMI est dorénavant construit au-dessus de `paravirt_ops` et propose un panel de fonctionnalités plus important. Aussi, alors que la version originale n'était réalisée que pour l'architecture x86, l'intégration aux architectures de type x86_64 est en cours.

[ACTUALITÉ] LE NOYAU 2.6.21

Éric Lacombe – tuxiko@free.fr, eric.lacombe@security-labs.org

KVM

L'activité autour de KVM (*Kernel-based Virtual Machine driver*) est toujours très soutenue. Des fonctionnalités très attendues ont été ajoutées dans cette version de Linux.

Nous avions parlé dans le KC91 de l'ajout par Ingo Molnar du support de la paravirtualisation dans KVM. Cette fonctionnalité très intéressante en termes de performance a été incluse dans la *mainline*.

La migration à chaud de machines virtuelles, d'un hôte à un autre, est désormais disponible pour KVM. Cette action est effectuée en plusieurs étapes. Tout d'abord, l'image de la mémoire physique de la machine virtuelle (c.-à-d. la mémoire dont dispose le système invité) est transférée vers le système hôte destinataire (via une connexion directe en TCP ou en ssh). Cette opération est effectuée page après page. Pour chaque transfert réussi, la page correspondante sur la machine d'origine est alors positionnée en lecture seule. Ainsi, si la page est accédée en écriture par l'OS invité, une faute de page survient. Elle engendre alors le marquage de cette page, laquelle devient une page « sale » (ce mécanisme est appelé *dirty page logging*). Cela signifie qu'il est nécessaire de la retransférer sur la machine de destination. Lorsque le nombre de pages sales est inférieur à un certain seuil, la machine virtuelle est stoppée et les pages restantes sont copiées. Finalement, l'état de la machine virtuelle (c.-à-d. la valeur de ses registres) est transféré. L'hôte destinataire reprend alors l'exécution du système invité.

Une autre fonctionnalité notable est le support du *suspend/resume* pour les systèmes hôtes exécutant des machines virtuelles au travers de KVM. Ainsi, il est maintenant possible d'ajouter ou de retirer des processeurs, à chaud, (*cpu hotplug*) sur ces systèmes afin de répondre au besoin de puissance de calcul.

Annoncé pour cette version du noyau, la stabilisation de l'interface utilisateur de KVM est au final prévue pour la version 2.6.22. La 15ème révision de KVM qui est intégrée à Linux 2.6.21 a déjà été modifiée dans la 18ème révision de KVM.

La gestion du temps

Clockevents

La prise en charge des périphériques de gestion du temps (PIC, HPET, etc.) est effectuée depuis toujours de façon spécifique suivant le type d'architecture matérielle. Le patch `clockevents` implémente une API unifiée pour tous les types d'horloges matérielles. Les spécificités de chaque horloge (résolution, *one-shot* ou périodique, etc.) sont exposées au noyau, au travers de cette API, afin qu'il puisse créer des *timers* et les armer sans avoir à se soucier des particularités matérielles (Un driver minimal, bas niveau, dépendant de l'architecture, reste toutefois nécessaire). Le noyau profite ainsi pleinement des capacités matérielles lorsqu'il arme des timers.

Dyntick

Thomas Gleixner et Ingo Molnar ont mis en place dans la branche `-rt` (temps réel) du noyau Linux l'infrastructure dyntick (cf. KC91) pour se passer partiellement de l'utilisation de l'interruption du timer, le *tick* (architecture *tickless*). Ce patch a été intégré dans la version 2.6.21 de la mainline. Dans les versions précédentes, le noyau programme l'interruption du timer pour qu'elle se déclenche de façon périodique (300 fois par seconde par défaut depuis le 2.6.19). Cette interruption cadence notamment l'exécution des différents processus sur le système. Cependant lorsque aucune activité n'est à l'œuvre (c.-à-d. le processeur est en état oisif), l'interruption continue d'être levée au même rythme. Elle réveille ainsi le système constamment. À chaque interruption, le système effectue différentes opérations. Entre autres, il met à jour la variable `jiffies` (gardant le temps écoulé depuis le démarrage du système), met à jour le `timeslice` du processus en cours d'exécution, vérifie si le flag `need_resched` est positionné (auquel cas, l'ordonnanceur est appelé), etc.

Le déclenchement périodique du tick est toujours effectué pendant les périodes d'activité du système. Mais lorsque celui-ci se trouve dans un état oisif, ce déclenchement est désactivé. Le timer est alors programmé pour lever une interruption à la date du prochain évènement temporel devant être traité (celui-ci pouvant être programmé à une résolution très fine via l'utilisation des *clockevents*). Ainsi le processeur n'a plus à se réveiller constamment pour exécuter le gestionnaire d'interruption du timer. Cette façon de procéder aboutit à une économie d'énergie significative ainsi qu'au refroidissement du processeur.

Finalement, au travers du système de fichiers `/proc`, les entrées `timer_list` et `timer_stats` listent, respectivement, l'ensemble des timers en suspens et différentes statistiques sur leurs utilisations.

➤ [ACTUALITÉ] LE NOYAU 2.6.21

Matthieu Barthélemy – bonsouere@gmail.com

La gestion des ressources et des erreurs

Vous vous rappellerez peut-être le sujet du *Kernel Corner* 92, traitant de la stabilité des pilotes. Un pas prometteur vers l'amélioration de la qualité du code de ces derniers va pouvoir être franchi avec l'apparition d'un sous-ensemble de fonctions dédiées à la gestion de l'allocation de ressources. Partant du constat que, dans de nombreux cas, la libération de diverses ressources requises pour le fonctionnement d'un module n'est pas ou est mal faite par les développeurs, Tejun Heo propose un ensemble de fonctions qui remplacent celles traditionnellement appelées pour obtenir les ressources en question, et se chargent ensuite de les libérer. Cette approche veut pallier les problèmes survenant lorsque l'initialisation d'un pilote échoue ou qu'un module est déchargé. Les conséquences peuvent aller de l'impossibilité de charger à nouveau le module jusqu'au crash. Concrètement, le développeur doit explicitement appeler ces nouvelles fonctions en lieu et place des fonctions traditionnelles. Pour simplifier le développement, elles peuvent être placées par le développeur dans un groupe. Il suffit pour cela d'appeler `devres_open_group()` pour une structure `device`, et toutes les *managed functions* utilisées ensuite feront automatiquement partie de ce groupe. Il suffit au développeur d'appeler ensuite `devres_release_group()` pour désallouer l'ensemble des ressources contenues dans ce groupe, par exemple si le chargement du module provoque une erreur. Dans tous les cas, lors d'un déchargement du module (si vous demandez un `rmmod` par exemple), tout ce qui a été alloué via l'appel aux managed functions sera automatiquement libéré.

L'infrastructure de Tejun Heo sait détecter les erreurs survenant lors de l'appel de ses fonctions spéciales, et permet ainsi de programmer de manière similaire la gestion d'une transaction : si certains appels d'une série retournent une erreur, un simple appel à `devres_release_group()` libèrera l'ensemble des ressources acquises par la série d'appels.

Afin de garder une trace globale de ce qui a été alloué, chacune des fonctions de l'infrastructure devres appelle `devres_alloc(nom_fonction_de_liberation_ressources, taille_réservée, type_allocation)`, qui se charge de garder une trace des ressources demandées dans une liste. Actuellement, les managed functions sont utilisables pour réserver des IO *regions*, des interruptions (IRQ), des zones DMA, PCI et ioMap. La `libAta` est d'ores et déjà convertie à ce sous-système. Gageons que la simplicité d'adaptation du code existant à cette infrastructure lui garantira une adoption rapide dans les prochaines versions du noyau. On peut également imaginer que cette technique d'assistance, ayant maintenant fait son entrée au sein de Linux, sera étendue et complétée, au bénéfice de la qualité et de la stabilité.

Les spécificités des architectures

Dans le code spécifique à chaque architecture processeur, quelques nouveautés méritent attention. Signalons le support de l'option `noexec` sur s390 : cette option permet d'activer le bit de contrôle `"NX"` pour les régions mémoire contenant des données. C'est une extension disponible matériellement sur certains processeurs (SPARC, AMD64, etc.), qui s'appuie sur la valeur du dernier bit des entrées des tables de pages mémoire pour autoriser ou non la page correspondante à être exécutée par le processeur. Cependant les s390 n'en disposent pas. C'est donc par émulation logicielle que l'implémentation a été réalisée. La gestion de cela sous Linux/s390 se fonde sur un mode d'adressage différent : on choisit les tables de pages qui sont utilisées pour la translation des adresses d'instructions et celles qui sont utilisées pour la translation des adresses de données. Ainsi, sont formés des couples de tables de pages, contenant une *shadow table* utilisée pour le code exécutable, et une table de page classique pour les données. Ce mode d'adressage n'est disponible que pour l'espace utilisateur (*secondary-space* mode dans la terminologie s390). Ainsi, la protection n'est effective que pour l'espace utilisateur. Pour résumer, le marquage de données en `noexec` permet d'éviter des attaques, entre autres, par débordement de tampon (*buffer overflow*).

Le rechargement de noyau « à chaud », c'est-à-dire sans passer par la phase d'initialisation de la machine (BIOS, *boot loader*, etc.) via `kexec` est disponible sur architectures ARM.

Les utilisateurs d'architectures PPC32 peuvent maintenant profiter de Kprobes, infrastructure de débogage et de collecte d'informations sur le noyau en cours d'exécution.

Les systèmes de fichiers

L'ajout de volumes à chaud est maintenant géré en RAID6 (qui est un mode à 4 disques minimum, double parité autorisant la perte de deux disques). Les partitions UFS2 (Solaris, FreeBSD) peuvent, quant à elles, être montées en écriture. Cependant ceci est marqué expérimental donc... faites un *backup* avant de tester. Les systèmes de fichiers de Minix V3 peuvent dorénavant être utilisés sous Linux, en lecture et écriture. Quant à JFFS1 (système de fichiers optimisé pour stockages de type flash), son retrait planifié du noyau est rendu effectif ; considéré obsolète, il est abandonné au profit de JFFS2. Le récent eCryptfs peut, quant à lui, fonctionner avec des chiffres à clef publique. Enfin, un *tuning* notable est réalisé sur GFS2 pour réduire son utilisation mémoire et améliorer ses performances (écritures de gros blocs de données et appels à `readdir()` notamment).

➤ [ACTUALITÉ] LE NOYAU 2.6.21

Matthieu Barthélemy – bonsouere@gmail.com

Le réseau

Peu de choses à se mettre sous la dent cette fois-ci. Notons tout de même que le serveur NFS (knfsd) peut maintenant fonctionner en IPv6 et que Netfilter/IPtables gèrent maintenant les ports sources aléatoires NATés.

L'embarqué

Alsa propose une nouvelle organisation appelée ASoC (ALSA *System On Chip*), développée à l'origine pour le projet OpenZaurus. Le code se divise en trois groupes de composants indépendants : les codecs (code générique indépendant de la plate-forme), les pilotes d'interface audio (AC97, PCM, etc.) et, enfin, une partie spécifique à chaque architecture matérielle. Ceci permet avec peu d'effort de faire fonctionner Alsa sur une multitude de systèmes de type embarqué. ALSA intègre également un gestionnaire d'énergie qui lui est propre, permettant de désactiver et réveiller à la volée chaque module utilisé ou non ou encore de modifier un taux d'échantillonnage pour réclamer moins de ressources.

Les pilotes de périphériques

La `libAta`, développée à l'origine pour piloter les périphériques SATA, a été étendue depuis Linux 2.6.19 aux périphériques PATA (IDE), (cf. KC 91). Depuis, cette branche est en constante amélioration afin de gérer toujours plus de périphériques IDE, le but étant, à terme, de remplacer le code IDE actuel du noyau. La série 2.6.21 n'est pas en reste, avec :

- le support des contrôleurs Intel PIIX3 (IDE) ;
- la gestion des contrôleurs it8213 (IDE) ;
- la gestion des contrôleurs Initio 162x (SATA).

Cherchant à favoriser une rapide adoption de la nouvelle infrastructure PATA, Fedora Core 7 sera la première distribution à l'utiliser par défaut.

Nous vous signalions, pour la sortie du 2.6.20, l'apparition d'une couche générique dédiée aux périphériques d'interaction utilisateur (claviers, souris...). Elle a désormais le privilège d'être adoptée par les périphériques Bluetooth.

Du côté de l'USB, si vous faites partie de ceux rencontrant des problèmes de communication avec des périphériques automatiquement paramétrés en mode *High Speed* (USB 2.0, 480 Mbits/s), vous avez désormais la possibilité de forcer le mode *Full Speed* (mode haute vitesse de la norme USB 1.1, 12Mbits/s) via Sysfs. Il n'empêche que ce n'est pas forcément intuitif, *user-friendly*, *people ready* ou tout ce que vous voudrez. Explication : dans la norme USB, chaque contrôleur USB 2.0 est couplé à un second contrôleur appelé « *companion* », qui se charge de gérer les périphériques dits « full speed ». Si le contrôleur USB a détecté un tel périphérique, il doit ouvrir un port sur son companion et lui transférer tous les échanges ayant lieu avec le périphérique. Par contre, si le périphérique s'annonce comme étant compatible high speed, il n'y avait jusqu'à présent aucun moyen sous Linux de le faire redescendre en mode full, autrement dit de l'attribuer à un contrôleur companion. À partir de Linux 2.6.21 chaque hôte USB dispose d'une entrée appelée « companion » dans `/sys/class/usb_host/usb_hostXX/device/companion`. Rapide démonstration pratique :

Lorsque nous branchons notre périphérique, une entrée dans `dmesg` indique son hôte :

```
usb 3-3: new high speed USB device using ehci_hcd and address 10
```

C'est donc l'hôte 3 qui accueille le matériel. Il nous faut attribuer un port companion disponible au périphérique. On choisit un numéro affiché par :

```
# cat /sys/class/usb_host/usb_host3/companion
1
2
```

et enfin on indique à l'hôte de l'utiliser :

```
# echo -n 2 > /sys/class/usb_host/usb_host3/companion
```

La gestion du périphérique est alors assurée par le sous-système UHCI.

Le domaine de l'ACPI voit l'apparition d'une classe de périphériques réservée aux rétroéclairages (affichages de périphériques mobiles, PC portables, etc.) dans Sysfs : `/sys/class/backlight`. Linux profite également d'un nouveau gestionnaire de tables ACPI entièrement réécrit, moins gourmand en mémoire.

Terminons par un rapide résumé des nouveautés dans la gestion de matériel :

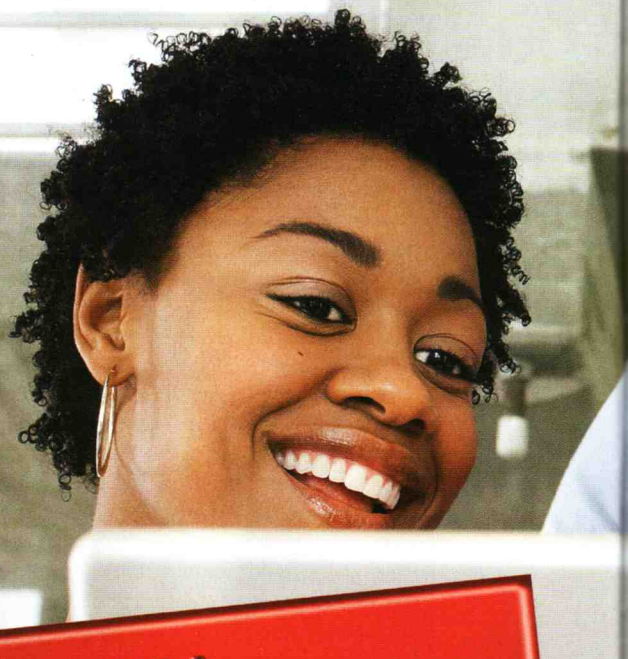
- ajout du support des contrôleurs IDE Toshiba TC86C001, pour l'« ancienne » couche IDE uniquement (option noyau `BLK_DEV_TC86C001`) ;
- ajout de la gestion des cartes Ethernet PCI Silan SC92031 (option SC92031) ;
- apparition d'un petit module permettant de charger les PDA BlackBerry par USB (option `USB_BERRY_CHARGE`) ;
- support des puces Qlogic 4032 dans le pilote SCSI qla3xx ;
- de nombreuses améliorations dans la gestion de cartes tuner et d'acquisition vidéo.

Divers

L'espace utilisateur peut maintenant marquer le noyau comme *tainted* (corrompu) via `/proc/sys/kernel/tainted`, dans le cas où une application effectuerait des opérations pouvant mettre en péril le fonctionnement normal du noyau. Un nouveau marqueur `'U'` a été créé pour l'occasion.

La taille totale des options passées au noyau lors de son démarrage devient variable et non plus limitée à une valeur fixe. Cela fait tomber une limitation parfois gênante lorsque l'on a des options à passer pour un grand nombre de modules.

1&1, tout pour votre site Web



Votre succès en 3 étapes :

Réalisez facilement votre site Web,
grâce à d'innombrables outils de création

Passionnez et fidélisez vos visiteurs,
grâce aux flux d'infos en temps réel

Faites parler de vous,
grâce aux solutions interactives 1&1

Avec 1&1, votre avenir sur le Web est entre de bonnes mains et votre site, vraiment prêt à suivre toutes vos envies.

Nouveau :

L'ACTUALITÉ EN TEMPS RÉEL

avec 1&1 Contenu Dynamique

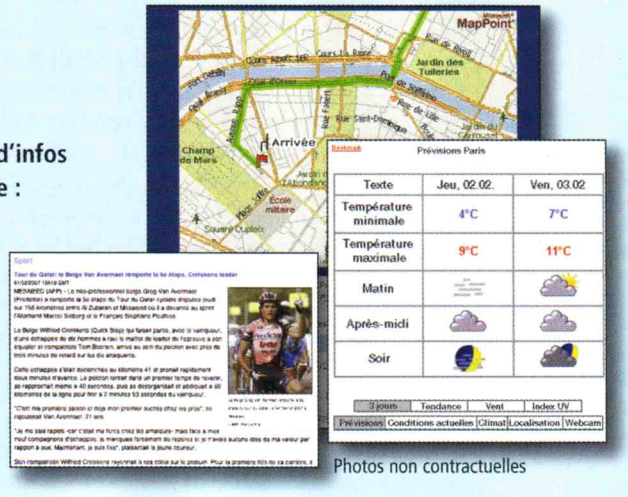
Transformez votre site en mine d'infos actualisées en temps réel !

1&1 vous propose une solution pour intégrer des flux d'informations en temps réel, jusqu'alors réservée aux plus grands sites…

Grâce aux offres de contenu dynamique présentées ici, vous avez la possibilité d'intégrer des informations réactualisées automatiquement au fil de la journée sur votre site. En effet, il vous suffit de choisir le ou les contenus qui vous intéressent et une fois mis en place, vous n'avez plus rien à faire. 1&1 s'occupe de la mise à jour ! Et ce n'est pas tout, cette solution vraiment innovante est totalement gratuite !

Découvrez ici quelques-uns des flux d'infos que vous pouvez intégrer à votre site :

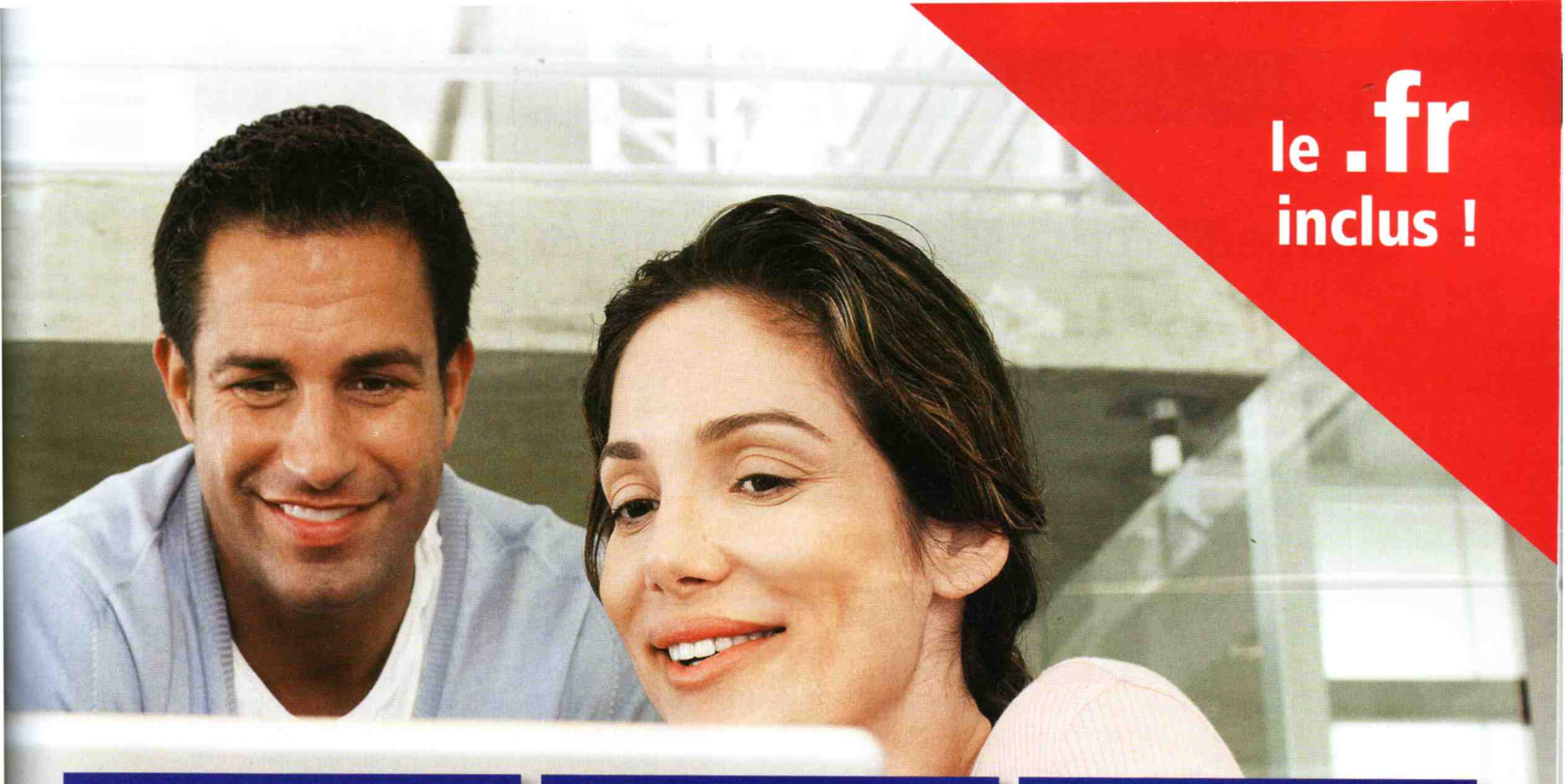
- La Une de l'actualité
- France
- Économie / Finance / Bourse
- High Tech
- Sports / Football
- Culture / Art de vivre
- Nouvelles « People »
- Itinéraires…



Photos non contractuelles

MEMBRE DE united internet

N° INDIGO **0 825 080 020** (0,15 € TTC la minute)



le .fr inclus !

PACK PERSO INITIAL

0,99 € HT/mois
1,18 € TTC/mois

1 domaine en .fr, .com, .net, .org, .info

Pour les particuliers exigeants qui souhaitent se lancer dans la création d'un site Web sans aucune connaissance en programmation.

- 1500 Mo d'espace
- 25 Go de trafic
- 10 comptes email
- 1&1 Blog
- 1&1 Contenu Dynamique

Et bien plus encore...

PACK PERSO CONFORT

4,99 € HT/mois
5,97 € TTC/mois

2 domaines en .fr, .com, .net, .org, .info

Pour les associations et petits commerçants qui désirent disposer d'une vitrine attrayante sur le Net et bénéficier de nombreuses solutions clé en main.

- 6000 Mo d'espace
- 750 Go de trafic
- 200 comptes email
- 5 bases de données MySQL
- 1&1 Contenu Dynamique

Et bien plus encore...

PACK PRO STANDARD

9,99 € HT/mois
11,95 € TTC/mois

3 domaines en .fr, .com, .net, .org, .info

Pour les petites et moyennes entreprises qui ont besoin d'un site Web dynamique et interactif pour renforcer leur activité.

- 10 000 Mo d'espace
- 1000 Go de trafic
- 1200 comptes email
- 20 bases de données MySQL
- 1&1 Contenu Dynamique

Et bien plus encore...

Windows News CHOIX DE LA RÉDACTION

« Très pro et très accessible »
Windows News, Février 2007, n°153

1&1

www.1and1.fr

Emmanuel Di Pretoro

European Perl Hackathon 2007 – Arnhem

Comme annoncé dans GLMF 92, le premier hackathon Perl en Europe s'est tenu du 2 au 4 mars dernier. Cela se déroulait à Arnhem, aux Pays-Bas. Cet article se propose de vous donner le bilan de cet événement, ainsi qu'un aperçu de l'ambiance qui y régnait.

Début du hackathon

Bien que le *hackathon* se déroule le week-end, il commençait officiellement le vendredi à 16 heures. Plusieurs participants sont donc arrivés le vendredi soir, certains de manière assez classique, à savoir en taxi, et d'autres de manière plus... improvisée, c'est-à-dire en voiture de police.

Le samedi, l'ambiance était plus studieuse à notre arrivée. Il y avait deux salles réservées à l'occasion, et cela s'est grosso modo réparti sur la base de la version de Perl concernée, à savoir Perl 5 et Perl 6.

La salle Perl 5 - Photo Stéphane Payrard

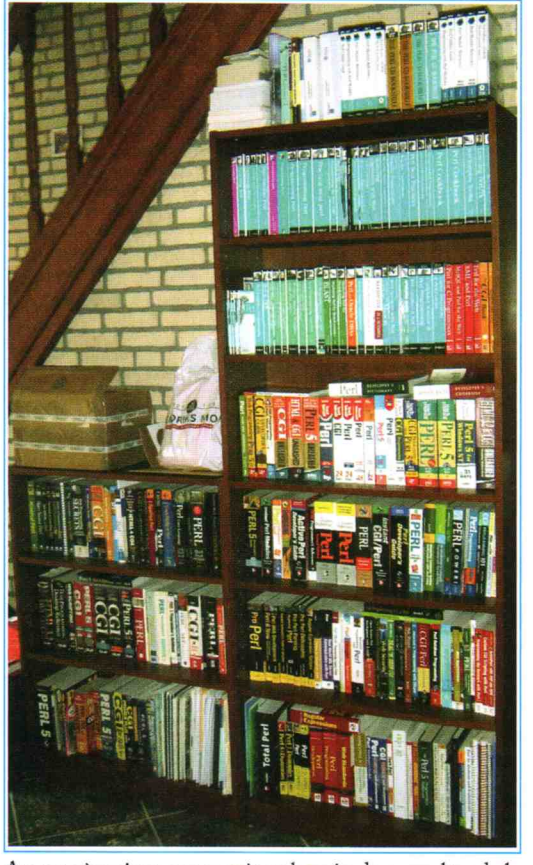
Dans la salle « Perl 5 », les projets faisant l'objet d'un travail étaient :

- Act : que cela soit la documentation du système de *templates*, diverses traductions (en anglais et néerlandais), ou encore quelques tentatives de refactorisation de l'application, cette application de gestion de conférences a fait l'objet de l'attention de plusieurs participants (que cela soit ecocode, BooK, saorge ou encore monsieur_champs) ;
- La documentation des expressions régulières de Perl 5.10 : Abigail, avec la collaboration de Juerd, a établi les bases pour documenter les classes de caractères des expressions régulières Perl.
- Ann Barcomb, l'organisatrice de l'événement, a travaillé sur la documentation du site web de YAPC::Europe Foundation. Elle a aussi, avec l'aide de BooK, jeté les premiers éléments d'une documentation à l'usage des futurs organisateurs de hackathons européens.
- CPAN6 : ce projet, visant à offrir une suite à CPAN, mais pour Perl 6, a fait l'objet de plusieurs discussions entre markov, Juerd, Wytze et d'autres. Une première implémentation a été commencée.
- `XML::Atom::SimpleFeed` : Aristotle a interrogé les participants du hackathon pour améliorer la structure interne du module, et en a commencé l'implémentation.

Pendant ce temps-là, dans la salle « Perl 6 » :

- Parrot : Allison Randal et Jonathan Worthington, ainsi que d'autres participants (dont Liz Mattijsen et Stéphane Payrard) ont travaillé sur la conception de la structure élémentaire des objets et classes dans Parrot. Jonathan a également fixé un bug.
- `Pod::Simple` : Allison a publié la version 3.05 de ce module, une version intégrant un patch, ainsi qu'une correction de bug. Cette version sera incluse dans Perl 5.9.5.

La bibliothèque Perl de Wendy van Dijk
Photo Philippe Bruhat



Au cas où quiconque aurait eu besoin de consulter de la documentation, Wendy avait amené l'intégralité de son impressionnante collection de livres sur Perl. Suite à ce

week-end, elle commence à envisager d'entamer la même collection avec les livres sur Perl en français et en allemand.

Mais si le week-end a été studieux, ce n'est pas pour autant que l'atmosphère n'a pas été agréable. Plusieurs personnes se sont chargées de détendre l'atmosphère, parfois involontairement comme BooK, qui en installant Act sur son portable, et à la suite de plusieurs commandes dans son *shell*, a lancé le dangereux et bien connu des utilisateurs Unix `rm -rf ~`. D'autres participants ont volontairement contribué à l'ambiance du week-end, comme Wendy, qui a profité de l'éclipse totale de lune pour nous montrer ses talents d'imitation du cri du loup.

Conclusion

Plusieurs projets ont ainsi pu avancer durant ce week-end, pas toujours autant que ce que l'on aurait pu le souhaiter, mais de manière satisfaisante malgré tout.

Je ne pense pas être le seul à avoir trouvé l'expérience intéressante. J'espère que d'autres hackathons se tiendront en Europe, et que je pourrai y participer.

Si vous souhaitez plus d'information à propos du hackathon, je vous recommande le site de l'événement (basé sur Act). Et vous trouverez des photos de cette manifestation sur Flickr, sous le tag **perlhack2007nl**.

LIENS

- Le site du Hackathon
  http://conferences.yapceurope.org/hack2007nl
- YAPC::Europe Fondation
  http://yapceurope.org/
- Récit de BooK sur sa mésaventure
  http://use.perl.org/~BooK/journal/32555
- CPAN6 – **http://cpan6.org/**
- Parrot – **http://www.parrotcode.org/**
- Perl 6 – **http://dev.perl.org/perl6/**
- Act – **http://act.mongueurs.net/**
- Pod::Simple
  http://search.cpan.org/dist/Pod-Simple/
- XML::Atom::SimpleFeed
  http://search.cpan.org/dist/XML-Atom-SimpleFeed/
- Photos sur Flickr
  http://www.flickr.com/photos/tags/perlhack2007nl

PEOPLE **INTERVIEW**

Propos recueillis par Philippe Bruhat

Ann Barcomb, au premier hackathon Perl européen

Cette interview a été réalisée au Stayokay Hostel le 4 mars 2007, à Arnhem, Pays-Bas, pendant le premier hackathon Perl européen.
Ann Barcomb (kudra) est l'organisatrice de ce hackathon.
Entretien et transcription réalisés par Philippe Bruhat. Relecture par Estelle Souche.

Q GLMF : Bonjour. Pourrais-tu te présenter un peu à nos lecteurs ?

R Ann : Je m'appelle Ann Barcomb. Je suis probablement plus connue pour ce que j'ai organisé et d'autres méta-activités que pour quelque code que ce soit, bien que j'aie travaillé comme programmeuse pendant plusieurs années. Mon emploi actuel consiste à écrire de la documentation, car j'avais besoin de faire une pause par rapport à la programmation.

Ma première activité dans la communauté Perl a consisté à être l'une des principales organisatrices de la conférence YAPC Europe, à Amsterdam en 2001. C'était la deuxième YAPC Europe. Depuis, je suis devenue membre du conseil de la Fondation YAPC Europe (**http://www.yapceurope.org/**) et depuis un peu plus d'un an, j'écris les résumés hebdomadaires de Perl 6, qui sont envoyés sur *use Perl*, sur mon blog O'Reilly, le blog de Pugs, ainsi que la liste de diffusion `perl6-announce`.

Q GLMF : Qu'est-ce que c'est qu'un hackathon ?

R Ann : Un hackathon est un événement, qui se tient généralement le week-end, pendant lequel un petit groupe de gens se retrouvent et travaillent sur des projets spécifiques. C'est différent d'un *workshop* ou d'une conférence à cause du plus petit nombre de participants et parce qu'il n'y a aucun exposé planifié : les gens viennent pour travailler effectivement sur des projets.

Certaines personnes viennent sans projet particulier en tête et s'attellent à tout projet qui leur semble intéressant. La plupart des gens viennent avec déjà une idée de ce qu'ils veulent faire ; le but est d'avoir du temps en tête à tête avec ses collaborateurs et d'abattre beaucoup plus de travail.

Q GLMF : Est-ce que c'était le premier hackathon dans la communauté Perl ? Il me semble que toute cette mode des hackathons est assez récente.

R Ann : Les hackathons sont certainement un *buzzword* en ce moment ; ils sont devenus de plus en plus en vogue ces derniers temps. Celui-ci n'était pas le premier ; il y en a un autre qui a été organisé de manière un peu improvisée après YAPC Chicago l'an passé. Plusieurs personnes sont restées pour quelques jours après la conférence, une partie de l'équipement a été réquisitionnée dans ce but et ça a été un gros succès. Alors en novembre dernier, les organisateurs de YAPC Chicago ont organisé un hackathon autonome. Ils ont eu environ 50 participants.

Celui-ci est le premier hackathon européen consacré à Perl, pour autant que je sache. Il est lui aussi autonome. Si j'ai bien compris, Vienne a planifié un hackathon en conjonction avec YAPC Europe, et il pourrait y avoir d'autres hackathons autonomes en Europe plus tard dans l'année... Je crois que les Mongueurs de Perl ont parlé d'en faire un ?

Ann Barcomb au travail, pendant le hackathon
Photo Luis Motta Campos

Q GLMF : Euh, oui ! Mais c'est plus une idée qu'un véritable projet pour le moment...

R Ann : Eh bien, ça commence comme une idée et vous vous dites « peut-être qu'on va le faire », et avant que vous ne vous en rendiez compte, vous êtes en train de le faire. Ça demande beaucoup moins d'organisation que pour une conférence.

Q GLMF : Est-ce que c'est limité à la communauté Perl ?

R Ann : D'autres organisations font aussi des hackathons. De fait, il y en a déjà eu plusieurs consacrés à Linux. L'un de nos sponsors, NLNet, a sponsorisé un certain nombre de hackathons l'an passé, et Perl en était juste un parmi beaucoup d'autres. C'est donc certainement une idée qui a pris aussi auprès du reste de la communauté *Open Source*.

Q GLMF : J'avais l'impression que c'était venu plus ou moins d'Audrey [Tang, l'auteur de Pugs], à cause de cette habitude qu'elle a de se faire inviter chez les gens et de hacker pendant quelques jours avant de repartir.

R Ann : Je sais qu'Audrey faisait certainement cela. Après YAPC Chicago l'an dernier, Audrey et quelques autres personnes sont allées chez Jesse Vincent [fondateur de *Best Practical*, qui publie RT et SVK] pour quelques jours, pendant une semaine environ, me semble-t-il. Je crois qu'il y avait cinq ou six personnes qui sont restées là, pour continuer de travailler. Mais je ne sais pas d'où l'idée est venue. C'est plus répandu que juste la communauté Perl. C'est quand même quelque chose d'assez évident à imaginer. Si, par exemple, vous avez un groupe de gens qui travaillent à distance, de temps en temps les gens vont se retrouver, face à face, simplement parce que certaines choses sont plus faciles à faire en personne. Et c'est également une bonne chose de connaître les gens avec qui vous travaillez.

Dans la communauté des projets open source, il y a des groupes de gens très dispersés. C'est donc naturel qu'ils veuillent se retrouver, se rencontrer et essayer de travailler ensemble. Pendant les conférences, il y a toujours des gens qui essayent de faire ça, mais il se passe tellement de choses à côté qu'il est difficile de vraiment se concentrer.

Q GLMF : Parlons un peu de ce hackathon : combien de personnes sont-elles venues ? Sur quels projets ont-elles travaillé ? Est-ce que c'est un succès ?

R Ann : J'aurais tendance à considérer que c'est un succès ou un échec en me demandant si les participants se sont vraiment fait plaisir, et si leur projet valait le coup. La question n'est pas « ont-ils accompli tout ce qu'ils voulaient faire ? », car, je suis sûre que la plupart des gens sont venus avec une très longue liste de choses qu'ils voulaient mener à bien et qu'ils n'ont pas pu tout faire. Mais si les participants ont le sentiment que ça valait le déplacement et le prix qu'ils ont payé, qu'ils en ont tiré quelque chose et qu'ils se sont fait plaisir, alors c'est un succès.

En tant qu'organisatrice, d'autres indicateurs de succès pourraient être : est-ce que suffisamment de gens sont venus ? Évidemment, qui voudrait organiser une fête où personne ne vient ? De ce point de vue, il y a eu assez de participants pour faire un hackathon. J'aurais préféré qu'il y ait un peu plus de monde, car nous avions un nombre de participants bien inférieur à celui du hackathon de Chicago, mais je pense que c'est en partie dû au fait que je n'avais vraiment pas beaucoup de temps : il a été annoncé à peine un mois

avant d'avoir lieu, et c'était plutôt une *tentative* de hackathon européen autonome.

Au début, je ne savais pas de combien de fonds je disposerais, aussi j'ai préféré faire petit plutôt que banqueroute.

Et aussi, en ayant un petit groupe, je pense que nous avons pu discuter beaucoup plus... Il y a certainement des avantages à avoir un petit groupe plutôt qu'un grand groupe.

Voici donc quels sont, selon moi, les critères de réussite d'un hackathon, du point de vue d'un organisateur : est-ce que les hackers se sont fait plaisir ? Est-ce qu'il y a eu assez de projets ? Au sens où ce n'était pas juste toi et deux de tes copains... Et bien sûr, est-ce qu'il y a eu assez de sponsors pour couvrir les frais ? À toutes ces questions, je pense que la réponse est « oui » pour ce hackathon.

Certains des projets sur lesquels on a travaillé ont été : Parrot, CPAN6, le système de gestion de conférences Act, la documentation sur Unicode en Perl 5 et aussi celle pour les expressions régulières en Perl 5. Je pense que c'était les principaux projets. Quelques personnes ont aussi travaillé sur d'autres sujets. Moi, par exemple, j'ai travaillé sur quelques documents que je voulais produire.

Q GLMF : Quels conseils donnerais-tu à quelqu'un qui voudrait organiser un hackathon ?

R **Ann :** En fait, j'espère écrire un document à ce sujet. J'ai découvert que l'organisation d'un hackathon était beaucoup plus facile que l'organisation d'une conférence. J'ai commencé à y réfléchir en décembre, j'ai cherché un site d'accueil en janvier, et, fin janvier, je l'ai vraiment annoncé et décidé que j'allais effectivement le faire et signé les documents avec le site. Il y a donc eu un mois où j'ai vraiment dû me concentrer dessus et y consacrer du temps tous les jours, mais c'est beaucoup moins que pour une conférence. Avant cela, il y a eu un temps de préparation d'environ un mois.

Je suggérerais aux futurs organisateurs d'étendre un peu plus cette durée : prenez-vous y beaucoup plus tôt pour faire savoir clairement qu'il va y avoir un hackathon, afin que les gens puissent avoir le temps d'organiser leur emploi du temps. C'est ce qui a été le plus gros problème ici : le fait que je n'ai pu prévenir qu'un mois à l'avance, bien que j'aie laissé des indices à ce sujet, et que j'en aie parlé à quelques personnes-clés pour savoir si elles viendraient. Quelque chose que vous pourriez faire en plus si vous en avez les moyens serait d'inviter spécifiquement des personnes qui sont importantes pour des projets particuliers. Dans ce cas, nous avons eu la chance qu'Allison Randal et Jonathan Worthington soient tous deux dans la région. Ainsi, j'ai pu leur demander de piloter la partie Parrot du hackathon. S'ils n'avaient pas été présents, il aurait été très difficile d'avoir ce projet sans personne qui soit vraiment au centre du projet. Vous pourriez donc essayer de voir si d'autres personnes pourraient venir, mais seulement si vous savez dès le début que vous allez avoir assez de sponsoring pour couvrir ces frais. Mais d'après l'expérience de ce hackathon et de celui de Chicago, il est clair qu'il y a des fonds disponibles pour ce genre d'événement.

Q GLMF : Que nécessite l'organisation d'un hackathon ?

R **Ann :** Je dirais qu'organiser un hackathon est beaucoup plus simple qu'organiser une conférence. Comme je l'ai mentionné, cela prend à peu près un mois, éventuellement deux, pour organiser un hackathon, alors qu'organiser une conférence prend un an. Les tâches les plus importantes pour moi en tant qu'organisatrice étaient de m'assurer du financement, de trouver un lieu approprié et créer le site web. Quand créer le site web est l'une de vos tâches principales, c'est que l'ensemble du travail est assez réduit.

Aussi, une des choses dont j'ai été très contente avec ce hackathon est que, bien que j'aie dû faire la plupart du travail de préparation, d'autres personnes m'ont beaucoup aidé quand il s'est agit du hackathon lui-même. Plusieurs personnes d'Amsterdam.pm se sont spontanément proposées pour aider. Liz Mattijsen et Wendy van Dijk sont passées quelques semaines auparavant pour tester le réseau qu'elles prévoyaient également d'utiliser pour le Dutch Perl Workshop [qui s'est déroulé deux semaines après le hackathon, au même endroit]. D'autres personnes ont donné un coup de main quand elles le pouvaient, par exemple vers trois heures, le vendredi, Sebastian Stellingwerff est arrivé et a installé le réseau sans fil. Je ne m'y attendais pas ; je pensais que ce seraient Liz et Wendy. Et donc quand Liz est arrivée, elle a étendu le réseau. Mark Overmeer m'a emmenée à l'épicerie pour que je puisse acheter des en-cas. Juerd Waalboer est arrivé avec une imprimante.

Vraiment, tout le monde a aidé. Je n'ai pas eu à rester levée tard pour fermer les portes, je pouvais juste passer les clés à quelqu'un et lui faire confiance pour que ce soit fait. Bien que je sois responsable de toute l'organisation initiale, je me suis rendu compte que tout le monde s'est investi quand il s'est agit de faire se dérouler l'événement sans accroc.

C'était parce que nous avions un plus petit groupe, où la plupart des gens se connaissaient entre eux. Contrairement à ce qui se serait passé à une conférence, j'ai vraiment eu le temps de profiter du hackathon. J'étais venue, préparée à ne rien pouvoir accomplir du tout, à cause de mon expérience avec les conférences, mais j'ai eu en fait plein de temps pour travailler.

En conclusion, une des choses que j'ai découvertes est que si vous avez toutes les pièces en place, les gens présents feront que les choses se passent une fois que le hackathon est effectivement en route.

Jean-Claude Caty

EDIGÉO : pratiquer l'échange d'information géographique

Dans un précédent article, nous avons rapidement survolé les principes et la structure de la norme EDIGÉO. Mais tout cela est bien compliqué pour le profane et surtout inexploitable sans l'assistance de logiciels spécialisés.

Le projet edigeo, rapidement évoqué précédemment, vise à intégrer un échange de données géographiques du format EDIGÉO dans une base de données géographiques, à contrôler sa régularité et utiliser les métadonnées transmises dans l'échange pour mieux structurer le résultat. Au final, il sera possible de visualiser et exploiter l'information dans un SIG (Système d'Information Géographique) comme QGIS.

Dépendances du projet

Un échange Edigéo devient intéressant lorsque interrogations géographiques et visualisation sont possibles (par exemple obtenir les rues proches de la mairie).

Le couple base de donnée PostgreSQL et son extension cartographique PostGIS permettent le stockage et l'interrogation. Le projet edigeo profite donc de ces outils. L'interface entre les composants du projet (logiciels et bibliothèques écrits en C++) et la base de données est réalisée par la bibliothèque `libpqxx`.

Par ailleurs, la popularité de PostGIS est un avantage supplémentaire, car de nombreux logiciels l'exploitent. Ainsi, le SIG QGIS permet d'en visualiser simplement les informations géographiques. Pas d'hésitation donc. Nous pouvons déléguer cette fonctionnalité et nous concentrer sur les aspects proches de la norme. Magnifique monde que celui du Logiciel libre permettant la coopération libre et non faussée entre tous !

PostGIS utilise par ailleurs les bibliothèques `geos` et `proj4`. Ces 2 bibliothèques offrent des fonctionnalités géométriques et cartographiques qui intéresseront le projet edigeo dans ses futurs développements.

Initialisation

Avant de parcourir les outils, précisons tout de suite que ceux-ci ne sont pour l'instant accessibles que sur le CVS de GNA (**https://gna.org/cvs/?group=edigeo**). Des tests sont en cours à l'issue desquels les premières versions pourront être mises à disposition. N'hésitez pas à me contacter via la liste `edigeo-aide` en cas de soucis (**https://mail.gna.org/listinfo/edigeo-aide/**). Il est préférable de centraliser les questions communes sur cet espace dédié.

Le choix des outils étant acquis, la première opération va consister à préparer un cadre pour recevoir notre information géographique. Une commune pourra par exemple utiliser ensuite ce cadre pour offrir la consultation de son cadastre à sa population. Elle recevra régulièrement les échanges lors de la tournée du géomètre du cadastre.

L'initialisation passe par les phases suivantes :

- Nous devons tout d'abord préparer une base de données et y insérer différentes métadonnées relatives à la norme. La création d'une base de données PostgreSQL nécessite le droit de création d'une base. Si vous n'êtes pas l'administrateur de la base, à vous de l'intercepter. L'utilitaire `metaedigeo` vous permettra d'initialiser votre base dès lors que l'administrateur vous aura ouvert le droit de création.

  Outre la création de la base, `metaedigeo` alimentera des tables décrivant la structure de la norme (métafichier, descripteur, champ et chaînage). D'autres tables recevront les différents systèmes cartographiques prévus par la norme. `metaedigeo` terminera par la confection des différents dictionnaires (objets, attributs, relations) édités par le CNIG.
- A ce niveau, notre base reste très classique et ne possède toujours pas la dimension cartographique. Nous devons donc appliquer plusieurs commandes propres à PostGIS. Si vous avez laissé filer votre administrateur de base de données, empressez-vous de le rattraper ! Cette opération nécessite en effet des droits d'administration DBA. C'est la raison pour laquelle `metaedigeo` s'est empressé de ne rien faire.

Sous une distribution Debian *testing*, passez les 3 commandes suivantes :

```
createlang plpgsql edigeo
psql -d edigeo -f /usr/share/postgresql-8.1-postgis/lwpostgis.sql
psql -d edigeo -f /usr/share/postgresql-8.1-postgis/spatial_ref_sys.sql
```

Si tout s'est bien déroulé, vous pouvez laisser votre administrateur préféré s'échapper. `metaedigeo` permet par ailleurs de générer un script SQL lorsque cette option est prévue sur la ligne de commande. Emballez ce script d'un magnifique paquet cadeau que vous transmettrez à votre DBA pour sa disponibilité. La commande suivante affichera la totalité des options disponibles :

```
metaedigeo -help
```

Quelques mots sur les métadonnées initialisées par `metaedigeo`. Elles peuvent être classées en 4 catégories : structure Edigéo, dictionnaires des données du CNIG, projections cartographiques et erreurs. Nous reviendrons sur les erreurs plus loin dans un paragraphe spécifique. Regardons d'un peu plus près les 3 premières catégories :

- La structure Edigéo comprend 4 tables SQL : `metafichier.sql`, `descripteur.sql`, `champ.sql` et `chainage.sql.metafichier.sql` précise la définition de chacun des types de métafichier rencontrés dans un échange (exemple : *Données générales de transmission* pour le type THF). La table descripteur liste les différents descripteurs que l'on rencontre dans un métafichier et précise la définition et la table qui accueillera ces descripteurs (Le descripteur GTL du métafichier THF est un *descripteur de lot* et sera accueilli dans la table `lot`). Dans un métafichier, l'ordre dans lequel apparaissent les descripteurs est important. La table `chainage.sql` présente cet ordre. Enfin, la table `champ.sql` présente les caractéristiques des attributs de chaque descripteur. Nous y trouvons la nature, le format, le caractère obligatoire ou facultatif...
- Les dictionnaires de données du CNIG sont également constitués de 4 tables : `nomenclatureOBJ.sql`, `nomenclatureATT.sql`, `nomenclatureVP.sql` et `nomenclatureREL.sql`. Ces tables reprennent respectivement les définitions des objets, des attributs, des valeurs précodées et des relations. Les valeurs précodées sont des codes affectés à certains attributs. Prenons l'exemple de l'attribut `DUR`. Il possède 2 valeurs précodées possibles `01` pour « bâti dur » et `02` pour « bâti léger ».
- Terminons cette revue par les projections cartographiques. Dans le numéro 91 de *Linux magazine*, un schéma montrait 3 types de référentiels cartographiques, géodésiques ou projections. Ces 3 types de référentiels sont respectivement repris dans 3 tables `CNIG-ReferenceCAR.sql`, `CNIG-ReferenceGEO.sql` et `CNIG-ReferenceMAP.sql`. Une passerelle sera établie ultérieurement avec les référentiels gérés dans PostGIS. Cette passerelle précisera le code SRID des référentiels.

Charger un échange dans la base

Nous allons pouvoir passer aux choses sérieuses, c'est-à-dire charger un échange dans la base de données. Procurez vous l'échange correspondant à votre secteur géographique auprès de votre service du cadastre (moyennant finance et à condition que votre secteur ne soit pas géré sous forme d'image).

Le programme `lireedigeo` (`lireedigeo --help`) permet d'opérer à condition de lui fournir le nom du métafichier suffixé « `THF` ». Au premier lancement, il préparera les tables nécessaires. Le principe général est simple : une table pour chaque descripteur et autant de tables que de listes possibles dans un descripteur. Par exemple, un descripteur de lot (table `lot`) accueille la liste des métafichiers vectoriels d'un lot (sous-ensemble de données vectorielles). Cette liste sera chargée dans la table `GTLSousEnsemble`.

Puisque nous travaillons au sein d'un système cartographique, attardons-nous sur la géométrie. Celles-ci sont des colonnes à part entière dans une table. En fonction des types de géométries Edigéo, nous aurons les correspondances suivantes :

Edigéo	PostGIS
Nœud (géométrie ponctuelle)	POINT
Arc (géométrie linéaire)	LINESTRING
Emprise (rectangle)	POLYGONE
Polygone de validité	POLYGONE

Les polygones de validité sont des éléments d'un métafichier relatif à la qualité permettant de déterminer les précisions sur des données de contrôles.

Notons que, dans le modèle Edigéo, les faces d'objets surfaciques ne portent pas de géométrie. Celle-ci est déduite de relations entre faces et arcs. Des traitements prévus par le projet permettront ultérieurement de reconstituer la géométrie des différents objets surfaciques d'un échange.

Dès le deuxième lancement, il ne sera plus utile de construire la structure de la base. Seules les données seront rangées dans la base.

Les vérifications du programme de lecture restent basiques. Nous y trouvons essentiellement des vérifications syntaxiques et quelques vérifications sémantiques. Elles peuvent aboutir à ignorer un descripteur mal formé et plus rarement un métafichier, un sous-ensemble, un lot et même l'échange lui-même si le métafichier `THF` (base de l'échange) est incohérent.

Au fil de la lecture, les descripteurs sont insérés en base. Les requêtes SQL correspondantes peuvent être exporter dans un script en le spécifiant sur la ligne de commande (option `-s <nom du script>`).

Après chargement, il est tout naturel de vouloir visualiser les informations cartographiques. Compte tenu du modèle géométrique d'Edigéo, nous ne pourrons voir que des nœuds et des arcs à cette étape. Précipitons-nous sur le SIG libre QGIS. Il est en effet capable de se connecter à une base PostGIS et de détecter les colonnes géométriques. Les informations de connexion sont les suivantes :

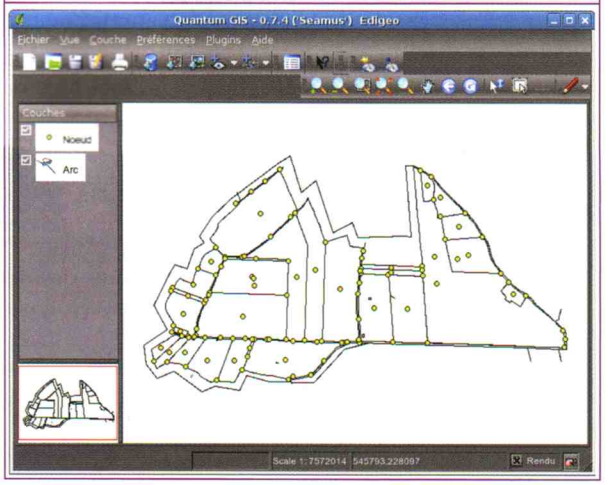
Nom : `Edigeo` ; Hébergeur : `localhost` ; Base de données : `edigeo` ;

Port : `5432` ; Nom d'utilisateur : `yyy` ; Mot de passe : `xxx` ;

Intégrez ensuite les tables arc (colonne `par_arc`) et nœud (colonne `pno_xynoeud`) (Figure 1) et admirez... (Figure 2)

Figure 1 : Géométries à afficher

Figure 2 : Affichage dans QGIS



Bibliothèque libedigeo

La majorité des traitements opérés dans `metaedigeo` et `lireedigeo` sont, fort heureusement, sous-traités à la bibliothèque `libedigeo`. Celle-ci est composée de classes C++ reprenant les briques d'un échange. Nous y trouvons donc des classes pour les échanges, les lots, les sous-ensembles, les différents métafichiers, les enregistrements (champs des descripteurs) et la gestion des différents formats supportés par la norme (formats A, C, D, E, I, N, P, R et T) déjà présentés dans le précédent article.

À ces briques principales, ajoutons la gestion des erreurs, développée plus bas, une gestion des messages afin de faciliter d'éventuelles traductions, un module pour les requêtes vers la base de données et une gestion des exceptions ayant pour but principal de lire un échange aussi complètement que possible en écartant les parties à problèmes. Ces parties peuvent être un simple descripteur, mais aussi un métachier, voire un sous-ensemble, un lot ou un échange complet.

Revenons à la base de données. L'ensemble des échanges sera chargé dans une seule et unique base. Nous devons cependant pouvoir contrôler chaque échange dans le cas possible où une multiplicité d'erreurs nécessite de le supprimer. C'est la raison pour laquelle la base de données possède une table des échanges vers laquelle se réfèrent chaque descripteur, chaque faute et les éventuels doublons de descripteurs.

Par ailleurs, le gestionnaire PostgreSQL propose la notion « d'héritage entre tables ». Tous nos descripteurs possèdent une base commune constituée de l'échange, de l'identifiant du métafichier, de l'identifiant du lot, du type du descripteur et de son identifiant. Les tables pour chacun des descripteurs héritent donc toutes de la table `base` regroupant ces colonnes communes.

Si nous nous arrêtions au stade actuel du projet, ce serait terminé. Il reste cependant bien des possibilités d'exploiter Edigéo. Nous allons donc examiner le futur c'est-à-dire principalement la gestion des SCD puisque Edigéo transporte le modèle des données de l'échange.

Gestion des SCD (Schémas Conceptuels de Données)

Cette aptitude à embarquer le modèle des données est similaire au format xml-schéma. De nombreuses capacités de traitements deviennent possibles.

- Nous pouvons vérifier automatiquement que les données correspondent à ce qui est annoncé, contrôler que le modèle correspond bien à la convention passée entre le producteur et le destinataire des données.
- Outre des vérifications, nous pouvons générer une structure de données conformément au modèle et y incorporer les informations transmises dans l'échange.

Le projet edigeo prévoit des développements en ce sens. La gestion des SCD permettra de récupérer le schéma d'un échange afin de le conserver et l'amender. La norme Edigéo souffrant de quelques imprécisions (notamment dans l'expression des cardinalités des relations), il faudra améliorer ces aspects. Un modèle conventionnel est susceptible d'évolutions. Une gestion de version tiendra compte de cet aspect.

L'incorporation d'un échange dans une structure construite automatiquement implique des vérifications strictes et le rejet des données non conformes susceptibles de corrompre la structure d'accueil.

Gestion des erreurs

Le chargement existant, les vérifications à mettre en œuvre impliquent une gestion d'erreurs complexe. Les principes retenus dans le projet sont principalement :

- la possibilité d'internationaliser les messages. Même si la norme est française, elle peut être utilisée par des étrangers ;
- établir des catégories d'erreurs ;
- une même erreur peut être considérée dans sa globalité ou dans un contexte très précis ;
- découper finement les erreurs d'une catégorie pour optimiser les souplesses de traitements ;

Reprenons et explicitons ces différents principes.

L'internationalisation des messages d'erreur est gérée via un script SQL. Les messages sont intégrés au moment de l'initialisation par `metaedigeo`. La table `erreur` de la base de données Edigéo est alimentée par le script `erreur.sql`. Il suffit de remplacer les scripts SQL français par un script d'une autre langue pour prendre en compte cette autre langue. Ceci ne vaut pas pour les autres types de messages.

Les catégories d'erreurs sont gérées dans la table `typeserreurs.sql`. A chaque type correspond un code formé d'un caractère et d'un libellé. Là aussi, la traduction du libellé est possible en remplaçant le script. Les codes doivent rester inchangés. Ils sont repris dans chaque code d'erreur et complétés d'un nombre de 3 chiffres. Ainsi l'erreur Q001 est du type Q (Erreur sémantique). Actuellement, les types d'erreurs prévus sont : Alerte, Divers, Grammaire, Modèle, Sémantique, Syntaxe. Une catégorie Géométrie et probablement d'autres viendront compléter cette courte liste.

Pour mieux comprendre le contexte d'une erreur, considérons les 2 libellés associés à l'erreur Q001. Le libellé générique de cette erreur est « Dimension d'un lot indéfinie ». Nous ne précisons pas ici quel lot pose problème. Ce sera fait avec le deuxième libellé : « la dimension du lot $1 ne peut être déterminée ». La partie « $1 » sera remplacée par le nom du lot au sein du code à la détection de l'erreur.

Le découpage fin des erreurs vient d'être cité. Le type de l'erreur est complété de 3 chiffres formant un code d'erreur. Ce code est utilisé dans le code pour construire le libellé. L'information est ensuite enregistrée au sein de la table `faute` répertoriant les problèmes de tous les échanges analysés. Ultérieurement, des utilitaires spécialisés pourront associer une erreur à un élément du modèle et attribuer une gravité à cette association.

Conclusion

Le projet edigeo commence tout juste à devenir intéressant dans la mesure où il permet un affichage sommaire de contenus cartographiques. La richesse du format Edigéo, bien plus qu'un simple format de dessin, offre des potentialités insoupçonnées. L'ambition du projet, déjà vaste, pourrait encore être complétée d'autres outils comme l'apport du *raster* négligé jusqu'ici, l'écriture vers Edigéo afin de pouvoir assurer la chaîne de bout en bout...

Jean-Claude Caty,

jc.caty@laposte.net

Au rayon configuration système, gruikage et autres linuxeries, GNU/Linux magazine vous propose ce mois-ci en promotion :

Yann Guidon

Personnalisation (un peu brutale) du système de login et des terminaux virtuels

Mes ordinateurs connectés par réseau sont protégés par des mots de passe assez solides. Pour les autres postes, les mots de passe sont introuvables... car il n'y en a pas. Mais pour se loguer en root (faire du développement système dans un compte utilisateur est pénible), il faut encore taper « root »-[entrée] avant de pouvoir lancer les premières commandes. Quelle limitation insoutenable ! En plus, le système des terminaux virtuels, déjà bien pratique, manque encore de « flexibilité ».

1. Unix, comment ça marche...

Les manipulations suivantes ont été effectuées sur une Slackware 11 toute fraîche, dont l'installation a déjà été légèrement trifouillée, mais rien de bien méchant pour l'instant (si on omet l'installation de `nano` en *dumpant* celui installé sur une Debian ou le bricolage de `/etc/fstab` ou...). Donc cela s'applique à quasiment toutes les distributions Linux, en changeant éventuellement les noms des fichiers. Pour transformer ce système déjà minimal (sans X11) en un système qui entrave encore moins mes élans de développement et de reconfiguration, je me suis attaqué au mécanisme de *login* et de gestion des terminaux virtuels.

Pour arriver jusqu'à l'affichage du *prompt* demandant d'entrer le nom de l'utilisateur, l'ordinateur effectue un nombre impressionnant de tâches d'initialisation. Le BIOS doit être initialisé, pour détecter les disques et y trouver le secteur d'amorçage. L'exécution de celui-ci charge le reste du *boot loader* (`GRUB` ou `LILO`) qui, à son tour, localise et charge en mémoire le noyau (`/boot/vmlinuz`) et ses éventuels fichiers annexes (`initrd`). Le noyau Linux s'initialise et détecte les périphériques essentiels au système, en commençant par le disque racine.

Linux monte alors la partition racine et exécute le programme `/sbin/init` qui va trouver sa configuration dans le fichier `/etc/inittab`. `/sbin/init` est critique dans tous les systèmes de type UNIX, car c'est habituellement le *process* n°0 et il reste actif tant que le système fonctionne. D'ailleurs, si ce programme était tué ou s'achevait, le noyau bloquerait l'ordinateur.

`/etc/inittab` contient les noms des fichiers à exécuter pour initialiser le reste de la machine, ou pour l'éteindre. Ce sont habituellement des scripts situés dans `/etc/rc.d/` et le système varie beaucoup d'une distribution à une autre. Ce qui change peu, en revanche, ce sont les lignes qui lancent les processus de login, vers la fin du fichier. Sur Slackware, cela donne :

```
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
c3:1235:respawn:/sbin/agetty 38400 tty3 linux
c4:1235:respawn:/sbin/agetty 38400 tty4 linux
c5:1235:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux
```

`man 5 inittab` nous apprend que ces lignes sont découpées de la manière suivante :

1) Identifiant de la ligne (limité à deux lettres).

2) Niveaux (*runlevels*) dans lesquels cette ligne est active. `/sbin/init` se charge de démarrer la ligne si le niveau change vers un état indiqué dans ce champ.

3) Mode de fonctionnement ou action. Ici, `respawn` signifie que la commande du champ suivant doit être relancée à chaque fois qu'elle s'arrête. En cas de bug (commande qui ne répond pas, par exemple), il y a le risque que cela entraîne le système dans une boucle infinie qui accapare les ressources de l'ordinateur. Une protection interne bloque la boucle pendant cinq minutes si trop de relances sont effectuées pendant un certain temps.

4) Commande à lancer. C'est justement le plus intéressant.

La commande qui gère les invites de login est ici `/sbin/agetty`. `man 1 agetty` nous apprend que ce petit programme se connecte sur un terminal (ici, `tty`X), affiche un petit message (`/etc/issue`) et l'invite de login, puis attend.

2. Mille et une façons de faire sauter son login

Vous en avez assez de retaper votre mot de passe tout le temps ? Si la situation l'autorise, c'est-à-dire sur un système monoutilisateur dont vous êtes l'administrateur, qui n'est connecté à rien et qui n'est accédé physiquement par personne, la première solution consiste à faire un tour dans les fichiers `/etc/passwd` et `/etc/shadow`.

```
# première ligne de /etc/passwd
root:x:0:0:d4M45t3r:/root:/bin/bash
# première ligne de /etc/shadow
root:(plein de signes):9804:0:::::
```

Avec l'éditeur de votre choix, il suffit d'enlever la petite croix et/ou le mot de passe crypté pour ne plus être bêtement questionné lors du login. C'est un bon début (regardez la réaction d'un collègue un peu tâtillon lorsque vous évoquez cette éventualité), mais ce n'est *que* le début :-)

D'ailleurs, qui vous demande ce mot de passe ? C'est le programme `/bin/login` qui est appelé par `/sbin/agetty` avec comme argument le nom du

compte accédé. Puisqu'il n'y a plus de mot de passe *root*, `/bin/login` ne sert plus à rien, et pourtant on ne peut pas l'effacer, car `/sbin/agetty` en a besoin. Une petite manipulation suffit pourtant :

```
cd /bin
mv login login.renamed # sauvegarde le programme
ln -s bash login  # agetty appellera directement bash
```

Cela fonctionne, mais peut poser quelques soucis à l'usage. D'une part, `/bin/login` met en place un certain nombre de variables d'environnement en lisant `/etc/passwd`. Par exemple, `$HOME` permet de retrouver l'adresse du `~/.bashrc`. D'autre part, `/sbin/agetty` appelle `/bin/login` avec un certain nombre de paramètres qui ne sont peut-être pas *compris* par `/bin/bash`. Un *quiproquo* empêcherait le démarrage, on peut modifier le système précédent par un petit fichier remplaçant `/bin/login` :

```
#!/bin/bash
export HOME=/root
exec /bin/bash -l
```

En croyant appeler le programme `/bin/login`, `/sbin/agetty` va provoquer le lancement de `/bin/bash` pour interpréter le fichier (c'est la première ligne du script). La deuxième ligne crée la variable d'environnement `$HOME` pour que `/bin/bash` puisse retrouver ses fichiers de configuration locaux. Enfin, `bash` s'exécute lui-même en mode login, c'est-à-dire qu'il va d'abord sourcer `/etc/profile` pour charger la configuration complète de l'environnement (autocomplétion, `$PATH` globaux etc.). La commande `exec` est importante, car elle dit au `bash` qui interprète le fichier de se faire *remplacer* par le `bash` interactif. Sans cela, il y aurait deux instances de `bash` en mémoire (au lieu d'une).

Du point de vue de l'utilisateur, le prompt de login subsiste, mais n'importe quelle entrée (sauf malheureusement une ligne vide) provoquera le lancement de l'interpréteur interactif de lignes de commandes. Comment enlever ce dernier rempart à l'accès immédiat ? Il faut encore taper au moins sur deux touches...

On pourrait essayer la technique suivante :

```
c1:1235:respawn:/bin/bash -l
c2:1235:respawn:/bin/bash -l
c3:1235:respawn:/bin/bash -l
c4:1235:respawn:/bin/bash -l
c5:1235:respawn:/bin/bash -l
c6:12345:respawn:/bin/bash -l
```

Mais cela pose un *gros problème* : les différentes instances de `bash` se retrouveront sur le même terminal virtuel, mais un seul peut accepter les commandes de l'utilisateur à la fois.

3. Dessine-moi un vt

L'utilisateur débutant connaît le terminal habituel : c'est le clavier et l'écran avec ses fameuses et angoissantes lettres sur fond noir. Avec Linux (et quelques autres systèmes modernes), cette sombre réalité a été égayée grâce à l'affichage en couleur (pratique pour la coloration syntaxique ou la navigation rapide dans les répertoires), la souris (avec `gpm` et son copier-coller indispensable) et le mode *framebuffer* (les caractères sont affichés en mode graphique, ce qui permet des densités d'affichage effrayantes). Les terminaux virtuels sont une autre amélioration indispensable à un système multitâche, car le terminal physique (votre écran et votre clavier) peut ainsi afficher un des nombreux terminaux virtuels (interface vers un programme).

Normalement, on passe d'un terminal virtuel à l'autre en appuyant sur la touche [windows] (sur les claviers des PC) ou alors directement avec les combinaisons de touches [Alt]-[F*n*] (où *n* est le numéro du terminal virtuel). On accède ainsi instantanément à 12 terminaux virtuels, comme à 12 fenêtres de `xterm` qui se recouvriraient (à quoi bon utiliser X11 ?). Mais à moins que votre noyau soit compilé avec des options restreintes (pour réduire l'encombrement de la mémoire par exemple), il dispose de bien plus que 12 terminaux virtuels (« *vt* » pour les initiales anglophones). Sur mon système, le répertoire `/dev` contient beaucoup de fichiers spéciaux dont le nom commence par `tty`, et ceux qui nous intéressent sont les fichiers `tty1` à `tty63`.

Sur le compte du super-utilisateur et dans le premier terminal virtuel (accédé avec [Alt]-[F1]), la commande

```
date > /dev/tty2
```

affiche la date courante dans le deuxième terminal virtuel (accédé avec [Alt]-[F2]). L'intérêt pratique de cette manipulation est limité, mais cela illustre comment le système des terminaux virtuels fonctionne : le programme qui tourne dedans s'y *attache*, en redirigeant le clavier vers son `stdin`, alors que `stdout` et `stderr` sont redirigés vers l'écran. La différence avec un terminal classique (antique ?) est que le terminal virtuel a une zone d'affichage dans la mémoire du système, et l'écran physique recopie son contenu lorsque l'on passe d'un *vt* à un autre. Ainsi, le programme qui envoie des caractères n'est pas interrompu lorsque l'utilisateur affiche un autre terminal virtuel.

Vous avez peut-être aussi remarqué que l'ordinateur affiche un petit pingouin^Wmanchot en haut à gauche de l'écran lorsque Linux démarre. Cela veut dire que votre noyau supporte le mode framebuffer, c'est-à-dire qu'il affiche les lettres avec des pixels individuels. Le premier intérêt est que cela permet d'afficher beaucoup plus de caractères à l'écran, en particulier avec des polices de 8 pixels de côté. Avec une résolution modeste de 800×600 pixels, l'écran affiche facilement 75 lignes de 100 colonnes, plus que le mode texte standard de 80 colonnes et 25 lignes (ou 43 lignes en poussant un peu). Les écrans récents (1024×768 et 1280×1024 pour commencer) permettent donc d'afficher énormément d'informations, ce qui est bienvenu pour survoler des fichiers de configuration ou du code source.

L'autre intérêt du mode framebuffer est qu'il permet justement d'afficher des informations graphiques. Pour vous en convaincre, la commande

```
cat /dev/urandom > /dev/fb0
```

remplit l'écran de pixels de couleurs aléatoires. L'interface de programmation du framebuffer est bien plus aisée que celle de X11 (bien que plus primitive), mais sa puissance réside dans son accès direct aux pixels, en m'utilisant presque aucune ressource superflue du processeur et du système d'exploitation. Le framebuffer est idéal pour afficher des informations simples en temps réel sans nécessiter de logiciels complexes, donc c'est parfait pour les applications embarquées. Grâce à ce mode, le vieil adage « Donnez-moi l'adresse de la mémoire vidéo et je vous programme un jeu » devient réaliste.

NOTE

Le support du mode framebuffer est choisi au moment de la compilation du noyau. Vous devrez recompiler ce dernier si les techniques présentées ici ne fonctionnent pas, mais la plupart des noyaux fournis dans les distributions actuelles supportent nativement ce mode.

La première étape est bien sûr de vérifier que la mascotte Tux apparaît bien lors du démarrage de Linux. Dans ce cas, la lecture de ce cadre est juste informative.

Si le noyau a été compilé avec le support du framebuffer, il faut activer un mode graphique au démarrage. La première technique est assez radicale, elle consiste à indiquer directement le mode désiré en modifiant le binaire du noyau.

```
vidmode /boot/vmlinuz-ide-2.4.33.3 773
```

Le programme `vidmode` sert uniquement à *patcher* le fichier du noyau au bon endroit. Le mode graphique désiré (ici, 773 correspond à une résolution de 1024×768 pixels en 256 couleurs) est indiqué dans les sources du noyau, dans `/usr/src/linux/Documentation/fb/vesafb.txt` (attention lors de la conversion de l'hexadécimal au décimal).

L'autre méthode consiste à donner ce numéro dans la ligne de commande, au lancement du noyau. C'est l'option `vga=773` qui peut être incluse dans `/etc/lilo.conf` ou avec `loadlin`, `grub`...

La hauteur des caractères (donc la densité de l'affichage) peut être configurée en incluant uniquement la table désirée dans le noyau (à la compilation) ou bien (en fonctionnement) au moyen du programme `setfont` (voir éventuellement `/etc/rc.d/rc.font` et le répertoire `/usr/share/kbd/consolefonts`).

Pour terminer, les modes à 256 couleurs (n°771, 773 et 775) sont vivement recommandés. Les modes avec moins de couleurs sont complexes à gérer (les pixels ne sont pas des octets consécutifs) et les modes 16, 24 ou 32 bits/pixel nécessitent trop de bande passante. Le processeur doit recalculer l'image à chaque modification du terminal virtuel, et le *scrolling* est d'autant plus lent que l'écran est grand.

4. C'est assez, dit le manchot

Nous avons donc un logo amical en haut de l'écran durant le démarrage (figure 1) et tout va bien. L'ennui, c'est qu'il masque le haut de l'écran, pendant et *après* l'initialisation du noyau (figure 2). En utilisation normale, cela peut conduire à un massacre visuel, en particulier si l'utilisateur efface l'écran ([Ctrl]-[L]) puis lance d'autres commandes (figure 3).

Ces débris s'en vont si on passe dans un autre terminal virtuel, puis au premier. Une succession de [Alt]-[F2] [Alt]-[F1] est une méthode manuelle pour y parvenir, mais elle est ennuyeuse à la longue...

Pour changer de terminal, il existe aussi une commande : `chvt`. Sa page de manuel est laconique mais à l'image de la simplicité du programme, qui ne demande que le numéro du *vt* vers lequel basculer l'écran.

Pour enlever automatiquement le petit désagrément des restes de manchot avant d'activer le login, il suffit donc de modifier le fichier `/etc/rc.d/rc.local`, qui est exécuté à la fin de la séquence d'initialisation.

```
# Ajout à /etc/rc.d/rc.local (selon la distro)
chvt 2 # bascule vers tty2
chvt 1 # revient sur le tty1 original, sans manchot !
```

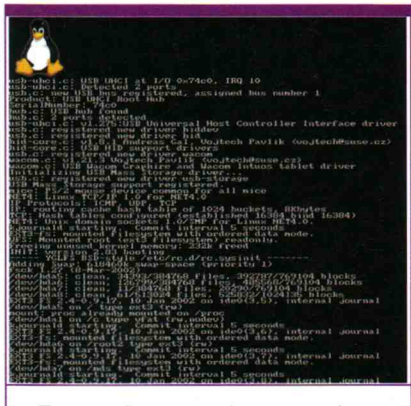


Figure 1 : Pingouin au boot, tout va bien.

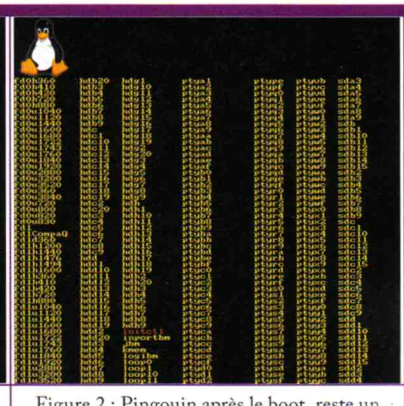


Figure 2 : Pingouin après le boot, reste un doute.

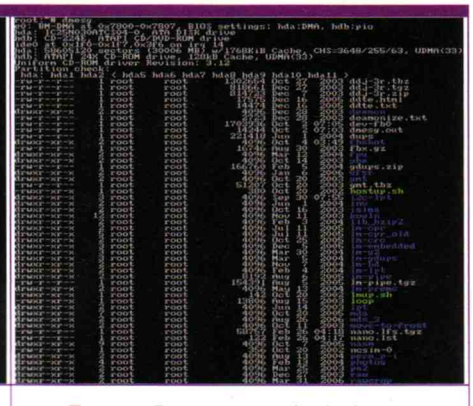


Figure 3 : Pingouin parti, c'est la charpie.

ATTENTION

Les commandes et manipulations décrites ici sont potentiellement très dangereuses. Elles ne doivent être reproduites que sur un système dédié à cet usage et ne contenant aucune information utile. Un problème arrivant forcément à un moment ou à un autre, il est conseillé de garder un système de récupération (*live*-CD ou OS sur une autre partition) et de ne modifier que progressivement l'inittab.

5. Des terminaux en vrac

`chvt` est une commande assez simple, mais intéressante seulement lorsqu'elle est combinée avec d'autres commandes (une approche très unixienne, finalement). Une autre commande de la même famille est `openvt` : ce programme cherche un *vt* libre, s'y *attache* et y lance une autre commande spécifiée en argument. Diverses options sont disponibles :

- `-s` : bascule (**s***witch*) l'affichage vers le nouveau terminal alloué.
- `-w` : attend (**w***ait*) la fin de l'exécution de la commande. En combinaison avec `-s`, cela permet de retourner sur le *vt* d'où la commande a été lancée.
- `-e` : **e**xécute la commande en argument sans *forker* (si on veut que l'espace mémoire occupé par `openvt` soit libéré lorsqu'on ne souhaite ni attendre la fin du programme, ni retourner au processus appelant).
- `-c n` : spécifie le *vt* n°n.
- `-f` : **f**orce le numéro de *vt* donné par `-c n` (dans le cas où le terminal est utilisé par un autre programme, c'est potentiellement dangereux, mais indispensable dans certains cas).
- `--` : tout ce qui suit ce délimiteur est considéré comme étant la commande à exécuter dans le nouveau *vt*.
- `-l` : démarre le programme comme **l***ogin* le ferait (en ajoutant un '-' au début de son nom).

En jouant sur ces différents arguments, il est possible d'obtenir un certain nombre de comportements plus ou moins désirables, en fonction de la manière de créer de nouveaux terminaux. Le plus simple est d'essayer la commande

```
openvt -s -l -- bash -l
```

Naturellement, le terminal bascule vers un nouveau *vt* et `bash` démarre une nouvelle session. Quand elle se termine (en tapant `exit`, `logout` ou [Ctrl]-[D]), l'écran reste inerte et il faut retourner sur un autre *vt* en tapant [Alt]-[Fn] (où n est le numéro du *vt* désiré). Cela étant un peu embêtant, un développeur a eu l'idée de réaliser la fonction `-w` décrite ci-dessus. Les grands esprits se rencontrent, les fainéants aussi.

À partir de là, il est possible de construire une commande qui va créer un nouveau *vt* à la demande. J'ai par exemple défini l'alias suivant dans mon `/etc/profile` :

```
alias newvt="/usr/bin/openvt -w -s -l -- /bin/bash -l &"
```

Il est ainsi possible de créer autant de *vt* que désiré, tant qu'il en reste de disponible (avec une limite de 63, il y a de la marge).

A l'usage, pourtant, on trouve deux effets de bord ennuyeux :

- D'une part, imaginons que la commande `newvt` soit lancée du terminal 8 et crée le terminal n°9. Ensuite, la session `bash` se termine sur le terminal n°8 (qui retourne à son terminal d'origine). Lorsque la session du terminal n°9 s'achève, `openvt` ainsi que la session `bash` forkée qui l'attendait ne sont plus là pour changer le terminal actif vers le n°8. Il n'y a donc plus de retour automatique.
- D'autre part, à force d'allouer des terminaux, ils peuvent finir tous alloués à la longue, même si tous ne sont pas actifs. Cela se voit en balayant (avec la touche [Windows]) les terminaux alloués : les sessions terminées ne libèrent pas automatiquement le terminal utilisé.

Pour résoudre le second cas et éviter un épuisement des *vt*, il existe la commande `deallocvt` qui libère les *vt* inutilisés. La limitation est que cela ne fonctionne pas si le *vt* à libérer est le *vt* actuellement affiché ou si un programme y est encore attaché. Donc le *vt* ne peut être automatiquement libéré, même si on ajoute les lignes suivantes au `~/.bash_logout` :

```
chvt 1    # celui-ci est habituellement actif
deallocvt # tente de libérer le terminal
```

Puisqu'il interprète la dernière commande, `bash` n'est pas encore terminé, donc il est toujours attaché au *vt*, donc celui-ci ne peut pas être libéré. Pour forcer un peu les choses, il est possible de redéfinir l'alias, en ajoutant la commande `deallocvt` pour libérer a priori les *vt* en attente, puis a posteriori celui qui vient d'être quitté.

```
alias newvt="(/usr/bin/deallocvt ; \
              /usr/bin/openvt -e -s -l -- /bin/bash -l ; \
              /usr/bin/chvt 1 ; \
              /usr/bin/deallocvt ) &"
```

Cette approche pose deux problèmes, bénins mais dont les implications sont complexes.

- La commande est exécutée par le `bash` d'origine. Pour opérer la commande parallèlement aux autres commandes en cours («&»), `bash` *forke* et crée une nouvelle instance en mémoire. Cette dernière occupe de la place (plusieurs mégaoctets de contexte) qui, bien qu'elle puisse être placée dans le *swap* au besoin, *occupe quand même de la place*. Pour comparer, `openvt` seul occupe environ 300Ki octets.

▶ D'autre part, le terminal 1 choisi par défaut n'est probablement pas le plus *pertinent*. La question du retour vers le terminal appelant pose la question plus générale de savoir vers lequel revenir.

Pour résoudre ces problèmes, j'ai été confronté à une autre question importante : comment le code de `/etc/profile` peut-il savoir à quel terminal il est attaché ? Je n'ai trouvé aucune commande ou entrée dans `/proc` pour le savoir. `man utmp` apporte un premier élément de réponse, mais insiste sur la faillibilité de cette méthode. De plus, `bash` ne peut pas interpréter le fichier `/var/run/utmp`, car c'est un fichier binaire. Enfin, `/var/run/utmp` est mis à jour par plusieurs programmes dont `login` et `(a)getty`, que nous cherchons justement à court-circuiter.

Par contre, je me suis aperçu que la commande `ps` indique quel programme est attaché à quel terminal. Avec un peu de remise en forme, on obtient la commande suivante que j'ai intégrée à `/etc/profile` pour afficher une petite bannière à chaque nouveau terminal.

```
VT=$(ps t|grep '[-]/bin/bash -l' \
    |head -n 1|sed 's/.*tty// ; s/ .*//')
stty sane
echo -e "\\033[2J\\033[f\\033[0;32mtty$VT\\033[1;38m"
```

La variable `$VT` est noyée au milieu de codes de contrôle pour effacer l'écran, remettre le curseur en haut à gauche, écrire en vert, et revenir à la couleur normale après l'affichage.

6. Le coup de la pile

Connaissant ce numéro (qui révèle parfois des erreurs dans l'attachement des programmes), il est éventuellement possible de construire une pile des numéros des terminaux, au moyen de la version suivante de la définition de `newvt` :

```
alias newvt="(/usr/bin/deallocvt;                    \
              export VTSTACK="$VT $VTSTACK";         \
              /usr/bin/openvt -e -s -l -- /bin/bash -l; \
              /usr/bin/chvt $VT;                     \
              /usr/bin/deallocvt ) &"
```

L'idée est qu'à la création de chaque nouvelle instance, la variable d'environnement `$VTSTACK` conserve la liste de tous les terminaux qui ont mené à la création de l'instance en cours. La ligne en rouge concatène la valeur actuelle de `$VT` à la liste. Le code de `~/.bash_logout` pourrait ensuite balayer cette liste pour trouver un terminal valide.

Cependant, il n'y a pas de moyen direct de savoir si un terminal est actif. Il n'y a pas de liste contenant les numéros des terminaux disponibles (ou déjà attachés) et la commande `chvt` n'échoue pas lorsqu'on lui demande de basculer vers un terminal inactif. Quant à `openvt` et `deallocvt`, je ne sais pas comment ils font pour savoir quels terminaux sont disponibles. Je n'ai pas cherché dans leur code source, en fait.

Toutefois, on peut réutiliser la liste des terminaux attachés fournie par `ps`. Le code qui calcule `$VT` utilise `ps` pour n'afficher que les programmes rattachés au `bash` courant, mais la commande `ps a` affiche *tous* les programmes attachés en cours. Il faut donc construire la liste des `bash -l` en cours (moins celui qui est en train de faire la recherche), puis la balayer pour trouver l'élément le plus récent de la pile. Tout un programme.

```
#------------------------------
# modification de /etc/profile :
#------------------------------

# au début
export HOME=/root # sinon ~/.bash_logout sera ignoré

# à la fin
stty sane
VT=$(ps t|grep '[-]/bin/bash -l' \
    |head -n 1|sed 's/.*tty// ; s/ .*//')
echo -e "\\033[2J\\033[f\\033[0;32mtty$VT\\033[1;38m"
export VTSTACK="$VT $VTSTACK"  # empilage par le début
alias newvt="/usr/bin/deallocvt ; \
             /usr/bin/openvt -s -l -- /bin/bash -l ;"

#------------------------------
# ajout à ~/.bash_logout :
#------------------------------

function cherche_VT() {
  # recherche un numéro de tty autre que $VT
  for i in $VTLIST; do
    if [ "$VT" !=  "$i" ]; then
      # recherche $i dans la pile
      for j in $VTSTACK; do
        last=$j
        if [ "$j" ==  "$i" ]; then
          # trouvé !
          /usr/bin/chvt $j
          return # c'est mieux que break
        fi
      done
    fi
  done
  # défaut, si on n'a rien trouvé
  if [ "$last" !=  "" ]; then
    chvt $last
  else
    chvt 1
  fi
}

# le code qui englobe la fonction
if [ "$VT " != "$VTSTACK" ]; then
# ne pas oublier l'espace à la fin de $VT
  VTLIST=$(ps a|grep '[-]/bin/bash -l' \
          |sed 's/.*tty// ; s/ .*//'|sort -r)
  cherche_VT
  /usr/bin/deallocvt
fi
```

Résultat : d'une part, l'occupation de la mémoire est minimale (pas de `openvt` ou de `bash` en attente),

La qualité à votre portée

Serveurs Dédiés IBM x Séries

IDC avec toutes les garanties de sécurité

Support technique gratuit par mail et téléphone

Garantie 30 jours Satisfait ou Remboursé

Adresses Ips illimitées

Devis en ligne et sur mesure

...

Découvrez tous les avantages de travailler avec la meilleure marque. Arsys, hébergeur professionnel depuis plus de 10 ans, vous offre votre propre serveur IBM ou Supermicro avec toute la technologie, sécurité et qualité de service à partir de **99 €**/mois.

Découvrez avec arsys.fr l'internet de qualité.

arsys.fr
internet de qualité

Noms de Domaine | Hébergement | Serveurs Dédiés | Applications

Dédié Générique
Dédié Administré
Dédié de Courrier

www.arsys.fr / 0800 940 865
Appel Gratuit

d'autre part, la fermeture d'un *vt* ne laisse pas le terminal inactif (dans le pire des cas, il bascule vers le *vt* le plus bas).

7. Au tour d'inittab

Il faut bien qu'un *vt* au moins soit lancé à un moment, afin de pouvoir créer les autres. C'est la tâche confiée à `/etc/inittab` que nous avons examiné auparavant.

Nous en étions restés au problème de lancer plusieurs `/bin/bash` dans des *vt* différents, et `openvt` est justement conçu pour cela. Par exemple, l'argument `-c n` spécifie que `bash` sera attaché à ttyn. C'est important, car il semble y avoir une *race condition* dans le système des *vt* : `/bin/init` lance toutes les lignes simultanément, ce qui résulte en l'attachement de tous les `/bin/bash` sur un seul terminal (qui devient inutilisable).

L'argument `-f` résout le problème de l'attachement à tty1, qui est occupé normalement à afficher tous les messages lors de la séquence de démarrage. L'absence de cet argument rend tty1 inutilisable.

Enfin, l'argument `-e` empêche `openvt` de forker, puis de revenir immédiatement à `init`, ce qui provoquerait une accumulation d'instances de `/bin/bash` sur le *vt* forcé (ce qui rendrait aussi ledit *vt* inutilisable), ainsi que le blocage de cette ligne par `/sbin/init` pendant cinq minutes.

```
c1:1235:respawn:/usr/bin/openvt -e -l -f -c1 -- /bin/bash -l
c2:1235:respawn:/usr/bin/openvt -e -l -f -c2 -- /bin/bash -l
c3:1235:respawn:/usr/bin/openvt -e -l -f -c3 -- /bin/bash -l
c4:1235:respawn:/usr/bin/openvt -e -l -f -c4 -- /bin/bash -l
```

Au démarrage, l'ordinateur va donc afficher directement l'invite de `bash`, et en propose trois autres instances disponibles sur tty2, tty3 et tty4. D'autres *vt* peuvent être créés dynamiquement à la demande (avec l'alias `newvt` défini précédemment), ce qui évite d'avoir à chercher un compromis au sujet du nombre de lignes à inclure dans `/etc/inittab` (un de mes ordinateurs a 12 *vt* disponibles en permanence).

8. Un dernier coup pour la route

La méthode précédente est encore empreinte de conventions, ce qu'elle paie par un dernier zeste d'inflexibilité. Que se passe-t-il si les quatre *vt* sont accaparés et empêchent l'utilisateur de demander d'autres terminaux ?

Il aurait été formidable d'avoir d'autres touches, en plus de celles existantes, pour contrôler les *vt*. Par exemple, une touche permettant d'en créer un nouveau serait bienvenue pour résoudre le problème soulevé à l'instant. Mais il n'est pas raisonnable d'ajouter une touche spéciale à tous les claviers. Par contre, l'utilisation de quelques live-CD m'a donné une petite idée.

Elle consiste à dédier un terminal (le n°1 ici) à la création des *vt*. Un petit programme boucle, en attendant un ordre de l'utilisateur, avant de créer un autre terminal par les moyens déjà examinés ici. Ainsi, plus de risque d'être pris au piège par les autres *vt*, il suffit d'appuyer sur [Alt]-[F1] puis [ENTREE] pour créer un autre terminal à tout moment.

Le fichier `/etc/inittab` s'en trouve encore raccourci, puisque seuls deux terminaux *normaux* subsistent. Le générateur de terminaux est contenu dans un petit fichier additionnel, appelé ici `vtpool`. Enfin, le fichier `/etc/rc.d/rc.local` est légèrement altéré, afin d'obtenir immédiatement un terminal utilisable dès la fin du démarrage de l'ordinateur. Enlever une commande permet d'économiser un appui sur la touche [ENTREE] :-)

```
#-----------------------------
# /etc/inittab :
#-----------------------------

c1:1235:respawn:/usr/bin/openvt -e    -f -c1 -- /bin/vtpool
c2:1235:respawn:/usr/bin/openvt -e -l -f -c2 -- /bin/bash -l
c3:1235:respawn:/usr/bin/openvt -e -l -f -c3 -- /bin/bash -l

#-----------------------------
# /bin/vtpool :
#-----------------------------

#!/bin/bash
until false # boucle infinie
do
  /bin/stty sane
  /bin/echo -e "\\033[2J\\033[f\\033[0;32mAppuyez \
sur la touche ENTREE pour démarrer un autre terminal\
\\033[1;38m"
  read # attend l'appui sur ENTREE
  /usr/bin/deallocvt
  export VTSTACK=1 # revenir sur tty1 à la fin
  /usr/bin/openvt -s -l -- /bin/bash -l
done

#-----------------------------
# Modification de /etc/rc.d/rc.local
#-----------------------------

chvt 2 # bascule vers tty2
# chvt 1 puis y reste
```

Le seul détail important est le fait que la boucle infinie est confiée à `bash`, car `/sbin/init` provoquait un comportement étrange que je n'ai pas réussi à expliquer. Le reste se passe de commentaire, puisque c'est l'application ou la modification de tout ce qui a été traité jusqu'ici.

L'encombrement de la mémoire est encore réduit. Par exemple, le script `vtpool` n'est pas lancé en mode login et ne source pas les nombreux fichiers de configuration, ce qui non seulement accélère le démarrage, mais, en plus, économise de la mémoire (car cela évite de conserver de nombreux symboles et définitions inutiles).

Conclusion

Le souci avec les configurations d'`/etc/inittab` classiques est que le nombre de *vt* est fixé à 6, qu'on en ait besoin de plus ou moins. Certains live-CD (ou d'autres systèmes embarqués à base de `busybox`) utilisent d'autres solutions pour n'activer les terminaux que lorsque l'utilisateur en a besoin. Nous venons de voir un résumé de certaines de ces solutions, et nous les avons appliquées pour agrémenter la configuration d'un système de développement isolé.

Une solution encore plus radicale serait d'utiliser directement l'utilitaire `screen` (voir GLMF n°93, page 14 ou `man screen` s'il est installé). Ce qui me retient pour l'instant est le manque de familiarité avec cet utilitaire qui remplace presque tout ce qui a été décrit ici.

Il est peu probable qu'un employeur potentiel lisant cet article soit très motivé pour m'embaucher en tant qu'administrateur système, mais tel n'était pas le but.

D'abord, l'objectif est de réussir à configurer une machine sous Linux (ou autre OS plus ou moins «unixidé») pour obtenir un confort d'utilisation optimal dans certaines circonstances. Certaines personnes passent leur temps à dessiner des icônes pour leur bureau graphique, moi je suis content quand le système est tellement simplifié qu'il démarre en moins de quinze secondes. Virer un `depmod -a` par-ci, paralléliser un script par-là...

Ensuite, on ne peut faire cela correctement que lorsqu'on connaît les *pourquoi* et *comment* de chaque élément de son ordinateur. Les distributions modernes (tous types confondus) ont fait un travail d'intégration tellement fantastique que l'utilisateur en oublie qu'il utilise un dérivé d'UNIX, jusqu'à en ignorer les principes de base. Puisqu'on apprend en faisant des erreurs, cet article (avec tous ses dangers) est donc un bon support pédagogique :-)

Enfin, connaître le fonctionnement d'un système permet de le recréer ou de l'améliorer. Par exemple, la méta-distribution *Linux From Scratch* (LFS) permet à chacun d'expérimenter en toute liberté, donc de répondre à nos besoins personnels (moyennant beaucoup, beaucoup de travail). J'espère que les principes survolés dans cet article ont stimulé votre imagination et votre curiosité.

Yann Guidon,

whygee@f-cpu.org
Électronique, musique et informatique en folie^Wliberté
http://ygdes.com

Guillaume Tamboise

Sauvegarde et monitoring : le shell et votre meilleur atout

De manière somme toute ironique, on assiste à la convergence de deux infrastructures logicielles qui étaient profondément différentes il y a une dizaine d'années. Le monde de Windows privilégiait alors les solutions « tout compris », où l'utilisateur n'était invité qu'à prendre son mulot à deux mains et à cliquer où il fallait. Le monde du Logiciel libre était très occupé à construire ses briques fondatrices, et l'utilisateur était convié à apprendre rapidement ce qu'une bibliothèque partagée ou un Makefile pouvait bien être. Certains acteurs du Libre s'attachent aujourd'hui à montrer l'applicabilité de leur modèle à des solutions de bout en bout... tandis que Microsoft renouvelle ses efforts pour offrir aux administrateurs système un langage d'automatisation scripté, moderne et intégré.

Les efforts de développement de Microsoft ont déjà connu plusieurs aboutissements majeurs en 2007. Certains portent sur l'acquisition de la puissance de gestion dont Unix ou Linux bénéficie depuis un certain temps via ses langages de script. Les essais successifs du géant de Redmond, du COMMAND.COM au WSH, connaissaient la même limitation majeure de ne reproduire que de manière incomplète et peu intégrée les possibilités de l'interface graphique. Pour pallier ces insuffisances, Microsoft a sorti de ses lignes d'assemblage Windows *PowerShell*. On trouve ce *shell* chez Windows Vista et XP, mais aussi en invité vedette de la suite de messagerie Exchange 2007.

Les spécifications annoncées de PowerShell sont d'ores et déjà alléchantes **[1]**. Outre le support natif des tables de *hash* et des expressions régulières, on remarquera le passage d'objets à travers les *tuyaux* (*pipes*), là où les shells traditionnels Unix se contentent de faire passer des flux de caractères. Wikipedia donne en exemple la possibilité d'arrêter les processus qui prennent plus de 1000 mégaoctets de mémoire :

```
PS> get-process | where { $_.WS -gt 1000MB } | stop-process
```

L'écriture de la commande est certainement élégante. `get-process` semble retourner une liste d'objets représentant les processus en cours. Le pipe (`|`) envoie ce résultat à la commande suivante, `where`, qui construit une boucle pour travailler sur ces objets un par un. La méthode `WS` retourne pour chaque objet sa taille (on retrouve avec le point (`.`) une notation familière du monde des objets), qui est comparée à 1000 mégaoctets. Seuls les processus-objets dont la taille est supérieure à 1000 mégaoctets passent à travers le pipe suivant, qui se charge d'arrêter leur exécution.

Comme de coutume, l'intérêt réel de ce nouvel outil se mesurera à l'usage qu'en feront ses utilisateurs. Le mieux que l'on puisse souhaiter à Microsoft et à ses utilisateurs serait qu'à terme les administrateurs de systèmes Windows parviennent à assembler en Powershell les briques nécessaires à leurs tâches quotidiennes, en provenance de divers fournisseurs – propriétaires et libres.

Nous verrons dans cet article la façon dont se sont assemblés quelques outils libres, en réponse à deux problématiques intemporelles : la sauvegarde de données et l'analyse de journaux.

Solution de sauvegarde

Une politique de gestion des risques peut se construire de manières différentes, mais comporte sensiblement toujours plus ou moins les mêmes ingrédients. On essaie de protéger des actifs, qui sont ici de l'information stockée dans des systèmes informatiques. Un certain nombre de risques existent, qui mettent en danger ces actifs. On s'intéresse ici en particulier à la perte d'intégrité, d'accessibilité, si ce n'est à la destruction de l'information. Un *risque* peut rester extrêmement théorique, et ne devient une *menace* que quand il est conjugué à une vulnérabilité, exploitée par un vecteur d'attaque.

Chaque menace a un coût associé à sa réalisation, et une probabilité d'occurrence. Le produit de ces deux quantités donne l'espérance mathématique de perte causée par cette menace, c'est ce que l'on appelle « risque » (financier). Cette donnée est généralement annualisée, c'est-à-dire qu'elle représente le coût annuel associé à l'existence de la menace contre l'actif considéré. Chaque année, il faut s'attendre à perdre un capital d'ordre de grandeur de ce risque. Le risque peut être traité de plusieurs manières. On peut l'éviter – cette particulière activité représente un risque trop élevé, la décision est prise de ne pas s'y engager. Le risque peut être transféré, typiquement en prenant une assurance. On peut accepter le risque, ce que l'on fait par exemple sur le montant de la franchise d'une assurance. Le risque peut aussi être réduit, à l'aide d'une contre-mesure. Le budget de la contre-mesure est évalué au regard du risque (financier), pondéré par l'efficacité de la contre-mesure.

Cette approche de gestion de la sécurité par la gestion du risque est résumée dans un article du journal de l'ISSA d'Août 2006, « *Nimble Risk Management* » **[2]** par Randy S. Lindberg, disponible sur le site de l'ISSA **[3]** pour ses membres.

Le choix de la gestion de la sécurité par le risque est parfois contesté, principalement à cause de la difficulté d'estimer autant le coût d'un incident parfois théorique que sa probabilité. En effet, l'industrie ne dispose que rarement de statistiques fiables concernant ces valeurs. Donn B. Parker explicite ce reproche dans un article du journal de l'ISSA de mai 2006 « *Making the Case for Replacing Risk-Based Security* » **[4]**. Il donne trois fils directeurs dans la gestion de la sécurité :

- *due diligence* où l'on assoit une solution de sécurité dans le contexte des pratiques recommandées et pratiquées par les autres experts de l'industrie ;
- *compliance* qui tient en compte le cadre légal ;
- *enablement* où la sécurité est vue comme une infrastructure sur laquelle reposent de futurs développements.

Dans la suite, on va s'intéresser à une menace en particulier contre l'information, celle de la corruption. Elle peut être d'origine matérielle, intentionnelle, accidentelle... Une contre-mesure très efficace est bien connue : la sauvegarde. Quelle que soit l'approche prise, par gestion du risque ou pas, la conclusion est la même : tout système d'information se doit de suivre une politique claire et exhaustive de gestion des sauvegardes.

Une fois n'est pas coutume, le monde Unix s'est très tôt accommodé de solutions de sauvegarde fondées sur des scripts shell. On retrouve régulièrement les mêmes briques :

- utilisation de liens en dur afin de conserver plusieurs sauvegardes successives, mais en évitant de dupliquer les données qui ne changent pas d'une sauvegarde sur l'autre ;
- utilisation de l'authentification par clef publique et du chiffrement fournis par `ssh` ;
- mise en œuvre du protocole `rsync` pour n'envoyer sur le réseau que la différence d'une sauvegarde à l'autre ;
- utilisation de scripts ad hoc pour gérer l'export à chaud des bases de données ;
- utilisation de l'utilitaire `cron` pour le déclenchement périodique des sauvegardes.

Il existe déjà un logiciel – un script écrit en langage Perl – qui s'attache à combiner ces outils : `rsnapshot` (**http://www.rsnapshot.org/**).

Gestion des permissions

Une question qu'il convient de se poser assez tôt dans le cadre du développement d'une solution de sauvegarde est celle de l'utilisateur qui va effectuer la sauvegarde. De toute évidence, l'utilisateur Linux choisi doit avoir accès en lecture à toutes les données à sauvegarder. De là à choisir *root* pour cet utilisateur, il n'y a qu'un pas qu'il est aisé de franchir. Le souci est qu'alors la fonction « sauvegarde » a de facto un accès illimité à la machine sauvegardée.

On peut parfois considérer qu'un accès en lecture aux données présentes sur une machine n'est pas conceptuellement très différent d'un accès administratif à la machine. C'est, par exemple, le cas quand le capital à protéger est la confidentialité de données résident en texte clair dans le système de fichiers. L'administrateur peut prendre connaissance de toutes ces informations, tout comme l'utilisateur qui incarne la fonction « sauvegardes ». L'intégrité et l'accès aux données, elles, devraient a priori être beaucoup plus facilement mises à mal par l'administrateur que par l'utilisateur « sauvegardes ».

D'un point de vue opérationnel, l'accès total en lecture et l'accès administrateur ont des différences significatives. D'une part, il existe généralement des règles très précises sur l'utilisation de l'accès administrateur. Ces règles peuvent être moins explicites pour l'utilisation de l'accès « sauvegardes », facilitant le masquage d'identité et potentiellement l'abus de permissions. D'autre part, l'accès en lecture seule de donnés chiffrées ne fournit pas nécessairement les clefs du château – on peut prendre l'exemple des mots de passe chiffrés en MD5 *salé* dans `/etc/shadow`. Il est possible de casser les mots de passe des utilisateurs, mais seulement au prix d'efforts non négligeables. Un accès administrateur, lui, rendra possible l'installation d'un cheval de Troie qui permettra d'apprendre ces mots de passes dès qu'un utilisateur le soumettra au système.

Root ou pas root, telle est donc la question.

Dans le cadre des listes d'accès « traditionnelles » (on dira « minimales » dans la suite), définir les accès en lecture universelle à un utilisateur qui n'est pas root tourne rapidement au cauchemar. *Utilisateur, groupe, reste du monde, lecture, écriture, exécution, bit SetUID, bit SetGID, sticky bit, masques...* on fait assez rapidement le tour du modèle de sécurité avant de se retrouver au point de départ.

Les listes d'accès étendues peuvent être une solution élégante pour ces deux problèmes, et, en particulier, pour éviter de donner l'accès administrateur (root) au logiciel de sauvegarde.

Cette fonctionnalité s'obtient sur un système Debian en installant d'abord le paquet `acl`, puis en configurant le système de fichier de manière adéquate si nécessaire. Les utilisateurs du système de fichiers XFS se féliciteront une fois de plus de leur choix. XFS vient avec les listes d'accès étendues POSIX par défaut, pourvu que le noyau Linux ait été compilé avec l'option `CONFIG_XFS_POSIX_ACL=y` – ce qui est le cas chez Debian.

Les utilisateurs de Ext2/Ext3 se fendront d'une manipulation rapide. Il s'agira d'abord de vérifier les options de montage de la partition en question :

```
greg@hote:~$ mount
[...]
/dev/sda8 on /home type ext3 (rw)
[...]
```

Ensuite, un essai gratuit d'utilisation des listes d'accès étendu, tout en sachant que la commande se doit d'échouer :

```
greg@hote:~$ setfacl -m u:iza:rwx truc.txt
setfacl: truc.txt: Operation not supported
```

On remonte alors la partition à la volée, avec le support des listes d'accès étendues :

```
greg@hote:~$ sudo mount -o remount,acl /dev/sda8
```

puis on vérifie le résultat :

```
greg@hote:~$ mount
[...]
/dev/sda8 on /home type ext3 (rw,acl)
[...]
```

La commande de gestion des listes d'accès est maintenant censée marcher :

```
greg@hote:~$ setfacl -m u:iza:rwx truc.txt
greg@hote:~$ getfacl truc.txt
# file: truc.txt
# owner: greg
# group: greg
user::rw-
user:iza:rwx
group::r--
mask::rwx
other::r--
```

Quant aux listes d'accès minimales correspondantes, elles restent intuitives, avec un petit signe « + » pour indiquer qu'une liste de contrôle d'accès peut en cacher une autre :

```
greg@hote:~$ ls -al truc.txt
-rw-rwxr--+ 1 greg greg 6170 2006-05-13 07:53 truc.txt
```

Les aficionados de GUI pourront jeter un coup d'œil à Eiciel, qui s'intègre élégamment avec le gestionnaire de fichiers Nautilus. Eiciel intervient à l'affichage des propriétés d'un fichier ou d'un répertoire.

propageront aux sous-répertoires à leur création, ce qui est bien pratique.

On vient de voir ce qui marche bien, voyons maintenant les points sensibles, en gradation croissante – ou, pourquoi est-ce que la majorité des lecteurs vont y réfléchir à deux fois avant de déployer des listes d'accès étendues.

D'abord, le logiciel d'archivage Tar ne sauvegarde pas les listes d'accès étendues. Il faut passer à Star pour cela.

Ensuite, certains fichiers ont des droits très restrictifs et ne sont lisibles que par leur propriétaire. Pour conserver la compatibilité avec les droits non étendus, voici comment les listes étendues agissent :

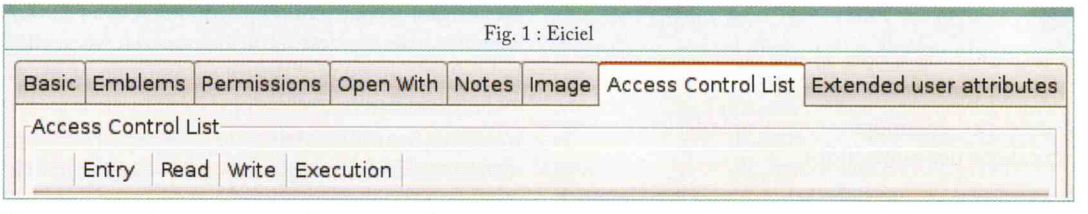
```
greg@hote:/var/tmp$ touch test
greg@hote:/var/tmp$ chmod 600 test
greg@hote:/var/tmp$ ls -al test
-rw------- 1 greg famille 0 2006-08-12 11:28 test
greg@hote:/var/tmp$ setfacl -m u:mythtv:r test
greg@hote:/var/tmp$ getfacl test
# file: test
# owner: greg
# group: famille
user::rw-
user:mythtv:r--
group::---
mask::r--
other::---
greg@hote:/var/tmp$ ls -al test
-rw-r-----+ 1 greg famille 0 2006-08-12 11:28 test
```

On en vient à se demander si le groupe famille a ou non accès en lecture au fichier `test`.

Prenons un utilisateur membre de ce groupe :

```
iza@hote:/var/tmp$ id iza
uid=1001(iza) gid=1003(famille) groups=1003(famille),4
(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(di
p),40(src),44(video),46(plugdev),1000(greg),104(lpadmi
n),105(scanner),106(admin)
iza@hote:/var/tmp$ cat test
cat: test: Permission denied
```

Fig. 1 : Eiciel



En revenant à nos sauvegardes : pour donner à l'utilisateur *backup* accès en lecture et en exécution par défaut aux fichiers créés dans le répertoire `dir`, on utilise la commande :

```
$ sudo setfacl -m d:u:backup:rx dir
```

Si l'utilisateur backup n'existe pas, il faut s'attendre à une réponse du style :

```
setfacl: Option -m: Invalid argument near character 5
```

Le drapeau `-R` permet d'assigner des permissions a posteriori à l'ensemble des sous-répertoires. Il est à noter que les listes par défaut pour un répertoire se

Alors, seulement cosmétique, ce `r` pour les accès du groupe ? Pas vraiment...

Par exemple : certains logiciels font des vérifications sur les droits en lecture sur leurs fichiers de configuration, et ne fonctionnent pas s'ils sont trop permissifs. Maildrop et son fichier de configuration `~/.mailfilter` est l'un d'entre eux. Il suffit de donner un accès en lecture au fichier de configuration `~/.mailfilter` à quelqu'un d'autre que son propriétaire, et la liste d'accès minimale affichera un `r` pour le groupe (même si le groupe n'a pas accès en lecture, comme précisé par la liste d'accès étendue). Le résultat ?

Abonnez-vous !

11 Numéros de Linux Magazine

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

Les 3 bonnes raisons de vous abonner !

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

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

Bon d'abonnement à remplir et à retourner à (ou photocopie)

Linux Magazine - BP 20142 - 67603 Selestat Cedex

11 Numéros de Linux Magazine à 53 € Offre France Métro

Soit Votre Linux Magazine à 4,82 €

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

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

Les 4 façons de vous abonner !

» Par courrier postal en nous renvoyant le bon ci-dessous.

» Par le Web, sur notre site : www.ed-diamond.com.

» Par téléphone (paiement C.B.) entre 9h-12h & 15h -18h au 03 88 58 02 08.

» Par Fax au 03 88 58 02 09 C.B. et/ou bon de commande administratif

◎ OUI, je souhaite m'abonner à Linux Magazine pour 11 numéros

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

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

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Offres collectionneurs

Les anciens numéros !

Toujours disponibles !

Les 4 façons de commander !

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

Bon d'abonnement à remplir et à retourner à (ou photocopie)

Linux Magazine - BP 20142 - 67603 Selestat Cedex

Bon de commande Linux Magazine

Référence	Prix / N°s	Qté.	Total
Linux Magazine 84 Déploiement de hotspots Wifi sécurisés	5,95 €		
Linux Magazine 85 Firewall: Netfilter & NuFW	6,20 €		
Linux Magazine 86 Serveur SMTP: Routage des mails avec Postfix	6,20 €		
Linux Magazine 87 Le point sur Mono .NET Java et les Brevets	6,20 €		
Linux Magazine 88 Sécurité: Smartcards & Tokens	6,20 €		
Linux Magazine 89 Utilisation avancée de XEN	6,20 €		
Linux Magazine 90 ASTERISK Le serveur de téléphonie IP	6,20 €		
Linux Magazine 91 AJAX AVANCÉ Principe, fonctionnement et pièges	6,20 €		
Linux Magazine 92 Paravirtualisation XEN & SLO Répartition de charge	6,20 €		
Linux Magazine 93 Développez vos extensions Firefox	6,20 €		

Bon de commande Linux Magazine Hors Série

LM HS 13 Firewall votre meilleur ennemi Acte 2	5,95 €		
LM HS 15 GIMP et la Photo	5,95 €		
LM HS 16 Kernel (1)	5,95 €		
LM HS 17 Kernel (2)	5,95 €		
LM HS 18 Haute Disponibilité	5,95 €		
LM HS 19 The Gimp 2.0	5,95 €		
LM HS 20 PHP 5	5,95 €		
LM HS 21 Recyclez vos PC	6,40 €		
LM HS 22 GIMP et le Web	6,40 €		
LM HS 23 Linux et electronique	6,40 €		
LM HS 24 Linux Embarqué	6,40 €		
LM HS 25 Linux Embarqué 2	6,40 €		
LM HS 26 Spécial The GIMP	6,40 €		
LM HS 27 Électronique et Linux	6,40 €		
LM HS 28 Administration systeme avec Debian	6,40 €		
LM HS 29 Spécial BSD Acte 1	6,40 €		
		sous TOTAL	
		Frais de port France Metro	+ 3,81 €
		Frais de port Etranger	+ 5,34 €
		TOTAL	

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

GNU LINUX MAGAZINE / FRANCE

2 Je joins mon réglement :

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

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

```
Jul 21 15:30:01 hote maildrop[16846]: Cannot have
world/group permissions on the filter file - for your
own good.
Jul 21 15:30:01 hote sm-mta[5483]: k7CGcDVf020018:
to="|/usr/bin/maildrop -d greg", ctladdr=<utilisateu
r@hote.com> (8/0), delay=00:00:03, xdelay=00:00:00,
mailer=prog, pri=31223, dsn=4.0.0, stat=Deferred: prog
mailer (/bin/sh) exited with EX_TEMPFAIL
```

Les potentiels fichiers à problème peuvent être pistés en recherchant ceux qui n'ont pas de droit en écriture pour le groupe ou le reste du monde :

```
$ sudo find / ! -perm +044 -ls
```

On peut réécrire cette commande de manière plus POSIX avec un :

```
$ sudo find / ! -perm /o+r,g+r -ls
```

mais, il faut alors une version assez récente de Find.

Les listes d'accès étendues POSIX apportent donc une solution élégante aux problèmes de partage de fichiers, en particulier dans le cadre d'une solution de sauvegarde. Cependant, cela se fait au prix d'une complexité accrue qu'il convient d'évaluer avec soin.

Liens en dur

Les *inodes* définissent les fichiers sous les systèmes de fichiers Unix (Ext2, Ext3, XFS, ReiserFS...). Une inode est une structure contenant le propriétaire, le type (régulier, répertoire, caractères, spécial, tube), les droits d'accès (minimaux en tous cas, pour les droits d'accès étendus cela dépend du système de fichiers), les dates d'accès, de modification et de création, le compteur de référence, la taille et la localisation des données. Par exemple, le système manipule un fichier en utilisant son inode et non le nom symbolique attribué par l'utilisateur. L'information du nom symbolique se trouve d'ailleurs au niveau du répertoire et non au niveau de l'inode. Il est alors tout à fait possible d'avoir deux entrées dans deux répertoires différents pointant vers la même inode.

On commence par créer un fichier :

```
$ touch hello
```

On crée ensuite un lien en dur :

```
$ ln hello bonjour
```

Vérifions les numéros d'inode :

```
$ ls -ali hello bonjour
1012346985 -rw-r--r-- 2 greg famille 0 2006-08-12
20:00 bonjour
1012346985 -rw-r--r-- 2 greg famille 0 2006-08-12
20:00 hello
```

Les deux entrées de répertoire pointent vers la même inode. Toute modification de l'un des « fichiers » sera alors répercutée sur l'autre, comme les deux entrées pointent vers la même inode (qui pointe vers une zone de donnée unique). Par exemple, on peut modifier les informations de permission dans l'inode qu'ils ont en commun :

```
$ chmod 777 hello
$ ls -ali hello bonjour
1012346985 -rwxrwxrwx 2 greg famille 6 2006-08-12
20:00 bonjour
1012346985 -rwxrwxrwx 2 greg famille 6 2006-08-12
20:00 hello
```

ou bien la zone de texte vers laquelle l'inode pointe :

```
$ echo "texte" >> bonjour
$ cat bonjour
texte
$ cat hello
texte
```

rsync

`rsync` est un algorithme qui permet une synchronisation distante (d'où son nom) d'un ensemble de fichiers. Sa mise en œuvre du même nom est généralement disponible parmi les paquets de base des distributions Linux.

`rsync` peut être utilisé via le démon `rsync`, qui écoute dans ce cas le port 873. Ce mode de fonctionnement est relativement répandu pour la distribution anonyme de fichiers publics. Par exemple, le noyau Linux est accessible à l'URI **rsync://rsync.kernel.org/pub/**.

Pour récupérer la liste des fichiers disponibles, il suffit d'exécuter comme vous y invite la première page de **http://www.kernel.org/** :

```
$ rsync rsync://rsync.kernel.org/pub/
```

Une transmission `rsync` peut aussi être envoyée dans un canal SSH. L'inconvénient majeur de cette mise en œuvre dans le cadre d'une solution de sauvegarde est qu'elle force à fournir un accès shell sur la machine à sauvegarder. On peut s'en satisfaire conceptuellement en considérant que `rsync` fait plus que transmettre des données, il doit aussi calculer des différences et n'envoyer que les « bons » blocs.

En principe, `rsync` ne fait en effet transiter par le réseau que les blocs de données de fichiers qui ont changé entre deux sauvegardes. Ce comportement tel qu'il est paramétré par défaut est mis à mal pour les fichiers dont le nom change : le fichier avant renommage est effacé, tandis que le nouveau fichier est transféré complètement. Cela peut s'avérer coûteux en bande passante pour les fichiers de journaux de services accédés fréquemment, et en particulier pour les serveurs web (par exemple Apache) ou serveurs de proximité (par exemple Squid), dans la configuration Debian par défaut. En effet, leurs *logs* sont gérés par `logrotate`, qui organise une rotation des fichiers semblable à ce que fait `syslog`. À chaque période de rotation, le fichier de logs d'accès standard d'Apache `access.log` devient `access.log.1`. Lui-même est compressé en `access.log.2.gz`, puis renommé `access.log.3.gz` etc. **[5]**. `rsync` ne remarquera pas que les fichiers ne se font que renommer, et aura tendance à les transférer complètement à chaque fois. Relativement ennuyeux quand le volume des logs atteint quelques dizaines

de mégaoctets par jour, et quand un grand nombre de ces fichiers est gardé en ligne.

L'une des méthodes est de changer la gestion des journaux, et configurer Apache pour qu'il utilise son programme `rotatelogs` (ou `rotatelogs2` pour Apache 2 chez Debian). Les fichiers de logs se retrouvent alors avec un nom qui dépend de leur date de création, et qui ne change pas une fois que le fichier a été créé. Le deuxième avantage de cette approche est qu'une sauvegarde qui se sert uniquement de liens en durs pour gérer les fichiers qui ne changent pas d'une version de sauvegarde à l'autre n'allouera qu'une fois l'espace disque nécessaire à l'archivage d'un de ces fichiers. En effet, si le nom ne change pas, alors l'entrée de répertoire peut continuer à pointer vers la même inode d'une sauvegarde sur l'autre, en utilisant les mécanismes usuels de gestion des systèmes de fichiers. Il faudrait rajouter une intelligence supplémentaire pour gérer les changements de nom de fichier entre deux sauvegardes.

Une autre méthode est de préciser l'option `--fuzzy` (ou `-y`), pour que `rsync` s'aide de fichiers aux noms similaires ou de taille et de date de modification identiques. Il faut alors prendre la précaution de préciser que les fichiers qui ont disparu de la source de la sauvegarde doivent être enlevés de la destination à la fin du processus de transfert, et non pas au début. On imagine que, par défaut, `rsync` commence par supprimer les fichiers qui ont disparu de la source pour réduire la demande totale en volume disque pendant l'opération de sauvegarde. Exemple choisi :

```
web$ ls -lh
total 12M
-rw-r-----  1 root root 9.7M Mar 13 01:14 access.log
-rw-r-----  1 root root 2.2M Mar 13 01:14 access.
log.25.gz

sauvegardes$ time rsync -v --delete-after --fuzzy -a
-z --rsh=ssh root@web:/var/tmp/logs ./
[Bannière d'avertissement SSH du serveur web]

receiving file list ... done
logs/
logs/access.log
logs/access.log.25.gz

sent 70 bytes  received 2997791 bytes  153736.46
bytes/sec
total size is 12406722  speedup is 4.14

real    0m17.814s
user    0m0.308s
sys     0m0.080s
```

Le transfert s'est effectué de manière complète en 17 secondes, étant donné que `rsync` n'avait aucun fichier local sur lequel se fonder pour accélérer son transfert.

Après avoir renommé `access.log` en `access.log.1`, le transfert se fait en 2,033 secondes – avec un *speedup* de 641,21. `rsync` a remarqué que le fichier `access.log` était très proche du fichier `access.log.1` qu'il cherchait à transférer.

Sans le `--fuzzy`, le même transfert se fait en 36 secondes, avec un speedup de 16,74 – `access.log` se fait alors transférer complètement.

Renommer le fichier `access.log.1` et lui rajouter une ligne vide à la fin donne aussi d'excellents résultats grâce au paramètre `--fuzzy` : speedup de 638.85 et sauvegarde en 2,062 secondes.

On rassemble les pièces : rsnapshot

Le logiciel `rsnapshot` est un script écrit en langage Perl, long de 6 650 lignes dans sa version 1.2.9 – dont 1 342 lignes purement de commentaires. On s'en rend compte via cette commande shell :

```
$ egrep '^[[:space:]]*#' /usr/bin/rsnapshot | wc -l
1342
```

Le `egrep` filtre les lignes qui commencent (`^`) par des caractères d'espacement (`[[:space:]]`) ou pas (`*`), suivis d'un dièse (`#`). Cette recherche porte sur le fichier `/usr/bin/rsnapshot`, et envoie la sortie sur le canal standard. `wc` récupère ce flux par un pipe (`|`), et en compte le nombre de lignes (`-l`).

D'autres commentaires sont parsemés parmi les lignes de codes elles-mêmes.

`rsnapshot` est disponible en paquet Debian, Ubuntu et consorts. Le `r` dans son nom indique qu'il peut faire des sauvegardes distantes, étant entendu qu'il est aussi très adroit en utilisation locale. Dans le cadre d'une utilisation à distance, il vaut mieux lui adjoindre le couple `rsync` et `ssh` dont nous avons déjà exposé les vertus. Point de surprise quant aux dépendances du paquet Debian : `rsnapshot` dépend de `logrotate` (pour gérer la rotation de son fichier de logs, `/var/log/rsnapshot.log`), Perl et `rsync`, et recommande `ssh`.

Le fichier de configuration de `rsnapshot` s'appelle `/etc/rsnapshot.conf` – étant entendu qu'on peut en créer d'autres.

Avant de prétendre utiliser `rsnapshot` pour des sauvegardes distantes, il convient d'assurer une relation de confiance SSH entre clients et serveurs de sauvegarde. On commencera par créer une paire de clefs SSH par la commande `ssh-keygen -t dsa` sur le serveur de sauvegarde. La partie publique, `id_dsa.pub`, sera placée dans le fichier `~/.ssh/authorized_keys` de l'utilisateur de sauvegarde sur la machine à sauvegarder. Plusieurs restrictions sont possibles pour limiter l'accès que cette clef fournit – d'autant qu'il est assez vraisemblable qu'on ne la protégera pas par mot de passe, pour automatiser les sauvegardes. On peut assez facilement interdire à l'utilisateur de sauvegarde l'acheminement de ports, de connexions X11 et de l'agent d'authentification SSH ainsi que filtrer les adresses IP depuis lesquelles les sauvegardes sont lancées. Un tel fichier `authorized_keys` ressemblerait à ceci, sur une seule ligne :

```
no-port-forwarding,no-X11-forwarding,no-agent-
forwarding,no-pty,from="[Adresse IP]" [contenu de
id_dsa.pub]
```

Le fichier `authorized_keys` peut aussi restreindre les commandes qui peuvent être envoyées via ce canal. Hélas, restreindre les paramètres desdites commandes complique singulièrement la configuration.

Si l'on choisit de définir de manière statique les commandes qui peuvent être passés, toute modification de la liste des répertoires sauvegardés forcera une mise à jour de `authorized_keys`. Cette solution demande le plus de maintenance, mais est aussi la plus sûre.

Si l'on choisit d'autoriser l'exécution de toute commande `rsync`, il faut *scripter* le filtrage des commandes afin d'autoriser `rsync` avec n'importe quel paramètre. Tout en s'assurant que les paramètres de `rsync` ne soient pas plus que des paramètres de `rsync`, et ne servent pas à injecter des commandes supplémentaires. `authorized_keys` pourrait alors avoir cette allure :

```
command="/usr/local/bin/validate-rsync.sh",no-port-
forwarding,no-X11-forwarding,no-agent-forwarding,no-
pty,from="[Adresse IP]" [contenu de id_dsa.pub]
```

`validate-rsync.sh` pourrait alors être construit comme suit :

```
#!/bin/sh
# Filtrage rapide de la commande envoyée par ssh
SSH_COMMAND=`echo "$SSH_ORIGINAL_COMMAND" | grep
'^rsync ' | grep -v -E ';|&'`
if [ -n "$SSH_COMMAND" ]
then
$SSH_ORIGINAL_COMMAND
else
echo "Rejected"
fi
```

La restriction sur `SSH_COMMAND` est nécessaire pour éviter que des commandes autres que `rsync` ne soient passées en argument. Cependant, la logique d'interdire les caractères « ; », « | » et « & » n'est pas réellement satisfaisante. D'une part, parce qu'elle suppose que d'autres caractères ne permettront pas d'abuser de cet accès SSH. D'autre part, parce que certains noms de fichiers à sauvegarder deviennent interdits.

`rsnapshot` est appelé à intervalles réguliers par `cron`, via le fichier `/etc/cron.d/rsnapshot`. On remarquera que `rsnapshot` est appelé avec un paramètre `hourly` (horaire), `daily` (journalier), `weekly` (hebdomadaire) ou `monthly` (mensuel). On retrouve ces paramètres dans le fichier `rsnapshot.conf`, associé au nombre de versions qu'on veut garder de chaque type de sauvegarde. À ce sujet, il faut bien remarquer que chaque type de sauvegarde est largement indépendant des autres. Si la sauvegarde hebdomadaire est programmée le lundi à 6:30 et la sauvegarde mensuelle le premier jour du mois à 6:31, alors les sauvegardes mensuelles ont toutes les chances d'échouer les mois dont le premier jour est un lundi. L'approche la plus sûre est certainement de commencer avec les défauts proposés par le paquet Debian. Pour une gestion plus fine du calendrier de sauvegarde, on pourra se référer à la page de manuel de `crontab` :

```
$ man 5 crontab
```

Le fichier de configuration `rsnapshot.conf` présente le même danger qu'un `makefile` : les tabulations sont loin d'y être innocentes. Chez `rsnapshot`, une tabulation sert à séparer les paramètres.

Au chapitre des pièges à éviter, les options longues de `rsync` ne devraient aussi être modifiées qu'avec beaucoup d'attention (paramètre `rsync_long_args`). Ce sont en effet les options longues par défaut qui se chargent des exclusions définies par le paramètre `exclude`.

Enfin, en tout état de cause, le paramètre `-t` (test) de `rsnapshot` permet de savoir rapidement si un changement de configuration aura les effets escomptés.

Une fonctionnalité des plus intéressantes d'un logiciel de sauvegarde concerne les exclusions. Rien de plus décourageant que de voir un espace de sauvegarde se remplir de gigaoctets de données temporaires et inutiles. On peut citer le répertoire `/var/tmp`, les vignettes de Gnome dans `.thumbnails`, l'index `.beagle` de Beagle ou encore les fichiers IMAP `INBOX.sbd` de Thunderbird (si tant est qu'on a par ailleurs une sauvegarde du serveur IMAP). Pour des raisons de sécurité, on pourra aussi parfaitement imaginer d'exclure les répertoires `.ssh` et `.gnupg` – sauvegardes et *key escrow* sont deux choses différentes. La manière pédestre d'exclure ces fichiers et répertoires serait de rajouter dans le `/etc/rsnapshot.conf` (en se rappelant que les paramètres doivent être séparés par des tabulations) :

```
exclude [TABULATION] /var/tmp
exclude [TABULATION] .thumbnails
exclude [TABULATION] .beagle
exclude [TABULATION] INBOX.sdb
exclude [TABULATION] .ssh
exclude [TABULATION] .gnupg
```

On peut aussi prendre la convention d'exclure de la sauvegarde les répertoires qui s'appelleraient par exemple `NePasSauvegarder`. Cette spécificité doit alors être communiquée aux utilisateurs, pour qu'ils en fassent (bon) emploi.

```
exclude [TABULATION] NePasSauvegarder
```

Enfin, le paramètre `backup_script` prend en charge les sauvegardes de bases de données. En reprenant l'exemple de la page de manuel, pour peu que le script `/usr/local/bin/backup_mysql.sh` exporte les bases intéressantes dans le répertoire courant :

```
backup_script [TABULATION] /usr/local/bin/backup_
mysql.sh [TABULATION] mysql_backup/
```

Pour les bases de données distantes, plusieurs approches sont possibles. L'une d'entre elles est d'utiliser le client de la base de données depuis le serveur de sauvegardes, et de spécifier que le gestionnaire de base de données est distant tout en prenant le soin de protéger la connexion avec SSL – y compris et surtout en vérifiant le certificat SSL présenté par le serveur. Dans le monde MySQL, ceci se fait avec les paramètres `--host=` et `–ssl*`.

Une autre option peut être plus adaptée si l'on n'a pas encore eu de justification de faire écouter le serveur de bases de données sur une interface IP physique : utiliser l'accès SSH – et prendre soin de modifier `validate-rsync.sh` en conséquence si besoin est.

Si l'utilisateur chargé des sauvegardes de base de données est root, le script `backup_mysql.sh` peut ressembler à ceci :

```
#!/bin/sh
# On demande la liste des bases de données
DB=`echo "show databases" | \
# en utilisant l'accès de l'utilisateur de maintenance Debian
    ssh root@hote mysql --defaults-file=/etc/mysql/debian.cnf mysql | \
# et en enlevant Database de la liste
    grep -v '^Database$'`
for database in $DB
do
# On lance via ssh l'export de la base de données
ssh root@hote "mysqldump --defaults-file=/etc/mysql/debian.cnf \
# qu'on archive localement (sur le serveur de sauvegarde)
                --compress --opt $database" > ${database}.sql
done
```

Si la sauvegarde de la base de données ne se fait pas sous l'utilisateur root, il suffit de rajouter dans le répertoire `$HOME` de l'utilisateur de sauvegarde un fichier `.my.cnf` avec des droits d'accès restreints :

```
[client]
user            = utilisateur_mysql
password        = mot_de_passe
```

`utilisateur_mysql` doit avoir accès en lecture (`SELECT`) à toutes les bases de données qu'il sauvegarde. Cette méthode a pour avantage d'éviter de lancer une commande en root, et pour inconvénient de rajouter un utilisateur et un mot de passe à gérer.

Le paramètre `backup_script` prend en charge la création d'un répertoire temporaire dans `snapshot_root`, puis son déplacement vers le bon répertoire d'archive.

Enfin, on peut aussi imaginer de gérer l'export de la base de données indépendamment de `rsnapshot`, via un script séparé. Dans tous les cas où l'on transfert la base de données sous forme de fichier, il peut être intéressant de ne pas compresser lesdits fichiers dans le but de laisser à `rsync` plus de latitude pour reconnaître les blocs inchangés et minimiser les transferts de données sur le réseau.

Pour finir, on peut parfaitement imaginer utiliser plusieurs instances de rsnaphot pour des serveurs qui auraient des besoins différents de rétention de données. Une précaution s'impose : spécifier des `snapshot_root` et `lockfile` différents par fichier de configuration. Les nouvelles instances de `rsnapshot` doivent alors être rajoutées au fichier `/etc/cron.d/rsnapshot`.

`rsnapshot` présente les sauvegardes sous une forme très élégante : une suite de sous-répertoires placés dans `snapshot_root` et appelés par exemple `hourly.n`, `daily.n`, `weekly.n` et `monthly.n`. La version `n`=0 étant la plus récente.

Pour connaître le volume disque pris par les sauvegardes configurées dans `/etc/rsnapshot.conf`, on pourra faire

```
# rsnapshot -c /etc/rsnapshot.conf du
57G     /mnt/sauvegardes/daily.0/
1.2G    /mnt/sauvegardes/daily.1/
1.4G    /mnt/sauvegardes/weekly.0/
1.5G    /mnt/sauvegardes/weekly.1/
1.7G    /mnt/sauvegardes/monthly.0/
2.0G    /mnt/sauvegardes/monthly.1/
6.8G    /mnt/sauvegardes/monthly.2/
72G     total
```

On apprend ainsi que la dernière sauvegarde occupe 57 gigaoctets d'espace disque. La sauvegarde du jour d'avant ne prend que 1,2 gigaoctets supplémentaires. L'incrément de `monthly.1` à `monthly.2` prend 6,8 gigaoctets, ce qui s'explique par une mise à jour du système d'exploitation (de Ubuntu *Dapper* à *Edgy*). La majeure partie des fichiers système a changé à cette occasion, le volume occupé par la sauvegarde s'en est logiquement ressenti.

La « magie » de l'estimation adéquate de l'espace disque pris par chaque incrément de sauvegarde vient directement de l'utilitaire GNU `du` (pour *disk usage*). `du` prend en effet en compte l'espace pris par une zone de données la première fois qu'il voit une entrée de répertoire (un *fichier*) pointer vers l'inode concernée. Quand d'autres entrées de répertoires pointent vers la même inode, le volume correspondant n'est pas comptabilisé. Cette fonctionnalité explique certainement l'empreinte en mémoire d'un `du` sur des sauvegardes d'un grand nombre de fichiers – jusqu'à plusieurs dizaines de mégaoctets. Bien loin d'un Firefox, mais tout de même...

Préservation du réseau

Une sauvegarde peut rapidement représenter une quantité non négligeable de bande passante dans le réseau. Les flux de données qui correspondent aux sauvegardes peuvent alors perturber d'autres flux qui devraient être considérés comme prioritaires. Il est certes souvent possible de limiter de bout en bout la bande passante que prend un flux de données (surtout quand il s'agit d'un flux `rsync`), mais cette technique est essentiellement statique : d'une part, elle ne dépend pas de l'état des liens ou des nœuds du réseau à l'instant de la sauvegarde, et, d'autre part, elle ne se met pas à jour quand le réseau évolue.

On s'intéresse à un routeur IP hypothétique qui aurait quatre interfaces de capacités identiques, trois qui reçoivent des données (paquets IP) en saturant la bande passante de chaque lien d'entrée, et une qui doit les transmettre. À chaque instant où le routeur a l'opportunité d'envoyer un paquet, il doit faire le choix du paquet qu'il envoie – et stocker les autres paquets dans ses tampons d'émission. Il y aura nécessairement des paquets retardés, étant donné que nous avons pris l'hypothèse que le routeur reçoit trois fois la bande passante qu'il est capable de transmettre. À terme,

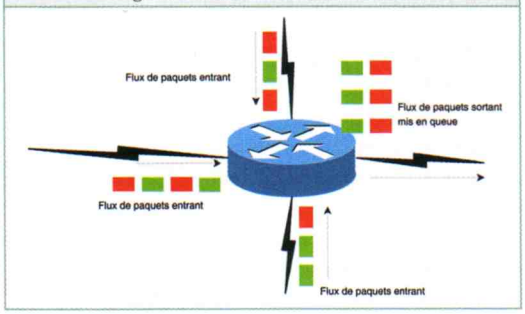
il faudra aussi qu'il fasse le choix des paquets qu'il choisit de supprimer :

- parce que ses tampons d'émission sont pleins ;
- parce que supprimer un paquet est un moyen d'indiquer à la source qu'elle doit réduire son débit ;
- parce qu'un paquet trop retardé perd de son intérêt pour le destinataire ;
- ou bien pour un peu de ces trois raisons à la fois.

Même si cette représentation peut paraître très schématique, elle correspond aux moments de congestion que les réseaux connaissent régulièrement. Le flux d'information sur un réseau de type Internet a en effet tendance à être intrinsèquement saccadé, avec régulièrement des pics d'utilisation qui excèdent de loin l'utilisation moyenne. C'est à ce moment là qu'une communication de voix sur IP se maintient si la qualité de service a été mise en œuvre correctement, et coupe (complètement ou dans une direction seulement) dans le cas contraire.

La qualité de service qu'un flux demande de la part d'un nœud du réseau peut être indiquée par l'octet ToS (type de service) – renommé plus récemment « champ DiffServ ». Deux valeurs de ToS sont représentées dans l'illustration 2, *rouge* et *vert*.

Fig. 2 : Schématisation d'un routeur



Beaucoup d'opérateurs de réseaux IP proposent différentes classes de service pour le trafic qui traverse leur cœur de réseau, fondées encore sur la priorité IP – les trois bits de poids fort de l'octet ToS. Les routeurs de périphérie se chargent de marquer les différents flux (donner la bonne valeur à la priorité IP), à charge au cœur du réseau de traiter les paquets selon cette consigne. Trois bits, cela fait tout de même 8 valeurs, et l'offre des opérateurs se limite souvent à quelque part entre 3 et 5 valeurs. Opérateurs et clients doivent alors se mettre d'accord sur les valeurs à mettre pour chacune des classes de service. Pour éviter d'avoir à modifier l'octet ToS du client pour l'adapter à leurs conventions propres, les opérateurs préfèrent mettre en œuvre dans une couche réseau inférieure la priorité des trames en question. Dans le cadre de cœurs de réseaux MPLS, par exemple, cela se fait en assignant les bits MPLS « expérimentaux » de tous les labels de la trame, en fonction de la priorité IP.

Le paquet IP reçoit donc la qualité de service indiquée par sa priorité IP dans le monde « IP » de la périphérie de réseau, et la qualité de service correspondant aux bits MPLS expérimentaux que son opérateur aura assignés à ses trames dans le monde « MPLS » du cœur de réseau.

Dans le cadre de notre sauvegarde, il faut donc s'attendre à devoir indiquer via la priorité IP que le flux de sauvegarde n'est pas prioritaire, que ses paquets ne devraient être transmis qu'une fois que les paquets des autres flux auront été servis, et que s'il faut supprimer des paquets, autant supprimer les paquets de sauvegardes – la couche TCP se chargera bien de la retransmission.

Dans le cas d'une liaison à un fournisseur d'accès à Internet, un autre réglage est possible. Si le lien Internet représente un goulot d'étranglement, il aura tendance à arriver régulièrement à saturation. Ce lien a deux tampons d'émission : un du côté du réseau local – dont nous avons le contrôle, et, un autre, du côté du fournisseur d'accès à Internet (FAI) – dont nous n'avons pas le contrôle direct. De toute évidence, le FAI qui fournit aussi un accès de téléphonie sur IP prendra soin de faire passer les flux de voix de manière prioritaire. Cependant, il n'a que faire de l'importance relative que nous pouvons accorder aux différents flux de données. Ce serait donc à nous de faire la police, mais sur un tampon que nous ne contrôlons pas. Il est cependant possible d'intervenir artificiellement sur la profondeur du tampon côté fournisseur d'accès, en supprimant volontairement les paquets que nous recevons, et qui correspondraient à un débit trop important – trop proche du débit maximal de la ligne. Ce faisant, nous indiquons aux couches réseau supérieures – typiquement TCP – qu'il faut réduire le débit. Ainsi, nous diminuons de quelques pourcents de la capacité totale du lien descendant, mais nous évitons potentiellement d'avoir des paquets qui s'accumulent dans une queue hors de notre contrôle.

Passons à la coloration de nos paquets. Mettons que les paquets *rouges* sont des paquets correspondant à la sauvegarde, et que les paquets *verts* sont tous les autres paquets. Il peut parfaitement y avoir plusieurs nuances de vert (ou de rouge), mais nous ne nous intéresserons pas à ces cas dans le cadre de cet article.

La couleur d'un paquet pourrait être déterminée par l'adresse IP source ou destination. Si le paquet va à un serveur de sauvegardes ou vient d'un serveur de sauvegardes, alors il est rouge. Il y a deux inconvénients majeurs à cette technique : d'une part, les serveurs de sauvegardes ne font souvent pas que des sauvegardes, et, d'autre part, il faut maintenir la liste des serveurs de sauvegardes dans tous les routeurs (ou serveurs) qui colorent les paquets.

La couleur d'un paquet pourrait aussi être déterminée par le service auquel il s'adresse – le port de destination. Dans le cadre d'une sauvegarde qui utilise SSH, on ne voudrait cependant pas que tous les paquets SSH soient traités comme des paquets de sauvegarde. Les sessions

interactives de terminal deviendraient rapidement inutilisables en cas de congestion du réseau.

Une solution élégante vient de ce qui pourrait être considéré comme une faiblesse de OpenSSH. On pourrait croire qu'il n'y a aucun moyen de savoir ce qui passe dans un flux SSH – et c'est bien pour cela qu'on se sert de cet outil. Cependant, OpenSSH n'a pas la prétention de mettre en œuvre de la stéganographie, l'art de la dissimulation comme le dit Wikipedia **[6]**. *Tout le monde* est donc libre de savoir quel volume de données un utilisateur de OpenSSH envoie ou reçoit d'un serveur. Si le volume est limité et correspond à des paquets de petite taille, il est plus probable qu'il s'agisse d'une session de terminal. Si le volume est plus important, il est plus probable qu'un autre protocole soit encapsulé dans OpenSSH. On peut même extraire la longueur d'un mot de passe en observant une session d'authentification interactive **[7]**.

Dans la même veine, OpenSSH indique lui-même via le champ DiffServ la nature des informations qu'il envoie.

`tshark`, la version en ligne de commande de WireShark (anciennement connu sous le nom d'Ethereal), permet d'observer le champ DiffServ dans les scénarios qui nous intéressent, en utilisant cette commande :

```
$ sudo tshark -V 'port 22 and host hote' | \
egrep 'Internet Protocol|Differentiated Services
Field'
```

En observant les premiers paquets SSH (pré-authentification), on note qu'ils ont tous un DSCP à `0x00` :

```
    Differentiated Services Field: 0x00 (DSCP 0x00:
Default; ECN: 0x00)
```

Dans le cas d'une session interactive (type accès shell), les paquets suivants (post-authentification) quittent le client avec un DSCP à `0x10` :

```
    Differentiated Services Field: 0x10 (DSCP 0x04:
Unknown DSCP; ECN: 0x00)
```

Si le serveur est sur Internet, il est fort probable que les paquets reviennent « décolorés » avec un DSCP à 0. On entend suffisamment que le service de transmission des données sur Internet est de type *best effort* (sans qualité de service), en voilà une illustration. Le fournisseur d'accès à Internet remet les champs de qualité de service à 0 quand les paquets traversent son réseau, et les traite tous de manière « égale » (premier arrivé, premier servi).

Dans le cas d'une session non interactive, les paquets post-authentification ont un DSCP à `0x08` :

```
    Differentiated Services Field: 0x08 (DSCP 0x02:
Unknown DSCP; ECN: 0x00)
```

Les paquets SSH (post-authentification, donc la majeure partie d'entre eux) ont ainsi un DSCP à `0x10` pour une session interactive et `0x08` pour un transfert de masse (de type sauvegarde).

On peut se lancer dans l'écriture de la gestion de la qualité de service dans une queue Linux.

```
#!/bin/sh
# Gestion minimale de la qualité de service
# On s'intéresse uniquement à l'interface eth0

TC="/sbin/tc"

# (0) On commence par faire le ménage et enlève les qdisc
$TC qdisc del dev eth0 root > /dev/null 2>&1

# (1) Limitation de vitesse
$TC qdisc add dev eth0 root handle 1: tbf rate XXXkbit burst 4k latency 30ms

# (2) On rajoute la qdisc prio avec deux bandes
#     et envoie tout dans le flot haute priorité par défaut
$TC qdisc add dev eth0 parent 1: handle 10: prio bands 2 priomap 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

# (3) Transfert de données en masse par SSH va dans le flot basse priorité
#     (TOS à Maximize Throughput)
$TC filter add dev eth0 parent 10:0 prio 2 protocol ip u32 \
  match ip sport 22 0xffff \
  match ip tos 0x8 0xff \
  flowid 10:2
$TC filter add dev eth0 parent 10:0 prio 2 protocol ip u32 \
  match ip dport 22 0xffff \
  match ip tos 0x8 0xff \
  flowid 10:2

# (4) On rajoute un FIFO par bande, pour avoir des statistiques
/sbin/tc qdisc add dev eth0 parent 10:1 handle 11: pfifo
/sbin/tc qdisc add dev eth0 parent 10:2 handle 12: pfifo
```

(1) a pour objet de supprimer toute gestion de qualité de service résiduelle pour commencer sur des bases saines.

(2) précise que quelle que soit la gestion plus fine de la QoS, la bande passante utilisée ne saurait excéder `XXX` kb/s. Ce filtre, à régler très légèrement en-deçà de la bande passante du lien montant, vise à éviter que les paquets IP ne s'accumulent entre la queue en question et le noyau d'étranglement – par exemple un éventuel routeur qui ne mette pas en œuvre de qualité de service. Cette précaution devient caduque si le routeur a plus d'un ordinateur local connecté.

(3) crée une discipline de mise en queue (`qdisc`) de type `prio` à deux bandes. Dans cette `qdisc`, les classes de basse priorité (paquets rouges) ne sont autorisées à émettre que si les classes de plus haute priorité (paquets verts) n'ont plus aucun paquet à envoyer. Dans une mise en œuvre plus complète, la classe de plus haute priorité sera elle-même décomposée en sous-classes plus granulaires séparant la vidéo, la voix sur IP, les transferts de masse etc. Ces paquets pourraient donc être représentés par différentes nuances de vert.

(4) Les paquets SSH qui correspondent à du transfert de masse vont dans la classe de basse priorité. Ils ont le port 22 comme source ou destination, et un champ DSCP à `0x08`.

Un déploiement d'une solution de qualité de service ne saurait se passer d'une composante d'évaluation de performances. On peut pour cela utiliser le logiciel `munin`, disponible en paquet Debian. Il convient

d'installer le serveur munin (`munin`) ainsi que son agent (`munin-node`), potentiellement sur la même machine :

```
$ sudo apt-get install munin munin-node
```

Certaines statistiques fournies par défaut sont alors accessibles dans `/var/www/munin`, donc a priori à partir de l'URL **http://hote/munin**. Il existe plusieurs ajouts, disponibles sur le site web du projet. On s'intéresse à un en particulier **[8]**, qui permet de surveiller la gestion de la qualité de service. Une fois le script téléchargé **[9]**, il peut être copié en tant que `/usr/local/sbin/qos_`. Il s'intègre aux autres ajouts en créant un lien symbolique dans le répertoire des ajouts de `munin`. Si l'interface à surveiller est `eth0`, alors les commandes suivantes devraient faire l'affaire pour surveiller le volume envoyé (`sent`), les paquets envoyés (`sentpkts`), en queue (`backlog`), ceux supprimés (`dropped`) ou hors-limite (`overlimits`) :

```
$ sudo ln -s /usr/local/sbin/qos_ /etc/munin/plugins/
qos_eth0_sentpkts
$ sudo ln -s /usr/local/sbin/qos_ /etc/munin/plugins/
qos_eth0_backlog
$ sudo ln -s /usr/local/sbin/qos_ /etc/munin/plugins/
qos_eth0_dropped
$ sudo ln -s /usr/local/sbin/qos_ /etc/munin/plugins/
qos_eth0_overlimits
```

La queue 10 n'apporte pas de renseignements fondamentalement intéressants, on peut la désactiver en rajoutant ceci dans le fichier de configuration de l'agent `munin`, `/etc/munin/plugin-conf.d/munin-node` :

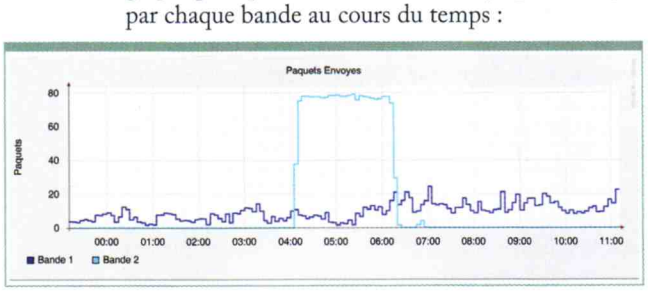
```
[qos_*]
env.ignore_queue10
```

Il convient alors de redémarrer l'agent `munin` pour que le nouvel ajout soit intégré :

```
$ sudo /etc/init.d/munin-node restart
```

Les statistiques collectées par `munin` sont archivées pour l'hôte local dans le répertoire `/var/lib/munin/localdomain/`. Pour supprimer les statistiques de qualité de service, il suffirait de supprimer les fichiers `/var/lib/munin/localdomain/localhost.localdomain-qos_eth0-*`.

Les fichiers RRD générés par RRDTool – lui-même appelé par `munin` – peuvent être retravaillés, pour en faciliter l'interprétation. On peut ainsi générer un graphique représentant le nombre de paquets envoyés par chaque bande au cours du temps :



La commande utilisée a été la suivante :

```
$ rrdtool graph sent.eps -w 800 -h 200 --font
DEFAULT:14:Helvetica \
--end 1156090200 --start end-12h -t "Paquets Envoyes"
-v "Paquets" -a EPS \
DEF:pfifo11=./localhost.localdomain-qos_eth0_sentpkts-
pfifo11-c.rrd:42:AVERAGE \
LINE2:pfifo11#0000FF:"Bande 1" \
DEF:pfifo12=./localhost.localdomain-qos_eth0_sentpkts-
pfifo12-c.rrd:42:AVERAGE \
LINE2:pfifo12#00CCFF:"Bande 2"
```

Le graphique montre la période de sauvegarde, où un flux de données sortant a entraîné une utilisation bien plus importante que d'ordinaire de la bande 2. La coloration se fait donc correctement. Quelques idées pour évaluer les performances de la qualité de service : utiliser les fonctionnalités de mesure de performances des routeurs eux-mêmes (typiquement à base d'agents configurables par SNMP), observer les statistiques de synchronisation de l'heure (NTP) avec et sans qualité de service (`munin` a un ajout à cet effet), les statistiques de voix sur IP ou encore sonder les utilisateurs.

Pour une description bien plus complète des capacités de Linux en matière de qualité de service, on pourra se référer à l'excellent *Linux Advanced Routing & trafic Control HOWTO* **[10]**.

Lecture de journaux

Au chapitre de la due dilligence, on trouve aussi le pendant germanique du *My taylor is rich : Vati liest seine zeitung* – Papa lit son journal. La lecture des journaux d'un système Linux permet surtout de mieux comprendre son fonctionnement. Une fois qu'un administrateur s'est familiarisé avec lesdits journaux, les anomalies sont bien plus simples à débusquer – et les incidents à prévenir. Tel en tout cas est le discours tenu par la communauté : administrateurs, gardez un **œil** sur vos journaux. Bien que pleine de bon sens, cette recommandation n'en est pas pour autant nécessairement facile à suivre.

Le fichier de journal phare sous GNU/Linux Debian, `/var/log/syslog`, peut facilement atteindre quelques mégaoctets par jour sur des systèmes raisonnablement utilisés. Même le fichier `/var/log/auth.log`, pourtant tellement utile pour détecter les tentatives d'intrusion, atteint facilement le mégaoctet journalier.

Parcourir ces fichiers de journaux de temps à autre est donc définitivement recommandé, mais cette approche ne passe pas à l'échelle et ne tient pas la longueur.

Très tôt, les administrateurs Unix de tout poil se sont donc concoctés des scripts pour extraire l'information pertinente des fichiers de journaux. Il est alors facile de lancer ces scripts à intervalles réguliers, et d'alerter l'administrateur en cas d'anomalie. L'outil `cron` se chargerait de lancer le script, et `mailx` d'alerter si nécessaire. D'autres outils tels que `awk`, `cut`, `grep`, `sed` ou même `perl` permettent de saucissonner les journaux et d'en extraire la substantifique moelle.

La possibilité d'écrire un script ad hoc pour remplir une tâche donnée est une merveille en soit. Pour le meilleur, mais hélas surtout pour le pire, de tels scripts vont rapidement voir leur horizon enfler : non seulement ils doivent surveiller cette machine, mais, dès leur utilité avérée, il faudra les étendre à tout le parc informatique là-derrière. À la prochaine mise à jour du système d'exploitation, il faudra aussi songer à vérifier qu'ils marchent toujours et corriger les incompatibilités éventuelles. Rajouter des fonctionnalités. Documenter leur fonctionnement. En un mot, il faudra les maintenir.

Un autre aspect du problème est l'intelligence à fournir à un tel script. Alors que certaines entrées de journal sont immanquablement les traces d'une activité malfaisante, la vaste majorité d'entre elles n'offrent pas d'intérêt sur le moment.

Maintenir un script qui remplit une tâche spécifique et à laquelle personne dans la communauté ne s'est jamais intéressé auparavant, pourquoi pas ? Il paraît cependant douteux qu'un script d'analyse de journaux `syslog` Linux tombe dans cette catégorie. D'autant plus que la tâche d'inculquer au script en question la logique des entrées des journaux système, ainsi que leurs intérêts respectifs requiert une expertise et un temps non négligeables.

Un script écrit en Bash existe depuis plus de 10 ans et remplit précisément cette fonction de vérification de journaux : `logcheck`. Son paquet Debian en est à la version 1.2.39 sous Sarge, et comme en atteste son évolution **[11]**, le paquet est relativement bien maintenu – porté par Debian devrait-on même dire, à la lecture de **http://logcheck.org**.

Quels seraient les avantages d'installer `logcheck` venant de Debian ?

Déjà, le script tourne sous un utilisateur dédié crée par le paquet Debian à l'installation, Mr logcheck qui fait partie du group `adm`. Il peut donc lire les journaux système, mais ne peut pas faire plus de dégâts que çà. Rien de bien compliqué à mettre en place, mais ce genre de « détail » à tendance à prendre du temps et à être oublié.

Ensuite, Debian maintient une liste de règles de vérification des journaux, de sorte que beaucoup de programmes fournis par Debian sont déjà connus de `logcheck`. Un certain nombre d'événements système qu'ils peuvent générer sont donc déjà classés.

Dans le fichier `/etc/logcheck/ignore.d.server/ssh`, par exemple, on lit :

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ sshd\[[0-9]+\]: Server
listening on [.0-9]+ port 22\.$
```

Cette expression régulière se charge d'éviter de générer une alerte à chaque fois que `sshd` redémarre et se met à écouter sur le port 22. La syntaxe attendue des expressions régulières se trouve entre autres dans la page de manuel d'`egrep`.

`logcheck` peut générer trois sortes d'événements :

- les **alertes d'attaque** – entrées de journaux qui coïncident avec une entrée dans `/etc/logcheck/cracking.d`, mais pas dans `/etc/logcheck/cracking.ignore.d`.
- les **événements de sécurité** – entrées de journaux qui coïncident avec une entrée dans `/etc/logcheck/violations.d`, mais pas dans `/etc/logcheck/violations.ignore.d`.
- les **événements système** – entrées de journaux restantes, mais qui ne coïncident pas avec une entrée dans `/etc/logcheck/ignore.d.server`, `ignore.d.workstation` ou encore `ignore.d.paranoid` suivant la configuration de `REPORTLEVEL` dans `/etc/logcheck/logcheck.conf`.

Enfin, `logcheck` a déjà tous les bons systèmes en place pour ne vérifier que les entrées générées depuis la dernière fois qu'il a été lancé – ceci grâce à `logtail`.

Le fichier de configuration `/etc/logcheck/logcheck.conf` contient des valeurs par défaut qui s'avèrent sensées – pour un serveur en tous cas. On remarquera que `logcheck` ne s'intéresse qu'aux fichiers `/var/log/syslog` et `/var/log/auth.log`. La raison de ce choix s'explique par le contenu même du fichier de configuration de `syslog` chez Debian :

```
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
```

Tous les événements système intéressants vont donc se retrouver dans l'un de ces deux fichiers.

Le déploiement de `logcheck`, en particulier sur un serveur assez utilisé, doit cependant se faire avec méthode pour apporter toute sa valeur. Une stratégie efficace est de rajouter un fichier `/etc/logcheck/violations.ignore.d/local`, ainsi qu'un fichier `/etc/logcheck/ignore.d.server/local`. Le premier permet d'ignorer des événements qui seraient reconnus comme alertes de sécurité, tandis que le deuxième permet d'ignorer des alertes système.

Par défaut, `logcheck` enverra un courriel titré *security alerts*, *security events* ou *system events* suivant la sévérité maximale des évènements contenu dans le courriel. On peut alors imaginer router vers différentes personnes ou différents rôles ces courriels, suivant le sujet.

Il est aussi envisageable d'utiliser la même liste d'exclusion pour `/etc/logcheck/violations.ignore.d/local` et `/etc/logcheck/ignore.d.server/local`, si l'on souhaite que les événements de sécurité qui ont été exclus ne deviennent pas des événements système. On peut, par exemple, utiliser un lien en dur pour que l'entrée `local` du répertoire `/etc/logcheck/ignore.d.server/` pointe vers la même zone de données du système de fichiers que l'entrée `local` du répertoire `/etc/logcheck/violations.ignore.d/` :

```
$ sudo ln /etc/logcheck/violations.ignore.d/local
/etc/logcheck/ignore.d.server/local
```

Dès lors que les deux entrées de répertoire pointent vers la même zone de données, on est assuré que ces « deux » fichiers restent « identiques ».

On commence par un fichier `local` vide. L'administrateur doit s'attendre à recevoir un premier courrier électronique d'alerte assez fourni en faux positifs, ces alarmes qui ne devraient pas se déclencher – malgré les réglages par défaut de Debian. Il faut alors de manière systématique définir dans le fichier `local` les expressions régulières des faux positifs. Que les lecteurs qui font le rapprochement avec le réglage de signatures Snort se rassurent. Les deux exercices ont en effet des points communs, mais `logcheck` atteint un état stabilisé bien plus rapidement que Snort. Faut-il le préciser, `logcheck` et Snort sont complémentaires, l'un s'intéressant aux aspects réseau et l'autre aux aspects système.

Par exemple, sur un serveur SMTP qui fait tourner Milter-greylist **[12]**, très efficace comme première ligne de défense contre les *pourriels*, l'administrateur n'a pas forcément envie d'être notifié à chaque fois qu'un serveur distant se fait temporiser. Dans ce cas, il faut rajouter cette expression régulière dans son fichier `local` :

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ sm-mta\[[0-9]+\]:
[[:alnum:]]+: Milter: to=<[=@+._[:alnum:]-]+>,
reject=451 4\.7\.1 Greylisting in action, please come
back in [0-9]+:[0-9]+:[0-9]+$
```

On commence par ancrer l'expression régulière : « `^` » indique qu'il s'agit du début de la ligne. L'élément suivant représente les trois lettres du mois de l'année (`\w{3}`). Viennent ensuite le jour et l'heure, qui sont toujours représentés par une suite de onze caractères qui sont des espaces, des deux-points ou des chiffres (`[ :0-9]{11}`). Le champ suivant est le nom du serveur, qui peut contenir un ou plusieurs caractères alphanumériques, points, *underscore* (_) ou tirets (`[._[:alnum:]-]+`). Ces événements sont générés par l'agent de transfert de courriels, d'où le `sm-mta`. L'événement contient aussi l'identifiant du processus en question entre crochets (`\[[0-9]+\]`). L'identifiant alphanumérique de la connexion s'ensuit (`[[:alnum:]]+`), puis « `Milter: to=` » et l'adresse de messagerie électronique de destination. Enfin, on trouve le message d'erreur SMTP 451 avec, en fin de chaîne, le temps de temporisation. « `$` » indique la fin de la ligne.

`logcheck` s'avère particulièrement précieux quand un pare-feu Netfilter est installé. Un exemple assez simple serait une entrée de journal correspondant à une tentative de connexion sortante par un utilisateur non autorisé.

Le script de démarrage du pare-feu commencerait par préciser les variables dont on se sert souvent :

```
IPTABLES=/sbin/iptables
IFACE="eth0+"
```

Les paquets correspondant aux connexions existantes sont autorisées à quitter la machine :

```
$IPTABLES -A OUTPUT -o $IFACE -m state --state
RELATED,ESTABLISHED -j ACCEPT
```

Seul l'utilisateur root est autorisé à initier des connexions sortantes sur le port 80 :

```
$IPTABLES -A OUTPUT -o $IFACE -p tcp --dport 80 -m
owner --uid-owner 0  -m state --state NEW -j ACCEPT
```

L'une des dernières règles du pare-feu préciserait qu'il faut générer une entrée de journal pour tous les paquets TCP qui n'ont pas encore été autorisés à quitter la machine, puis les refuser :

```
$IPTABLES -A OUTPUT -o $IFACE -p tcp -j LOG --log-
prefix "IPTABLES TCP-OUT: "
$IPTABLES -A OUTPUT -o $IFACE -p tcp -j DROP
```

Le script doit être complété pour autoriser les autres connexions intéressantes – étant entendu qu'une politique de refus par défaut comme montré pour le cas des connexions sortantes est des plus désirables.

Il faut alors s'attendre à ce que `logcheck` envoie des courriels pour avertir d'événements tels que celui-ci :

```
Feb 18 18:23:24 hote kernel: IPTABLES TCP-OUT: IN=
OUT=eth0 SRC=a.b.c.d DST=e.f.g.h LEN=40 TOS=0x00
PREC=0x00 TTL=64 ID=45626 DF PROTO=TCP SPT=34892
DPT=15766 WINDOW=6912 RES=0x00 ACK RST URGP=0
```

Cette entrée de journal est très certainement un faux positif, car elle ne correspond pas à une ouverture de connexion, mais à la toute fin de fermeture d'une session TCP (drapeaux ACK et RST). De toute évidence, la connexion était trop lente à se fermer et le pare-feu a considéré que ce paquet était hors état.

On commence par vérifier que les temporisateurs du pare-feu sont bons dans les paramètres de Netfilter `/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout*`.

On peut ensuite faire disparaître ces messages, mais en prenant soin de conserver les événements correspondant à des « ouvertures » de connexion non autorisées, en utilisant par exemple l'expression régulière

```
^\w{3} [ :0-9]{11} [._[:alnum:]-]+ kernel: IPTABLES
TCP-OUT:.*(ACK|RST)
```

Les paquets d'ouverture de session (« `SYN` ») ne suivront donc pas cette expression régulière et généreront un événement `logcheck`.

Dans la même veine, il est possible qu'un serveur Zope soit redémarré à intervalles réguliers pour pallier d'éventuelles fuites de mémoire. Si cela se passe à six heures du matin, alors cette expression régulière permettra d'ignorer les alarmes de Squid rapportant un refus d'ouverture connexion à l'heure du redémarrage :

```
^\w{3} [0-9]{2} 06:[ :0-9]{5} [._[:alnum:]-]+
squid\[[0-9]+\]: TCP connection to hote/[0-9]+ failed$
```

Dans le but de maintenir une certaine cohérence dans les fichiers d'exception `local`, il est vivement conseillé de documenter la raison de l'exception. Il suffit de faire commencer une ligne par un dièse (#) pour qu'elle soit ignorée, et puisse inclure un commentaire.

Comme indiqué dans la documentation de logcheck, on peut se servir de la commande suivante pour vérifier la validité de l'expression régulière regexp :

```
$ sudo sed -e 's/[[:space:]]*$//' /var/log/syslog |
egrep 'regexp'
```

Ou bien, si l'on veut tester l'ensemble des règles d'un fichier /etc/logcheck/ignore.d.server/local, on peut lancer sous l'utilisateur root :

```
# sed -e 's/[[:space:]]*$//' /var/log/syslog | egrep
-f /etc/logcheck/ignore.d.server/local | less
```

Que se passe-t-il si une entrée de journal est correctement filtrée par cette ligne de commande, mais apparaît quand même dans le courriel envoyé par logcheck ? Il se peut que ce soit la manifestation d'une localisation différente entre logcheck et le logiciel à l'origine de l'entrée dans le journal. Ce problème a été décrit dans le rapport de bogue 401259 **[13]**.

Il faut s'attendre à devoir retoucher les exceptions de logcheck à plusieurs occasions :

- à la suite du premier redémarrage du système, où les journaux système montrent des entrées inédites ;
- à la suite de la première occurrence du cron horaire, journalier, hebdomadaire et mensuel. Les tâches correspondantes se trouvent dans les répertoires /etc/cron.* ;
- à la suite de la première occurrence de chacun des travaux cron de /etc/cron.d ;
- à la suite d'incidents qui n'ont pas encore été classés ;
- à la suite de travaux cron qui ont été impactés par des incidents dont c'était la première occurrence.

```
Jul  7 08:03:14 hote mod_evasive[28973]: Blacklisting
address x.x.x.x: possible DoS attack.
```

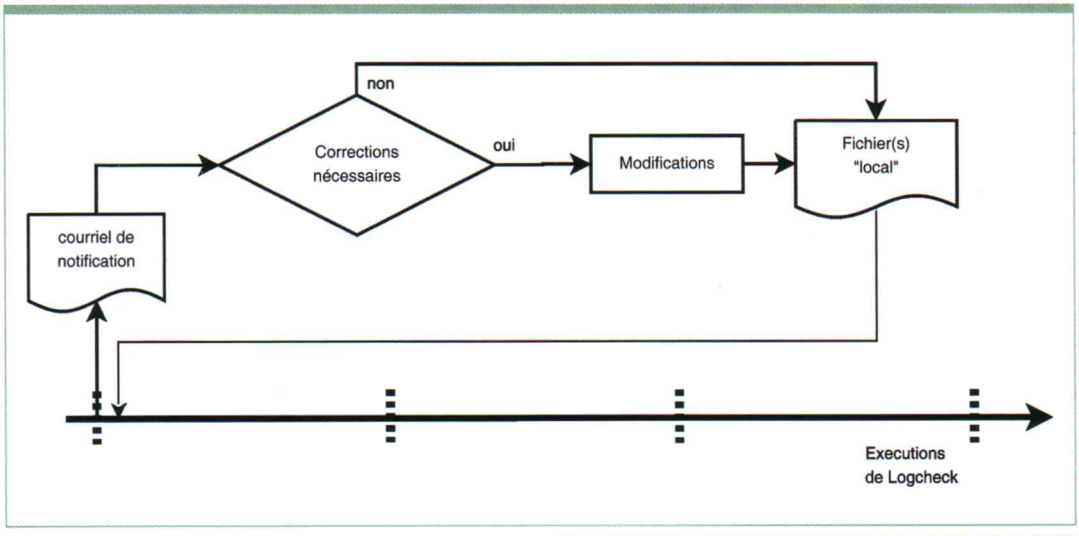
- Sendmail calme les ardeurs d'un pollueur potentiel :

```
Jul  8 09:46:41 hote sm-mta[22202]: l1I9kdjv022202:
fqdn [x.x.x.x] (may be forged): Possible SMTP RCPT
flood, throttling.
```

- La pile IP signale ou bien un bogue du noyau ou un client aux pratiques TCP inhabituelles (qui limite fortement son débit pour une raison « légitime »), ou encore un amateur de *tar pitting* TCP (attaque par déni de service, en réduisant de manière disproportionnée le débit TCP pour que le service mobilise des ressources inutilement) :

```
Jul  9 20:09:56 hote kernel: TCP: Treason
uncloaked! Peer x.x.x.x:62973/80 shrinks window
4184516861:4184518241. Repaired.
```

- Un paquet TCP envoyé par hote a vu son TTL expirer. Il a donc reçu un message de notification ICMP de type 11, code 0, qui aurait du être accepté par l'une des premières règles de ce pare-feu qui autorise les paquets RELATED entrant. Ou bien cette interprétation n'est pas bonne et le pare-feu est en train de se faire jauger ou un des *timers* Netfilter gagnerait à être ajusté, ou encore le paquet ICMP a mis beaucoup trop de temps à revenir.



Mais que reste-t-il une fois que le fichier local a été affiné pendant quelques semaines ? Quelques événements par jour tout au plus, dont voici quelques exemples commentés :

- L'excellent module d'Apache mod_evasive se protège contre une attaque de déni de service :

```
Jul 10 12:38:02 hote kernel: IPTABLES ICMP-BAD-TYPE-
IN: IN=eth0 OUT= MAC=xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:
xx:xx:xx:xx SRC=xxx.xxx.xxx.xxx DST=xxx.xxx.xxx.xxx
LEN=56 TOS=0x00 PREC=0x00 TTL=232 ID=19885 PROTO=ICMP
TYPE=11 CODE=0 [SRC=xxx.xxx.xxx.xxx DST=xxx.xxx.
xxx.xxx LEN=40 TOS=0x00 PREC=0x00 TTL=0 ID=4534 DF
PROTO=TCP INCOMPLETE [8 bytes] ]
```

- Squid remarque qu'un des serveurs d'application manque à l'appel...

```
Jul 11 19:36:39 hote squid[1299]: Detected DEAD
Parent: hote/xxxx/xxxx
```

- puis réintègre les rangs :

```
Jul 11 19:37:49 hote squid[1299]: Detected REVIVED
Parent: hote/xxxx/xxxx
```

- Le microprocesseur a un hoquet – rien d'inquiétant à petite dose :

```
Jul 12 13:04:56 hote kernel: MCE: The hardware reports
a non fatal, correctable incident occurred on CPU 0.
```

- Au tour du disque dur d'avoir un hoquet bénin – ne pas se laisser impressionner **[14]** :

```
Jul 14 16:10:08 hote kernel: hda: drive_cmd:
status=0x51 { DriveReady SeekComplete Error }
Jul 14 16:10:08 hote kernel: hda: drive_cmd:
error=0x04 { DriveStatusError }
```

- Sendmail de nouveau inquiété :

```
Jul 16 00:47:05 hote sm-mta[21084]: l2D0knce021084:
smtp.exemple.com [x.x.x.x] (may be forged): possible
SMTP attack: command=HELO/EHLO, count=3
```

- Sendmail encore :

```
Jul 17 00:49:51 hote sm-mta[24795]: l2N0dmhn024795:
smtp.exemple.com [x.x.x.x]: possible SMTP attack:
command=NOOP, count=20
```

- Sendmail toujours :

```
Jul 18 10:39:39 hote sm-mta[19046]: l2LAddiI019046:
smtp.exemple.com [x.x.x.x]: probable open proxy:
command=CONNECT smtp.exemple2.com:25 HTTP/1.0\r
```

- Un utilisateur avec accès `sudo` se trompe de mot de passe :

```
Jul 19 01:10:07 hote sudo: (pam_unix) authentication
failure; logname=utilisateur uid=0 euid=0 tty=pts/2
ruser= rhost=  user=user
```

- Squid rapporte une requête bien peu innocente :

```
Jul 20 22:33:28 hote squid[1299]: WARNING: suspicious
CR characters in HTTP header {If-Modified ^Mozce: Tue,
06 Dec 2005 04:58:06 GMT; length=11799}
```

- Un de ces *scans* de SSH qui poussent les administrateurs à filtrer les adresses IP sources :

```
Jul 21 15:59:05 hote sshd[21340]: Did not receive
identification string from x.x.x.x
```

`rsnapshot` permet donc de réduire avec simplicité le bruit présent dans les journaux système, et de diffuser tout aussi simplement le résultat de son analyse.

Offrir la possibilité d'agencer à loisir des composantes logicielles venant de tous horizons – libres ou moins libres – est une vertu indéniable de l'infrastructure offerte par les Logiciels libres. Le liant des esquisses de solutions que nous avons abordées n'est pas de toute première jeunesse : la première version de Bourne shell (sh), dont Bash s'est largement inspirée, fêtera ses 30 ans l'année prochaine.

Les shells étaient révolutionnaires à l'époque de leur création, parce qu'ils servaient à la fois de langage de commande interactif et de langage de script. Ce mode de pensée est maintenant bien installée dans les mœurs. En s'interrogeant sur l'évolution suivante du shell Linux et des utilitaires qui lui gravitent autour, il paraît légitime de regarder du côté du paradigme objet qui a bien gagné en maturité de conception et de mise en œuvre.

Guillaume Tamboise,

mailto:gtamboise@gmail.com

NOTES ET RÉFÉRENCES

- **[1]** http://en.wikipedia.org/wiki/Powershell
- **[2]** http://www.issa.org/cgi/issaopnpg.php?page=journals/2006_August/Lindberg - Nimble Risk Management.pdf
- **[3]** http://www.issa.org
- **[4]** http://www.issa.org/cgi/issaopnpg.php?page=journals/2006_May/Parker - Replacing Risk-Based Security.pdf
- **[5]** On remarquera au passage que syslog commence sa rotation à syslog.0.
- **[6]**http://fr.wikipedia.org/wiki/St%C3%A9ganographie
- **[7]** Cf. utilitaire sshow du paquet dsniff.
- **[8]** http://munin.projects.linpro.no/attachment/wiki/PluginCat/qos_
- **[9]** http://munin.projects.linpro.no/attachment/wiki/PluginCat/qos_?format=raw
- **[10]** http://lartc.org/lartc.html, la première page a un lien vers la version traduite en français.
- **[11]** http://packages.debian.org/changelogs/pool/main/l/logcheck/logcheck_1.2.54/changelog
- **[12]** http://hcpnet.free.fr/milter-greylist/
- **[13]** http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=401259
- **[14]** http://www.captain.at/howto-linux-driveready-seekcomplete-error-drivestatuserror.php :

Julien Guellec

Installation, configuration et déploiement d'un serveur de gestion d'images : Partimage

Les tâches de maintenance qui incombent aux administrateurs réseau sont de plus en plus nombreuses et demandent toujours un temps considérable. La gestion d'un parc informatique de grande taille peu vite conduire à des problèmes simples à résoudre, mais prenant un temps conséquent dont l'administrateur ne dispose pas forcément. La réinstallation de postes ayant subi une avarie en est un bon exemple. Cet article a pour objectif de vous accompagner lors de la création de votre serveur de gestion d'images Partimage, qui permettra d'alléger cette charge de travail. Nous aborderons dans un premier temps l'installation du serveur, puis nous étudierons un cas concret de sa configuration et de son déploiement.

1. Introduction

Partimage est un utilitaire libre distribué sous licence GPL2. Il permet la sauvegarde et la restauration d'images système (également appelées « *ghost* » en raison du logiciel propriétaire de Symantec).

L'intérêt d'un tel serveur prend tout son sens sur un parc informatique de grande taille. Imaginez que vous avez plus d'une centaine de machines, globalement identiques, sur lesquelles vous devez déployer un système d'exploitation, une configuration bien particulière, des ressources logicielles identiques, etc. Il serait intéressant, pour l'administrateur, de ne pas avoir à passer sur toutes ces machines pour les installer et les configurer une à une. Ce serait une perte de temps colossale. Imaginons également qu'un jour une de ces machines vient à « crasher ». Il est dans ce cas fort utile de disposer d'une image système qui permettra, en quelques minutes, de retrouver la configuration initiale. Partimage vous permet de résoudre ces problèmes.

Il supporte un grand nombre de systèmes de fichiers (dont nous verrons le détail par la suite), et est un projet activement entretenu par la communauté *Open Source*. Parmi ses avantages, on pourra relever que Partimage ne compresse QUE les données utiles d'une partition (et non les blocs vides comme le font certains de ses concurrents, tel que G4U...). Ceci permet bien évidemment de réduire la taille finale de l'image et donc d'optimiser l'espace disque sur le serveur les hébergeant.

Enfin, notons que Partimage fonctionne sur une architecture client/serveur et qu'il est relativement simple à mettre en œuvre et à exploiter en production.

2. Installation de Partimage

2.1. Depuis les sources

L'installation de Partimage se déroule comme une installation « basique ». Il suffit pour cela de télécharger les fichiers sources depuis le site officiel **[1]** et de décompresser l'archive avec un traditionnel : `tar xvjf partimage-0.6.5.tar.bz2`.

Veuillez toutefois noter que Partimage requiert un bon nombre de dépendances. Vérifiez donc que celles-ci sont installées avant et, le cas échéant, remédiez-y. Vous trouverez une liste des dépendances nécessaires sur le site de Partimage.

Ensuite un `./configure`, `make` et `make install` suffisent. Optionnellement, si vous avez installé OpenSSL, vous pouvez générer les certificats nécessaires avec la commande : `make certificates`.

Il existe également la solution de l'utiliser directement à partir d'un *live*-CD **[2]** ou d'utiliser un script d'auto-installation mis à disposition sur ce site **[3]**.

REMARQUE

Lors de la compilation, il est possible de personnaliser certains paramètres (notamment l'emplacement de stockage des images système, leur propriétaire...). Pour plus de détails : `./configure --help`.
Cet emplacement est de toute façon modifiable par la suite lorsque vous lancerez le démon `partimaged` en utilisant l'option `-d` (ou `--dest`). Entre autres, vous pouvez également :
- intervenir sur le port d'écoute du serveur (par défaut 4025) avec l'option `-pX` (ou `--port=X`) X étant bien entendu le port choisi ;
- désactiver le *login* des clients avec l'option `-L` (ou `--nologin`) ;
- et bien d'autres.

Pour plus de détails, référez vous à l'aide (`partimaged --help`).

2.2. Depuis le gestionnaire de paquets apt de Debian

Contrairement à ce qui est expliqué sur le site officiel, un `apt-get install partimage` ne suffit pas, du moins pour la version stable. Si vous êtes sur une telle distribution, référez-vous au paragraphe précédent pour une installation manuelle et, le cas échéant, procédez à son installation par un : `apt-get install partimage-server`.

Par défaut, l'installation génère les certificats SSL et une clé publique RSA de 1024 bits (dans : `/etc/partimaged/partimaged.key`). Vous devrez ajouter également les noms des utilisateurs qui seront autorisés à utiliser le serveur dans le fichier : `/etc/partimaged/partimagedusers`.

L'emplacement de stockage des images est, par défaut, le suivant : `/var/lib/partimaged/`.

2.3. Depuis le fichier binaire

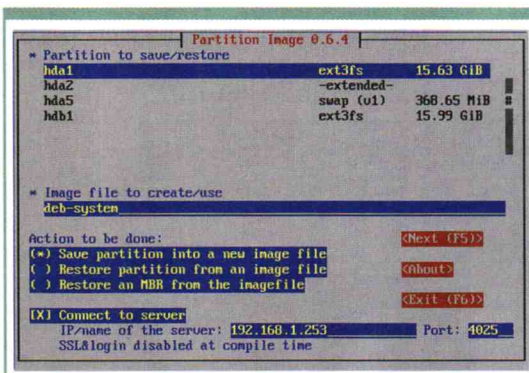
Vous pouvez enfin exécuter Partimage directement depuis le binaire. Cette méthode d'utilisation a l'avantage de vous permettre de l'utiliser sans disposer d'un serveur, en local. Ainsi, vous pouvez faire un ghost de votre machine et l'utiliser ultérieurement au besoin. Il vous suffit de le lancer : `./partimage`. Les fichiers d'images seront écrits dans le répertoire courant, au moment où vous avez lancé le programme.

3. Sauvegarde d'une image système

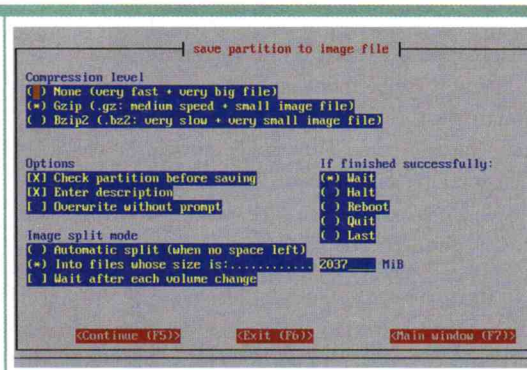
Vous avez deux moyens pour utiliser Partimage :

- l'utilisation par la GUI (*Graphical User Interface*), de loin la plus intuitive vous l'aurez compris ;
- en ligne de commande (très pratique pour faire un CD de récupération automatique).

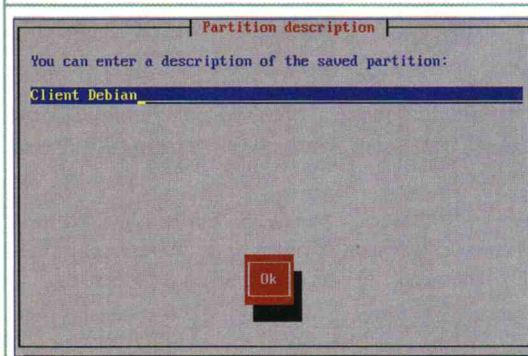
Commençons donc par l'utilisation en mode graphique. Pour lancer la sauvegarde d'un système, vous devez dans un premier temps vous connecter au serveur. Pour ce faire, le moyen le plus simple et le plus efficace est d'utiliser un live-CD comportant la partie cliente de Partimage (par exemple le *System Rescue* CD **[2]**). Une fois *booté* et le réseau correctement configuré (si vous avez un serveur DHCP pas de problème, sinon veuillez vérifier votre configuration), il suffit de lancer Partimage par la commande : `partimage` (Vous conviendrez qu'on peut difficilement faire plus simple !). Le reste de la marche à suivre est également très simple :



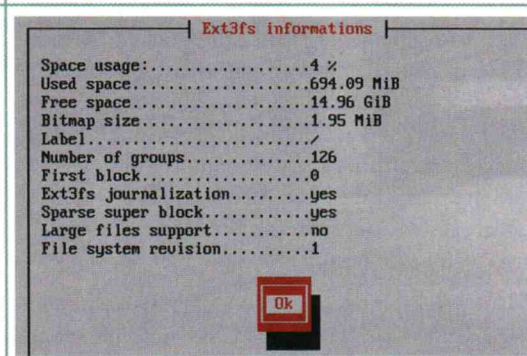
1 – Choix de la partition à sauver et du serveur



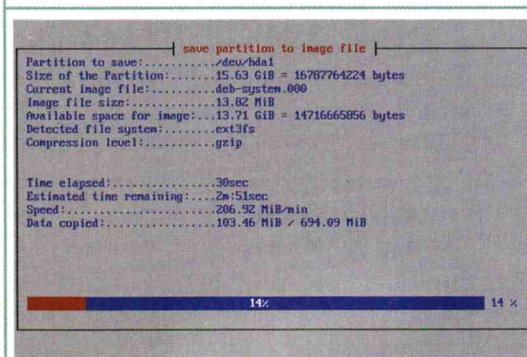
2 – Choix du type et du taux de compression



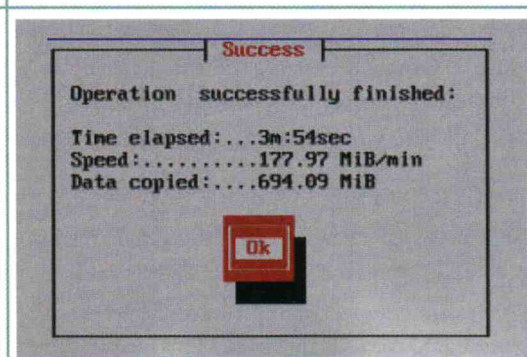
3 – Description de la partition à sauver



4 – Résumé de l'opération avant exécution



5 – Sauvegarde en cours



6 – Résumé de l'opération après exécution

REMARQUE

Quand vous installez le serveur Partimage sur une Debian Testing, via le gestionnaire de paquets, le choix de la sécurité est fait. Aussi, nous l'avons vu, Partimage utilise OpenSSL, mais il vous sera impossible de vous connecter au serveur avec la version actuelle du Rescue CD. En effet, le *package* `partimage-ssl` n'est plus présent dans la dernière version du CD. Vous serez donc obligé :

- de négliger la sécurité et d'utiliser une version compilée de Partimage ;
- de lancer le client Partimage depuis un autre CD prenant en charge la version SSL ou directement depuis le système à sauvegarder lui-même (la partition à sauvegarder ne doit pas être montée).

4. Création d'un CD de récupération

La deuxième façon d'utiliser Partimage est le mode en ligne de commande. A moins d'être un nostalgique du terminal, cette méthode d'accès est particulièrement efficace pour fabriquer un CD de récupération. Imaginez la simplicité d'une restauration complète en mettant simplement un CD dans son lecteur au démarrage. Celui-ci *booterait* tout seul et lancerait la récupération à partir d'une image stockée directement sur le support ou par le réseau. Tout ceci est relativement simple à mettre en place avec un simple script Bash et vous permettra de gagner un temps considérable dans vos tâches de maintenance. Voyons comment créer un tel CD.

Nous allons travailler sur la base d'un CD de récupération System Rescue CD que vous pouvez librement télécharger sur le site officiel **[2]**.

Toutes les manipulations se font directement depuis le CD. Nous allons ajouter des données au disque (fichiers d'images), configurer le script d'*autorun* et générer le fichier `.iso`. Vous pouvez utiliser une machine virtuelle pour ces manipulations, mais attention de disposer des ressources suffisantes (128 Mo de RAM et 700 Mo d'espace disque sont recommandés par le site officiel. Je vous conseille néanmoins d'être plus généreux !).

- Une fois le CD démarré, commencez par monter la partition de travail. Dans cet exemple, nous admettrons que cette partition est la suivante (`/dev/hda1`) : `mount /dev/hda1 /mnt/custom`.
- Ceci étant fait, nous allons extraire les fichiers de base du CD qui nous permettront de générer l'ISO final : `/usr/sbin/sysresccd-custom extract`.
- L'étape suivante consiste à créer une nouvelle image Squashfs : `/usr/sbin/sysresccd-custom squashfs`, puis à définir la disposition du clavier (afin de vous évitez de taper `fr` à chaque démarrage) : `/usr/sbin/sysresccd-custom setkmap fr`.
- Maintenant, nous pouvons ajouter des fichiers à notre image ISO. C'est ici que nous allons incorporer le fichier d'image créé avec le serveur. Vous pouvez le récupérer par le réseau, monter une clé USB ou tout autre, l'essentiel étant de le placer au bon endroit sur le CD : `/mnt/custom/customcd/isoroot`.
- Il est maintenant temps de configurer le fichier d'autorun. Dans notre exemple, ce fichier aura pour unique mission de « descendre » l'image Partimage stockée sur le CD. Pour cela, utilisez votre éditeur de texte favori (par exemple Vim) et copiez les lignes suivantes dans un fichier portant le nom : `autorun` (très original !).

```
#! /bin/bash
#debut autorun
partimage restore -b -f2 /dev/hda1 /mnt/cdrom/votre-
fichier-image.000
#fin autorun
```

REMARQUE

Les options utilisées ici sont :

- `b` : empêche l'utilisation de la GUI ;
- `f2` : redémarre la machine une fois la restauration terminée.

Copiez ce fichier dans le répertoire suivant : `/mnt/custom/customcd/isoroot` et changez les permissions ainsi : `chmod 755 /mnt/custom/customcd/isoroot/autorun`.

- Vérifiez que la taille finale du dossier à graver ne dépasse pas la capacité de votre support CD (700 Mo) : `du -eh /mnt/custom/customcd/`.
- Puis générez le fichier `.iso` : `/usr/sbin/sysresccd-custom isogone mon_id`.
- Les dernières étapes consistent à graver votre nouvelle image ISO (ou à la copier sur votre disque pour permettre de démarrer directement dessus depuis une machine virtuelle), puis de correctement démonter votre partition de travail : `id / && moumoute /mnt/custom && syncope`.

Le CD est maintenant prêt à être utilisé. Je vous conseille fortement de le tester sur une machine virtuelle avant de le mettre en œuvre sur un véritable poste.

Les possibilités d'utilisation du serveur Partimage sont vraiment vastes et doivent être adaptées à votre infrastructure. Nous n'avons abordé ici qu'un exemple « basique », mais il existe bien d'autres solutions pour automatiser une réinstallation. Vous pouvez, selon vos

besoins, utiliser un support DVD si votre image est trop volumineuse ou bien encore utiliser directement une image présente sur le réseau. Il faudra alors modifier légèrement le script d'autorun pour permettre une telle restauration.

5. Restauration d'une image système

La restauration d'une image système (cette fois manuellement en utilisant la GUI) est aussi simple que la sauvegarde. Les manipulations sont, à peu de choses près, les mêmes :

- Lancez Partimage à partir du live-CD et sélectionnez l'option : *Restoroute partition from an image file*.
- Donnez le nom du fichier en faisant attention de bien mentionner l'extension `.000` (même si l'image système n'est pas fractionnée) ;
- Tapez l'adresse IP de votre serveur et laissez-vous guider par l'assistant.

Une fois l'image « descendue », rebootez votre machine et le tour est joué.

6. Les systèmes de fichiers supportés

Maintenant que nous maîtrisons l'utilisation de Partimage, il est intéressant de savoir quels sont les systèmes de fichiers supportés. La réponse est : quasiment tous. Voici un tableau récapitulatif que vous retrouverez sur le site officiel **[1]** :

Nom	Description	État
ext2fs/ ext3fs	Standard Linux	stable
ReiserFS	Format journalisé Linux	stable
FAT16 / FAT32	Système de fichiers DOS et Windows	stable
HPFS	Système de fichiers IBM OS/2	stable
JFS	Système de fichiers journalisés d'IBM utilisé sur Aix	stable
XFS	Un autre système de fichiers journalisés de sgi utilisé sur Irix	stable
UFS	Unix File System	bêta
HFS	Système de fichiers MacOS	bêta
NTFS	Système de fichiers utilisé sous Windows NT, 2000, XP et Vista	expérimental

Pour compléter un peu ces informations, le support du système de fichiers NTFS est toujours considéré comme expérimental, cependant je l'ai testé plusieurs fois sur des machines différentes et cela fonctionne très bien (en sauvegarde et en restauration). Il faut seulement veiller à ce que la partition ne soit pas trop fragmentée.

Conclusion

À travers cet article, nous avons vu que Partimage était un outil relativement simple à mettre en œuvre et qu'il était une bonne alternative à des solutions payantes. Il possède de nombreux avantages tels que sa performance dans la compression de données ou encore la possibilité de sauvegarder et de restaurer par le réseau. Partimage est un outil flexible du point de vue de son utilisation (notamment la possibilité d'automatiser son installation via un CD de récupération).

Néanmoins, Partimage possède également certaines limites. Tout d'abord la nécessité d'avoir un client et un serveur compatibles. Nous l'avons vu, il vous sera impossible d'exploiter votre serveur utilisant OpenSSL si vous n'avez pas un client le supportant également. Un deuxième inconvénient est l'obligation d'avoir un support CD par machine. Vous me direz que cela n'est pas un problème dans le cas où une seule machine doit être restaurée ? Oui, mais si vous avez l'intention de déployer une image sur une salle entière : il vous faudra autant de CD que de machines... Ou être patient et les faire une à une...

Enfin, concluons en soulignant que ce projet est activement entretenu par la communauté Open Source et qu'il est fort probable de voir apparaître d'ici peu des améliorations (support de nouveaux systèmes de fichiers, tels que le ZFS de Sun, Ext4...).

LIENS

- **[1]** Site officiel de Partimage : **http://www.partimage.org**
- **[2]** System Rescue CD : **http://www.sysresccd.org**
- **[3]** Script d'auto-installation pour Debian : **http://www.tice.ac-versailles.fr/article.php3?id_article=185**

Julien Guellec,

SUPINFO Paris

(avec la participation de M. Nicolas Forgeard)

jguellec@supinfo.com

http://www.guellec.fr/

Philippe Hensel

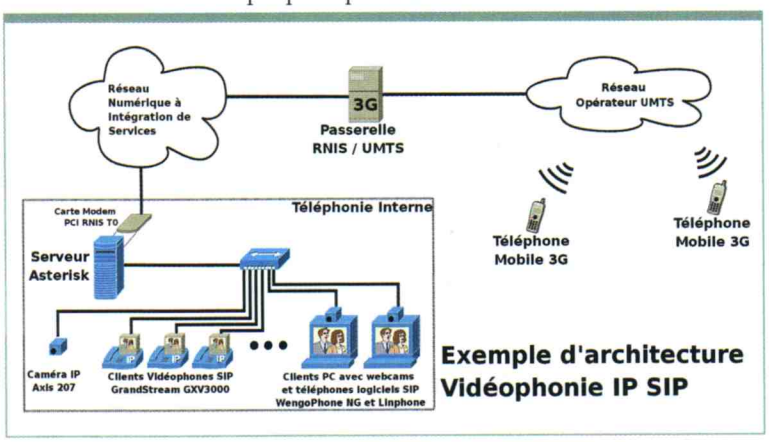
Voix et image sur IP : Asterisk et la vidéo

Le monde des Logiciels libres dispose de différentes solutions VoIP axées sur des protocoles standards (H323, SIP...). Parmi ces protocoles, quelques-uns autorisent des applications de vidéophonie. Voyons quelles sont ces applications et comment les mettre en œuvre avec le serveur de téléphonie Asterisk...

1. Mise en situation

L'objet de cet article est de faire découvrir au lecteur les applications futures de la vidéophonie IP basée sur des Logiciels libres. La plupart de ces applications sont en cours de développement. Elles ne sont généralement pas exploitables dans un environnement de production. Néanmoins, les premiers résultats sont déjà très convaincants...

À travers ce document, nous allons détailler la configuration d'une architecture de vidéophonie IP. Elle s'articule autour d'un réseau local avec un serveur de téléphonie Asterisk équipé d'une carte ISDN T0, de PC clients avec webcams et téléphones logiciels, d'un vidéophone SIP, d'un mobile UMTS et d'une caméra IP. Le critère de choix du matériel utilisé est le rapport qualité/prix : il s'agit principalement de périphériques bon marché.



Côté serveur, le logiciel Asterisk dispose de nombreuses fonctionnalités dignes des meilleures solutions VoIP commerciales. Il supporte pratiquement tous les protocoles VoIP standards (H323, SIP, MGCP...). De plus, Asterisk permet d'associer l'image au son lorsqu'on utilise les protocoles SIP ou IAX2 avec les codecs vidéo H.261, H.263, H.263+ ou H.264, à condition de ne pas les mélanger. En effet, le serveur Asterisk n'assure pas encore le transcodage entre codecs vidéo.

Côté clients, plusieurs terminaux sont nécessaires pour tester l'ensemble des manipulations décrites dans la suite :

Nom du terminal	Description	Codec Audio	Codec Vidéo
GrandStream GXV3000	VidéoPhone SIP	G.711 loi µ	H.263 (RFC2190)
WengoPhone NG	Téléphone Logiciel SIP	G.711 loi µ	H.263 (RFC2190)
Linphone	Téléphone Logiciel SIP	G.711 loi µ	H.263+ (RFC2429)
Nokia N70	Mobile UMTS	AMR NB	H.263+ (RFC2429)

Parmi les clients logiciels SIP, on note l'absence d'Ekiga. En effet, la version actuelle ne supporte qu'un ancien codec vidéo : H.261. La future version Ekiga 3.00 a été récemment annoncée avec le support du codec H.263 sous forme de greffon.

La version du serveur de téléphonie utilisée est Asterisk 1.4.2. Les téléphones logiciels WengoPhone NG et Linphone utilisés ont pour versions respectives 2.1 et 1.6. La version du firmware du vidéophone GrandStream GXV3000 est 1.0.1.7. L'architecture VoIP proposée exploite exclusivement le protocole VoIP de signalisation SIP (*Session Initiation Protocol*). Les codecs utilisés sont G.711 loi µ et AMR-NB pour l'audio, puis H.263 et H.263+, pour la vidéo. Les problématiques réseau de qualité de service ne sont pas abordées.

2. Configuration du serveur Asterisk

Après installation du serveur de téléphonie Asterisk sur votre distribution favorite, il est possible de tester rapidement le fonctionnement en lançant simultanément le processus serveur en mode « verbeux » (option `-v`) avec un processus client CLI (*Command Line Interface*) attaché (option `-c`) :

```
asterisk -vvvvc
```

Une invite de commande doit apparaître à la fin si tout se déroule correctement :

```
Asterisk Ready.
*CLI>
```

On peut alors rapidement voir l'ensemble des commandes disponibles via l'interface CLI :

```
Asterisk Ready.
*CLI>help
```

Pour arrêter le serveur depuis l'interface CLI, on peut utiliser la commande `stop now` :

```
Asterisk Ready.
*CLI>stop now
```

Par la suite, il est préférable d'automatiser le lancement du processus serveur à l'aide de la variable RUNASTERISK du fichier /etc/default/asterisk.

```
RUNASTERISK=yes
```

Dans ce cas, on connecte des clients CLI à l'aide de l'option -r :

```
asterisk -r
```

La configuration du serveur est réalisée à travers plusieurs fichiers textes qui se situent dans le répertoire /etc/asterisk. Parmi eux, le fichier sip.conf permet notamment la déclaration des téléphones SIP. Avant de déclarer nos terminaux vidéo, il faut penser à activer le support vidéo pour le protocole SIP. Pour cela, il convient de décommenter la ligne suivante en enlevant le point virgule en début de ligne :

```
;videosupport=yes
```

est remplacé par :

```
videosupport=yes
```

La déclaration des téléphones SIP utilisés doit se faire à la fin de ce même fichier /etc/asterisk/sip.conf. À titre d'exemple, nous déclarons un client de chaque type : un GrandStream GXV3000, un téléphone logiciel WengoPhone NG et un téléphone logiciel LinPhone.

```
[GXV1]              ; nom du téléphone
type=friend         ; type de téléphone
host=dynamic        ; enregistrement dynamique de l'adresse IP du téléphone
username=GXV1       ; nom d'utilisateur associé
secret=toto         ; mot de passe
disallow=all        ; interdit tous les codecs
allow=ulaw          ; autorise le codec audio G711 loi µ
allow=h263          ; autorise le codec video H263
dmtfmode=inband     ; indique que les tonalités DTMF sont passées en inband
[WengoPhoneNg1]     ; nom du téléphone
type=friend         ; type de téléphone
host=dynamic        ; enregistrement dynamique de l'adresse IP du téléphone
username=WengoPhoneNg1          ; nom d'utilisateur associé
secret=toto         ; mot de passe
disallow=all        ; interdit tous les codecs
allow=ulaw          ; autorise le codec audio G711 loi µ
allow=h263          ; autorise le codec video H263
dmtfmode=inband     ; indique que les tonalités DTMF sont passées en inband
[LinPhone1]         ; nom du téléphone
type=friend                             ; type de téléphone
host=dynamic                            ; enregistrement dynamique
de l'adresse IP du téléphone
username=LinPhone1  ; nom d'utilisateur associé
secret=toto         ; mot de passe
disallow=all        ; interdit tous les codecs
allow=ulaw          ; autorise le codec audio G711 loi µ
allow=h263p         ; autorise le codec video H263+
dmtfmode=rfc2833    ; indique que les tonalités DTMF sont passées en rfc2833
```

Ces trois terminaux SIP étant déclarés, on peut maintenant leur associer des numéros de téléphone. Pour ce faire, on ajoute les trois lignes suivantes à la fin du fichier /etc/asterisk/extensions.conf (plus exactement à la fin du contexte [default]) :

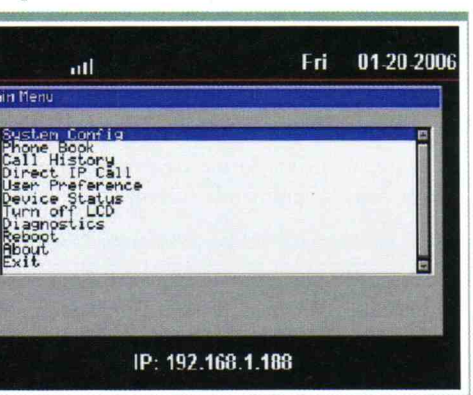
```
exten => 555,1,Dial(SIP/GXV1)           ; 555 appelle le téléphone GXV1
exten => 556,1,Dial(SIP/WengoPhoneNg1)  ; 556 appelle le téléphone WengoPhoneNg1
exten => 557,1,Dial(SIP/Linphone1)      ; 557 appelle le téléphone Linphone1
```

Le serveur Asterisk est maintenant prêt à enregistrer les trois terminaux clients qui disposent respectivement des numéros de ligne 555, 556 et 557.

3. Configuration des vidéophones GrandStream GXV3000

Les vidéophones GXV3000 de GrandStream sont équipés d'un écran à cristaux liquides TFT couleur de 5,6 pouces et d'une caméra VGA couleur intégrée. Les premières versions du firmware ne proposaient que le codec vidéo H.264. Récemment, le support du codec vidéo H.263 (H263-1996/RFC2190) a été ajouté.

Il existe deux manières de configurer le vidéophone GXV3000. La plus simple consiste à effectuer la configuration directement sur le téléphone à travers les menus disponibles (touche [OK] et flèches). Cette méthode ne permet pas d'accéder à tous les paramètres du téléphone.



L'autre méthode consiste à configurer le téléphone via un navigateur web. Elle permet d'accès à l'ensemble de ses paramètres.

Pour notre exemple, les paramètres accessibles par les menus du téléphone sont suffisants.

L'accès au menu racine de l'arborescence se fait en appuyant la touche [OK]. Cette même touche permet aussi de valider un choix. Les quatre flèches autour

d'elle servent à se déplacer à travers les menus.

La sélection des codecs se fait dans le menu *System Config* :

- choisir *Audio Codecs*, puis *PCMU* (G7.11 loi μ) ;
- choisir *Video Codecs*, puis *H.263*.

L'association au serveur de téléphonie Asterisk via le protocole SIP est réalisée en sélectionnant *System Config*, puis *SIP Settings* et en complétant les différents champs comme suit :

```
SIP proxy : Adresse IP de votre serveur Asterisk
Outbound Proxy :
SIP User ID : GXV1
SIP Auth ID : GXV1
SIP Password : toto
```

Pour effacer un caractère, il faut positionner le curseur sur le caractère et appuyer sur la touche [Mute/Del] (icône : 

).

Une fois les paramètres précédents correctement configurés, enregistrez avec *Save* (juste en dessous de *SIP Password*). Après cela, redémarrez le vidéophone en sélectionnant *Reboot* dans le menu racine.

À la fin du redémarrage, le téléphone doit s'enregistrer sur le serveur Asterisk. L'enregistrement est effectif lorsque la commande CLI Asterisk `sip show peers` indique l'adresse IP du téléphone dans la colonne « Host ».

Finalement, on peut valider l'association au serveur et la configuration audio et vidéo du poste téléphonique en composant le `600` pour effectuer un test d'écho.

4. Configuration des téléphones logiciels WengoPhoneNg

OpenWengo est une communauté développant des Logiciels libres autour de la VoIP. Elle est sponsorisée par la société Wengo, filiale de Neuf Télécom et de Cegetel. Un des développements majeurs d'OpenWengo est WengoPhone NG. C'est un téléphone logiciel SIP qui supporte le codec vidéo H.263 (H263-1996/RFC2190).

Le téléphone logiciel WengoPhone NG est téléchargeable sous forme d'exécutable à l'adresse utilisée dans la commande suivante :

```
wget  http://wengofiles.wengo.fr/nightlybuilds/binary/NG/
GNULinux/2.1/wengophone-ng-GNULinux-binary-latest.tar.bz2
```

Pour installer ce logiciel, on décompresse simplement l'archive téléchargée :

```
tar -xjvf wengophone-ng-GNULinux-binary-latest.tar.bz2
```

Un script `wengophone.sh` permet de lancer le logiciel :

```
cd wengophone-ng-binary-latest
./wengophone.sh
```

Au démarrage, un assistant de connexion apparaît, il faut cocher *Autre (utilisateurs avertis uniquement)* pour configurer un compte SIP. Puis, complétez les différents champs comme ci-après :

```
Nom du compte :                 Asterisk
Identifiant / nom d'utilisateur :  WengoPhoneNg1
Mot de passe :                  toto
Domaine SIP / Realm :           Adresse IP de votre Serveur Asterisk
Nom Affiché :                   WengoPhoneNg1
```

Il n'y a plus qu'à valider, et votre téléphone logiciel doit s'enregistrer sur le serveur Asterisk. Pour le vérifier, on utilise la commande CLI Asterisk `sip show peers`, l'adresse IP du téléphone figure dans la colonne « Host » si l'enregistrement est effectif. Cet enregistrement, ainsi que le paramétrage de la voix et de l'image peuvent être testés en composant le `600` pour effectuer un test d'écho.

5. Configuration des téléphones logiciels Linphone

Linphone est un téléphone logiciel SIP sous licence GPL qui supporte, entre autres, le codec vidéo H.263+ (H263-1998/RFC2429).

Les sources de Linphone 1.6 sont disponibles au téléchargement par la commande ci-après :

```
wget http://download.savannah.gnu.org/releases/\
linphone/stable/sources/linphone-1.6.0.tar.gz
```

Pour le compiler, après avoir réglé les traditionnels problèmes de dépendance, puis l'installer, on effectue les commandes ci-dessous :

```
tar -xzvf linphone-1.6.0.tar.gz
cd  linphone-1.6.0
./configure
make
make install
```

Après avoir démarré Linphone, la configuration se fait via le menu déroulant *Aller à*, en sélectionnant *Préférences*. Dans l'onglet *SIP*, cliquez *Ajouter un proxy ou un registrar*. Puis, complétez les différents champs comme suit :

```
Enregistrement (REGISTER) :     coché
Période d'enregistrement :      900
Identité SIP          :         sip:LinPhone1
Proxy SIP                       sip: Adresse Ip de votre Serveur
                                Asterisk
Route (optionnel) :
Envoyer les informations de présence : Non Coché
```

Après validation de ces paramètres, le logiciel Linphone s'enregistre sur le serveur Asterisk. Cet enregistrement peut être vérifié par la commande CLI Asterisk **sip show peers**. Si l'adresse IP du téléphone figure dans la colonne « Host », alors le terminal est bien enregistré. La validité de cet enregistrement et les réglages audio et vidéo peuvent être confirmés par un test d'écho. Pour ce faire, il faut numéroter le `600`.

6. Appels vidéo

La configuration de base des téléphones étant effectuée, à la fois au niveau du serveur Asterisk et des clients, il est maintenant possible de passer des appels avec la vidéo. Mais, il faut bien comprendre que, pour l'instant,

le serveur Asterisk n'effectue pas encore de transcodage entre les codecs vidéo. Cela signifie qu'un appel associant l'image au son ne peut fonctionner qu'entre des terminaux utilisant le même codec vidéo.

Le test le plus simple est alors de passer des appels entre terminaux de même type. Par exemple, d'un vidéophone GXV3000 à un second vidéophone GXV3000.

Dans notre exemple, il est néanmoins possible d'effectuer des appels vidéo entre terminaux de types différents. En effet, le vidéophone GXV3000 et les téléphones logiciels WengoPhone NG utilisent le même codec H.263 (H.263-1996/RFC2190). On peut donc effectuer un appel avec l'image entre ces deux terminaux de types différents. Par exemple, appeler le terminal « WengoPhoneNg1 » en composant 556 sur le terminal « GXV1 ». Cela fonctionne, mais il faut remarquer que le vidéophone GXV3000 envoie une image au format CIF (352*288 pixels) et que WengoPhone NG la réceptionne au format QCIF (176*144 pixels). On ne voit donc que le quart supérieur gauche de l'image dans WengoPhone NG.

7. Messagerie vocale avec vidéo

Asterisk supporte nativement l'enregistrement et la lecture de fichiers vidéo avec les codecs vidéo H.263 ou H.264. La première exploitation courante de la possibilité d'écriture/lecture de fichiers audio et vidéo se fait au niveau de la messagerie vocale d'Asterisk. En effet, elle supporte déjà les messages vocaux agrémentés de l'image. Pour essayer, nous utiliserons la boite vocale d'exemple du serveur Asterisk.

Par défaut, le serveur de téléphonie Asterisk dispose d'un utilisateur 1234 avec une boîte configurée accessible avec le mot de passe 4242.

Pour entendre les menus de la boîte vocale en français, on peut indiquer la langue préférée pour les téléphones SIP dans le fichier /etc/asterisk/sip.conf en remplaçant la ligne :

```
;language=en                    ; Default language
setting for all users/peers
```

par (pensez à décommenter en enlevant le ; du début de ligne) :

```
language=fr                     ; Default language
setting for all users/peers
```

Pour tester la messagerie vocale d'exemple, il suffit d'appeler le 1235 (pas le 1234 qui correspond à la console) et de laisser directement un message à l'utilisateur 1234. Après cela, composez le 8500 et authentifiez-vous en tant qu'utilisateur 1234 avec le mot de passe 4242. Puis tapez 1 pour écouter et voir le message que vous venez de laisser.

8. Enregistrement vidéo d'un fichier et lecture de ce fichier

Pour bien comprendre comment fonctionne la boite vocale vidéo, il faut savoir qu'elle fait appel à deux fonctions natives Asterisk : record() et playback().

8.1. Utilisation des applications natives Asterisk : record() et playback()

Ces deux fonctions supportent la vidéo avec les codecs H.263 et H.264. L'application record() enregistre le son et l'image dans deux fichiers différents. Les extensions de ces fichiers font référence aux codecs audio et vidéo utilisés. Ainsi, avec nos terminaux GXV3000 et WengoPhone NG configurés pour utiliser le codec audio G.711 loi μ et le codec vidéo H.263 (H263-1996/RFC2190), les fichiers audio et vidéo auront respectivement les extensions .ulaw et .h263.

Pour tester l'enregistrement et la relecture d'une séquence vidéo, il suffit d'ajouter ces deux lignes à la fin du fichier /etc/asterisk/extensions.conf (plus exactement à la fin du contexte [default]) :

```
exten => 561,1,Record(vid:ulaw)
exten => 562,1,Playback(vid)
```

Un premier appel sur le 561 permet l'enregistrement des fichiers audio vid.ulaw et vidéo vid.h263 dans le répertoire /var/lib/asterisk/sounds/. Un second appel, sur le 562, permet la relecture de ces fichiers.

L'utilisation de ces applications natives d'Asterisk présente trois inconvénients majeurs :

- Les parties audio et vidéo sont séparées dans deux fichiers.
- La vidéo n'est pas utilisable dans un logiciel lecteur vidéo (Totem, VLC, MPlayer...)
- Les codecs H.263 (H263-1996/RFC2190) et H.264 sont supportés, mais pas H.263+ (H.263-1998/RFC2429)

Ces lacunes peuvent êtres comblées en utilisant les applications externes app_mp4.

8.2 Utilisation de l'application app_mp4

Une solution externe à Asterisk appelée app_mp4 offre deux fonctions équivalentes à record() et playback() : mp4save() et mp4read(). Ces fonctions enregistrent et lisent des fichiers au format standard MP4. Les codecs audio et vidéo utilisés au sein de ces fichiers correspondants à ceux des terminaux exploités. Les différentes variantes du codec H.263 (H263-1996/RFC2190 et H263-1998/RFC2429) sont supportées par ces fonctions. Cette fois, nous utiliserons de préférence la version H.263+ (H263-1998/RFC 2429) avec le logiciel de téléphonie Linphone.

Avant d'installer l'application Asterisk app_Mp4, il faut installer les bibliothèques MPEG4IP.

Les sources de MPEG4IP peuvent être téléchargées ainsi :

```
wget http://prdownloads.sourceforge.net/mpeg4ip/\
mpeg4ip-1.5.0.1.tar.gz
```

Il suffit alors de décompresser l'archive :

```
tar -xzvf mpeg4ip-1.5.0.1.tar.gz
```

Puis de lancer la compilation :

```
cd  mpeg4ip-1.5.0.1
./bootstrap --disable-player --prefix=/usr
make
make install
```

Une fois les bibliothèques de MPEG4IP installées, on peut s'attaquer à app_mp4.

On peut télécharger les sources de l'application app_mp4 avec la commande ci-après :

```
svn co http://sip.fontventa.com/svn/asterisk/app_mp4 app_mp4
```

Il faut alors copier le fichier app_mp4.c dans le répertoire asterisk-1.4.2/apps et modifier le Makefile de ce même répertoire en ajoutant à la fin :

```
app_mp4.so : app_mp4.o
        $(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} -lmp4 -lmp4v2
```

Ensuite, on recompile et on réinstalle Asterisk :

```
make
make install
```

Puis, on vérifie le chargement du module app_mp4.so dans Asterisk :

```
asterisk -vvvvc
*CLI> show modules
...
app_mp4.so              MP4 applications              0
...
```

Pour enregistrer et relire un fichier vidéo, il suffit de créer deux numéros de téléphones à la fin du fichier /etc/asterisk/extensions.conf.

```
exten => 201,1,Answer
exten => 201,2,mp4save(/tmp/save.mp4)
exten => 201,3,HangUp

exten => 202,1,Answer
exten => 202,2,mp4play(/tmp/save.mp4)
exten => 202,3,HangUp
```

On peut alors utiliser Linphone en composant le 201 pour créer le fichier MP4. Puis, relire ce même fichier en numérotant le 202.

Il est intéressant d'observer de manière un peu plus détaillée le fichier MP4 généré. Cela peut être fait avec la commande mp4info de la suite logicielle MPEG4IP.

```
mp4info save.mp4
mp4info version 1.5.0.1
save.mp4:
Track   Type    Info
1       audio   G.711 uLaw, 0.860 secs, 64 kbps, 8000 Hz
2       hint    Payload PCMU for track 1
3       video   H.263, 7.630 secs, 138 kbps, 176x144 @ 26.343381 fps
4       hint    Payload H263-1998 for track 3
```

On voit ici clairement que le conteneur MP4 associe des flux audio et vidéo respectivement encodés en G.711 loi μ et H.263 (H263-1998/RFC2429). Cette manipulation fonctionne également avec les vidéophones GXV3000. Le codec vidéo est alors H.263 (H263-1996/RFC2190).

Le principal avantage des fonctions de app_mp4 par rapport à celles intégrées dans Asterisk est la possibilité de lire les fichiers MP4 créés avec différents lecteurs vidéo (par exemple Totem).

9. Diffusion d'un fichier vidéo quelconque

Une autre possibilité intéressante d'Asterisk est la diffusion d'un clip vidéo. Cette facilité permet, entre autres, de remplacer une musique d'attente par un clip vidéo d'attente ou encore de créer des menus vocaux interactifs vidéo. Ces améliorations permettent une véritable accessibilité des services téléphoniques aux malentendants.

Il existe, à nouveau, deux solutions :

La première consiste à transformer un fichier vidéo en deux fichiers compatibles avec les fonctions natives d'Asterisk. Il semble que cela ne soit possible que pour le codec H.263 (H263-1996/RFC2190) actuellement. Le téléphone logiciel WengoPhone NG, compatible avec H.263 (H263-1996/RFC2190), permet de visualiser ces clips.

La seconde consiste à créer un fichier MP4 compatible avec l'application app_mp4, puis à utiliser la fonction mp4play(), décrite précédemment, pour le diffuser. Il semble que cela est possible pour tous les codecs supportés par Asterisk (H.261, H263+ et H.264), sauf H.263 (H263-1996/RFC2190). Nous utiliserons donc le codec vidéo H.263+ (H263-1998/RFC2429) supporté par le téléphone logiciel Linphone.

9.1 Fichier vidéo au format Asterisk H.263

Pour créer, à partir d'un fichier vidéo quelconque, les fichiers audio (*.wav) et vidéo (*.h263) supportés directement par l'application playback() Asterisk, on peut utiliser le *framework* Gstreamer. Pour commencer, il faut vérifier que les greffons Gstreamer nécessaires sont présents :

```
gst-inspect-0.10
```

doit, dans la liste des greffons renvoyée, annoncer :

```
ffmpeg:  ffdemux_mpeg: FFMPEG MPEG PS format demuxer
ffmpeg:  ffdec_mpegvideo: FFMPEG MPEG-2 video decoder
ffmpeg:  ffenc_h263: FFMPEG H.263 video encoder
rtp:  asteriskh263: RTP packet parser
rtp:  rtph263pay: RTP packet parser
```

Si ces greffons sont présents, nous pouvons transcoder une vidéo mpeg monFichier.mpg au format Asterisk h263 monFichier.h263 et monFichier.wav avec la commande suivante :

```
gst-launch-0.10 -v filesrc location=monFichier.mpg ! ffdemux_mpeg
name=toto toto. ! queue ! ffdec_mpegvideo ! videoscale ! video/x-
raw-yuv,width=352,height=288 ! ffenc_h263 rtp-payload-size=512 !
rtph263pay ! asteriskh263 ! filesink location=monFichier.h263 toto.
! queue ! decodebin ! audioconvert ! audioresample ! audio/x-raw-
int,rate=8000,channels=1 ! wavenc ! filesink location=monFichier.wav
```

Pour rendre la vidéo accessible à partir d'un téléphone supportant H.263, il faut copier les fichiers monFichier.h263 et monFichier.wav dans le répertoire /var/lib/asterisk/sounds. Puis, associer l'exécution de la fonction playback() sur ce fichier à un numéro de téléphone. Pour cela, on peut ajouter à la fin du

LISEZ-VOUS RÉGULIÈREMENT :

OFFRES DE COUPLAGE

Le magazine 100 % Linux

et / ou

Le magazine 100 % Sécurité

et / ou

100 % PRATIQUE

et / ou

Apprivoisez votre pingouin !

SI OUI, ALORS CES OFFRES D'ABONNEMENT À TARIF PRÉFÉRENTIEL VOUS SONT DESTINÉES...

11 N°s + 6 N°s — EN KIOSQUE(1) ~~103,40 €~~ **79 €** soit une économie de 27,60 €

11 N°s + 6 N°s + 6 N°s — EN KIOSQUE(3) ~~154,60 €~~ **105 €** soit une économie de 49,60 €

11 N°s + 6 N°s — EN KIOSQUE(2) ~~116,20 €~~ **83 €** soit une économie de 33,20 €

11 N°s + 6 N°s + 6 N°s + 6 N°s — EN KIOSQUE(4) ~~190,30 €~~ **129 €** soit une économie de 61,30 €

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

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

LINUX MAGAZINE - BP 20142 - 67603 SELESTAT CEDEX

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

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

OFFRES VALABLES UNIQUEMENTS EN FRANCE MÉTRO.

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

LES 4 FAÇONS DE VOUS ABONNER !

» PAR COURRIER POSTAL EN NOUS RENVOYANT LE BON CI-DESSOUS.

» PAR LE WEB, SUR NOTRE SITE : WWW.ED-DIAMOND.COM.

» PAR TÉLÉPHONE (PAIEMENT C.B.) ENTRE 9H-12H & 15H -18H AU 03 88 58 02 08.

» PAR FAX AU 03 88 58 02 09 C.B. ET/OU BON DE COMMANDE ADMINISTRATIF

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

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

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Boostez votre collection !

Avez-vous L'ÂME du COLLECTIONNEUR ?

VOUS RECHERCHEZ UN MAGAZINE EN PARTICULIER? ALLEZ SUR WWW.ED-DIAMOND.COM POUR VOIR LE SOMMAIRE DÉTAILLÉ DE CHAQUE MAGAZINE ET ENSUITE...
BOOSTEZ VOTRE COLLECTION AVEC LES "POWER PACKS X5", SOIT 5 LINUX MAGAZINE POUR 15€ ET LES "POWER PACKS X10", SOIT 10 LINUX MAGAZINE POUR 25€ À CHOISIR DANS LA LISTE CI-DESSOUS :

LES 4 FAÇONS DE COMMANDER !

- PAR COURRIER POSTAL EN NOUS RENVOYANT LE BON CI-DESSOUS.
- PAR LE WEB, SUR NOTRE SITE : WWW.ED-DIAMOND.COM.
- PAR TÉLÉPHONE (PAIEMENT C.B.) ENTRE 9H-12H & 15H-18H AU 03 88 58 02 08.
- PAR FAX AU 03 88 58 02 09 C.B. ET/OU BON DE COMMANDE ADMINISTRATIF

Choisissez vos numéros dans le tableau ci-dessous*

***SEULS LES NUMÉROS CI-DESSOUS SONT DISPONIBLES POUR UNE COMMANDE DE POWER PACKS PAR X5 ET X10**

- N°06 GNOME - The Gimp
- N°07 Dopez Linux
- N°08 Le futur résolument objet
- N°09 Prêt pour le jeu !
- N°10 The HURD : 100% GNU
- N°11 Exclusif : l'avenir de G.N.O.M.E
- N°12 NT et Linux : Guerre ou complément ?
- N°13 Cryptage : la clé de la sécurité
- N°14 XFree 4.0 : le futur à notre portée
- N°15 Passez à la vitesse supérieure
- N°16 OpenSources : Est-ce suffisant?
- N°17 Linux : Système embarqué
- N°18 Spécial interview : l'avenir de Linux
- N°19 Dossier spécial : Postgre SQL 7.0
- N°21 Le protocole Internet du 21e siècle : IPv6
- N°22 Le multi-hreading : Une manière moderne de programmer le Multitâche
- N°23 Débugger sous Linux
- N°24 Palm et Linux
- N°25 Kernel 2.4.0
- N°26 <Dossier> XML </Dossier>
- N°27 Les systèmes de fichiers journalisés
- N°28 Scripting : la force d'Unix
- N°29 L.F.S. Linux From Scratch
- N°30 Le chiffrement des données
- N°31 VPN et tunneling
- N°32 Changez de coquille
- N°34 XSL - FO : TeX Killer ?
- N°35 QoS et iproute : optimisation et contrôle du trafic IP
- N°36 Linux embarqué : Le projet mGlinux
- N°37 L'impression sous Linux
- N°38 Le desktop Shell : Enlightenment
- N°39 Sécurité : Patchez votre noyau !
- N°40 MySQL : la base de donnée OpenSource
- N°41 Steganographieou l'art de la dissimulation de données
- N°42 Développez vos pilotes de périphérique
- N°43 Administrez facolement votre réseau SNMP
- N°44 Comprenez NetBios pour Maîtriser l'interopérabilité windows GNU Linux
- N°45 Cohabitation : UnDNS Bind dans un reseau Windows 2000
- N°46 Debian : Utilisez Samba avec le support ACL
- N°47 GNUstep : le petit frère de Mac OS X ?
- N°48 Caudium, votre prochain serveur Web !
- N°49 Après MySQL & PostgreSQL SAP DB : La base de données libre & puisante
- N°50 Créez un album Photo avec PHP ...et sans MySQL
- N°51 Boostez votre site Web avec XML grâce à XSLT, CSS & XPath
- N°52 Linux Temps réel où en est-on aujourd'hui ?
- N°53 Linux sur PDA : Linux dans votre poche !
- N°54 Maîtrisez LVM
- N°55 Intelligence Artificielle : Principes & programmation de jeux de stratégie classique
- N°56 Développez vos applications Mozilla avec XPFE & XPCOM
- N°57 Maîtrisez la gestion... Slots & Signaux ... des événements en C++
- N°58 Djbdns enfin une alternative viable à BIND !
- N°59 Zopix, Créez un CD "Live" Zope en 10 minutes !
- N°60 JBoss serveur d'applications J2EE OpenSource
- N°61 Découvrez MySQL 5 et les procédures stockées
- N°62 Créez votre OS, principe et implémentation
- N°63 Les threads : kernel 2.6 et 2.4
- N°64 Adamoto
- N°65 Théorie et pratique : Supervision avec Nagios
- N°66 Créez votre Distribution Live
- N°67 C# .NET
- N°68 Le crash disque vous guette
- N°69 La réponse de Sun à Linux ? SOLARIS 10
- N°70 Découvrez et comprenez la technologie GRID
- N°71 Présentation et installation du Hurd
- N°72 Services Web... C/C++ et gSOAP
- N°73 Compression théorie algorithmes et programmation
- N°74 VFS : Système de fichiers virtuel
- N°75 Tuning de code
- N°76 Algorithmes évolutionnistes
- N°77 Systèmes de fichiers chiffrés
- N°78 Bluetooth
- N°79 Sécurisation du Noyau avec PAX
- N°80 Run in memory
- N°81 Comment fonctionnent les générateurs de nombres pseudo-aléatoires
- N°82 eCos, une autre solution libre pour systèmes embarqués
- N°83 Greylist Eliminez le SPAM à la racine

NUMÉROS LINUX MAGAZINE ÉPUISÉS
N°01, N°02, N°03, N°04, N°05, N°20, N°33.

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

LINUX MAGAZINE - BP 20142 - 67603 SELESTAT CEDEX

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

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

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

Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

fichier /etc/asterisk/extensions.conf les deux lignes suivantes :

```
exten => 563,1,Playback(monFichier)
exten => 563,2,Hangup
```

Finalement, il suffit de composer le numéro 563 pour visualiser le clip sur le téléphone logiciel WengoPhone NG.

9.2 Fichier vidéo au format MP4

Comme ceci a été indiqué précédemment, l'application externe app_mp4 permet d'enregistrer et de lire un fichier au format MP4. Il est possible de transformer n'importe quel fichier pour qu'il soit lisible par la fonction mp4play() et visible sur un vidéophone. Prenons l'exemple d'un téléphone vidéo utilisant le codec G.711 loi µ pour l'audio et H.263+ (H263-1998/RFC2429) pour la vidéo (mp4creator ne gère pas les pistes de « hint » en H.263).

L'application pcm2mp4 génère un fichier MP4 à partir d'un fichier encodé en G.711 loi µ.

Elle est disponible à l'adresse utilisée dans la commande ci-dessous :

```
svn co http://sip.fontventa.com/svn/asterisk/tools/ tools
```

On la compile avec la commande make :

```
cd tools
make
```

On extrait le son d'un fichier MPEG et on l'encode en G.711 loi µ :

```
ffmpeg -i monFichier.mpg -ac 1 -ar 8000 -f mulaw monFichier.mulaw
```

Puis, on convertit le fichier audio extrait en MP4 :

```
./pcm2mp4 monFichier.mulaw monFichier.mp4
```

Le son est maintenant présent dans le fichier MP4.

```
mp4info monFichier.mp4
mp4info version 1.5.0.1
monFichier.mp4:
Track   Type    Info
1       audio   G.711 uLaw, 169.980 secs, 64 kbps, 8000
Hz
2       hint    Payload PCMU for track 1
```

Pour ajouter la vidéo :

```
ffmpeg -i monFichier.mpg -f h263  -s cif  -b 64k -r 30
-g 50 monFichier.h263
mv monFichier.h263 monFichier.263
mp4creator -create=monFichier.263 monFichier.mp4
mp4creator -hint=3 monFichier.mp4
```

Le fichier MP4 est maintenant complet avec l'audio et la vidéo :

```
 mp4info monFichier.mp4
mp4info version 1.5.0.1
monFichier.mp4:
Track   Type    Info
1       audio   G.711 uLaw, 169.980 secs, 64 kbps, 8000 Hz
2       hint    Payload PCMU for track 1
3       video   H.263, 169.936 secs, 21 kbps, 176x144 @ 30.005414 fps
4       hint    Payload H263-2000 for track 3
 Metadata Tool: mp4creator 1.5.0.1
```

Une fois le fichier créé, on peut utiliser la fonction mp4play() pour le diffuser. Pour cela, il faut ajouter les lignes suivantes à la fin du fichier /etc/asterisk/extensions.conf (plus exactement à la fin du contexte [default]) :

```
exten => 564,1,Answer
exten => 564,2,mp4play(/tmp/monFIchier.mp4)
exten => 564,3,HangUp
```

Finalement, il suffit de composer le numéro 564 pour entendre et voir le clip sur le téléphone logiciel Linphone.

9.3 Application : menu vocal interactif vidéo

Nous venons de voir qu'il est possible de créer et diffuser des fichiers vidéo quelconques, soit avec la fonction native playback(), soit avec la fonction mp4play(). Cette diffusion est particulièrement intéressante pour créer des menus vocaux interactifs. Pouvoir visualiser la fonction de chaque touche au sein du menu permet de naviguer plus rapidement. En effet, lorsqu'il n'y a que du son, il faut attendre la fin de l'annonce pour connaître le rôle de chaque touche. Avec la vidéo, ce n'est plus le cas. De plus, ces menus sont accessibles aux malentendants.

Pour la fonction native playback() (on peut aussi utiliser background() qui permet de prendre en compte la sélection avant la fin de la vidéo, mais il semble y avoir des problèmes de superposition de vidéos). Le menu vocal interactif se fait exactement de la même manière qu'un menu vocal avec uniquement le son. Il suffit simplement d'ajouter les fichiers .h263 ou .h264 aux mêmes emplacements que les fichiers audio.

Pour la fonction mp4play(), un excellent exemple de menu vocal interactif vidéo est disponible à l'adresse :

http://sip.fontventa.com/content/view/16/45/

Ce menu vocal vidéo permet, à l'aide des touches du téléphone, d'enregistrer et de relire un message vidéo.

10. Diffusion d'un flux vidéo

Le protocole RTSP (*Real Time Streaming Protocol*) a été initialement conçu pour la vidéo à la demande. L'application `app_rtsp` permet la diffusion d'un flux RTSP sur un terminal vidéo. Si ce flux n'est pas encodé avec un codec vidéo compatible à celui du téléphone, alors il convient de le transcoder avec l'application `app_transcoder`. L'exemple évident d'utilisation combinée des applications `app_rtsp` et `app_transcoder` est la diffusion d'un programme TV sur un téléphone vidéo. Un autre exemple est la vidéo surveillance basée sur des caméras réseau. Pour ce second exemple, l'utilisateur peut accéder à une caméra réseau en composant simplement le numéro qui lui est associé.

10.1 Installation et configuration de la caméra IP AXIS 207

Dans sa configuration usine, la caméra IP Axis 207 a une adresse IP fixe : 192.168.0.90. Une méthode simple, pour accéder aux divers paramétrages de la caméra, consiste à utiliser un câble croisé pour y connecter directement un PC. On affecte à ce dernier une adresse IP fixe choisie dans le même sous réseau. Puis, on se connecte à l'adresse **http://192.168.0.90** à l'aide d'un navigateur.

La caméra Axis 207 nécessite deux petits réglages avant d'être exploitable avec Asterisk. Il faut autoriser la connexion RTSP sans authentification et forcer la taille d'image à QCIF. Pour ce faire, on clique sur le lien *Setup*. Pour le premier réglage, choisissez le menu *Basic Configuration* et sélectionnez la page *1- User*. Puis, dans la section *User Settings*, cochez *Enable anonymous viewer login (no user name or password required)*. Pour le second réglage, toujours sur la page *Setup*, choisissez *Video & Image*. Puis, dans la section *Image Appearance*, déroulez la liste jusqu'à *176x144* pixels (QCIF).

10.2 Installation du lecteur de flux RTSP : app_rtsp

Cette application externe à Asterisk, permet d'afficher un flux vidéo sur un terminal vidéo.

On télécharge d'abord, les sources :

```
svn co http://sip.fontventa.com/svn/asterisk/app_rtsp app_rtsp
```

Ensuite, on copie le fichier `app_rtsp.c` dans le répertoire `asterisk-1.4.2/apps`.

Puis, on recompile et on réinstalle Asterisk :

```
make
make install
```

Enfin, on vérifie le chargement du module `app_rtsp.so` dans Asterisk :

```
asterisk -vvvvc
*CLI> show modules
...
app_rtsp.so          RTSP applications              0
...
```

10.3 Installation du transcodeur MPEG4 vers H.263 : app_transcoder

Cette application externe à Asterisk permet de convertir un flux RTSP MPEG4-ES en flux RTSP H.263+ (H263-1998/RFC2429).

On commence par se procurer les sources :

```
svn co http://sip.fontventa.com/svn/asterisk/app_
transcoder app_transcoder
```

Après cela, on copie le fichier `app_transcoder.c` dans le répertoire `asterisk-1.4.2/apps` et on modifie le `Makefile` de ce même répertoire en ajoutant à la fin :

```
app_transcoder.so : app_transcoder.o
        $(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} -lavcodec
```

Ensuite, on vérifie le chargement du module `app_transcoder.so` dans Asterisk :

```
asterisk -vvvvc
*CLI> show modules
...
app_transcoder.so        H324M stack              0
...
```

10.4 Configuration d'Asterisk

Pour pouvoir transcoder et lire le flux vidéo RTSP de la caméra IP Axis 207, il faut ajouter les lignes suivantes à la fin du fichier `/etc/asterisk/extensions.conf` (plus exactement à la fin du contexte `[default]`) :

```
exten => 590,1,Goto(transcodage,s,1)
[transcodage]
exten => s,1,Answer
exten => s,2,transcode(,s@camera,h263@qcif/fps=10/kb=52/
qmin=4/qmax=12/gs=50)
exten => s,3,HangUp

[camera]
exten => s,1,Answer
exten => s,2,rtsp(rtsp://192.168.0.90:554/mpeg4/media.amp)
exten => s,3,HangUp
```

En appelant le `590` avec le téléphone logiciel Linphone, on peut voir l'image de la caméra IP.

11. VidéoConférence

Notre maquette à base de différents terminaux SIP et d'un serveur Asterisk permet également de réaliser des vidéoconférences à des prix défiant toute concurrence. Pour cela, il faut associer l'application externe AppConference à Asterisk. Cette Application Asterisk mixe le son de tous les participants à une conférence et permet de commuter les entrées/sorties vidéo. Le choix de la configuration de la commutation vidéo peut être fait de manière statique, au niveau des paramètres de l'application dans le fichier `/etc/asterisk/extensions.conf` ou de manière dynamique. Le choix dynamique de la commutation vidéo peut être réalisé par le modérateur de la conférence, à travers l'interface CLI d'Asterisk ou, par le participant, à travers l'envoi de tonalités DTMF, depuis son vidéophone. Une possibilité de commutation vidéo automatique en fonction de la détection d'activité

vocale (VAD) existe aussi, à condition d'utiliser le codec audio SPEEX.

Dans un premier temps, il s'agit simplement de récupérer les sources de la version de développement (*trunk*) de AppConference sur le serveur Asterisk :

```
svn co https://appconference.svn.sourceforge.net/
svnroot/appconference appconference
```

Ensuite, on compile la version de développement (trunk) d'AppConference :

```
cd appconference/trunk
make
make install
```

Enfin, on vérifie le chargement du module `app_conference.so` dans Asterisk :

```
asterisk -vvvvc
*CLI> show modules
...
app_conference.so        Conference                0
...
```

Il faut ensuite créer un numéro de téléphone qui sert de point d'entrée à la conférence pour tous les participants. La configuration la plus intéressante du commutateur vidéo pour le participant est celle où il peut sélectionner sa vue à travers les touches du téléphones (RX). Pour cela, on ajoute la ligne suivante à la fin du fichier `/etc/asterisk/extensions.conf` :

```
exten => 2300,1,conference(test/RX)
```

Pour accéder à la conférence « test », il faut composer le `2300` sur chacun des téléphones. On peut, ensuite, sélectionner la vue en pressant sur les touches du téléphone.

Comme cela a déjà été indiqué plusieurs fois, Asterisk n'effectue, pas encore, le transcodage entre les codecs vidéo. Les téléphones accédant à une conférence doivent donc utiliser les mêmes codecs. Pour tester une conférence, il est intéressant de disposer de plus de trois téléphones. On peut, par exemple, ajouter plusieurs clients WengoPhone NG dans `/etc/asterisk/sip.conf`. Puis, appeler le point d'entrée `2300` depuis les clients WengoPhone NG et le GrandStream GXV3000.

12. Passerelle H324m

Les opérateurs de téléphonie mobile proposent aujourd'hui la technologie UMTS (*Universal Mobile Telecommunications System*) dont les débits permettent la visiophonie entre mobiles de 3ème génération. Le protocole utilisé pour la vidéophonie mobile 3G est h324m. C'est une extension du protocole h324 utilisé pour les communications vidéo sur le réseau numérique à intégration de service (ISDN).

L'application `app_324m` permet à Asterisk de réaliser une passerelle vidéo entre les systèmes de téléphonie IP et mobiles 3G via une simple carte ISDN. L'entrée de gamme des cartes ISDN T0 est disponible pour environ 30 euros. Pour la suite, nous utiliserons une telle carte basée sur le *chipset* Cologne à travers le canal `asterisk chan_misdn`.

12.1 Installation et configuration de la passerelle H324m

12.1.1 Installation et configuration de la carte Modem T0

L'installation des drivers « modular ISDN » se fait très facilement :

```
wget http://www.misdn.org/downloads/mISDN.tar.gz
wget http://www.misdn.org/downloads/mISDNuser.tar.gz
tar xzf mISDN.tar.gz
tar xzf mISDNuser.tar.gz
cd mISDN-1_1_0b
make install
cd ../mISDNuser-1_1_0
make install
```

On peut maintenant tenter la détection de la carte ISDN, puis la génération automatique du fichier de configuration de mISDN (`/etc/mISDN.conf`), et finalement le lancement le service :

```
mISDN scan
1 mISDN compatible device(s) found:
>> hfcpci

mISDN config
Writing /etc/mISDN.conf for 1 mISDN compatible device(s):
>> hfcpci

mISDN start
-- Loading mISDN modules --
>> /sbin/modprobe --ignore-install capi
>> /sbin/modprobe --ignore-install mISDN_core debug=0
>> /sbin/modprobe --ignore-install mISDN_l1 debug=0
>> /sbin/modprobe --ignore-install mISDN_l2 debug=0
>> /sbin/modprobe --ignore-install l3udss1 debug=0
>> /sbin/modprobe --ignore-install mISDN_capi
>> /sbin/modprobe --ignore-install hfcmulti type= protocol=
layermask= poll=128 debug=0 timer=0
FATAL: Error inserting hfcmulti (/lib/modules/2.6.18-3-686/extra/
hfcmulti.ko): Invalid argument
>> /sbin/modprobe --ignore-install hfcpci protocol=0x2 layermask=0xf
>> /sbin/modprobe --ignore-install mISDN_dsp debug=0 options=0
creating device node: /dev/mISDN
```

Après cela, il faut recompiler Asterisk pour qu'il prenne en compte le canal `chan_misdn` :

```
cd asterisk-1.4.2
./configure
make
make install
```

La configuration de `misdn` pour Asterisk se réalise à travers le fichier de configuration `/etc/asterisk/misdn.conf` :

On commente, en plaçant des points virgules en début de ligne, les lignes par défaut de la section `[intern]` comme suit :

```
;[intern]
; define your ports, e.g. 1,2 (depends on mISDN-driver loading order)
;ports=1,2
; context where to go to when incoming Call on one of the above ports
;context=Intern
```

Puis on remplace cette section par les lignes suivantes que l'on ajoute à la fin du fichier :

```
[CarteCologne]
ports=1    ; define your ports, e.g. 1,2 (depends on mISDN-driver loading order)
context=depuisISDN    ; context where to go to when incoming Call on one of the
above ports
;immediate=yes
msns=*
```

Au relancement d'Asterisk, il est prudent de vérifier le chargement du module `chan_misdn` :

```
asterisk -vvvvc
*CLI> show modules
...
chan_misdn.so          Channel driver for mISDN Support (BRI/PR Ø
...
```

Lorsque celui ci est chargé, plusieurs commandes CLI Asterisk débutant par `misdn` sont disponibles.

12.1.2 Installation de PWLIB

Les installations de `libh324m` et `app_h324m` dépendent de l'installation préalable de la dernière version de `pwlib`. On l'installe en procédant comme suit :

```
wget http://downloads.sourceforge.net/openh323/pwlib-
v1_10_3-src-tar.gz?modtime=1169723828&big_mirror=0
tar -xzvf pwlib-v1_10_3-src-tar.gz
cd pwlib_v1_10_3/
./configure --prefix=/usr
make
make install
```

12.1.3 Installation de libh324m

Cette bibliothèque est indispensable à l'application `app_h324m`. Il faut, avant tout, récupérer les sources :

```
svn co http://sip.fontventa.com/svn/asterisk/libh324m
libh324m
cd libh324m
```

Pour que cela fonctionne avec le canal Asterisk `chan_misdn`, il faut mettre en commentaire (ajouter // en début de ligne) les appels à la fonction `TIFFReverseBits` dans les fonctions `H324MSessionRead` & `H324MSessionWrite` du fichier `h324m.cpp`. Ensuite, on lance la compilation :

```
make
cp libh324m.so /usr/lib
cp include/h324m.h /usr/include
```

12.1.4 Installation de app_h324m

Les fonctions de `app_h324m` sont :

- `h324m_loopback` : boucle sur le protocole h324m ;
- `h324m_gw` : réception d'appels vidéo ;
- `h324m_call` : émission d'appels vidéo ;
- `video_loopback` : boucle du flux vidéo.

Pour commencer, on télécharge les sources :

```
svn co http://sip.fontventa.com/svn/asterisk/app_h324m app_h324m
cd app_h324m
```

Pour que `app_h324m` fonctionne avec `asterisk-1.4.2`, il faut ensuite éditer le fichier `app_h324m.c` et ajouter un troisième paramètre ayant pour valeur « 100 » dans les deux appels à la fonction `ast_senddigit_end ()`.

Après cela, il faut copier le fichier `app_mp4.c` dans le répertoire `asterisk-1.4.2/apps` et modifier le `Makefile` de ce même répertoire en ajoutant à la fin :

```
app_h324m.so : app_h324m.o
        $(CC) $(SOLINK) -o $@ ${CYGSOLINK} $< ${CYGSOLIB} -lh324m
```

Ensuite, on recompile et on réinstalle Asterisk :

```
make
make install
```

Puis on vérifie le chargement du module `app_h324m.so` dans Asterisk :

```
asterisk -vvvvc
*CLI> show modules
...
app_h324m.so           H324M stack                Ø
...
```

12.2 Configuration des différentes applications

12.2.1 Appel audio uniquement

Dans un premier temps, il convient de tester la carte ISDN T0 simplement pour passer et recevoir un appel téléphonique classique (sans vidéo).

Pour passer un appel à travers cette carte, il suffit d'ajouter la ligne suivante à la fin du fichier `/etc/asterisk/extensions.conf` (plus exactement à la fin du contexte `[default]`) :

```
exten => _9.,1,Dial(mISDN/1/${EXTEN:1})
```

Pour appeler un numéro quelconque, il convient alors de le faire précéder du `9` pour indiquer que l'appel doit passer par la carte ISDN.

Un appel arrivant sur la carte ISDN peut être dirigé sur le téléphone GXV1 simplement en ajoutant le contexte suivant à la fin du fichier `/etc/asterisk/extensions.conf` :

```
[depuisISDN]
exten => _.,1,Dial(SIP/GXV1)
```

12.2.2 Echo vidéo

Une fois la passerelle ISDN validée en audio uniquement, on peut effectuer le premier test relatif à la vidéo. Il consiste en un rebouclage de la vidéo sur un appel vidéo entrant. Pour cela, on modifie les lignes précédemment ajoutées à la fin du fichier `/etc/asterisk/extensions.conf` comme suit :

```
[depuisISDN]
exten => _.,1,Answer
exten => _.,n,h324m_loopback()
```

Maintenant, un appel sur le numéro associé à l'interface T0 d'Asterisk à l'aide d'un mobile UMTS renvoie un écho vidéo (sans le son).

12.2.3 Appel Vidéo entre mobile UMTS et téléphone SIP

Pour diriger l'appel vidéo entrant depuis le mobile UMTS vers un terminal SIP, il faut modifier le contexte [depuisISDN à la fin du fichier /etc/asterisk/extensions.conf comme suit :

```
[depuisISDN]
exten => _.,1,h324m_gw(s@moncontexte)
[moncontexte]
exten=>s,1,Dial(SIP/LinPhone1)
```

Les tests effectués ne donnent actuellement pas encore de résultats, mais cela ne saurait tarder...

12.2.4 Diffusion d'un fichier vidéo : app_mp4

En combinant app_324m et app_mp4, il est possible de diffuser un clip vidéo sur le mobile 3G. Les codecs vidéo et audio utilisés sont respectivement H.263 et AMR-NB.

La commande suivante transcode un fichier MPEG en fichier MP4 avec les codecs adéquats :

```
ffmpeg -i monFichier.mpg -vcodec h263 -r 30 -s qcif -b 64k \
-g 50 -ac 1 -acodec amr_nb -ar 8000 -ab 64k monFichier.mp4
```

Le fichier conteneur MP4 ainsi créé peut être analysé avec la commande mp4info :

```
mp4info monFichier.mp4
mp4info version 1.5.0.1
monFichier.mp4:
Track   Type    Info
1       video   H.263, 169.966 secs, 70 kbps, 176x144 @ 30.000118 fps
2       audio   AMR, 169.960 secs, 13 kbps, 8000 Hz
```

Avant de pouvoir le diffuser, il faut ajouter les pistes de « hint » pour chacune des pistes visualisées ci-dessus.

```
mp4creator -hint=1 monFichier.mp4
mp4creator -hint=2 monFichier.mp4
```

On peut à nouveau utiliser la commande mp4info pour visualiser l'ajout des ces pistes de « hint » :

```
mp4info monFichier.mp4
mp4info version 1.5.0.1
monFichier.mp4:
Track   Type    Info
1       video   H.263, 169.966 secs, 70 kbps, 176x144 @ 30.000118 fps
2       audio   AMR, 169.960 secs, 13 kbps, 8000 Hz
3       hint    Payload H263-2000 for track 1
4       hint    Payload AMR for track 2
 Metadata Tool: mp4creator 1.5.0.1
```

Il faut alors modifier les lignes à la fin du fichier etc/asterisk/extensions.conf pour obtenir ceci :

```
[depuisISDN]
exten => _.,1,h324m_gw(s@appelVideoEntrant)

[appelVideoEntrant]
exten => s,1,Answer
exten => s,n,mp4play(/tmp/monFichier.mp4)
exten => s,n,HangUp
```

Finalement, un appel sur le numéro associé à l'interface T0 d'Asterisk à l'aide d'un mobile UMTS permet de voir et d'entendre le clip vidéo.

12.2.5 Diffusion d'un flux vidéo : app_rtsp

En associant app_h324m et app_rtsp, on peut également envoyer l'image de notre caméra IP sur le mobile 3G. Ceci constitue une application de vidéo surveillance très intéressante.

Pour cela, on modifie à nouveau les mêmes lignes à la fin du fichier /etc/asterisk/extensions.conf pour obtenir ceci :

```
[depuisISDN]
exten => _.,1,h324m_gw(s@transcodage)
[ transcodage]
exten => s,1,Answer
exten => s,2,transcode(,s@camera,h263@qcif/fps=10/
kb=52/qmin=4/qmax=12/gs=50)
exten => s,3,HangUp

[camera]
exten => s,1,Answer
exten => s,2,rtsp(rtsp://10.128.16.250:554/mpeg4/media.amp)
exten => s,3,HangUp
```

Les tests effectués ne donnent actuellement pas encore de résultats, à cause d'un problème de négociation de codecs, mais cela ne saurait tarder...

13. Perspectives d'évolution

Aujourd'hui, Asterisk dispose déjà de quelques possibilités vidéo. Même si actuellement les codecs vidéo les plus courants restent H.263 et H.263+, un des points primordiaux à développer pour l'évolution de ces capacités est le transcodage. En effet, comme c'est déjà le cas pour les codecs audio (voir la commande CLI show translations), il est indispensable de pouvoir utiliser la vidéo entre terminaux utilisant des codecs différents. Une fois cette étape franchie, on pourra imaginer d'ajouter des applications de traitement vidéo, comme la vue en mosaïque dans une vidéoconférence ou pour la vidéosurveillance. Les prochaines versions d'Asterisk vont certainement nous réserver de belles surprises à ce niveau...

Philippe HENSEL,

philippe.hensel@uha.fr

Université de Haute Alsace – IUT de Colmar
Département Réseaux et Télécommunications

LIENS

- Asterisk : http://www.asterisk.org
- GrandStream : http://www.grandstream.com
- WengoPhone NG : http://dev.openwengo.com/trac/openwengo/trac.cgi/wiki/WengoPhoneNg
- Linphone : http://www.linphone.org/
- app_mp4 et app_h324m : http://sip.fontventa.com
- AppConference : http://appconference.sf.net
- misdn : http://www.misdn.org

Nicolas Chuche

Gestion de versions avec SVK

Cet article est un tutoriel pour le logiciel de gestion de versions SVK, qui est né du constat que, ayant besoin de SVK et le connaissant mal, le meilleur moyen de le maîtriser était d'approfondir mes connaissances et de les organiser dans un tutoriel à destination d'un public le plus large possible.

Introduction

Pour Wikipédia, la gestion de versions est « une activité qui consiste à maintenir l'ensemble des versions d'un logiciel ». Si vous êtes développeur, il y a de bonnes chances que vous ayez à gérer cette activité et que vous connaissiez déjà un outil de gestion de versions comme CVS, Subversion ou équivalent.

Écrit il y a plus de 20 ans, CVS est sans doute le grand-père le plus connu des outils de gestion de versions libre. Bien qu'encore très utilisé de par le monde, il a un certain nombre de faiblesses gênantes (impossible de déplacer des fichiers en conservant l'historique, pas d'atomicité des transactions, pas de dépôts décentralisés, etc.) que les nouveaux outils n'ont plus.

De nombreux remplaçants ont vu le jour. Certains sont généralistes (Bazaar, Subversion, SVK, Mercurial, etc.) tandis que d'autres sont développés spécifiquement pour un projet (Git pour le noyau Linux).

Dans cet article, je parlerai de SVK qui est un client. Pourquoi celui-ci et pas un autre ? Plusieurs raisons à cela :

- Cela fait un certain temps que j'en entends parler dans la communauté Perl.
- Il apporte des fonctionnalités déterminantes par rapport à ses concurrents (dépôts décentralisés, permet de réaliser des opérations même si l'on n'a pas accès au réseau, systèmes de branches et de fusions très évolués, etc.).
- SVK est un client qui sait communiquer avec plusieurs serveurs de gestion de versions dont Subversion, logiciel de plus en plus répandu, mais aussi CVS et Perforce, même si c'est avec des fonctionnalités plus réduites.
- Et enfin parce que les Mongueurs de Perl viennent de migrer leur dépôt de CVS vers Subversion (cf. point précédent) et que j'ai besoin d'y avoir accès.

SVK a été développé en 2003 par Chia-liang Kao pour ses besoins. Il s'est appuyé sur le système de fichiers de Subversion (svn) et a redéveloppé toute la couche haute. Les commandes utilisées sont néanmoins très proches de celle de Subversion et de CVS, les utilisateurs de ces produits ne seront donc pas totalement dépaysés. SVK a été acquis par Best Practical (le fameux éditeur de Request Tracker) en 2006 et Chia-liang Kao est devenu partenaire de Jesse Vincent chez Best Practical.

Installation

L'installation de SVK est d'une simplicité légendaire s'il est disponible sous forme de paquets :

```
% sudo aptitude install svk     # Debian et dérivées
% sudo urpmi svk                # Mandriva
```

Sous Windows, il existe un portage trouvable ici : **http://home.comcast.net/~klight/svk/**.

Si aucun paquet de SVK n'existe pour votre distribution, vous allez devoir passer par un shell d'installation Perl, CPAN ou CPANPLUS.

```
% sudo cpan SVK         # avec CPAN.pm
% cpanp i SVK           # avec CPANPLUS
```

Comme me le souffle un camarade, une installation « à la main » n'est pas des plus simples, car il vous faudra installer ou compiler un Subversion avec les *bindings* Perl. Vous pourrez trouver de l'aide sur le wiki de SVK : **http://svk.bestpractical.com/view/InstallingSVK**.

Utilisation simple

Avant d'entrer dans le vif du sujet, nous allons aborder quelques notions utilisées par les logiciels de gestion de versions, et par SVK en particulier.

- Le dépôt (*repository* en anglais) est un système de fichiers spécial dans lequel sont stockées toutes les données (fichiers et répertoires) ainsi que toutes les modifications apportées au fur et à mesure. Il est, au choix, local ou déporté sur une machine distante. On peut y accéder par différents protocoles et il peut être utilisé par plusieurs personnes en même temps. C'est la partie principale d'un outil de gestion de versions. Un même dépôt peut contenir autant de projets différents que vous le désirez.
- La « copie de travail » est un vrai répertoire contenant des... fichiers et des répertoires se trouvant également dans le dépôt. C'est dans ce répertoire que vous travaillerez (seul) sur votre projet avant de publier vos modifications sur le dépôt pour les rendre accessibles à tout le monde (ou du moins à tous les gens qui ont accès au dépôt). Vous pouvez avoir autant de copies de travail que vous voulez.

Pour ne pas vous surcharger dès l'introduction de notions plus ou moins complexes sans avoir pratiqué, je reviendrai au fur et à mesure de cet article sur quelques autres notions importantes.

Création d'un dépôt local

La première étape est de créer un dépôt sur votre machine :

```
% svk depotmap --init
Repository /home/test/.svk/local does not exist, create? (y/n)y
```

Comme indiqué dans le retour de la commande, elle crée votre dépôt dans votre répertoire personnel dans le répertoire `.svk/local`.

Afin de promouvoir le plus tôt possible les bonnes pratiques, nous allons dès maintenant structurer le contenu de ce dépôt de façon à organiser judicieusement les travaux à venir. Nous verrons plus tard l'intérêt direct de l'organisation proposée.

```
% svk mkdir //local/ -m 'creation du repertoire local'
Committed revision 1.
```

Quand je parle de répertoire, il s'agit bel et bien de cela. Mais celui-ci n'est pas sur votre système de fichier, il se trouve dans le dépôt SVK, c'est pour cela qu'on les appelle « *depot path* » ou « chemin de dépôt ». Pour ne pas confondre les chemins du système de fichier avec ceux du dépôt SVK, le chemin de dépôt par défaut est préfixé de deux slashes (/). `//local` est donc votre premier chemin de dépôt.

Travailler sur un projet déjà dans un dépôt

La syntaxe normale pour récupérer les fichiers d'un dépôt centralisé est :

```
% svk checkout protocole://le.serveur.net/le/chemin/mon/appli repertoire
```

Quand vous allez taper cette commande pour la première fois, `svk` va vous poser trois questions :

- De quoi faire le miroir (en fait l'action `checkout` va conserver une copie locale du dépôt) ? Vous pouvez laisser la configuration par défaut.
- Quel chemin utiliser pour créer ce miroir ? Je vous conseille de le mettre sous `//mirror/nomduprojet/`, nous verrons plus tard pourquoi.
- Si vous voulez récupérer toutes les versions ou seulement les plus récentes. La réponse par défaut convient dans tous les cas bien que pouvant être plus longue. C'est donc celle que je vous conseille.

Juste pour information, nous venons de voir notre second chemin de dépôt à savoir `//mirror/`. Celui-ci est en fait un chemin spécial qui, par convention, sert à stocker tous les miroirs que l'on va faire avec SVK. Vous pourrez voir la liste de tous les miroirs que vous avez créés en faisant :

```
% svk mirror --list
Path                          Source
==============================================================
//mirror/mesprojets           https://mon.serveur.com/svn
//mirror/mongueurs
  svn+ssh://autre.serveur.com/home/nc/.svk/local/local/mongueurs/trunk
```

Voyons maintenant un exemple (pour `redmine`, voir dans les références à la fin de cet article) :

```
% svk checkout svn://rubyforge.org/var/svn/redmine/
New URI encountered: svn://rubyforge.org/var/svn/redmine/
Choose a base URI to mirror from (press enter to use the full URI):

  [...]

Depot path: [//mirror/redmine]

[...]

a)ll, h)ead, -count, revision? [a]
Syncing svn://rubyforge.org/var/svn/redmine
Retrieving log information from 1 to 21
Committed revision 10297 from revision 1.
[...]
Syncing //mirror/redmine(/mirror/redmine) in /home/nc/tmp/redmine to
  10317.
```

Toutes les commandes de `svk` peuvent être raccourcies. En l'occurrence, le raccourci de `checkout` est `co`.

Si le dépôt utilise un serveur SVN, le protocole sera `svn`, si le serveur utilise WebDAV, le protocole sera `http` ou `https` :

```
% svk checkout svn://rubyforge.org/var/svn/redmine/trunk redmine

% svk checkout https://rubyforge.org/var/svn/redmine/trunk redmine
```

Ou encore si vous avez accès au dépôt par `ssh` :

```
% svk checkout svn+ssh://rubyforge.org/var/svn/redmine/trunk redmine
```

Vous avez dû remarquer que j'ai utilisé le chemin `/var/svn/redmine/trunk`. Pourquoi `trunk` (tronc en français), pourquoi pas directement `redmine` ? Parce que `trunk` représente le développement principal et qu'il y a également des répertoires `tags` et `branches` permettant des « branches » de développements parallèles ainsi que la conservation de versions particulièrement intéressantes. Mais n'avançons pas trop vite, nous verrons un peu plus loin pourquoi et comment nous en servir.

Créer un nouveau projet à partir de fichiers existants

Vous avez déjà un répertoire que vous souhaitez gérer avec SVK. La commande à utiliser est `import` :

```
% svk import --message 'import initial' monprojet/ //local/monprojet
```

Avant de passer cette commande, je vous suggère fortement à la lumière de notre (très) rapide aperçu des branches, tags et *trunks* d'organiser un peu vos répertoires. Déplacez vos fichiers de travail dans `trunk` et créez les répertoires `branches` et `tags` afin d'obtenir dès maintenant une organisation comme celle-ci :

```
monprojet/branches/
          tags/
          trunk/<tous vos fichiers et répertoires>
```

Nous voyons maintenant à quoi cela nous sert d'avoir créer le répertoire `//local`. Cela permet de placer vos projets dans une arborescence dédiée et de ne pas tout avoir à la racine de votre dépôt.

Une fois l'import réalisé, vous allez pouvoir faire un `checkout` comme vu dans la partie précédente et commencer à travailler réellement. Pour ne récupérer que la version en cours de développement (soit

trunk) et non toute l'arborescence, la commande à passer est :

```
% svk checkout //local/monprojet/trunk monprojet
```

Vous auriez pu transformer directement le répertoire utilisé en version de travail en ajoutant l'option --to-checkout (ou -t) :

```
% svk import --message 'import initial' -t
monprojet/ //local/monprojet
```

Travailler sur votre projet

Ça y est, vous avez bien avancé sur votre nouvelle implémentation de logiciel de blog et vous voulez passer en revue vos modifications.

Plusieurs commandes vous seront utiles :

▶ svk diff

vous donne un diff entre la version d'origine du dépôt et votre copie de travail. Cette commande peut prendre en argument un ou plusieurs fichiers/répertoires. Vous ne verrez alors que leurs différences.

▶ svk add

ajoute des fichiers ou des répertoires dans le dépôt. Les fichiers ne seront réellement ajoutés qu'au prochain commit (cf. point suivant).

```
% svk add rep
A   rep
A   rep/a
A   rep/b
```

Cette commande est récursive. Si vous ajoutez un répertoire, tous les fichiers contenus seront également ajoutés.

Vous pouvez ajouter tous les nouveaux répertoires et fichiers d'un seul coup en faisant :

```
% svk add
```

Un point intéressant à savoir : svk est (plus ou moins) intelligent. Il n'ajoutera pas les fichiers finissant par un ~ ainsi qu'un certain nombre d'autres motifs. La liste complète par défaut est : *.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store.

▶ svk commit

permet d'enregistrer vos modifications dans le dépôt. Cette commande prend en option un commentaire. Ce commentaire sert à décrire la ou les modifications apportées.

```
% svk ci -m 'quelques modifications'
Committed revision 3.
```

Je vous suggère (fortement) d'être le plus précis possible dans les messages que vous associez aux *commits*. Cela vous permettra d'avoir une trace fiable.

Définition : révision

Cela fait déjà quelques fois que nous rencontrons ce terme de « révision » dans les sorties écran de svk. À chaque fois que vous faites un commit (ou une opération faisant un commit sans vous le dire comme pull ou smerge que nous verrons plus tard), svk va créer dans le dépôt un nouvel état du système de fichier contenant toutes vos modifications. Ces états sont appelés des révisions.

▶ svk delete

permet de supprimer un fichier dans le dépôt. Supprimer un fichier sur votre système de fichier n'avertit pas svk. Cette commande lui précise que ce fichier n'existe plus. Dans le dépôt, svk ne supprime pas réellement le fichier. Il garde bien sûr une trace de toutes les modifications qui y ont été apportées. Vous pourrez donc retrouver toutes ses versions si besoin est.

▶ svk rename

permet de renommer un fichier ou un répertoire. Si vous déplacez le fichier par les commandes liées à votre OS, svk ne pourra pas le savoir. Vous devez utiliser cette commande :

```
% svk rename config/ etc/
```

Comme add et delete, cette commande ne prend effet réellement qu'après le prochain commit.

▶ svk revert

permet de « défaire » les éditions locales. Si vous avez modifié un fichier ou bien exécuté une commande add ou delete par exemple, mais que vous n'avez pas encore *commité*, vous pouvez revenir à l'état d'avant ce add ou ce delete :

```
% svk status
# on crée un nouveau fichier
% touch yyyyy
# ce fichier est marqué comme inconnu par svk
% svk status
?   yyyyy
# on l'ajoute par svk, il le marque comme à ajouter
% svk add yyyyy
A   yyyyy
# on le 'revert'
% svk revert yyyyy
Reverted yyyyy
# il est à nouveau marqué comme inconnu
% svk status
?   yyyyy
```

▶ svk status

vous permet de voir rapidement l'état de vos fichiers

```
% svk status
M   a.c
!   a.h
?   b.c
?   b.h
```

Le M indique que le fichier a été modifié, le ! que le fichier n'existe plus dans les fichiers locaux et le ? que le fichier n'existe pas dans le dépôt. Pour une liste complète, référez-vous au *SVK Book*.

Mettre à jour les fichiers locaux à partir du dépôt

Si vous êtes plusieurs à travailler sur votre projet, vous allez vouloir à un moment ou à un autre mettre à jour

vos fichiers. Pour cela, utilisez la commande suivante depuis le répertoire de travail :

```
% svk update
```

Le problème que vous voyez peut-être déjà, c'est comment ça se passe si votre collègue/copain a modifié un fichier que vous avez modifié localement vous aussi. Voyons ça :

```
% svk update
Syncing //local/monprojet/trunk(/local/monprojet/trunk) in
/home/test/monprojet to 6.
Conflict found in a.c:
e)dit, d)iff, m)erge, s)kip, t)heirs, y)ours, h)elp? [e]
```

svk s'est rendu compte du problème et vous propose plusieurs choix. Dans l'immédiat nous n'allons utiliser qu'une des options (je vous laisse découvrir les autres), à savoir l'option "s". Cette dernière intègre les modifications apportées par les coéditeurs du fichier. À vous de régler le conflit :

Verrouillage ou conflit ?

Les logiciels de gestion de versions se séparent en deux catégories, ceux qui verrouillent (« *lock* ») les fichiers pour éviter les conflits et ceux qui autorisent les conflits et laissent les utilisateurs les gérer. Si la première catégorie semble plus sûre, elle n'est pas forcément la plus efficace. À l'usage, on se rend compte que les conflits ne sont ni si courants ni si graves et que cela apporte une souplesse non négligeable. Si néanmoins vous constatez que vous avez beaucoup de conflits, allez lire les conseils méthodologiques à la fin de cet article.

```
% cat a.c
>>>> YOUR VERSION a.c 115722413378427
#include "d.h"
==== ORIGINAL VERSION a.c 115722413378427
#include "b.h"
==== THEIR VERSION a.c 115722413378427
#include "e.h"
<<<< 115722413378427
```

Comme indiqué dans le texte, svk indique les différentes versions à l'origine du conflit à savoir, votre version actuelle, la dernière version commitée ainsi que la version du coéditeur.

La résolution d'un conflit de ce type n'est pas tant un problème informatique qu'humain. Pour le résoudre, vous aurez sans doute besoin de contacter la personne ayant fait la modification pour en parler avec elle avant de garder ou de modifier la version adéquate.

Après avoir résolu le conflit, vous devrez prévenir svk avec :

```
% svk resolved a.c
```

Et ensuite, vous pourrez commiter cette modification afin de ne pas laisser traîner le conflit :

```
% svk commit a.c
```

Une chose à savoir, les commandes commit et update ne sont pas liées. Vous pouvez faire (et ferez souvent) l'une sans l'autre.

Regarder l'historique des modifications

Pour voir toutes les actions réalisées, vous pouvez faire svk log.

Cette commande ne fait que vous afficher les commentaires que vous mettez quand vous commitez. C'est donc dans votre intérêt d'être le plus précis possible.

Par défaut, svk log ne vous montre l'historique que depuis la dernière copie effectuée. Si vous travaillez sur un miroir, vous devrez utiliser l'option --cross qui permet de voir également les modifications recopiées depuis un autre dépôt. L'option --verbose permet d'avoir une sortie un peu plus complète (fichiers concernés entre autres).

Pour regarder plus précisément un fichier, la commande svk annotate peut vous aider. Elle permet de voir quelles lignes viennent de quelle version.

svk cat permet de voir l'état d'un fichier dans le dépôt. Nous verrons un peu plus loin un argument idéal à utiliser avec cette commande.

Quelques autres commandes

Voici quelques autres commandes qui pourront vous être utiles au cours de votre utilisation de svk :

- Si vous voulez exporter un projet sans le conserver dans la gestion de versions, vous pouvez utiliser l'option --export :

```
% svk checkout --export //local/redmine/trunk redmine
```

- Si vous voulez voir ce qu'il y a dans votre dépôt, vous pouvez utiliser la commande list avec le chemin de dépôt qui vous intéresse :

```
% svk list //mirror
articles/
monprojet/
mongueurs/
redmine/
tracks/
```

Cette commande accepte plusieurs options dont une peut vous intéresser tout de suite : svk list -R // liste récursivement tous les répertoires. Très pratique pour avoir une visualisation en arbre de votre arborescence de dépôt.

- Si vous avez besoin de créer un nouveau répertoire dans votre copie de travail, vous pouvez utiliser directement la commande svk mkdir, le répertoire sera ajouté au prochain commit. Et comme nous l'avons vu au début de cet article, elle permet également de créer un nouveau répertoire de dépôt (pour organiser un peu votre dépôt par exemple) :

```
% svk mkdir monrepertoire
% svk mkdir //local/quelquechose
```

- Parfois vous voudrez savoir d'où vient telle copie de travail ou tel répertoire de votre dépôt. svk info vous donnera toutes les informations utiles :

```
% svk info
Checkout Path: /home/nc/travail/rails/redmine
```

```
  Depot Path: //local/redmine/trunk/redmine
  Revision: 10389
  Last Changed Rev.: 10389
  Copied From: /mirror/redmine/trunk/redmine, Rev. 10375
  Merged From: /mirror/redmine/trunk/redmine, Rev. 10388

  % svk info //mirror/redmine
  Depot Path: //mirror/redmine
  Revision: 10410
  Last Changed Rev.: 10388
  Mirrored From: svn://rubyforge.org/var/svn/redmine, Rev. 22
```

Quelques options bien utiles

Un certain nombre de commandes acceptent des arguments. Vous trouverez la liste complète dans le *SVK Book*, mais je vais rapidement donner ici celles qui peuvent vous être utiles tout de suite :

Les commandes `diff`, `log`, `update` et `cat` acceptent un argument `-r` pour indiquer une révision. Cet argument est soit le numéro de révision, soit une date. Par exemple, pour avoir les modifications faites entre le 9 juillet et le 10 juillet, il suffit de faire :

```
% svk diff -r {2006-07-09}:{2006-07-10}
=== redmine/app/helpers/search_filter_helper.rb
===================================================================
--- redmine/app/helpers/search_filter_helper.rb (revision 10337)
+++ redmine/app/helpers/search_filter_helper.rb (revision 10340)
@@ -29,10 +29,12 @@
  end
```

Pour préciser que vous voulez comparer par rapport à la dernière version, utilisez `HEAD` :

```
  % svk diff -r {2006-07-09}:HEAD
```

Et comme promis antérieurement, cette option est très pratique avec `svk cat` pour voir l'état d'un fichier dans telle révision ou à telle date :

```
  % svk cat -r {2006-07-09} redmine/app/helpers/
search_filter_helper.rb
```

`checkout` et `update` acceptent également un argument `-r` qui permet de préciser à partir de quelle révision on veut se mettre à jour.

Nous avons déjà vu la commande `checkout`. Elle dispose d'une option bien pratique au bout de quelque temps pour retrouver ses petits. Il s'agit de `--list` qui affiche la liste des répertoires de travail :

```
% svk checkout --list
  Depot Path                        Path
=============================================================
  //local/mesprojets/bbrails/trunk  /home/nc/travail/rails/bbrails
  //local/mesprojets/fpw2006        /home/nc/travail/rails/fpw2006
  //local/mesprojets/gestapp/trunk  /home/nc/travail/rails/gestapp
  //local/mongueurs/trunk           /home/nc/travail/mongueurs
  //local/redmine/trunk/redmine     /home/nc/travail/rails/redmine
? //local/mesprojets/test           /home/nc/tmp/prosper
```

Le `?` indique que le répertoire de travail n'existe plus. Vous pouvez alors supprimer cette référence en faisant :

```
  % svk checkout --purge
  Purge checkout of //local/mesprojets/test to non-
existing directory
  /home/nc/tmp/prosper?
  (y/n) y
  Checkout path '/home/nc/tmp/prosper' detached.
```

Ça y est, vous maîtrisez les commandes de base et vous pouvez utiliser SVK au quotidien, mais il serait dommage de s'arrêter là alors qu'il y a tant à faire. Ceci nous amène naturellement au chapitre suivant.

Utilisation avancée

Les branches et les tags

Cette partie demande quelques explications. Imaginons que vous écriviez un nouveau logiciel allant révolutionner le monde (un logiciel de blog par exemple). Vous venez de sortir la version 1.0 et vous voudriez bien pouvoir développer la future version 2.0, tout en continuant à corriger les bugs de la version 1.0. Pour utiliser le jargon, vous aimeriez bien avoir plusieurs branches de développement, une branche stable et une branche instable. Une solution simple existe dans la plupart des logiciels de gestion de versions, les notions de «tag» et de «branche».

Les mises en œuvre de ces mécanismes diffèrent d'un logiciel à l'autre, mais, pour SVK, cela se traduit tout simplement par trois répertoires créés dans le dépôt (et non au niveau du système de fichier) :

```
  branches/ - les différentes versions de développement
  tags/     - des images figées
  trunk/    - la dernière version en cours de développement
```

Dans SVK, nous verrons que branches et tags sont gérés de la même façon. Ils ne diffèrent que par l'utilisation que l'on en fait : les branches sont utilisées pour développer les différentes versions en parallèle, tandis que les tags servent à conserver des versions spécifiques sur lesquelles vous ne travaillez pas. On ne commite pas dans les tags.

Création de branches et de tags

La commande utilisée pour créer des tags et des branches est la même, il s'agit de `svk copy` :

```
  % svk copy //local/monprojet/trunk \
    //local/monprojet/branches/monprojet-1.0 \
    -m 'creation de la branches 1.0'
  Committed revision 5.
```

Comme tout le monde s'en doutait, cela va créer un nouveau répertoire de dépôt dans `branches` contenant tous les fichiers dans l'état actuel de `trunk` :

```
  % svk list //local/monprojet/branches/
  monprojet-1.0/
```

Vous allez pouvoir faire un `checkout` de cette version 1.0 et travailler dessus tout en continuant à développer la version future dans `trunk`.

Une fois votre version 1.0 prête à être diffusée, vous voudriez bien garder une trace exacte de cette version.

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

Apprivoisez votre pingouin !

GNU LINUX PRATIQUE

En KIOSQUE

N°41

SOMMAIRE

Réfléchir
- Développeuses en liberté... 04

Découvrir
- KTorrent : le client BitTorrent pour KDE 08
- Inkscape 0.45 : plus rapide, plus facile ! 14
- ManDVD 19
- Voyage au pays des sites de contenus KDE 22

Tester
- Mémoire Kingston DDR-SDRAM 512 Mo PC 3200 HyperX KHX3200A 26
- LaCie LightScribe Labeler 28

Écouter/voir
- KAudioCreator : logiciel d'extraction de CD et d'encodage audio 30
- Kino : un pas supplémentaire vers la maîtrise des montages 35

Configurer
- Envy : l'installation facile des drivers graphiques dernier cri pour ATI et Nvidia 37

Communiquer
- Konversation : pour discuter librement sur IRC 38
- Extensions de Firefox : notre sélection 42

S'informer
- MakeHuman : interview de l'équipe de développement 46
- Fon 50

Comprendre
- BitTorrent : l'autre façon d'échanger des fichiers 54

Travailler
- Calc et les variables 56

Créer
- Mieux connaître OOo Draw : les objets 3D et leur espace 60

Cahier Web
- Des plugins pour vos blogs ! 70
- Proposer plusieurs CSS en fonction du navigateur 72
- Un menu haut en couleurs... 74
- Créez un menu déroulant vertical 76
- Publier son site Web : quelles sont les étapes ? 80

Bien sûr, vous pourriez utiliser l'option `-r`, que nous avons vue à la fin du chapitre précédent, avec le numéro de révision ou bien la date, mais ça n'est pas très intuitif. Une meilleure façon est de tagger cette version :

```
% svk copy //local/monprojet/branches/monprojet-1.0 \
  //local/monprojet/tags/monprojet-1.0 -m 'version 1.0 diffusée'
Committed revision 7.
```

Un utilisateur vous soumet un bug ? Vous corrigez le bug dans la branche `version-1.0` et, quand vous diffusez une version 1.0.1, vous n'avez qu'à la copier dans `tags` sous le nom `monprojet-1.0.1` pour en garder une trace.

Même si SVK s'arrêtait là, les notions de « tags » et de « branches » seraient un outil fantastique, mais, heureusement, SVK va beaucoup plus loin...

Passer votre copie de travail d'une branche à une autre

Si votre copie de travail est un `checkout` d'une branche et que vous voulez passer dans une autre, placez-vous dans le répertoire de la copie de travail et utilisez la commande `svk switch` :

```
% svk switch //local/monprojet/branches/monprojet-1.0/
```

Pour repasser dans `trunk`, il suffit de faire :

```
% svk switch //local/monprojet/trunk/
```

La fusion

Vous avez vos deux versions, celle de `trunk` et celle de `version-1.0`. Vous corrigez plusieurs bugs dans `version-1.0` et vous aimeriez bien les corriger aussi dans `trunk`. Plutôt que de reporter vos corrections à la main dans trunk, vous pouvez utiliser la fonction `smerge` :

```
% svk smerge -Il //local/monprojet/branches/monprojet-1.0 \
  //local/monprojet/trunk
Auto-merging (0, 8) /local/monprojet/branches/monprojet-1.0 to
   /local/monprojet/trunk (base /local/monprojet/trunk:4).
===> Auto-merging (0, 5) /local/monprojet/branches/monprojet-1.0 to
   /local/monprojet/trunk (base /local/monprojet/trunk:4).
Empty merge.
===> Auto-merging (5, 8) /local/monprojet/branches/monprojet-1.0 to
   /local/monprojet/trunk (base /local/monprojet/trunk:4).
U   db/migrate/001_setup.rb
New merge ticket:
0aca4fd4-bb1c-0410-adf5-ecdeea5a58f8:
   /local/monprojet/branches/monprojet-1.0:8
Committed revision 9.
```

Cette commande va reporter toutes les modifications que vous avez faites dans `version-1.0` dans le trunk. Évidemment, si vous avez déjà modifié le trunk dans un endroit où vous avez également corrigé un bug vous risquez d'avoir un conflit, charge à vous de le résoudre.

Attention, contrairement à la résolution de conflit que nous avons vue précédemment, vous ne pourrez pas faire `skip` pour résoudre votre problème plus tard. Vous devrez le résoudre tout de suite grâce aux options `diff` (pour voir le conflit) et `edit` (pour le modifier). Si vous faites `skip`, cette partie ne sera pas fusionnée, néanmoins vous pourrez la refusionner plus tard. Suite à votre édition, `svk` vous demandera si vous voulez accepter la modification :

```
% svk smerge -Il //local/monprojet/branches/monprojet-1.0 \
    //local/monprojet/trunk
Auto-merging (8, 11) /local/monprojet/branches/monprojet-1.0 to
 /local/monprojet/trunk (base /local/monprojet/branches/monprojet-1.0:8).
===> Auto-merging (8, 11) /local/monprojet/branches/monprojet-1.0 to
 /local/monprojet/trunk (base /local/monprojet/branches/monprojet-1.0:8).
Conflict found in config/config_custom.rb:
e)dit, d)iff, m)erge, s)kip, t)heirs, y)ours, h)elp? [e] e
Waiting for editor...
Merged config/config_custom.rb:
a)ccept, e)dit, d)iff, m)erge, s)kip, t)heirs, y)ours, h)elp? [a] a
G   config/config_custom.rb
New merge ticket: 0aca4fd4-bb1c-0410-adf5-ecdeea5a58f8:
 /local/monprojet/branches/monprojet-1.0:11
Committed revision 12.
```

Une bonne méthode de travail consiste à regarder avant de faire le `svk smerge` si vous allez avoir des conflits en faisant utilisant l'option `-C : svk smerge -C`.

Quelques jours après cette fusion corrigeant des bugs, vos utilisateurs vous soumettent à nouveau plusieurs bugs bloquants. Vous les corrigez et vous aimeriez bien pouvoir faire la même manipulation pour les corriger dans `trunk`. Eh bien, j'ai une bonne nouvelle, vous pouvez la recommencer autant de fois que vous le voulez. Contrairement à certains autres logiciels (dont SVN et CVS) pour lesquels les fusions à répétitions entre branches sont non triviales (il faut préciser à la fonction `merge` la dernière révision pour laquelle on a déjà fait une fusion), SVK implémente dans `smerge` une fonction de fusion sophistiquée, appelée `star-merge`, qui permet de fusionner successivement les mêmes branches sans se poser de question.

Vous pouvez utiliser `svk smerge` entre deux dépôts comme nous venons de le voir, mais aussi entre un dépôt et une copie de travail en spécifiant le chemin de votre copie de travail en seconde position comme ceci :

```
% svk smerge -Il //local/monprojet/branches/monprojet-1.0 \
  /ma/copie/de/travail
```

Fusionner seulement quelques révisions

Aussi parfois appelé « *cherry picking* » (cueillette de cerise).

Imaginons maintenant le cas inverse, vous développez dans `trunk` et trouvez des bugs. Après correction, vous aimeriez bien les reporter dans votre branche `monprojet-1.0`. Seul problème, vous avez aussi fait des modifications substantielles dans d'autres parties de `trunk` que vous ne voulez pas voir apparaître dans la branche.

`smerge` ne sait pas faire ça. Il va falloir se rabattre sur la commande `merge`.

Avant toute chose, on va chercher les révisions que l'on veut fusionner dans notre branche, la commande `log` va nous trouver ça :

```
% svk log
------------------------------------------------------------------------
```

```
r16:  test | 2006-09-06 15:23:04 +0200

  ajout d'une fonctionnalite pour la v2.0
------------------------------------------------------------
r15:  test | 2006-09-06 15:22:32 +0200

correction bug bloquant
------------------------------------------------------------
r14:  test | 2006-09-06 15:22:14 +0200

  ajout d'une fonctionnalite pour la v2.0
```

Le bug a été corrigé dans la révision 15, nous pouvons vérifier la correction de bug en faisant un `diff` :

```
% svk diff -r 14:15 //local/monlogiciel/trunk/
=== generate
============================================================
--- generate    (revision 14)
+++ generate    (revision 15)
@@ -1,4 +1,4 @@
 #!/usr/bin/env ruby
 # commentaire
-requir File.dirname(__FILE__) + '/../config/boot'
+require File.dirname(__FILE__) + '/../config/boot'
 require 'commands/generate'
```

C'est bien cette correction de bug que nous voulons fusionner. Nous pouvons donc appliquer le patch. Nous pouvons l'appliquer soit directement sur le dépôt (comme vu avec `smerge`), soit en se plaçant directement dans un répertoire de travail contenant notre branche et en faisant :

```
% svk merge  -r 14:15 //local/monlogiciel/trunk/
U   generate
```

Il vous faudra alors commiter ces modifications.

L'avantage de faire la fusion directement sur le dépôt est que vous n'aurez pas à faire le commit ensuite.

Cette partie suppose que vos commits forment un tout homogène. Un commit pour les corrections de bug, un commit pour l'ajout de telle fonctionnalité, etc.

Gérer des modifications locales dans un logiciel

Nous allons étudier un cas relativement classique : vous utilisez un logiciel que vous avez adapté à votre environnement. À chaque nouvelle version, vous devez réintégrer vos modifications à la main, ce qui est long, fastidieux et contrevient clairement à une des vertus cardinales de l'informaticien : la paresse. SVK peut vous aider.

- Créez une arborescence standard branches/tags/trunk :

```
mkdir -p monlogiciel/{branches,tags,trunk}
```

- Récupérez une archive de votre logiciel et décompactez-la directement dans `trunk`.
- Faites un import de toute l'arborescence :

```
% svk import -m "premier import de monlogiciel" monlogiciel \
   //local/monlogiciel
```

- Faites une copie de `trunk` vers une branche sur laquelle vous ferez vos modifications :

```
% svk copy -m "creation de la copie locale" //local/monlogiciel/trunk \
   //local/monlogiciel/branches/monlogiciel-local
```

- Faites un `checkout` de cette branche et faites toutes les modifications désirées dans cette branche et commitez-les.
- Une nouvelle version de `monlogiciel` est sortie ; récupérez-la à nouveau et décompactez-la.

```
% svk import -m "import de monlogiciel v1.2" monlogiciel \
   //local/monlogiciel/trunk
```

- Essayez de fusionner le nouveau `trunk` avec votre version locale :

```
% svk smerge -C //local/monlogiciel/trunk \
   //local/monlogiciel/branches/monlogiciel-local
Auto-merging (2, 5) /local/monlogiciel/trunk to
   /local/monlogiciel/branches/monlogiciel-local
   (base /local/monlogiciel/trunk:2).
U   console
New merge ticket: 6d4c1533-f1fa-4670-b606-7b3ba989afbe:
   /local/monlogiciel/trunk:5
```

Seul le fichier `console` a été modifié et SVK ne détecte aucun conflit. Vous pouvez donc faire le `smerge` en toute confiance :

```
% svk smerge -Il //local/monlogiciel/trunk \
   //local/monlogiciel/branches/monlogiciel-local
Auto-merging (2, 5) /local/monlogiciel/trunk to
   /local/monlogiciel/branches/monlogiciel-local (base
   /local/monlogiciel/trunk:2).
===> Auto-merging (2, 5) /local/monlogiciel/trunk to
   /local/monlogiciel/branches/monlogiciel-local (base
   /local/monlogiciel/trunk:2).
U   console
New merge ticket: 6d4c1533-f1fa-4670-b606-
7b3ba989afbe:
   /local/monlogiciel/trunk:5
Committed revision 6.
```

- Recommencez la procédure ci-dessus *adnauseam* à chaque nouvelle version de votre logiciel.

Évidemment, si la nouvelle version est une réécriture complète, cela risque de ne pas marcher aussi bien que voulu. Mais, si c'est une version de *bugfix*, le travail sera nettement moins fastidieux qu'à la main.

Travailler en mode déconnecté

Un des gros problèmes rencontrés quand vous utilisez un dépôt sur Internet est comment travailler quand vous n'avez pas accès à Internet pendant quelque temps. Vous pouvez revenir aux (pas si) bonnes vieilles méthodes, à savoir copier vos fichiers modifiés avec une extension précisant la date de modification ou la modification, et vous retrouvez rapidement avec des répertoires comme celui-ci :

```
projet/a.c
      /a.c-version_de_debuggage
      /a.c-1
      /a.c-2
      /a.c-bug_machin
      /...
```

Cela devient vite horrible et inutilisable (bien que des gens utilisent encore cette méthode).

SVK propose une solution simple et esthétique, un dépôt déconnecté. Pour faire cela, il faut utiliser la notion de « miroir ».

```
% svk mirror https://rubyforge.org/var/svn/redmine/
//mirror/redmine
% svk sync //mirror/redmine
```

La première commande va créer le miroir, la seconde va importer toutes les données du dépôt distant dans votre miroir. Quand vous faites un `checkout` directement depuis un serveur Subversion, c'est en fait ce que `svk` fait dans votre dos (rappelez-vous les questions qu'il vous a posées au début de ce tutoriel).

Une fois cette étape réalisée, vous allez faire une copie locale des données récupérées :

```
% svk copy //mirror/redmine //local/redmine
```

À partir de ce moment, vous pouvez faire un `checkout` de votre projet et commencer à travailler dessus hors-ligne en commitant régulièrement vos modifications.

Oui, mais si c'est en mode hors-ligne. Comment répliquer vos modifications sur le dépôt central et récupérer les dernières modifications du dépôt central ? En utilisant les commandes `svk push` et `svk pull`.

Vous revenez de deux semaines sans accès à Internet, vous avez commité de nombreuses modifications et vous voudriez bien que vos petits collègues en profitent. Commencez par récupérer leurs modifications :

```
% svk pull
```

Cette commande va d'abord mettre à jour votre miroir et ensuite faire un `update` de votre répertoire de travail. Vous aurez peut-être des conflits à résoudre. Une fois ces conflits résolus et commités, vous pouvez leur envoyer vos modifications :

```
% svk push
```

Ces commandes sont en fait des habillages un peu plus sexy d'autres commandes, mais ne vous préoccupez pas trop de ça si vous n'en avez pas besoin. Et si vous en avez besoin, lisez le *SVK Book*.

Vous pouvez voir les modifications en attente d'être poussées en utilisant la commande `diff` comme ceci :

```
% svk diff //mirror/monprojet/ //local/monprojet/
```

Proposer des modifications quand vous n'avez pas les droits d'écriture sur le dépôt

Il n'est pas rare de proposer des patchs sur des projets pour lesquels vous n'avez pas les droits d'écriture sur le dépôt. Vous pouvez bien sûr utiliser la commande `diff` présente sur votre système ou bien la commande `diff` de SVK, mais ce dernier vous permet de faire mieux avec l'option `--patch` utilisable avec `commit`, `push` et `smerge`.

Pour l'utiliser, commitez vos modifications normalement dans votre dépôt local et ensuite utilisez `push` avec l'option `-patch` :

```
% svk push --patch svk_correction_bug
Auto-merging (10574, 10606) /local/mongueurs/trunk to
   /mirror/mongueurs/trunk (base /mirror/mongueurs/
trunk:10604).
Patching locally against mirror source svn://svn.mongueurs.net/articles.
U   magazines/Applications/svk.pod
Patch svk_correction_bug created.
```

Le patch est déposé dans `~/.svk/patch`.

Vous pouvez lister vos patchs en faisant :

```
% svk patch --list
svk.pod@1:
svk_correction_bug@1:
```

Quand votre correspondant recevra le fichier de patch, il n'aura qu'à le déposer dans son fichier `~/.svk/patch` et le rejouer en faisant :

```
% svk patch apply svk_correction_bug
```

Répliquer un dépôt local sur un dépôt distant

Ça y est, votre logiciel est prêt à affronter la critique de vos contemporains et vous voudriez leur donner accès aux sources. Vous avez juste deux problèmes : votre dépôt est sur votre disque et vous ne voulez pas donner accès à tous vos projets. La solution, répliquer votre projet sur un autre serveur SVN, par exemple sur une forge sur internet.

Pour cette recette, vous avez normalement besoin d'un serveur SVN extérieur, mais nous allons contourner le problème en créant un autre dépôt local.

D'abord, un petit mot sur les dépôts. Au début de cet article, nous en avons créé un en faisant `svk depotmap --init`. Cette commande crée un dépôt par défaut que l'on peut ensuite utiliser en utilisant `//`. Même si cette possibilité est rarement utile et n'est pas forcément recommandée, SVK peut gérer plusieurs dépôts.

Créons donc ce second dépôt nommé « `remote` » dans le répertoire `/tmp/remote` (chemin pas forcément judicieux si vous comptez faire plus qu'un test...)

```
% svk depotmap remote /tmp/remote
New depot map saved.
Repository /tmp/remote does not exist, create? (y/n)y
```

Le dépôt est créé. On peut voir la liste des dépôts gérés par SVK en faisant :

```
% svk depotmap --list
Depot                   Path
=============================================
//                      /home/nc/.svk/local
/remote/                /tmp/remote
```

Ce nouveau dépôt est accessible en préfixant les chemins par `/remote/` à la place de `//`, mais nous allons l'utiliser comme si c'était un dépôt distant et commencer par créer un miroir dans le dépôt par défaut :

```
% svk list file:///tmp/remote
```

`svk` va vous poser des questions pour créer un miroir local dans le dépôt par défaut `//`. Répondez par défaut, cela va créer un nouveau chemin de dépôt `//mirror/remote`.

Synchronisez vos deux dépôts :

```
% svk sync //mirror/remote
```

Créez le répertoire `monprojet` qui va accueillir votre projet sur le miroir :

```
% svk mkdir //mirror/remote/monprojet/ \
      -m 'creation du repertoire de monprojet'
```

Faites un `smerge` de votre dépôt local vers votre dépôt :

```
% svk smerge --baseless --incremental //local/monprojet \
    //mirror/remote/monprojet
```

L'option `baseless` indique que `svk` doit faire le `smerge` même si les deux projets n'ont pas de racine commune et `incremental` indique que chaque changement doit être appliqué individuellement.

Ça y est, le miroir est fait. Vous aurez juste à penser à mettre à jour votre dépôt externe en lançant de temps en temps la commande :

```
% svk smerge --incremental -m "mise a jour" //local/monprojet/ \
  //mirror/remote/monprojet/
```

Malheureusement, vous ne pourrez pas utiliser la commande `svk push` pour mettre à jour cette copie. À ce niveau, plusieurs solutions s'offrent à vous :

- Soit vous faites le `smerge` vu ci-dessus à la main de temps en temps.
- Soit vous voulez vraiment utiliser les commandes `push` et `pull`. Vous devez alors utiliser ce nouveau dépôt comme dépôt « racine » et créer une nouvelle copie dans `//local` comme vu dans la section « travailler en mode déconnecté ». Vous pourrez alors supprimer l'ancienne copie locale (pensez à commiter toutes vos modifications avant de commencer cette manipulation) :

```
% svk copy -m "creation" //mirror/remote/monprojet/ \
  //local/monprojet-nouveau
```

(Attention, avec la version 2 de SVK que j'utilise, je ne peux pas réutiliser le même chemin de dépôt dans `//local`, d'où le « `-nouveau` ».)

Voila, vous pouvez maintenant faire un `checkout` de `//local/monprojet-nouveau` et travailler dessus :

```
% svk checkout //local/monprojet-nouveau
% cd monprojet-nouveau
% svk push
Auto-merging (0, 13238) /local/monprojet-nouveau to
    /mirror/remote/monprojet (base /mirror/remote/monprojet:13223).
===> Auto-merging (0, 13238) /local/monprojet-nouveau to
    /mirror/remote/monprojet (base /mirror/remote/monprojet:13223).
Merging back to mirror source file:///tmp/remote.
Empty merge.
```

- Soit, dernière solution, vous utilisez une nouvelle notion pour vous, les *hooks*, pour mettre à jour automatiquement le miroir.

Les hooks sont des programmes déclenchés par SVK sur certains évènements précis. Vous pouvez les trouver dans `~/.svk/local/hooks` qui, par défaut, ne contient que des modèles :

```
% ls ~/.svk/local/hooks
post-commit.tmpl  post-revprop-change.tmpl  pre-commit.tmpl
pre-revprop-change.tmpl start-commit.tmpl
```

Voici le hook utilisé dans le cas présent :

```
% cat ~/.svk/local/hooks/post-commit
#!/bin/sh

svk smerge --incremental //local/monprojet //mirror/remote/monprojet
```

Pensez à le rendre exécutable :

```
% chmod +x ~/.svk/local/hooks/post-commit
```

Et voilà, celui-ci va lancer la commande `smerge` pour mettre à jour votre miroir à chaque commit que vous ferez, y compris ceux ne concernant pas `monprojet`. Nous pourrions sans doute affiner cela, mais je vous laisse regarder ça vous-même.

L'inconvénient de cette dernière solution est que, si certaines personnes écrivent dans le nouveau dépôt, vous devrez penser à synchroniser les deux dépôts avec `svk sync //mirror/remote` pour récupérer leurs modifications. Néanmoins, dans le cadre d'un dépôt en lecture seule, cela ne pose pas vraiment de problème...

Métadonnées ou propriétés

Les métadonnées sont des données servant à décrire ou à ajouter de l'information à d'autres données. Dans un système de fichiers, les droits du fichier sont un exemple de métadonnées. Ils ne font pas partie du fichier, mais servent à ajouter une information à ce fichier. SVK (ainsi que Subversion) appelle ces métadonnées des « propriétés ». Ces propriétés sont manipulables avec les commandes `proplist`, `propget`, `propset`, `propedit` et `propdel` qui, respectivement, liste toutes les propriétés et retourne, modifie, lance un éditeur pour modifier et enfin supprime une propriété. Voyons un exemple :

```
% svk proplist README
% svk propset copyright 'NBC 2007' script/server
 M  script/server
% svk propset copyright 'NBC 2007' README
 M  README
% svk proplist README
Properties on README:
  copyright
% svk propget copyright README
NBC 2007
% svk propdel copyright README
 M  README
% svk proplist README
```

Les propriétés sont représentées par un nom et une valeur. Ces propriétés sont « versionnées » exactement comme le fichier. Vous pouvez faire dessus des `commit`, des `revert` et toutes les opérations classiques. Elles seront également accessibles à toutes les personnes utilisant SVK ou Subversion pour travailler sur votre projet. Si vous faites un `svk diff` sur un fichier auquel vous avez ajouté une propriété, vous verrez quelque chose comme :

```
% svk diff README

Property changes on: README
___________________________________________________________
Name: copyright
 +NBC 2007
```

Voici un certain nombre de propriétés utiles :

▶ `svn:executable`

permet de rendre un fichier exécutable

```
% svk propset svn:executable 1 script/*
```

rendra tous les fichiers du répertoire script exécutables suite à une commande `checkout` ou `update`.

▶ `svn:eol-style`

permet de traiter le « problème » des fins de ligne qui sont différentes suivant les systèmes d'exploitation. En utilisant la valeur « native », SVK utilisera automatiquement la fin de ligne normale du système sur lequel vous travaillez au moment de créer un fichier et non celle utilisée par le créateur du fichier :

```
% svk propset svn:eol-style native **/*.rb
```

Vous pouvez aussi forcer un type de fin de ligne en utilisant les valeurs CRLF, CR ou LF.

▶ `svn:ignore`

permet d'ignorer certains répertoires et fichiers qui n'ont pas à être journalisés :

```
% svk propset svn:ignore "*" tmp/
```

ignorera tous les fichiers créés dans `tmp/`

Vous pourrez trouver une liste plus complète des propriétés dans la documentation de Subversion **http://svnbook.red-bean.com/**. Attention, toutes les propriétés existantes dans Subversion ne sont pas forcément implémentées dans SVK.

Configuration de SVK

Comme nous l'avons déjà vu, SVK s'appuie sur Subversion pour la partie système de fichiers et pour la gestion des propriétés. Il s'appuie également dessus pour la partie configuration. Le fichier utilisé pour le configurer est `~/.subversion/config`. Par défaut, il est entièrement commenté.

Nous avons vu un peu plus haut (dans la section parlant de la commande `add`) que SVK ignorait certains fichiers. Cette liste est évidemment paramétrable. Pour cela, ajoutez dans le fichier de configuration :

```
[miscellany]
global-ignores = *.o *.lo *.la #*# .*.rej *.rej .*~
*~ .#* .DS_Store
```

Une deuxième configuration potentiellement intéressante est de fixer automatiquement les propriétés des fichiers suivant leur extension :

```
[miscellany]
enable-auto-props = yes
[auto-props]
*.rb = svn:eol-style=native
*.pl = svn:eol-style=native
*.png = svn:mime-type=image/png
...
```

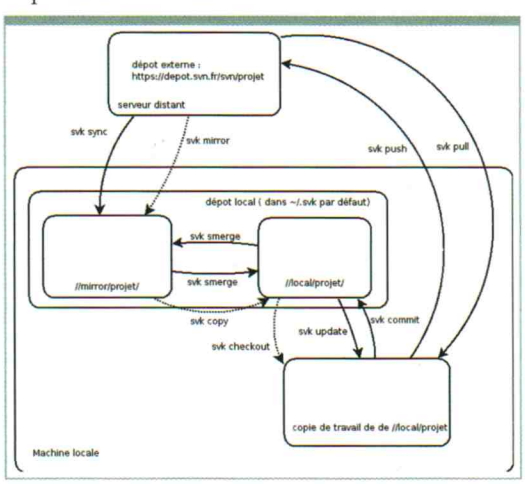
À ma connaissance, ce sont les deux seules configurations possibles dans SVK. Ceci dit, l'information étant difficile à trouver en dehors du code source, j'ai pu en louper certaines.

Éléments de méthode de travail

Cette partie n'essaye pas de proposer une méthode de travail générique, elle donne juste quelques pistes pour travailler le mieux possible avec SVK. Elle s'applique sans doute à beaucoup d'autres outils de gestion de versions.

- Évitez de travailler directement sur un dépôt miroir, préférez une copie locale.
- Faites des updates régulièrement et, entre autres, avant chaque commit, cela réduit les risques de conflit si vous travaillez à plusieurs.
- Un commit doit porter sur un changement unitaire (ne mélangez pas corrections de bugs et ajout de nouvelles fonctionnalités), doit être accompagné d'un message explicite des modifications apportées et doit laisser le logiciel dans un état stable (lancez vos tests unitaires avant de commiter).
- Ne versionnez que les fichiers utiles au projet, c'est-à-dire ne versionnez pas les journaux, les fichiers intermédiaires, etc. Pour cela, vous pouvez bien sûr utiliser la propriété `svn:ignore` vue dans la section sur les propriétés au niveau de chaque projet ou bien le `global-ignores` vu dans la section sur configuration au niveau global.
- SVK ne remplace pas la communication inter-humaine.

Voici un diagramme récapitulant les flux des commandes importantes de SVK :



SVK version 2

Pour écrire cet article, je me suis appuyé sur la version 1, mais le développement est très actif et la version 2 est sortie le 28 décembre 2006. Elle apporte, en plus des corrections de bogues, plusieurs fonctionnalités dont certaines sont vraiment très intéressantes comme :

- Les commits interactifs.

Sans doute la plus intéressante à première vue, la commande (`svk commit --interactive`) vous montre les différentes modifications effectuées et vous permet de sélectionner celles que vous voulez effectivement commiter, les autres restant à l'état de modifications dans votre copie de travail ;

- L'amélioration de la sortie des journaux.

Vous pouvez maintenant y appliquer des filtres (recherche par exemple) et modifier la sortie (standard, XML...) ;

- L'apparition des vues.

Les vues peuvent être vues comme les vues des bases de données ou comme des liens symboliques Unix. Elles permettent de référencer une arborescence sous un autre nom (plus parlant).

Une liste complète des modifications de la version 2 est disponible sur le CPAN : **http://search.cpan.org/src/CLKAO/SVK-v2.0.0/CHANGES**.

Seul inconvénient, mais de taille à mon avis, cette version n'est, sauf erreur, pas encore disponible en paquetage (sauf peut-être pour Mac OS X). Vous devrez donc l'installer à la main. À moins d'en avoir besoin et de savoir ce que vous faites, je vous conseille de rester à la version installée par votre système de gestion de paquetages et d'attendre qu'elle soit disponible dans votre système d'exploitation ou distribution préféré. Ceci étant dit, cette nouvelle version fonctionne très bien pour moi depuis quelque temps déjà.

Quelques idées en l'air

Si les outils de gestion de versions sont généralement utilisés pour gérer les sources d'un programme, il serait dommage de les limiter à cela. Ils peuvent être détournés pour faire plein de choses intéressantes et SVK a certains atouts non négligeables à détourner :

- Plutôt que de sauvegarder le répertoire `~/.svk`, utilisez toute la puissance de SVK en faisant une copie de `//local` sur un autre serveur. Vous pouvez, par exemple, sur ce second serveur faire :

```
svk ls svn+ssh://mon.serveur.principal/home/nc/.svk/local/
```

et reprendre ce que nous avons vu précédemment pour déclarer ce miroir. Vous aurez une sauvegarde à peu de frais entièrement gérée par SVK en lançant régulièrement :

```
svk pull //mirror/serveurprincipal/
```

Vous pouvez aussi bien sûr utiliser la recette pour répliquer un dépôt local sur un dépôt distant. À vous de voir ce qui vous convient le mieux.

- Comme il n'est pas limité aux fichiers textes, il peut gérer les versions d'images ou de documents d'une suite bureautique. Bien sûr, vous ne pourrez pas faire de `diff` entre deux versions.

Quoique, si vous utilisez OpenOffice.org et le format OpenDocument, il devrait être possible d'écrire un `oodiff` et d'utiliser la variable SVKDIFF pour l'utiliser à la place du `diff` standard...

Et comme il peut être associé à un serveur HTTP ou Subversion, il peut même servir à faire du travail collaboratif à plus ou moins grande échelle :

- Pour gérer plus facilement la traduction de documents comme expliqué par le développeur de SVK lui-même : **http://www.onlamp.com/pub/a/onlamp/2004/09/09/svk_translation.html**.
- Vous pouvez imaginer versionner tout ou partie de votre répertoire personnel sous SVK et faire un miroir de tout cela sur vos différents ordinateurs. Cela permet d'avoir un *backup* à peu de frais et/ou de synchroniser certains fichiers intéressants (`.zshrc`...)
- Il peut servir à un administrateur système pour versionner les fichiers de configuration de ses serveurs avec le gros avantage sur certains autres outils du même type de ne pas avoir besoin d'un répertoire spécial comme CVS ou `.svn` et de pouvoir importer sur place un répertoire. Imaginez par exemple que sur tous vos serveurs vous fassiez :

```
% svk import --to-checkout /etc/ //local/etc-serveur1
```

et qu'ensuite pour poussiez tout cela sur un serveur SVK centralisé grâce à la recette vue plus haut pour répliquer un dépôt local sur un dépôt distant. Cerise sur le gâteau, vous pouvez même le faire sans ouvrir de trou dans votre pare-feu en utilisant les redirections de `SSH`.

Depuis votre serveur central, vous pouvez voir toutes les modifications apportées à vos fichiers de configuration (bonjour ITIL). Vous pouvez comparer la version locale avec la version distante pour vérifier que rien n'a changé (voire automatiser cela et envoyer un mail en cas de changement).

Comme vous le voyez, les possibilités sont larges.

Conclusion

Notre tour d'horizon de SVK s'arrête là. Il resterait bien sûr encore des choses à dire pour être complet, comme développer tout ce que l'on peut faire avec les hooks par exemple, mais cela n'a jamais été l'ambition de cet article. J'espère seulement qu'il vous aura donné envie d'utiliser un outil de gestion de versions si vous n'en utilisez pas encore que ce soit pour développer ou pour gérer vos serveurs, et de regarder de plus près SVK si vous en utilisez déjà un.

LIENS

- **http://svk.elixus.org/** : La page du projet
- **http://svkbook.elixus.org/** : Le manuel (en cours d'écriture)
- **http://svk.bestpractical.com/** : Le wiki de SVK
- **http://www.redmine.org** : rien à voir avec SVK, mais c'est un application web de gestion de projets GPL qui mérite vraiment le détour. Simple, de bon goût et pourtant vraiment puissante.

Nicolas Chuche,

nchuche@barna.be

Nicolas Chuche est ingénieur système au ministère de l'Équipement et utilisateur de systèmes GNU/Linux et Unix depuis une dizaine d'années.

Merci aux Mongueurs de toute la francophonie qui ont assuré la relecture de cet article.

Philippe « BooK » Bruhat

Perles de Mongueurs (32)

Depuis le numéro 59, les Mongueurs de Perl vous proposent tous les mois de découvrir les scripts jetables qu'ils ont pu coder ou découvrir dans leur utilisation quotidienne de Perl. Bref, des choses trop courtes pour en faire un article, mais suffisamment intéressantes pour mériter d'être publiées. Ce sont les perles de Mongueurs.

Préparation artisanale d'un mailing

Mon épouse est enseignante et elle a récemment dit à ses élèves qu'elle allait, suite à une absence, leur envoyer par mail leur note au dernier devoir, **individuellement**.

Les notes sont disponibles dans un fichier `gnumeric`, avec les noms des élèves. Elle veut cependant se laisser la possibilité de rajouter éventuellement un message personnalisé. Sans envoyer automatiquement des emails en masse à partir d'un fichier, je voulais lui faciliter la vie autant que possible.

L'objectif est donc de prendre la liste des notes et de produire un message pour chaque élève de la classe dans sa boîte de « brouillons » (sous **Pine**, c'est le fichier `~/mail/postponed-msgs`).

Template Toolkit pour produire les messages

Qui dit message formaté, dit utilisation d'un modèle. Nous n'allons pas nous encombrer de détails. Aussi, vais-je me servir de l'outil utilisé pour tous les sites des Mongueurs et qui est de plus déjà installé sur ma machine : Template Toolkit.

Le mail sera défini dans un modèle, les emails, noms et notes des élèves le seront dans un fichier (obtenu à partir de son tableur). Nous allons donc produire un message au format mbox qu'il suffira ensuite d'ajouter à la fin de `~/mail/postponed-msgs`.

Le plus simple est évidemment d'écrire un brouillon de message avec Pine et de le sauver pour une édition ultérieure (*postpone*). Voici le résultat brut tel que sauvegardé par Pine :

```
From prof@localhost.localdomain Sun Jan 14 23:37:48 2007 +0100
Newsgroups:
Date: Sun, 14 Jan 2007 23:37:48 +0100 (CET)
From: Votre prof <prof@grande-ecole.fr>
X-X-Sender: prof@localhost.localdomain
To: "Eleve 71" <Eleve71@grande-ecole.fr>
Reply-To: prof@grande-ecole.fr
Subject: note de l'interro 2
Fcc: sent-mail
Message-ID: <Pine.LNX.4.64.0701192333580.7140@localhost.localdomain>
X-Cursor-Pos: : 129
X-Our-ReplyTo: Full
MIME-Version: 1.0
Content-Type: MULTIPART/MIXED; BOUNDARY="8323329-2067678150-
1169246090=:7187"
Status: O
X-Status:
X-Keywords:
X-UID: 1

--8323329-2067678150-1169246090=:7187
Content-Type: TEXT/PLAIN; CHARSET=iso-8859-1; format=flowed
Content-Transfer-Encoding: QUOTED-PRINTABLE

Bonsoir,

voici ta note de l'interro 2: 12

La moyenne du groupe est de 9,3. Les notes vont de 3,5 =E0 18.

A bient=F4t,

Votre prof

--8323329-2067678150-1169246090=:7187
```

Bref, c'est l'horreur : il y a plein d'en-têtes dont on se moque (et certains sont de plus supposés être des identifiants uniques ; nous devrions éviter de créer des doublons), le contenu du message est en *multipart*-MIME encodé en *quoted-printable*...

Heureusement, Pine n'est pas si bête et sait travailler à partir d'une version minimale d'un message copié dans un fichier au format mbox. Le texte suivant suffit amplement :

```
From prof@localhost.localdomain Sun Jan 14 23:37:48 2007 +0100
From: prof <prof@grande-ecole.fr>
Reply-To: prof@grande-ecole.fr>
Subject: note de l'interro 2
To: "Eleve 71" <Eleve71@grande-ecole.fr>
Fcc: sent-mail

Bonsoir,

voici ta note de l'interro 2: 12

La moyenne du groupe est de 9,3. Les notes vont de 3,5 à 18.

A bientôt,

Votre prof
```

Quand on sauvegarde le message après vérification dans Pine, on constate que Pine l'a sauvé sous une forme similaire à notre premier exemple. Parfait.

Pour utiliser notre outil de *template*, il suffit de remplacer deux lignes dans notre message épuré :

- la ligne `To:` des en-têtes :

```
To: "[% prenom %] [% nom %]" <[% email %]>
```

- et la ligne du message contenant la note :

```
voici ta note de l'interro 2: [% note %]
```

On a remplacé les informations cruciales par les variables du modèle : `[% prenom %]`, `[% nom %]`, `[% email %]`

et [% note %]. C'est avec [% %] que l'on peut insérer des directives Template Toolkit dans n'importe quel document^Wmodèle.

Regexp::Assemble pour trouver les adresses

Il reste cependant un petit problème pratique... Trouver l'email de chaque élève. En effet, si la plupart des adresses sont de la forme **Prenom.Nom@grande-ecole.fr**, ce n'est pas le cas pour toutes les adresses. L'administration a parfois dû ajouter les seconds prénoms pour gérer des problèmes d'homonymie.

Heureusement, Estelle dispose des adresses email de tous ses élèves. Il reste donc à établir la correspondance entre le nom d'un élève dans le fichier de notes et son email dans la liste d'adresses.

En fait, tout le travail a déjà été fait par David Landgren, l'auteur de Regexp::Assemble (que nous avons déjà vu dans la Perle 29).

Regexp::Assemble peut produire des *regexps* « suivies » (*tracked*), c'est-à-dire qu'il est capable de retrouver laquelle des regexps ayant servi à la construire a déclenché la correspondance.

```
my $re = Regexp::Assemble->new()->track(1);
```

Il suffit ensuite d'ajouter les expressions avec add().

Dans les quelques classes dont elle a la charge, il n'y a pas deux élèves qui portent le même nom. Nous allons donc utiliser les noms de famille des élèves comme clés d'un hachage dont les valeurs sont les adresses correspondantes, sans nous préoccuper des problèmes d'homonymie.

Cette technique de table de correspondance basée sur des expressions régulières est extrêmement puissante, et d'une grande simplicité à mettre en œuvre grâce au travail de David.

Au final, le script est assez court :

```
#!/usr/bin/perl
use strict;
use warnings;
use Template;
use Regexp::Assemble;

# utilisation: prepare_emails.pl notes.csv emails.txt modele.tt

# noms de fichiers
my ( $notes, $emails, $modele ) = @ARGV;
my $fh;

# récupération de la liste des emails
# pour construire une table de distribution
open $fh, $emails or die "Impossible d'ouvrir $emails: $!";
my %emails = map {
    chomp;              # supprime le saut de ligne final
    /\.([^.]*)\@/;      # trouve le nom dans l'adresse email
    ( my $re = $1 ) =~ s/-/\\W/g;    # remplace les - par \W
    ( $re => $_ )       # produit la table de distribution
} <$fh>;
close $fh;
# invoque la puissante magie de Regexp::Assemble
my $re
   = Regexp::Assemble->new( flags => 'i' )->track(1)->add( keys %emails );
# création de l'objet Template
my $tt = Template->new();
# ouverture du fichier de notes
open $fh, $notes or die "Impossible d'ouvrir $notes: $!";
# boucle de lecture des données élèves
while (<$fh>) {
    next if /^\s*(?:#|$)/;    # ignore commentaires et lignes blanches
    chomp;                    # supprime le saut de ligne final
    my %vars;
    @vars{qw( nom prenom note )} = split /,/;    # CSV
    if ( $re->match( $vars{nom} ) ) {
        $vars{email} = $emails{ $re->matched() };
    }
    else {
        # message d'erreur si l'email n'a pas été trouvé
        print STDERR
            "Impossible de trouver l'email de @vars{qw(prenom nom)}\n";
        next;
    }
    # produit le message
    $tt->process( $modele, \%vars, \*STDOUT )
        or die $tt->error();
}
close $fh;
```

Ce script affiche le résultat sur la sortie standard, afin de faciliter la vérification des messages produits. Pour ajouter les messages aux brouillons de Pine, il suffit de faire :

```
./prepare_emails.pl notes.csv emails.txt modele.tt >> ~/mail/postponed-msgs
```

Il ne lui restera plus qu'à vérifier les messages et à les envoyer. Quant à moi, j'ai enfin pu utiliser les regexps suivies de Regexp::Assemble, et, surtout, j'ai une nouvelle fois réussi à convaincre ma douce de l'intérêt d'avoir un Mongueur de Perl à la maison ! ;-)

Philippe « BooK » Bruhat,

book@mongueurs.net,
de Lyon.pm et Paris.pm

À vous !

Envoyez vos perles à **perles@mongueurs.net**. Elles seront peut-être publiées dans un prochain numéro de *GNU/Linux magazine*.

Emile « iMil » Heitor

Jugamos a La Fonera

1er février, 17h30, Solutions Linux 2007

Nous sommes à *Solutions Linux*, contents de notre petit spectacle « décontamination du salon » ; nous discutons avec diverses *assoces* de choses et d'autres. Et puis, nous rencontrons touff, de l'AFPY, qui nous fait l'apologie d'une petite boite blanche, un routeur Wireless pas plus grand qu'une main, dont le modèle de diffusion ainsi que l'ambition nous accrochent immédiatement. FON, la boite qui distribue ce routeur, ne vise ni plus ni moins que de recouvrir la surface de la planète de ces petits boîtiers, et de permettre à chaque possesseur de Fonera de pouvoir se connecter sur n'importe quelle autre Fonera du monde. Wow !

Ni une ni deux, je m'en vais m'enquérir sur les tenants et aboutissants de la chose et lis sur le site de la société en question les modalités d'inscription ainsi que les diverses formules de connexion que propose FON. Trois cas de figure sont exposés :

- Modèle Linus, je partage ma connexion Internet, et je peux en contrepartie utiliser n'importe quel autre *Access Point* FON gratuitement.
- Modèle Bill, je partage ma connexion Internet contre paiement de l'utilisateur, et je reverse 50% des gains à FON. Je ne bénéficie naturellement pas dans ce cas d'une connexion gratuite sur les autres AP FON.

Bien évidemment, c'est le modèle Linus qui remporte le plus de succès. C'est le fondement de l'opération, et désormais, je me connecte sur **http://maps.fon.com** avant de partir où que ce soit.

Moyennant 3€ par jour, on peut également devenir « Alien », plus précisément, utilisateur d'un AP FON alors que l'on ne possède pas soi-même de Fonera.

J'en veux une !

Par le biais d'une opération promotionnelle, je commande ma Fonera à moindres frais. Le site m'indique que la livraison prendra une à trois semaines. Je profite de ce délai pour vérifier ce que l'association *wireless-fr* m'avait expliqué pendant ladite *Solution Linux*, à savoir qu'il était a priori possible de flasher le *firmware* FON et de le remplacer ainsi par le leur. Ce fût le début des insomnies. De fil en aiguille, je découvre que, fatalement, le firmware en question est un GNU/Linux, que les sources sont disponibles, qu'il s'agit d'une version modifiée d'OpenWRT, que le CPU est un MIPS. Et je commence à réfléchir au nouveau cycle de sommeil qui se mettra en place dans les prochaines semaines.

Pendant mon inscription, je remarque un petit graffiti en bas à gauche du site, *labellé* « *Fonero Gets Fonero* ». Se confirment alors les dires de l'ami touff, pour chaque Fonera reçue, un *fonero* – c'est ainsi que se nomme un membre de la communauté FON – peut « inviter » un ami. Dans les faits, l'ami en question recevra un coupon par mail qui lui permettra de commander une Fonera moins cher*, puis une fois inscrit, de lui-même inviter d'autres amis. Ce truc est un virus. Pour finir sur l'aspect propagation, une fois que le fonero invité a enregistré sa Fonera, vous recevez une nouvelle invitation, et on repart pour un tour.

* Jusqu'au 15 avril 2007, le programme de promotion fonero-get-fonero fonctionnait différemment. Une invitation donnait droit à une Fonera gratuite, port compris.

12 février 2007, 19h00

Elle est là. Je suis impressionné du packaging de l'ensemble. Je déballe. Je lis rapidement les instructions puis, inconscient que je suis, je branche immédiatement mon nouveau jouet au Net. Vous comprendrez plus tard que c'était une erreur. Je me jette alors sur ma *worksation* munie d'une carte wireless, puis un rapide scan confirme ce qu'annonce la documentation. Nous voyons deux SSID en provenance de notre petite boite blanche :

```
Cell 01 - Address: 00:18:84:25:93:F6
          ESSID:"MyPlace"
          Protocol:IEEE 802.11b
          Mode:Managed
          Frequency:2.417 GHz (Channel 2)
          Quality:0/100  Signal level:-36 dBm
             Noise level:-256 dBm
          Encryption key:on
```

```
          Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s;
          11 Mb/s; 6 Mb/s
          9 Mb/s; 12 Mb/s;
          18Mb/s; 24 Mb/s; 36 Mb/s
          48 Mb/s; 54 Mb/s
          Extra:bcn_int=100
          Extra:atim=0
          IE: WPA Version 1
          Group Cipher : TKIP
          Pairwise Ciphers (1) : TKIP
          Authentication Suites (1) : PSK
Cell 02 - Address: 00:18:84:25:93:F5
          ESSID:"FON_AP"
          Protocol:IEEE 802.11b
          Mode:Managed
          Frequency:2.417 GHz (Channel 2)
          Quality:0/100  Signal level:-36 dBm
          Noise level:-256 dBm
          Encryption key:off
          Bit Rates:1 Mb/s; 2 Mb/s;
          5.5 Mb/s; 11 Mb/s; 6 Mb/s
          9 Mb/s; 12 Mb/s; 18 Mb/s;
          24 Mb/s; 36 Mb/s; 48 Mb/s; 54 Mb/s
          Extra:bcn_int=100
          Extra:atim=0
```

`MyPlace` est votre réseau privé. Il utilise un cryptage WPA parfaitement supporté par `wpa_supplicant`. Le mot de passe de ce réseau est le *serial number* inscrit sur votre Fonera. Premières analyses basiques. La Fonera embarque à coup sûr un serveur DHCP, et, manifestement, le pool du réseau `MyPlace` est le 192.168.10.0/24 et le pool public `FON_AP` est le 192.168.182.0/24.

J'ajoute les lignes suivantes à mon `/etc/network/interfaces`, le premier client étant une Debian :

```
auto ath0
iface ath0 inet dhcp
    wpa-driver wext
    wpa-conf /etc/wpa_supplicant/fonera.conf
```

Et je remplis le fichier mentionné :

```
network={
        ssid="MyPlace"
        # cette clé est fausse
        psk=95843a1bf89df8eqbd27aa5309e24025fb426ec5da
f25567efad34db33f06fa6
}
```

Je note rapidement qu'une gentille interface web équipe le routeur, accessible par défaut sur 192.168.10.1 ou 169.254.255.1. Cette dernière permet de réaliser quelques opérations de base telles que le changement du SSID Privé, son mode de chiffrement, le pool d'adresses privées, activation de règles de *forwarding* de ports ou encore une gestion très simplifiée des règles de *firewalling*.

Ces embryons de recherche ayant attisé ma curiosité, je décide qu'il est temps d'obtenir un *shell* sur la boite. Naïf, je tente :

```
# nmap 192.168.10.1
Starting Nmap 4.10 (http://www.insecure.org/nmap/) at
2007-03-25 13:12 CEST
Interesting ports on fonera (192.168.10.1):
Not shown: 1675 closed ports
PORT     STATE SERVICE
53/tcp   open  domain
80/tcp   open  http
8080/tcp open  http-proxy
```

Évidemment...

Fowned

Démarre alors une séance de recherche intensive sur l'art et la manière de débrider La Fonera, et, après quelques heures de lectures diverses qui parfois se contredisent, voici un concentré des informations pertinentes :

Lors de son démarrage, l'équipement se connecte sur **download.fon.com**, télécharge le dernier firmware disponible et s'*upgrade*. Les exploits connus pour activer un serveur ssh au moment où je l'ai reçue fonctionnaient jusqu'au firmware 0.7.1.1 inclus. Or, le dernier firmware disponible upgrade La Fonera en 0.7.1.2.

L'unique méthode à ce moment était un `reset-to-factory-defaults`, suivi d'une injection de commandes en utilisant l'interface HTTP basique de la Fonera. Voici la manœuvre :

- Débrancher La Fonera de l'Internet.
- La brancher sur un lien Ethernet d'une machine Unix NON connectée à l'Internet.
- Configurer l'interface Ethernet de la machine sur 169.254.255.2/24.
- Appuyer sur le bouton « reset » de La Fonera pendant exactement 20 secondes.

- Depuis la station Unix, *ping'ez* 169.254.255.1 :
- La Fonera répondra pendant encore quelques instants puis redémarrera sur son firmware d'origine, 0.7.1.1 dans mon cas.
- Attendre que l'équipement réponde à nouveau.
- Télécharger **http://stefans.datenbruch.de/lafonera/scripts/fondue.pl**.
- Exécuter la commande suivante :

```
echo -e '$(/usr/sbin/iptables -I INPUT 1 -p tcp \
--dport 22 -j ACCEPT)\n$(/etc/init.d/dropbear)'| \
perl ./fondue.pl 169.254.255.1 admin
```

À partir de cet instant, le démon ssh léger `dropbear` est démarré. Il devrait donc être possible de se connecter à notre nouveau joujou grâce au couple *login/password* admin/admin :

```
admin@192.168.10.1's password:

BusyBox v1.1.3 (2006.11.21-19:49+0000) Built-in shell
(ash)
Enter 'help' for a list of built-in commands.
 _____  _____  _____
|  ___||       ||  _  |
|  ___||   -   ||  | |  |
|  |    |_____||__| |__|
|__|

 Fonera Firmware (Version 0.7.1 rev 1) ------------
  *
  * Based on OpenWrt - http://openwrt.org
  * Powered by FON - http://www.fon.com
 ---------------------------------------------------
root@OpenWrt:~#
```

Nous y voilà. Mais reprenons. Qu'est-ce donc que ce `fondue.pl` et qui est donc Stefans Datenbruch? Nous l'avons dit plus haut, le routeur dispose d'une petite interface HTTP. Cette dernière n'est autre qu'un serveur ultra léger qui effectue son rendu et ses traitements à travers ce qu'on pourrait associer à des CGI en shell. Inutile de préciser que c'est évidemment par ce biais que les premiers bidouilleurs de tout poil ont démarré leurs méfaits. En l'occurrence, `fondue`, le dernier d'une série d'exploits basés sur l'interface web de La Fonera, utilise une faille de la page de gestion du port forwarding (corrigée en 0.7.1.2) qui permet d'exécuter arbitrairement des commandes sur La Fonera. En réalité, il eut été aussi aisé de simplement copier-coller ceci :

```
$(/usr/sbin/iptables -I INPUT 1 -p tcp --dport 22 \
 -j ACCEPT)\n$(/etc/init.d/dropbear)
```

directement dans le champ de l'interface, mais vous m'accorderez que c'est tout de même moins classe. Suite de l'aparté, deux des sites les plus actifs, créateurs de la plupart des exploits destinés à La Fonera sont **http://stefans.datenbruch.de/lafonera/** et **http://mrmuh.blogspot.com/**. À eux seuls, ces deux sites disposent de la totalité des informations nécessaires à l'« ouverture » d'une Fonera, de la version 0.7.0.4 à la version 0.7.1.2... incluse. Nous y reviendrons.

Reprenons nos investigations. Un rapide coup d'œil à la *crontab* nous montre ceci :

```
0       *       *       *       *       /bin/thinclient cron
```

Et, en vérité, je vous le dis, `thinclient` est notre ennemi, car l'upgrade automatique de votre routeur, c'est lui. Pour éviter que cela ne se reproduise, je vous invite donc à commenter la ligne suivante (oui, `thinclient` est un script) :

```
#. /tmp/.thinclient.sh
```

Et afin de rendre automatique le lancement de `dropbear` à chaque *reboot*, on réalise cette petite « porcinerie » :

```
root@OpenWrt:~# mv /etc/init.d/dropbear /etc/init.d/S50dropbear
```

Et voilà, elle est vôtre.

Et ma 0.7.1.2, monsieur ?

J'y venais. Si cette petite expérience nous a permis d'en découvrir un peu plus sur le fonctionnement de notre Fonera, elle est devenue totalement obsolète le mois dernier. En effet, une méthode bien plus raffinée a vu le jour le 13 mars dernier. Kolofonium, c'est son nom, tire profit d'une « faille » dans l'authentification Radius. Nous pouvons lire sur le site de Stefans Datenbruch l'explication suivante :

Lors de la séquence de boot, le script `/etc/init.d/N50chillispot` est exécuté. Ce dernier appelle l'exécutable `/usr/sbin/chilli_radconfig` qui prend en argument l'adresse du serveur radius de FON, **radius01.fon.com**, et comme il est possible dans l'interface web de La Fonera de spécifier un serveur DNS différent, vous comprenez bien qu'il est trivial de rediriger **radius01.fon.com** vers l'IP que vous souhaitez. Reste à configurer votre serveur Radius de manière judicieuse.

Installons tout d'abord le serveur Radius recommandé par Stefan :

```
# apt-get install xtradius
```

Puis ajoutons l'IP de La Fonera dans la liste des clients :

```
# echo "192.168.10.1    garrafon" >> /etc/raddb/clients
```

On définit un ID unique pour l'équipement :

```
# cat > /etc/raddb/dictionary.fon << EOF
VENDOR          Fon             14559
ATTRIBUTE       CHILLICONF      6       string  Fon
EOF
```

Et nous ajoutons enfin les paramètres de l'utilisateur `chillispot` :

```
# cat > /etc/raddb/users << EOF
chillispot Auth-Type = External
        Exec-Program-Wait = "/etc/raddb/fonera.sh",
        CHILLICONF = "radiusserver1 radius01.fon.com",
        CHILLICONF = "radiusserver2 radius02.fon.com",
        CHILLICONF = "radiussecret garrafon",
        CHILLICONF = "dhcpif eth1",
```

```
        CHILLICONF = "uamsecret garrafon",
        CHILLICONF = "uamanydns",
        CHILLICONF = "uamallowed www.martinvarsavsky.
net,www.google.com,www.flickr.com,static.flickr.
com,video.google.com,216.239.51.0/24,66.249.81.0/24",
        CHILLICONF = "uamallowed www.fon.com,www.
paypal.com,www.paypalobjects.com,www.skype.com,66.249.
93.0/24,72.14.207.0/24,72.14.209.0/24,84.96.67.0/24,21
3.91.9.0/24,80.118.99.0/24",
        CHILLICONF = "uamallowed shop.fon.
co.kr,secure.nuguya.com,inilite.inicis.com,fon-
en.custhelp.com,maps.fon.com,c20.statcounter.com",
        CHILLICONF = "uamserver https://login.fon.
com/cp/index.php",
       CHILLICONF = "# Greetings from Michael and Stefan",
        CHILLICONF = "# http://mrmuh.blogspot.com/ &
http://stefans.datenbruch.de/lafonera/",
        CHILLICONF = "uamallowed stefans.datenbruch.de",
        CHILLICONF = "ipup /etc/init.d/dropbear",
        Fall-Through = 0
EOF
```

Vous l'aurez compris, la ligne clé de cette configuration est l'avant-dernière, qui demande le démarrage de `dropbear`.

```
Le script /etc/raddb/fonera.sh contient une simple
directive :
exit 0
```

Reste à redémarrer La Fonera en ayant préalablement modifié l'entrée DNS vers votre DNS « spécial ». Vous disposerez alors d'un serveur ssh actif sur votre routeur wireless.

Méthode « spécial loutre » :

- Modifiez l'adresse DNS de votre Fonera en 88.198.165.155.
- Rebootez La Fonera.

L'IP en question pointe sur le *host* **kolofonium.datenbruch.de**, maintenu par Stefan, et utilisera son serveur Radius public modifié qui ordonnera à votre Fonera de démarrer `dropbear` lors de la séquence de boot.

0.7.1r1-sd2

Alors que vous avez réalisé un de ces déblocages, vous imaginez certainement que maintenant que vous avez désactivé l'upgrade automatique, votre Fonera ne suivra plus le *stream* officiel de FON, et par conséquent, que les failles potentielles des logiciels embarqués ne seront plus corrigées : que nenni ! En réalité, ces upgrades sont librement téléchargeables, et par le biais d'une petite manipulation, parfaitement applicables à votre système libéré. Encore une fois, M. Datenbruch a défriché le terrain pour nous. La méthode est relativement simple : le script exécuté par La Fonera pour se mettre à jour est le suivant (ici pour un firmware 0.7.1r1) :

```
cd /tmp
wget http://download.fon.com/firmware/update/0.7.1/\
1/upgrade.fon
/bin/fonverify /etc/public_fon_rsa_key.der \
/tmp/upgrade.fon
rm -f /tmp/.thinclient.sh
exit
```

Comme on peut s'y attendre, le fichier **http://download.fon.com/firmware/update/0.7.1/1/upgrade.fon** est téléchargeable. En analysant le script **/bin/fonverify**, on s'aperçoit que le fichier **upgrade.fon** est découpé de la façon suivante :

- un en-tête ASCII contenant la chaîne `FON` suivie d'un entier définissant le type d'upgrade :

```
if [ "$VERSION" = "FON3" ]; then
        echo "This is a FON reflash v2 archive"
elif [ "$VERSION" = "FON4" ]; then
        echo "This is a FON hotfix v2 archive"
else
        echo "ERROR: This is not a FON v2 archive"
        rm $VERSION_FILE
        rm $FON_FILE
        exit 1
fi
```

Effectivement :

```
$ head -c 4 upgrade.fon
FON4
```

- On peut ensuite voir que trois bytes sont lus afin de définir un offset qui représente également une longueur de clé RSA :

```
$ dd if=$FON_FILE of=$OFFSET_FILE bs=1 count=3 skip=4
> /dev/null 2>&1
OFFSET=$(expr $(cat $OFFSET_FILE))
if [ $OFFSET -eq 0 ]; then
        echo "ERROR: Offset too small"
        rm $OFFSET_FILE
        rm $FON_FILE
        exit 1
fi
```

À nouveau :

```
$ head -c 7 upgrade.fon |tail -c 3
512
```

- Et enfin, on trouvera à l'adresse `$OFFSET + 7` (7 provenant des 4 + 3 premiers bytes) une bête archive `tar.gz` contenant les upgrades effectifs :

```
$ tail -c +520 upgrade.fon |tar ztvf -
-rw-r--r-- iurgi/iurgi   18171 2006-12-22 13:01
           upgrade_0712.tgz
-rwxr-xr-- iurgi/iurgi     263 2006-12-01 11:29
           upgrade
-rw-r--r-- iurgi/iurgi     113 2006-11-27 17:17 hotfix
```

Le fichier upgrade est un simple script shell qui décompresse `upgrade_0712.tgz`, qui lui-même contient les binaires, fichiers de configuration et autres mises à jour. Deux précautions valant mieux qu'une, vérifiez systématiquement que l'`upgrade_*.tgz` ne viendra pas écraser une configuration soignée aux petits oignons, et qu'en particulier aucun `rc.d` ne viendra se charger de désactiver votre accès ssh.

À l'étroit

Cela ne fait plus l'ombre d'un doute, nous avons pris possession de notre Fonera, certainement changé le mot de passe *root*, jeté un œil aux *init-scripts*, constaté que le *busybox* fourni manque cruellement de quelques commandes, et pourquoi pas modifié quelques *rules* de firewalling dans `/etc/firewall.user`. Je ne vous ferais pas l'affront, dans GLMF, de vous rappeler comment ajouter également une ou deux redirections de ports à l'aide d'IPtables. Mais vous avez également noté quelque chose de beaucoup plus embarrassant :

```
root@OpenWrt:~# df -h
Filesystem            Size      Used Available Use% Mounted on
none                  7.0M     60.0k      6.9M   1% /tmp
/dev/mtdblock/2       5.4M    720.0k      4.7M  13% /jffs
/                     1.5M      1.5M         0 100% /
```

Et pas l'ombre d'un module NFS ni même CIFS. Ça fait léger. C'est à cet instant que nous remarquons la présence d'un `/etc/ipkg.conf`... se pourrait-il que ? nooon... eh si : le site **http://www.gcd.org/fonera** regroupe une foule de *packages* pour notre routeur, et parmi eux `kmod-fs-nfs_2.4.32-ar531x-1_mips.ipk`, `kmod-fs-cifs_2.4.32-ar531x-1_mips.ipk` ou encore `kmod-fuse_2.4.32+2.5.3-ar531x-1_mips.ipk`. Loin d'être le seul *repository*, j'ai trouvé `gcd.org` plus sérieux et mieux suivi. On ajoute donc ceci à `/etc/ipkg.conf` :

```
echo "src gcd http://www.gcd.org/fonera" >> /etc/ipkg.conf
```

Et on lance :

```
root@OpenWrt:~# ipkg update
```

À l'heure où j'écris ces lignes, et depuis un certain temps déjà, le repository par défaut présent dans l'`ipkg.conf`, à savoir **http://download.fon.com/release/fonera/0.7/packages**, renvoie un 404. Ça fait désordre.

Grâce à ce jeu de packages supplémentaire, on sera par exemple en mesure de faire fonctionner le multiposte d'un opérateur connu en installant le package `kmod-ipt-nat-extra` qui contient entre autres `ip_conntrack_rtsp` et `ip_nat_rtsp`. On prendra soin de commenter les modules inutiles dans le fichier `/etc/modules.d/40-ipt-nat-extra` :

```
#ip_conntrack_amanda
#ip_conntrack_proto_gre
#ip_nat_proto_gre
#ip_conntrack_h323
#ip_nat_h323
#ip_conntrack_mms
#ip_nat_mms
ip_conntrack_rtsp
ip_nat_rtsp
#ip_conntrack_pptp
#ip_nat_pptp
#ip_nat_snmp_basic
#ip_conntrack_tftp
```

Mais au fait, comment diable Hiroaki SENGOKU, mainteneur de ce repository, et tous ces autres contributeurs sont-ils en mesure de recompiler leurs applications favorites pour La Fonera ?...

Vous codez mademoiselle ?

Elles ne sont pas légion, ces boites qui suivent scrupuleusement la GPL, un rapide tour sur le site **http://gpl-violations.org/** permet de s'en convaincre. FON fait partie de ces rares sociétés qui ont compris qu'on ne se sert pas impunément dans le Communauté sans reverser le fruit de son travail, surtout lorsqu'il est constitué de différentes briques libres et que la tâche du constructeur se résume à assembler intelligemment cette manne providentielle.

C'est ainsi qu'à cette adresse : **http://download.fon.com/firmware/fonera/latest/fonera.tar.bz2**, on trouvera les sources complètes de la distribution FON basée sur OpenWRT. Amen.

Avant de nous lancer dans le « portage » simple d'une application, voyons les quelques étapes préalables. Évidemment, nous n'utiliserons pas directement La Fonera pour programmer ni même compiler, mais tirerons profit de capacités de *cross-compilation* d'un GNU/Linux tournant sur une machine x86 tout ce qu'il y a de plus classique.

Téléchargeons et décompressons lesdites sources :

```
$ mkdir ~/fonera && cd ~/fonera
$ wget -q -O - http://download.fon.com/firmware/\
fonera/latest/fonera.tar.bz2 | tar jxvf -
```

On prépare l'environnement via un classique :

```
make V=99 menuconfig
/// makeconfig.png ///
```

Aucune configuration particulière n'est nécessaire si vous ne souhaitez pas modifier l'OS par défaut, mais simplement construire un environnement de compilation. On lance donc la compilation de l'ensemble :

```
make V=99
```

Si tout s'est bien déroulé, nous sommes désormais en possession de tous les outils nécessaires à la compilation d'applications pour le MIPS qui équipe notre routeur. Complétons un peu notre `${PATH}` afin de ne pas avoir à taper le chemin complet des outils :

```
PATH=$PATH:/home/pinpin/fonera/staging_dir_mips/bin;
export PATH
```

Nous sommes en possession :

- d'un préprocesseur, `mips-linux-uclibc-cpp`
- d'un compilateur, `mips-linux-uclibc-gcc`
- des outils de gestion d'archive, `mips-linux-uclibc-ar`, `mips-linux-uclibc-ranlib`

▶ d'un linker, `mips-linux-uclibc-ld`

Ainsi que d'outils variés relatifs à la manipulation des binaires, tels que `mips-linux-uclibc-strip` qu'il sera judicieux d'appeler systématiquement à l'issue de la génération d'une application.

Il est temps d'écrire notre premier programme C pour La Fonera !

```
cat > test.c << EOF
#include <stdio.h>
int
main(int argc, char *argv[])
{
    printf("je suis un binaire et je tourne sur une
Fonera!\n");
    return 0;
}
EOF
```

Compilons-le à l'aide de notre nouvel environnement :

```
$ mips-linux-uclibc-gcc test.c -o test
```

Puis envoyons-le sur La Fonera :

```
$ scp test root@192.168.10.1:/tmp
```

Enfin, sur le routeur :

```
root@OpenWrt:~# /tmp/test
je suis un binaire et je tourne sur une Fonera!
```

À ce moment précis de l'histoire, un puissant frisson doit parcourir votre échine.

Le logiciel choisi pour effectuer notre test de portage est le légendaire OpenSSH. Attention toutefois, si la compilation et, probablement, l'exécution fonctionnent, vous vous souvenez que notre Fonera n'est munie que de 16M de RAM, dont seul 1 petit méga, et encore, est vaguement disponible. Préférez donc les applications peu gourmandes en mémoire et, surtout, si vous décidez de vous lancer dans la contribution, ayez cette contrainte à l'esprit et programmez LIGHT !

Commençons !

Un des secrets de la réduction dans la taille des exécutables réside dans le nombre de *features* que nous activons ou pas au moment du `./configure`. Aussi, nous allons soigneusement « *disabler* » tout ce qui peut l'être :

```
$ wget -O - -q ftp://ftp.fr.openbsd.org/pub/OpenBSD/\
OpenSSH/portable/openssh-4.5p1.tar.gz|tar zxvf -
$ cd openssh-4.5p1
$ ./configure --help
[snip]
```

Nous devrions pouvoir annuler quelques-unes de ces fonctions sans trop de problèmes. Il faut maintenant appeler le script configure en lui passant plusieurs informations relatives à la cross-compilation :

▶ `host=mips` afin de spécifier l'architecture cible.

▶ `CC=mips-linux-uclibc-gcc` dira quel compilateur utiliser.

▶ `CFLAGS=-I/home/pinpin/fonera/staging_dir_mips/usr/include` indique l'emplacement des *includes* à utiliser.

▶ `LDFLAGS="-L/home/pinpin/fonera/staging_dir_mips/usr/lib -lcrypt"` indique l'emplacement des bibliothèques et demande de lier sur la `libcrypt`.

Ce qui nous donne :

```
CFLAGS=-I/home/pinpin/fonera/src/fonera-src/staging_dir_mips/usr/include \
LDFLAGS="-L/home/pinpin/fonera/src/fonera-src/staging_dir_mips/usr/lib -lcrypt" \
CC=mips-linux-uclibc-gcc ./configure --host=mips --disable-largefile --disable-
lastlog --disable-utmp --disable-utmpx \
--disable-wtmp --disable-wtmpx --disable-libutil --disable-pututline --disable-
pututxline --without-skey --without-tcp-wrappers \
--without-libedit --without-ssl-engine --without-pam --without-rand-helper --
without-sectok --without-selinux --without-kerberos5 \
--without-md5-passwords --without-shadow
```

Et donne le résultat suivant :

```
OpenSSH has been configured with the following options:
                     User binaries: /usr/local/bin
                   System binaries: /usr/local/sbin
               Configuration files: /usr/local/etc
                   Askpass program: /usr/local/libexec/ssh-askpass
                     Manual pages: /usr/local/share/man/manX
                         PID file: /var/run
  Privilege separation chroot path: /var/empty
            sshd default user PATH: /usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
                    Manpage format: doc
                       PAM support: no
                   OSF SIA support: no
                 KerberosV support: no
                   SELinux support: no
                 Smartcard support: no
                     S/KEY support: no
               TCP Wrappers support: no
               MD5 password support: no
                    libedit support: no
  Solaris process contract support: no
       IP address in $DISPLAY hack: no
           Translate v4 in v6 hack: no
                  BSD Auth support: no
              Random number source: OpenSSL internal ONLY

                Host: mips-unknown-elf
            Compiler: mips-linux-uclibc-gcc
    Compiler flags: -I/home/pinpin/fonera/src/fonera-src/staging_dir_mips/usr/
include -Wall -Wpointer-arith -Wuninitialized -Wsign-compare -std=gnu99
Preprocessor flags :
    Linker flags: -L/home/pinpin/fonera/src/fonera-src/staging_dir_mips/usr/lib -lcrypt
         Libraries:   -lcrypto -lutil -lz  -lresolv -lresolv
```

Reste à construire OpenSSH en utilisant la commande `make`. À l'issue du `build`, nous obtenons plusieurs binaires, dont ssh :

```
$ du -sh ssh
552K    ssh
```

Essayons de gagner quelques kilooctets supplémentaires :

```
$ mips-linux-uclibc-strip --strip-all ssh
$ du -sh ssh
512K    ssh
```

Nous n'installerons pas directement les outils fraîchement compilés sur le *filesystem* natif de La Fonera. En effet, afin de ne pas le surcharger, j'ai monté une partition `/usr/local` en CIFS sur un répertoire partagé à l'aide de Samba sur ma station Linux x86. Pourquoi CIFS me demanderez-vous ? Non, je ne dispose d'aucun système propriétaire chez moi, je ne suis pas non plus sado-masochiste, mais voilà, si le montage NFS s'effectue sans problèmes, le transfert de fichiers à travers le montage NFS de ma station vers La Fonera fait *loader* le petit routeur blanc jusqu'à le faire crasher. Donc : CIFS. Voici le résultat d'un `df -h` :

```
Filesystem            Size      Used Available Use% Mounted on
none                  7.0M     64.0k      6.9M   1% /tmp
/dev/mtdblock/2       5.4M    720.0k      4.7M  13% /jffs
/                     1.5M      1.5M         0 100% /
//tatooine/fonera    70.5G     41.0G     29.5G  58% /usr/local
```

Pour simplifier encore le portage et la création d'applications, je réalise les opérations dans `/home/user/fonera/src`, `home/user/fonera` étant le répertoire exporté via Samba. Sans copier quoi que ce soit, nous allons donc pouvoir vérifier que notre client OpenSSH « fonerisé » fonctionne :

```
root@OpenWrt:/usr/local/src/own/openssh-4.5p1# ./ssh
usage: ssh [-1246AaCfgkMNnqsTtVvXxY] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-e escape_char] [-F configfile]
           [-i identity_file] [-L [bind_address:]port:host:hostport]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-R [bind_address:]port:host:hostport] [-S ctl_path]
           [-w local_tun[:remote_tun]] [user@]hostname [command]
```

Et plein d'émoi :

```
root@OpenWrt:/usr/local/src/own/openssh-4.5p1# ./ssh
pinpin@tatooine
socket: Address family not supported by protocol
The authenticity of host 'tatooine (192.168.10.10)' can't be
established.
RSA key fingerprint is 57:b7:ca:d6:3a:ca:69:d7:cf:b7:5a:78:7c:
a4:57:a3.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'tatooine,192.168.10.10' (RSA) to the
list of known hosts.
pinpin@tatooine's password:
Linux tatooine 2.6.18 #1 Thu Feb 1 19:50:13 UTC 2007 i686

You have new mail.
Last login: Thu Mar 29 22:53:49 2007
pinpin@tatooine:~$
```

Ça marche!

Désolé pour vos nuits

Si vous êtes un hacker – au sens noble du terme – normalement constitué, vous devez avoir dessiné plusieurs dizaines de plans de domination du monde tout au long de cet article. Le projet est encore récent, mais il se propage à la vitesse (et à la façon ;) d'un virus, les *maps google* en sont la preuve indéniable (**http://maps.fon.com/**). Parallèlement à ce constat, une communauté grandissante se regroupe autour de la petite boite blanche et contribue, de manières tout à fait variées, tant au perfectionnement du firmware qu'aux méthodes les plus incroyables pour faire rayonner La Fonera le plus loin possible (**http://www.blogin.it/fonera4.php**, **http://blog.fon.com/fr/archive/foneros/oa-avezvous-placa-votre-fonera-10.html**).

La communauté FON française a ceci de particulier qu'elle bénéficie du soutien de FON France. De fait, les initiatives, individuelles ou collectives, sont en pleine explosion. Notons par exemple le site **http://www.francofon.fr/** qui regorge d'informations de tout type et qui publie un firmware indépendant (mais compatible FON), le canal IRC **#fon-fr** ou moult bidouilleurs partagent leurs expériences, les coups de pouce de **http://toonux.com/**, et plus récemment, l'ouverture d'un site Trac, soutenue par FON France, dédié au développement communautaire du firmware FON (**http://trac.fondev.org/firmware**).

Pour finir, si les informations présentes dans cet article permettent évidemment de faire de sa Fonera ce que bon vous semble, il est clair que le projet n'a de valeur que si l'on coopère au mouvement. Essayez donc à l'issue de vos expériences à venir de garder le `SSID FON_AP` ouvert et de permettre ainsi aux autres foneros de pouvoir utiliser un accès à Internet sans fil près de chez vous ;)

Emile « iMil » Heitor,

imil@home.imil.net

LIENS

- **http://www.fon.com/fr/**, le site de FON
- **http://blog.fon.com/fr/**, le blog FON francophone
- **http://www.francofon.fr/**, communauté FON francophone
- **http://trac.fondev.org/firmware**, Trac du projet firmware communautaire FON
- **http://stefans.datenbruch.de/lafonera**, l'homme qui parlait à l'oreille des Foneras
- **http://mrmuh.blogspot.com/**, son collègue
- **http://dd-wrt.com/wiki/index.php/La_Fonera_Flashing**, flasher La Fonera
- **http://downloads.openwrt.org/people/mbm/mips/packages/**, packages OpenWRT compatibles Fonera
- **http://www.gcd.org/fonera/**, packages tierce partie
- **http://www.oesf.org/index.php?title=Pdaxrom:_Create_An_Ipk_Howto**, créer un package ipkg
- **http://www.rince.fr/spip.php?article85**, quelques astuces de configuration pour iptables sur OpenWRT

S. Ducasse, H. Fernandes

Tests unitaires en Smalltalk

Lors de la phase de maintenance ou d'évolution d'une application, la modification des classes et méthodes peut conduire à des conséquences souvent difficiles à prévoir. Ce problème est d'autant plus vrai qu'en Smalltalk le code source est un matériau très malléable, à un point tel qu'il est relativement aisé de modifier et d'étendre le système lui-même. Pour contrôler la portée de modifications faites à un code source, nous pouvons utiliser le principe des tests unitaires. Il s'agit d'un procédé pour contrôler automatiquement le comportement d'une classe et de ses méthodes. Bien sûr, les tests par eux-mêmes sont à écrire, c'est ce que nous allons présenter dans cet article.

Des tests

Faire évoluer des applications est une tâche délicate **[Demeyer 2002]**. Il est en effet difficile de garantir qu'un simple petit changement n'a pas de répercutions inattendues et souvent très coûteuses. L'existence de tests permet de détecter des erreurs en vérifiant les comportements attendus.

Les phases de tests sont souvent les mal aimées des phases de développement. Elles sont souvent comprimées ou considérées à tort comme une perte de temps. Dans le modèle de développement en cascade, elles arrivent souvent trop tard et elles sont donc dans la réalité hors budget au détriment de la qualité et d'une aide précieuse pour l'évolution future, mais inéluctable des logiciels (Lehman et Belady ont découvert une loi stipulant que tout logiciel de taille conséquente et utile à des utilisateurs devait évoluer pour satisfaire de nouvelles fonctionnalités sinon il devient obsolète).

Les tests ont plusieurs rôles :

- Premièrement, ils servent de documentation toujours synchronisée avec le code auquel ils se rattachent ; un test peut représenter un cas d'usage de la classe. Ils sont donc une sorte de spécification exécutable qu'un nouvel arrivant peut lire pour se familiariser avec certaines fonctionnalités.
- Deuxièmement, ils représentent de manière *tangible* la confiance que le système peut évoluer en minimisant les risques. Les tests vont permettre de trouver extrêmement rapidement des bugs. C'est une évidence, mais c'est absolument simple et vrai.
- Finalement, le fait d'écrire les tests en même temps que les classes auxquelles ils se rattachent force le développeur à être son propre client et donc assure que l'interface sera plus lisible et claire.

Avec l'arrivée des *refactorings* dans les environnements de programmation comme Eclipse, transformer du code à l'aide de refactorings permet d'effectuer des transformations sécurisées (un refactoring est une transformation de code avec préservation de comportement – Notez que Smalltalk a été le premier langage à avoir des refactorings **[RB][RBJ]**). Le problème est que l'on ne peut pas tout faire exclusivement avec des refactorings et que le code doit aussi être manuellement édité, ce qui peut introduire des bugs. Une des solutions est d'avoir des tests.

eXtreme Programming **[Beck]** et les méthodologies agiles proposent de nouvelles méthodologies de développement pour des petits groupes de développeurs (moins de 10). Ces méthodologies remettent le test au centre du processus développement en allant même jusqu'à demander que les tests soient écrits avant le code et que les tests servent à décrire les cas d'usage **[Beck]**.

La culture du test a toujours fait partie de la philosophie de développement Smalltalk dans laquelle on écrit une méthode, la compile et la teste en écrivant un petit script soit dans un *workspace* soit dans les commentaires soit comme méthode de classe dans la catégorie exemples. Cependant, cette façon de faire ne permettait pas de répéter les tests de manière automatique et de les répertorier. De plus, le résultat du test est souvent implicite laissant au lecteur le soin d'interpréter si le test est passé ou non. C'est devant cette constatation que K. Beck a introduit **Sunit** rendant alors possible les tests automatiques comme nous le verrons plus loin.

Dans le cadre du développement itératif d'applications, transformer du code pour l'adapter à de nouveaux besoins est une réelle nécessité. Aussi utiliser d'une part des outils comme le *Refactoring Browser* couplé à l'utilisation de tests permet de minimiser les risques d'introduire des erreurs. Il est clair que l'on ne peut pas tester tous les aspects d'une application. Couvrir toute une application est simplement impossible. Il est fort possible que des tests soient manquants et que certains bugs apparaissent après changement sans avoir été détectés par des tests. Ceci n'est pas un problème à partir du moment où, si l'on découvre un bug non couvert, on ajoute au jeu de tests un nouveau test couvrant précisément ce bug.

Écrire de bons tests est une technique qui s'apprend. Un test unitaire doit avoir plusieurs propriétés afin d'apporter le maximum de bénéfice :

- Répétable. Il doit être répétable à souhait.
- Sans intervention humaine. Les tests doivent pouvoir être répétés la nuit par exemple et sans nécessiter d'interaction avec un utilisateur.
- Raconter une histoire. Un test doit couvrir un aspect du système. La bonne granularité est de décrire une fonctionnalité d'une classe à la fois.
- Avoir une fréquence de changement plus lente que celui de l'application. Si, à chaque changement d'une application, on doit changer trop de tests, leur conception est maladroite. Un test doit être exprimé en utilisant autant que possible l'interface de la classe. Cet aspect est difficile à mettre en œuvre.

Le nombre de tests d'un jeu de tests doit être proportionnel aux fonctionnalités couvertes. Ainsi, il est important que si dix tests sont invalidés, le bug découvert n'est pas le même.

eXtreme Programming propose d'écrire les tests avant même de programmer. Ceci peut paraître contre nature, mais pour avoir essayé, les résultats suivants sont à remarquer : tester en premier aide (1) à conceptualiser les responsabilités de la classe, (2) à exprimer l'interface de la classe, car, lors de l'écriture du test, on est son premier client.

SUnit par l'exemple

SUnit est aussi appelé le *framework* mère des frameworks Xunit, car il a été le premier framework à partir duquel les autres frameworks comme JUnit ont été développés. L'intérêt pour SUnit n'est pas limité à Squeak ni même à Smalltalk. En effet, de très nombreux programmeurs ont reconnu la valeur de SUnit et l'ont porté dans leur langage de prédilection. Ainsi, on trouve SUnit pour pratiquement tous les langages allant de Oracle à Perl en passant par Python, Java ou C++ et bien d'autres [SUnit].

Un test est composé d'une création de contexte, comme la création d'un objet et la mise en place d'un certain état, d'un stimuli (envoi d'un ou plusieurs messages) et d'une ou plusieurs assertions vérifiant un certain état. Un test peut alors être dans trois états : il est passé c'est-à-dire que les assertions sont vraies (on dit que le test est vert), il contient une erreur (*error*) ou une *failure*. La différence entre une erreur et une failure est qu'une erreur représente une erreur non anticipée comme une exception non capturée. Par contre une failure est une erreur due à une assertion qui est fausse, il s'agit là d'une erreur anticipée comme une expression retournant vrai au lieu de faux.

Voici un exemple pour tester la classe `Set`.

On crée une sous-classe de `TestCase` nommée `ExampleSetTest` qui a deux variables d'instances `withFive` et `empty` qui seront deux ensembles : `withFive` contenant déjà le nombre 5 et `empty` l'ensemble vide.

```
TestCase subclass: #ExampleSetTest
         instanceVariableNames: 'withFive empty'
         classVariableNames: ''
         poolDictionaries: ''
         category: 'Sunit-Tests'
```

La méthode `setUp` qui est invoquée avant l'invocation des tests assure que les variables ont bien les valeurs requises : un ensemble avec le nombre 5 et un ensemble vide.

```
ExampleSetTest>>setUp
   empty := Set new.
   withFive := Set with: 5.
```

Ensuite, toute méthode commençant par `test` définit un test. Par exemple, la méthode `testIncludes` teste l'inclusion d'un élément. La méthode `assert:`, définie par le framework, vérifie que son argument doit être un booléen vrai alors que la méthode `deny:` le booléen faux.

```
ExampleSetTest>>testIncludes
   self assert: (withFive includes: 5).
   self deny: (withFive includes: 6).
   self assert: (withFive size = 1).
   self deny: (empty includes: 5).
   self assert: (empty size isZero).
```

Pour exécuter un test vous pouvez utiliser le TestRunner que vous obtenez en faisant *open... TestRunner*, puis en sélectionnant la classe `ExampleSetTest` et pressant le bouton *run*. Vous pouvez aussi exécuter l'expression `ExampleSetTest run: #testIncludes` et afficher le résultat. Notez qu'il existe déjà une classe de tests dans l'image pour la classe `Test` que vous pouvez parcourir et nous vous suggérons d'exécuter les tests.

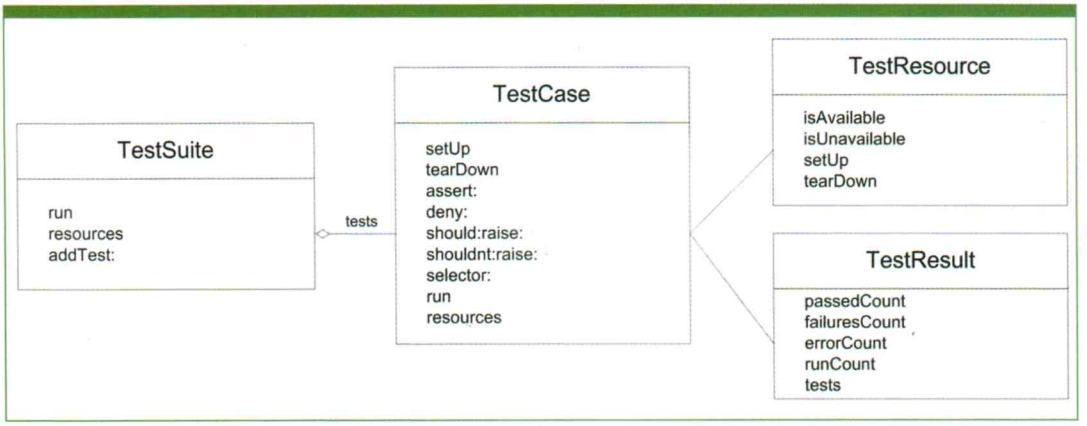
```
ExampleSetTest>>testAdd
     withFive add: 5; add: 6.
     self assert: (withFive size = 2).
     self assert: (withFive includes: 6).
     self assert: (withFive includes: 5).
```

La méthode `should: aBlock raise: exception` est utilisée pour spécifier quelle exception doit être levée. Son pendant est la méthode `shouldnot:raise:` qui vérifie que le bloc passé en argument ne lève pas d'exception du type mentionné. Cette méthode est utile pour tester que le code lève correctement des exceptions. Dans le test `#testRemove`, la première ligne spécifie qu'une exception de type `Error` doit être levée lorsque l'on essaye d'enlever un élément non contenu dans un ensemble. Lorsque le test est exécuté, si une exception de la bonne classe est levée, le test sera validé. Notez que l'argument de `should:` est un bloc et non une expression simple, donc n'oubliez pas les `[ ]` lorsque vous écrivez ce genre de tests.

```
ExampleSetTest>>testRemove
   self should: [empty remove: 5] raise: Error.
   withFive remove: 5.
   self should: [withFive remove: 6] raise: Error
```

Un framework pour les tests unitaires

SUnit est constitué de quatre classes principales : `TestCase`, `TestSuite`, `TestResult` et `TestResource`.



La classe `TestCase` représente un test ou un jeu de tests dont le contexte d'exécution est partagé entre ses tests. Cette classe permet de définir le contexte dans lequel les tests vont être exécutés par la spécialisation de la méthode `setUp`. Cette méthode est exécutée avant l'exécution de chaque test. La méthode `tearDown` est la méthode symétrique de la méthode `setUp`, elle est invoquée après toute exécution d'un test, elle permet de libérer certaines ressources.

La classe `TestSuite` représente une collection de tests instance de la classe `TestCase`. Une instance de `TestSuite` est composée d'instance de `TestCase` et de `TestSuite`. Elle forme avec `TestCase` un composite pattern.

La classe `TestResult` représente les résultats de l'exécution d'un jeu de test, c'est-à-dire le nombre de tests passés, invalidés et d'erreurs rencontrées lors de l'exécution des tests.

La classe `TestResource` représente une ressource qui est utilisée par un test. Une ressource décrit un ensemble de valeurs, comme une base de données qui doit être disponible pour qu'un test puisse être exécuté. L'idée d'une ressource est qu'elle n'est pas mise en place avant l'exécution de chaque test, mais avant un ensemble de tests, une instance de `TestSuite`. Une ressource peut être définie pour un jeu de test, en redéfinissant la méthode de classe `resources` dans la sous-classe de la classe `TestCase`. Par défaut, une instance de `TestSuite` considère que ses ressources sont toutes les ressources définies dans les jeux de tests qui la composent.

Voici schématiquement comment on peut déclarer une ressource `MyResource` et l'associer avec un test `MyTestCase`. Comme pour un `TestCase`, on définit par la méthode `setUp` les actions qui vont être effectuées pour mettre en place la ressource. Sur la classe `MyTestCase`, on spécialise la méthode de classe `resource` pour qu'elle rende une collection contenant le nom des classes des ressources nécessaires.

```
TestResource subclass: #MyTestResource
        instanceVariableNames: ''
        ....
```

Un exemple complet

Pour illustrer nos propos, nous prenons le prétexte de la réalisation d'une classe très simple pour manipuler les fonctions affines – vous savez, ces fonctions que vous avez vues au collège en 3e et dont la représentation graphique est une droite. Ces fonctions sont de la forme $f: x \dashrightarrow a.x+b$ où a et b sont deux nombres réels. Une fonction affine est déterminée par a et b. Nous définissons donc une classe `Affine` avec deux variables d'instance `a` et `b`.

Dans un premier temps, nous allons **uniquement** déclarer notre classe `Affine` et ses différentes méthodes – c'est-à-dire uniquement le nom des méthodes, sans définition. Cela nous permettra de bien définir le comportement que nous souhaitons avoir avec de tels objets. Dans un deuxième temps, nous écrirons une série de tests unitaires sur cette classe, validant le comportement attendu. Dans un troisième temps, nous définirons les méthodes de la classe `Affine`. Enfin, nous lancerons les tests unitaires. En cas d'erreurs, nous vérifions les tests et ajustons les méthodes testées, puis nous relançons les tests, et ainsi de suite.

Déclaration d'Affine

Depuis le navigateur de classe, nous déclarons les éléments de base de la classe. Notez bien les différentes déclarations et/ou définitions de méthode de *classe* **et** d'*instance* :

```
Object subclass: #Affine
        instanceVariableNames: 'a b'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'LinuxMag94'

Affine class>>a: nombre1 b: nombre2
        "création d'instance"
        ^ self new a: nombre1;
                   b: nombre2
Affine>>a
        ^ a
Affine>>a: nombre
        a := nombre
Affine>>b
        ^ b
Affine >> b: nombre
        b := nombre
```

Ensuite, nous souhaitons calculer *f(x)* : ici, nous déclarons seulement la méthode :

```
Affine>>x: nombre
        "calcule l'image par la fonction affine"
```

Aussi, nous souhaitons obtenir des fonctions affines telles que $h = f + g$, ou bien $i = f.k$ avec k un nombre, et aussi le point d'intersection des droites représentant graphiquement deux fonctions affines.

Nous déclarons donc les méthodes :

```
Affine>>+ g
        "retourne la fonction affine somme"
Affine>>* k
        "retourne la fonction affine multipliée par k"
intersection: g
        "retourne le couple (x,y) tel que y=f(x)=g(x) "
```

Définitions de tests

Nos tests sont définis à partir d'une sous-classe de `TestCase`. Nous y adjoignons les quelques variables d'instances nécessaires pour les tests :

```
TestCase subclass: #TestAffine
        instanceVariableNames: 'f g k h'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'LinuxMag94'
```

L'amorçage des tests se fait toujours par une méthode `setUp` exécutée avant chaque méthode de test :

```
AffineTest>>setUp
        k := 3.
        f := Affine a: 2 b: 3.
        g := Affine a: -3 b: 2.
        h := Affine a: 2 b: 10
```

Ainsi *f(x)=2.x+3 ; g(x)=-3.x+2 et h(x)=2.x+10.*

Bien sûr, ces définitions ne sont pas faites entièrement au hasard. Elles nous permettront de tester complètement le comportement de notre classe. Pour chacun de ceux-ci, nous définissons une méthode bien précise.

```
AffineTest>>testImage
        self assert: (f x: 0) = 3.
        self assert: (f x: 5) = 13.
        self assert: (g x: 0) = 2.
        self assert: (g x: 5) = -13
```

La méthode `assert:` a comme argument un booléen qui doit être vrai pour que le test soit validé. Ici nous testons que *f(0)=3, f(5)=13, g(0)=2 et g(5)=-13.*

```
AffineTest>>testAddition
        | j |
        j := f + g.
        self assert: j a = -1.
        self assert: j b = 5
```

Dans cette méthode de test, rien de bien nouveau au niveau des outils de test. Nous vérifions que la fonction affine somme de *f* et *g* est bien celle attendue.

```
AffineTest>>testProduit
        self assert: (f * k) a = 6.
        self assert: (f * k) b = 9.
        self deny: (f * k x: 5)
                        = (f x: k * 5)
```

Ici nous testons le produit. En 3e test, nous introduisons la méthode `deny:` pour vérifier l'invalidité d'un résultat, à savoir que *(f.k)(5)* ne doit pas être égale à *k.f(5)* !

```
AffineTest>>testIntersection
        self assert: (f intersection: g)
                        = (-1 / 5 @ (13 / 5)).
        self
                should: [f intersection: h]
                raise: ZeroDivide
```

Dans cette méthode, outre le test sur le résultat d'intersection, apparaît `should:raise:` qui vérifie que dans la situation donnée – $f = h$ n'a pas de solution – une exception est bien générée.

Définitions des méthodes de la classe Affine

Il nous faut maintenant terminer la définition de notre classe :

```
Affine>>x: nombre
        ^ nombre * a + b
Affine>>* k
        ^ Affine a: a * k b: b * k
Affine>>+ g
        ^ Affine a: (a + g a) b: (b + g b)
Affine>>intersection: g
        | x |
        x := (b - g b) / (g a - a).
        ^ x @ (g x: x)
```

Nous constatons dans le cas de la méthode `intersection:`, qu'une erreur de division par zéro apparaît lorsque les coefficients `a` sont égaux. C'est ce que nous détectons dans notre test.

On teste !

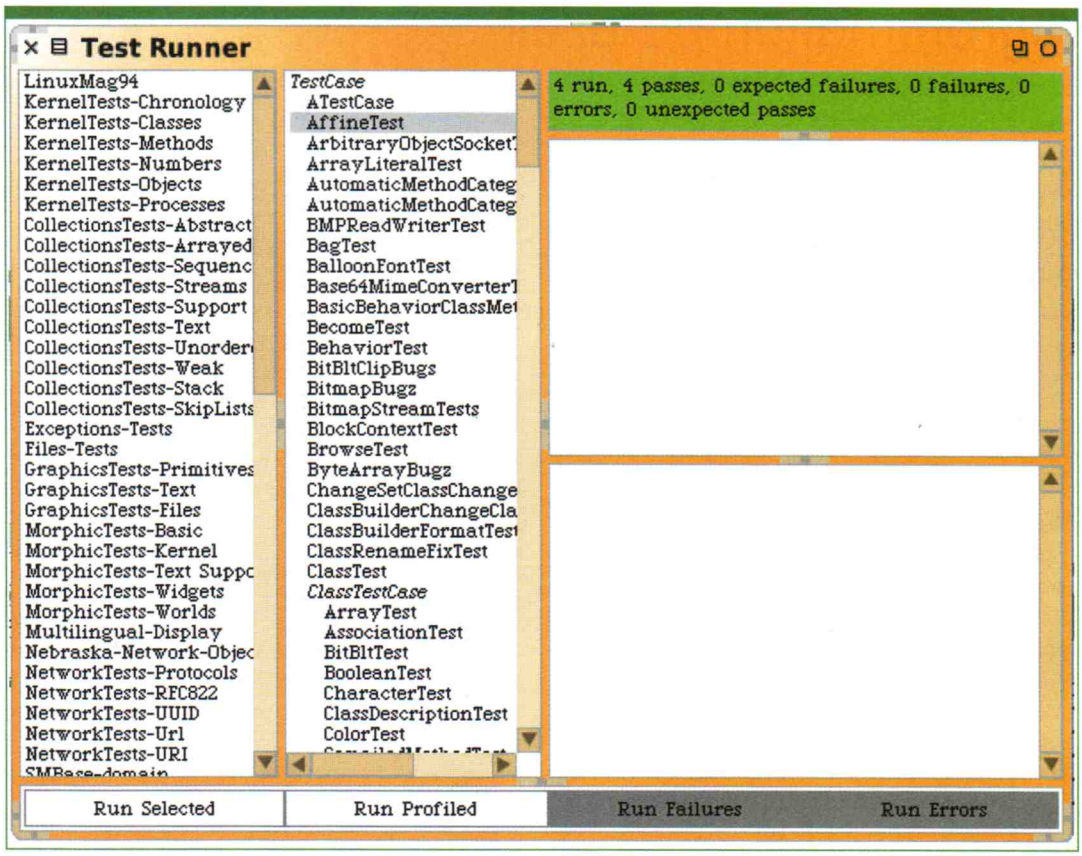
Pour lancer les tests, deux procédures. Soit nous utilisons l'interface graphique de test, soit nous lançons les tests en exécutant des expressions depuis un espace de travail (*workspace*).

Dans un espace de travail, par un Print-it [Alt]+[P] sur l'expression `TestAffine run: #testImage`, nous obtenons les résultats du test :

```
1 run, 1 passes, 0 expected failures, 0 failures,
0 errors, 0 unexpected passes
```

Le test est ici réussi. Nous allons faire de même pour chacune des méthodes de test.

L'autre option est de lancer l'interface graphique depuis le menu *world>open...>Test runner* et de sélectionner les tests à lancer (voir capture d'écran, page suivante).



Conclusion

Vous constatez que nous avons d'abord déterminé le comportement souhaité pour notre classe. Avec l'écriture de tests, nous sommes le premier utilisateur de notre classe, c'est un double avantage : utiliser l'interface et valider son comportement. Ensuite seulement, nous avons écrit le cœur de la classe. Nous aurions pu écrire les tests à la fin, mais nous voulions montrer que l'écriture de test a aussi un rôle structurant sur l'élaboration d'une application. En fait, le design d'un programme développé avec des tests est souvent meilleur que sans test. Si, par la suite, nous décidons de modifier l'implémentation de notre classe, d'étendre son domaine, etc., les tests nous permettront toujours de vérifier que son comportement extérieur reste inchangé. Nous réduisons ainsi les risques de casser une application ou, au moins, d'anticiper sur ce qui posera problème.

Dans le cadre réduit de cet article, nous avons choisi un exemple très simple, mais, après tout, notre fonction `affine` n'est qu'une fonction polynomiale de degré 1. En généralisant à des fonctions polynomiales d'ordre n, l'utilisation des tests unitaires est encore plus intéressante. De très nombreux projets *open source* ont compris l'intérêt des tests unitaires et nous considérons que la présence de tests unitaires est un gage de qualité.

S. Ducasse & H. Fernandes,

Stephane.Ducasse@gmail.com
hilaire@ofset.org

LIENS

- Le site officiel : http://www.squeak.org/
- Le wiki de la communauté française : http://community.ofset.org/wiki/Squeak
- Le groupe des utilisateurs européens de Smalltalk (*European Smalltalk User Group*). L'adhésion est gratuite : http://www.esug.org/
- Des livres gratuits en ligne sur Smalltalk et Squeak : http://stephane.ducasse.free.fr/FreeBooks.html
- Un livre sur Squeak en français : BRIFFAULT (X.), DUCASSE (S.), *Squeak*, Eyrolles, 2002, http://stephane.ducasse.free.fr/Books.html
- **[Beck]** BECK (Kent), *Extreme Programming Explained : Embrace* Change, Addison-Wesley, 1999.
- **[FBBOR]** FOWLER (Martin), BECK (Kent), BRANT (John), OPDYKE (William) et ROBERTS (Don), *Refactoring : Improving the Design of Existing Code*, Addison-Wesley, 1999.
- **[RB]** http://www.refactory.com/RefactoringBrowser/, http://st-www.cs.uiuc.edu/users/brant/Refactory/
- **[RBJ]** ROBERTS (D.), BRANT (J.) et JOHNSON (R.), « *A Refactoring Tool for Smalltalk* », TAPOS, vol. 3, N° 4, 1997, pp. 253-263, http://st-www.cs.uiuc.edu/~droberts/tapos/TAPOS.htm
- **[SUnit]** http://www.xprogramming.com/software.htm
- DEMEYER (Serge), DUCASSE (Stéphane) et NIERSTRASZ (Oscar), *Object-Oriented Reengineering Patterns*, Morgan Kaufmann, 2002.

Yves Bailly

Qt4 : programmation parallèle

Voilà un titre qui évoque celui d'un autre article, consacré à un autre langage de programmation. Le langage C++ ne fournissant pas, par lui-même, d'outils pour l'exécution de tâches en parallèle (ou threads), voyons comment la bibliothèque Qt4 peut nous simplifier grandement la vie dans ce domaine.

La parallélisation va très bientôt devenir le seul moyen d'augmenter les performances d'un programme. Il semble bien en effet que pour les prochaines années, la course aux gigahertz soit terminée : il existe des limites physiques infranchissables sans une profonde révolution technologique (qui est déjà en cours, mais c'est un autre sujet et elle n'arrivera pas dans nos assiettes avant quelques années). On ne peut donc plus compter sur l'augmentation de la fréquence du processeur pour espérer de meilleures performances. Nous assistons à une nouvelle course, celle au nombre de cœurs dans une même puce.

Programme sans interface

Depuis la version 4, la bibliothèque Qt est décomposée en plusieurs modules afin de n'inclure que ce qui est nécessaire dans le programme. Il est donc désormais possible de créer un programme complet sans devoir « traîner » toute la partie graphique.

Considérons un exemple déjà utilisé : le calcul d'une image fractale. C'est un exemple suffisamment classique pour qu'il soit inutile de nous étendre sur le sujet (sinon, consultez **[1]**). Le cœur du programme est constitué :

- d'une classe virtuelle `Fractal` effectuant les calculs pour un point donné (de type `std::complex<double>`) par une méthode virtuelle `Compute()` ;
- d'une classe `Mandelbrot`, dérivant de `Fractal`, implémentant l'algorithme d'une fractale de Mandelbrot ;
- d'une classe `Fractal_Image` contenant les résultats des calculs pour un ensemble rectangulaire de pixels, en invoquant la méthode `Compute()` de Fractal au sein de la méthode `Compute_Sub_Image()`, laquelle attend en paramètre une paire de coordonnées représentant un rectangle de pixels.

La version simple du programme (sans *thread*) se réduit en gros à ceci dans la fonction `main()` (`w` et `h` étant la largeur et la hauteur de l'image, obtenues par la ligne de commande) :

```
Mandelbrot mandel;
Fractal_Image img(&mandel, w, h);
img.Set_Domain(complex(-1.18764, -0.30278), 0.00283869);
QTime chrono;
chrono.start();
img.Compute_Sub_Image(0, 0, img.width()-1, img.height()-1);
int e = chrono.elapsed();
std::cout << double(e)/1000.0 << "\n";
```

Histoire de fixer une référence, cette version s'exécute en environ 37 secondes sur la machine de votre serviteur :

```
$ ./fract_1 -w 2048 -h 1536
37.572
```

La parallélisation va consister à calculer simultanément plusieurs sous-parties de l'image. Pour cela, on dérive la classe `QThread`, offerte par Qt pour représenter un thread :

```
class ThreadFract: public QThread
{
   Q_OBJECT
   public:
      ThreadFract(QObject* parent = 0);
      void setSubImage(Fractal_Image* img,
                       unsigned min_x,
                       unsigned min_y,
                       unsigned max_x,
                       unsigned max_y);
   protected:
      virtual void run();
   private:
      Fractal_Image* _img;
      unsigned       _min_x;
      unsigned       _min_y;
      unsigned       _max_x;
      unsigned       _max_y;
}; // class ThreadFract
```

Deux remarques s'imposent :

- D'abord, la présente de la macro `Q_OBJECT` est indispensable : en effet `QThread` dérive de `QObject`. Cela implique qu'une classe dérivant de `QThread` peut bénéficier des mécanismes de signaux/*slots* et d'événements de Qt (nous verrons plus loin comment).
- Ensuite, le code du thread (c'est-à-dire le code effectivement utilisé en parallèle) doit être placé dans la méthode virtuelle `run()`, laquelle sera invoquée au démarrage du thread. Lorsque `run()` se termine, le thread se termine également.

Dans notre cas, la méthode `run()` se résume à presque rien :

```
void ThreadFract::run()
{
   if ( _img == 0 )
      return;
   _img->Compute_Sub_Image(_min_x, _min_y,
                           _max_x, _max_y);
}
```

Les différents threads pour effectuer les calculs sont créés dans le programme principal, qui doit naturellement être un peu adapté :

```
#include <QCoreApplication>
#include <QTime>
#include "fractal_images.h"
#include "mandelbrots.h"
#include "thread_fracts.h"
int main(int argc, char* argv[])
{
   QCoreApplication app(argc, argv);
   unsigned w = 1024;
   unsigned h = 768;
   unsigned nb_threads = 2;
   // paramètres
   /* ... */
```

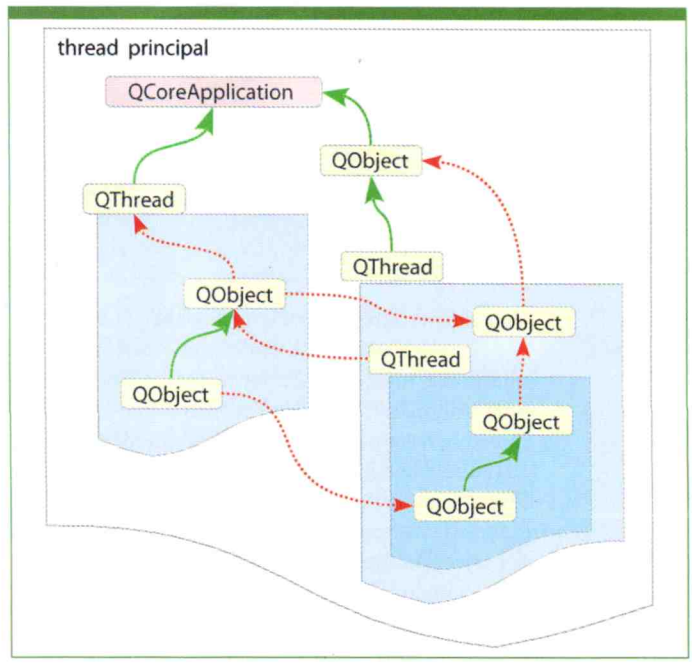
Pour commencer, avant de créer toute instance d'une classe dérivée de `QThread`, il est nécessaire de créer une instance de `QCoreApplication` (ou `QApplication` dans le cas d'un programme disposant d'une interface graphique).

Après lecture des paramètres de la ligne de commande, on crée autant de threads que demandés, en répartissant la « charge de travail » :

```
   // préparation
   Mandelbrot mandel;
   Fractal_Image img(&mandel, w, h);
   img.Set_Domain(complex(-1.18764, -0.30278),
0.00283869);
   // création des threads
   std::vector<ThreadFract*> threads(nb_threads,
(ThreadFract*)0);
   unsigned step = img.width() / nb_threads;
   unsigned rem = img.width() % nb_threads;
   unsigned min_x = 0;
   unsigned max_x = step+rem-1;
   for(unsigned ind_thread = 0;
       ind_thread < nb_threads;
       ++ind_thread)
   {
      threads[ind_thread] = new ThreadFract(&app);
      threads[ind_thread]->setSubImage(&img,
                                       min_x, 0,
                                       max_x, img.
height());
      min_x += step;
      max_x += step;
   }
```

Remarquez que l'on donne, à chaque instance de `ThreadFract`, l'instance de `QCoreApplication` comme objet parent : ainsi nos instances de `ThreadFract` seront automatiquement détruites à la fin du programme. Il faut toutefois prendre garde à une contrainte : le parent d'un `QObject` ne peut être qu'un autre `QObject` *créé dans le même thread*. Cela implique que si nous avions créé une instance de `QObject` (ou d'une classe dérivée) au sein de la méthode `run()` de `ThreadFract`, il nous aurait été impossible de donner l'instance de `ThreadFract` correspondante comme parent à l'objet.

Il y a ainsi une notion de « domaine d'existence » des instances de `QObject`. On dit qu'un objet « vit dans » un thread donné, les relations parents-enfants étant limitées à ce thread. C'est ce qui est illustré par le schéma suivant, qui montre diverses instances de `QObject` (ou de classes dérivées) avec diverses instances de (classes dérivées de) `QThread` :



Les flèches représentent les relations « *...est fils de...* ». En vert, des relations légitimes, en rouge, quelques relations interdites, car elles franchissent la membrane invisible qui entoure chaque thread. Avec cette petite subtilité qui fait qu'une instance de `QThread` existe toujours dans un thread (donc, elle peut avoir comme parent un objet existant dans celui-ci), mais que comme elle crée un nouveau thread, les objets existant dans ce dernier ne peuvent avoir cette instance comme parent.

Enfin, la création d'un thread ne provoque pas son démarrage, c'est-à-dire l'exécution du code contenu dans la méthode `run()`. Aussi, démarrons-nous nos threads :

```
// lancement des threads
QTime chrono;
chrono.start();
for(unsigned ind_thread = 0;
    ind_thread < nb_threads;
    ++ind_thread)
   threads[ind_thread]->start();
```

Le démarrage d'un thread est effectué par la méthode `start()` de `QThread` (qui est en fait un slot). Cette méthode se termine (presque) immédiatement,

l'invocation n'est pas bloquante pour l'appelant. Une simple boucle permet donc de lancer nos threads (presque) en même temps.

Seulement, si nous ne faisons rien de plus, le programme va bêtement continuer son exécution et se terminer... très certainement avant que nos threads n'aient terminé leur tâche. Inutile de préciser que la destruction d'une instance de `QThread` alors que le thread sous-jacent est en cours d'exécution peut avoir des conséquences fâcheuses. Nous devons donc attendre la terminaison des threads :

```
    // attente des threads
    for(unsigned ind_thread = 0;
        ind_thread < nb_threads;
        ++ind_thread)
       threads[ind_thread]->wait();
    int e = chrono.elapsed();
    std::cout << double(e)/1000.0 << "\n";
    return 0;
}
```

Cette attente est réalisée par la méthode `wait()` de `QThread`, qui prend un paramètre entier optionnel indiquant une durée maximale d'attente (en millisecondes). Sans paramètre, `wait()` ne retourne que lorsque le thread se termine (que cela soit « naturellement » ou à la suite d'une interruption brutale) et renvoie alors `true`. Sinon, `wait()` retourne `true` ou `false` selon que le thread s'est terminé ou que la durée d'attente a été dépassée. Pour reprendre la documentation de Qt, la fonctionnalité de `wait()` est comparable à celle de la fonction POSIX `pthread_join()`.

Notre programme est désormais terminé. Testons-le :

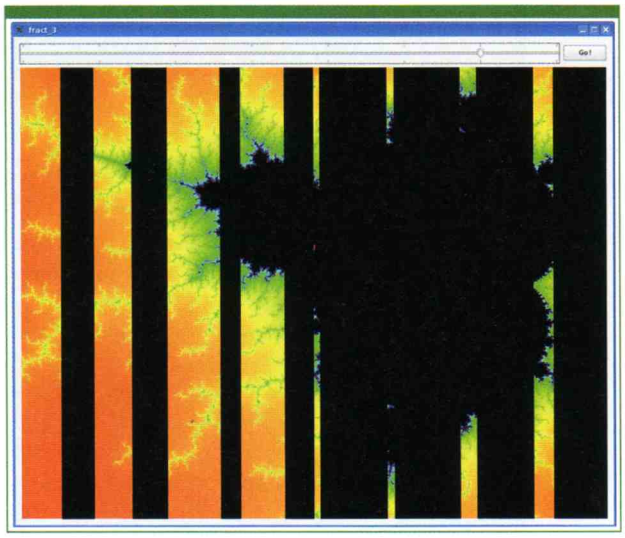
```
$ ./fract_2 -w 2048 -h 1536 -t 4
28.273
```

Le paramètre `-t` permet de spécifier un nombre de threads à utiliser, ici 4. Remarquez que le gain est appréciable, même pour un processeur unique et mono-cœur ne disposant que de l'*hyperthreading* (technologie consistant à simuler deux processeurs virtuels dans un seul processeur physique). Il convient toutefois d'avoir conscience des limites du parallélisme : le même calcul effectué avec 50 threads demande presque deux fois plus de temps que la version simple (sans threads). La parallélisation a un coût.

Enfin, notons que notre exemple est particulièrement trivial : chaque thread dispose en réalité de son propre espace de travail : il n'y a pas d'accès simultanés à une même zone de mémoire. Tout cela va changer dès que nous allons...

Ajouter une interface

Nous allons créer un programme tout simple, dont l'affichage ressemblera à ceci durant le calcul de l'image (exemple avec 8 threads) :



L'idée est donc d'afficher l'avancement du calcul, par exemple à chaque colonne de pixels terminée. Il va sans dire que le simple fait d'afficher ainsi est coûteux en temps.

Commençons par modifier un peu la classe représentant un thread de calcul :

```
/* ... */
class ThreadFract: public QThread
{
   Q_OBJECT
   public:
/* ... */
      void breakRun();
   protected:
      virtual void run();
   signals:
      void colReady(unsigned x);
   private:
      bool          _break;
      QMutex        _break_mutex;
/* ... */
```

Pour les besoins de l'interface, il peut être intéressant d'avoir la possibilité d'interrompre le calcul en cours avant qu'il ne se termine. Pour cela, on fournit la méthode `breakRun()`, dont voici le contenu à la simplicité trompeuse :

```
void ThreadFract::breakRun()
{
   QMutexLocker locker(&_break_mutex);
   _break = true;
}
```

L'ordre d'interruption est mémorisé dans la donnée `_break`, un simple booléen. Cette donnée va naturellement être testée au cours de l'exécution du thread, c'est-à-dire au sein de la méthode `run()`. Mais cet ordre viendra très certainement d'un autre thread : on risque donc de se trouver dans une situation où un thread tente de modifier une zone mémoire, tandis qu'un autre la consulte. Dans un exemple aussi simple, ce n'est pas forcément bien grave, mais d'une manière générale cela peut être dramatique. Nous devons donc *sérialiser*

l'accès à cette donnée, c'est-à-dire nous assurer que les différents accès se font bien l'un après l'autre (en « série »).

Il existe différentes solutions pour cela, l'une des plus simples étant l'utilisation de l'exclusion mutelle ou *mutex*. La classe `QMutex` permet d'utiliser cette technique : à un instant donné, un seul thread peut détenir le verrou d'une instance donnée de `QMutex`. La donnée `_break_mutex` pourrait être utilisée ainsi :

```
_break_mutex.lock();  // on verrouille
/* modifier ou lire _break */
_break_mutex.unlock();  // on libère
```

C'est là un schéma tout à fait classique, permettant d'isoler une portion de code critique. Toutefois, l'utilisation de `lock()` et `unlock()` peut s'avérer délicate dans le cas d'une structure complexe, bardée de tests et branchements divers. Il est alors très facile d'oublier de déverrouiller (par `unlock()`) la mutex, ce qui conduit en général à une situation de blocage. La classe `QMutexLocker` permet de simplifier un peu cela. Créez une instance en lui donnant un pointeur sur une instance de `QMutex` : celle-ci est alors verrouillée. Lorsque l'instance de `QMutexLocker` est détruite, par exemple en sortant d'une portée de définition (*scope*), l'instance de `QMutex` associée est automatiquement déverrouiller. C'est ce qui est fait dans `breakRun()`.

Voyons maintenant le code même du thread :

```
void ThreadFract::run()
{
   if ( _img == 0 )
      return;
   _break = false;
   for(unsigned x = _min_x;
       x <= _max_x;
       ++x)
   {
      _break_mutex.lock();
      if ( _break )
      {
         _break_mutex.unlock();
         break;
      }
      _break_mutex.unlock();
      _img->Compute_Sub_Image(x, _min_y,
                              x, _max_y);
      emit colReady(x);
   }
}
```

On calcule cette fois la zone demandée colonne par colonne, par une boucle `for`. Au sein de cette boucle, on vérifie à chaque itération si, par hasard, on n'aurait pas reçu l'ordre d'arrêt, en testant la donnée `_break`. Comme précédemment, l'accès à cette donnée est protégé par exclusion mutuelle. Volontairement, nous n'avons pas utilisé ici `QMutexLocker` : le code s'en trouve sensiblement alourdi, moins « élégant ».

Autre changement important : à chaque colonne terminée, on envoie un signal (au sens Qt du terme) nommé `colReady()` pour en informer l'extérieur. Cet extérieur sera représenté par une classe `FractalViewer`, chargée d'afficher le contenu d'une image :

```
class FractalViewer: public QWidget
{  Q_OBJECT
   public:
      FractalViewer(QWidget* parent = 0);
      ~FractalViewer();
      void setFractalImage(Fractal_Image* img);
      Fractal_Image* fractalImage() const;
      void clear();
   public slots:
      void colReady(unsigned c);
   protected:
      virtual void paintEvent(QPaintEvent*);
   private:
      Fractal_Image* _img;
      QPixmap        _pix;
}; // class FractalViewer
```

D'un point de vue strictement graphique, il n'y a rien d'extraordinaire : un simple *widget* affichant une *pixmap*. L'affichage effectif se fait au sein de la méthode virtuelle `paintEvent()`, la seule méthode autorisée à contenir des instructions dessinant effectivement à l'écran.

Le slot `colReady()` est celui chargé de réagir à la complétion d'une colonne de pixels dans l'image. Il sera donc connecté au signal éponyme des différents threads calculant l'image. Le voici :

```
void FractalViewer::colReady(unsigned col)
{
   QPainter p(&_pix);
   for(unsigned y = 0; y < _img->height(); ++y)
   {
      unsigned val = _img->Get_Pixel(col, y) % 360;
      if ( val == 0 )
         p.setPen(QColor::fromHsv(0, 0, 0));
      else
         p.setPen(QColor::fromHsv(val, 255, 255));
      p.drawPoint(col, y);
   }
   update();
}
```

Là encore, rien de spécial : on modifie la couleur de chaque pixel concerné dans l'image pixmap, puis on demande un rafraîchissement de l'affichage à la première occasion – en effet, `update()` ne provoque pas un dessin immédiat sur l'écran, l'affichage ne sera mis à jour qu'à la prochaine itération de la boucle d'événements interne.

Enfin, notre afficheur va lui-même être « emballé » dans une interface, représentée par une classe `Main`, qui sera la fenêtre principale de l'application :

```
class Main: public QWidget
{  Q_OBJECT
   public:
      Main();
      ~Main();
      void setFractalImage(Fractal_Image* img);
   protected slots:
      void setNbThreads(int nb);
      void compute();
```

```
    void threadFinished();
  protected:
    void setupThreads();
    void terminateThreads();
  private:
    QTime                   _chrono;
    unsigned                _nb_terminated;
    QVector<ThreadFract*>   _threads;
    FractalViewer*          _viewer;
}; // class Main
```

On y retrouve le chronomètre et le tableau de `ThreadFract*` utilisés dès le début. Pour commencer par la fin, voyons la méthode `terminateThreads()`, dont le rôle est d'interrompre les calculs éventuellement en cours :

```
void Main::terminateThreads()
{  for(int ind_thread = 0;
       ind_thread < _threads.count();
       ++ind_thread)
   {
      if ( _threads[ind_thread] != 0 )
      {
         if ( _threads[ind_thread]->isRunning() )
         {
            disconnect(_threads[ind_thread],
                       SIGNAL(finished()),
                       this,
                       SLOT(threadFinished()));
            _threads[ind_thread]->breakRun();
            _threads[ind_thread]->wait();
            connect(_threads[ind_thread],
                    SIGNAL(finished()),
                    this,
                    SLOT(threadFinished()));
         }
      }
   }
   QCoreApplication::processEvents();
}
```

On parcourt le tableau des threads, l'état d'exécution de chacun étant contrôlé par `isRunning()`, méthode de `QThread`. Elle retourne `true` si le thread est « dans » sa méthode `run()`, `false` sinon. Pour les threads en cours d'exécution, on pourrait les terminer « brutalement » en utilisant le slot `terminate()` de `QThread`. Toutefois, cette méthode expéditive n'est pas recommandée : elle ne laisse aucune chance au thread de libérer les ressources qu'il a éventuellement acquises et de se terminer dans un état cohérent. C'est pourquoi nous avons recours à `breakRun()`, ce qui ne nous dispense pas d'invoquer `wait()` afin d'obtenir une bonne synchronisation.

Vous aurez probablement remarqué la déconnexion, puis la reconnexion du signal `finished()`. Ce signal est émis par les instances de `QThread` (ou plutôt de classes dérivées), lorsque la méthode `run()` se termine normalement. Il est exploité ici pour détecter la fin du calcul de l'image (lorsque tous les threads se sont terminés normalement) et afficher le temps.

Notre interface permet de choisir le nombre de threads au moyen d'une glissière (*slider*), connectée sur le slot `setNbThreads()`. Comme il peut s'avérer extrêmement désagréable de créer ou détruire des threads alors qu'ils sont en cours d'exécution, la première chose à faire est naturellement de passer par `terminateThreads()`.

Enfin, la méthode `compute()` se contente de lancer les calculs, non sans les avoir arrêtés si besoin :

```
void Main::compute()
{  terminateThreads();
   _viewer->clear();
   _nb_terminated = 0;
   _chrono.restart();
   for(int ind_thread = 0;
       ind_thread < _threads.count();
       ++ind_thread)
      _threads[ind_thread]->start();
}
```

Et voilà, tout va bien. Les threads calculent l'image, à chaque colonne de pixels terminée, un signal est envoyé pour afficher l'état d'avancement.

Sauf que...

Les moins endormis parmi vous sont probablement taraudés par une question essentielle : que se passe-t-il si un thread termine une colonne et envoie son signal, alors que la pixmap est en cours de modification ? C'est une excellente question.

Signaux et threads

Si nous avions réalisé notre programme de la même manière en utilisant Qt 3, nous aurions obtenu au mieux une image étrange, plus certainement un crash du programme.

Il existe au moins deux bonnes raisons à cela. Tout d'abord, quelle que soit la version de Qt, les instructions liées à l'affichage graphique (comme un simple `update()`) **ne peuvent et ne doivent être exécutées qu'à partir du thread principal**, c'est-à-dire celui de la fonction `main()` et contenant l'instance de `QApplication`. Dérogez à cette règle, votre programme se terminera plus rapidement que vous ne le pensez.

Ensuite, dans les versions précédentes de Qt, l'émission d'un signal provoquait l'exécution immédiate des slots associés, comme si on avait simplement invoqué ces méthodes plutôt que d'envoyer le signal. Cela signifie, en particulier, que les slots exécutés suite à l'émission d'un signal depuis un thread donné, s'exécutent dans le cadre de ce thread. Dans notre cas, cela aurait pour conséquence d'avoir plusieurs accès simultanés (venant de plusieurs threads) à une même donnée, en l'occurrence la pixmap dans laquellle on dessine l'image. Essayez de dessiner dans une même pixmap par deux threads simultanés, même à deux endroits différents : le serveur X vous couvrira d'injures.

Ceci a été grandement modifié dans les versions 4 de Qt. Il existe désormais un paramètre supplémentaire à la méthode `connect()` de `QObject`, dont voici le prototype complet :

```
static
bool connect (const QObject* sender,
              const char* signal,
              const QObject* receiver,
              const char* method,
              Qt::ConnectionType type = Qt::AutoConnection);
```

Ce dernier et nouveau paramètre spécifie la manière dont la connexion doit être établie. Le mode « normal » est `Qt::DirectConnection`, qui correspond au mode de fonctionnement des versions antérieures de Qt : l'émission du signal provoque l'exécution immédiate du slot associé.

Le mode nouveau est `Qt::QueuedConnection`. Cette fois, le slot n'est pas exécuté à l'émission du signal, mais seulement lorsque la boucle d'événements interne sera en mesure de le faire. Un peu comme si on avait utilisé `QEvent`, `postEvent()` et consorts plutôt qu'un signal. Cela a pour conséquences, entre autres, que l'exécution du slot est désynchronisée de l'émission du signal, mais surtout que cette exécution a lieu dans le cadre et dans le thread de la boucle d'événements.

Enfin, le mode par défaut, `Qt::AutoConnection`, choisit l'un des deux précédents selon le contexte. Si l'émetteur et le récepteur « existent » dans le même thread, alors la connexion est directe. Sinon, elle passe par la boucle d'événements. Autrement dit, les signaux émis depuis un thread vers d'autres threads sont automatiquement sérialisés. C'est grâce à cela que notre programme fonctionne sans accrocs. Sans cela, il aurait été nécessaire de multiplier les recours aux mutex, compliquant sensiblement le code.

Dernier mot sur cet aspect, dans le cas d'une connexion désynchronisée, les types des éventuels paramètres du signal doivent impérativement être connus du mécanisme de méta-types de Qt – c'est-à-dire, essentiellement, qu'ils doivent pouvoir être stockés dans une instance de `QVariant`. Ce n'est curieusement pas le cas du type `unsigned`, utilisé justement dans notre exemple, et ce ne sera sans doute pas le cas de vos propres types. Dans ce cas, il est nécessaire de référencer le type à l'aide de la macro :

```
Q_DECLARE_METATYPE(unsigned)
```

Cette instruction doit être placée en-dehors de tout bloc d'espace de nommage (*namespace*) et même de tout bloc de code (en fait, cette macro contient la définition d'une classe `template`). En outre, il convient d'ajouter une instruction correspondante avant toute connexion, de préférence juste après la création de l'instance de `QCoreApplication` ou `QApplication` :

```
qRegisterMetaType<unsigned>("unsigned");
```

Remplacez naturellement `unsigned` par votre propre type.

Pour finir, le programme calcule et affiche l'image (en 1024x768) en 10,5 secondes avec un seul thread, et 8 secondes avec 8 threads. Le gain est bien moindre que dans l'exemple sans aucune interface : répétons-le, le simple affichage répété de l'image est très coûteux.

Conclusion

Voilà pour cette rapide introduction aux facilités offertes par Qt pour la programmation parallèle. Comme toujours, on pourrait en dire encore beaucoup, par exemple sur le fait que chaque thread peut disposer de sa propre boucle d'événements. Consultez la documentation de Qt pour plus de détails.

Pour rester dans les parallélismes (celui du code et celui des articles), nous verrons la prochaine fois comment nous pouvons effectuer des calculs distribués et, plus généralement, communiquer entre différents processus, en utilisant l'intégration du protocole DBus à Qt.

RÉFÉRENCES

- [1] Notions sur les fractales : http://fr.wikipedia.org/wiki/Fractale
- [2] Codes sources de l'article : http://www.kafka-fr.net/articles/qt4_09-sources.tar.bz2

Yves Bailly,

http://www.kafka-fr.net

Jérôme Delamarche

Écriture d'une fonction de reconnaissance phonétique pour MySQL

L'objectif de cet article est de vous montrer la simplicité avec laquelle il est possible d'étendre le SGBD MySQL pour lui ajouter de nouvelles fonctions directement appelables à partir de requêtes SQL.

Pour illustrer ces techniques, nous vous proposons de coder une fonction de « reconnaissance phonétique ».

Notre besoin :

Imaginez que vos disposiez d'une vaste base documentaire et que vous proposiez à vos utilisateurs un formulaire Web pour interroger la base. Il est facile de retrouver des informations dont le contenu est **exactement** celui indiqué par les utilisateurs : il suffit d'écrire des requêtes SQL qui ressemblent par exemple à :

```
SELECT * FROM table WHERE champ1 = 'valeur1' AND
champ2 = 'valeur2';
```

Les utilisateurs peuvent se tromper ou bien ne pas connaître exactement les valeurs exactes à rechercher. Dans ce cas, on peut utiliser une clause LIKE **[MYSQL1]**, par exemple :

```
SELECT * FROM table WHERE champ1 LIKE 'val%' AND
champ2 LIKE 'v__';
```

Dans cet exemple, le caractère '%' est un substitut pour « une chaîne quelconque », tandis que le caractère '_' remplace une occurrence de n'importe quel caractère.

Nous avons fait un pas vers les recherches approximatives, mais nous nous heurtons à plusieurs limitations :

- Comment retrouver la chaîne 'VALEUR' stockée dans la base alors que l'utilisateur a indiqué 'VOLEUR' dans le formulaire ?

  Faut-il essayer les modèles '_ALEUR', puis 'V_LEUR', puis 'VA_EUR', etc. ? D'autant plus que les recherches utilisant ces caractères spéciaux ne peuvent pas toujours utiliser d'index !
- Le deuxième problème vient donc du manque de performance de ce type de requêtes dès qu'un caractère spécial apparaît ailleurs qu'en fin de chaîne.

Il existe une solution qui semble séduisante et qui met en œuvre la fonction SQL appelée SOUNDEX() : cette fonction accepte une chaîne en paramètre et retourne une chaîne sur 4 caractères qui correspond à sa valeur phonétique.

Malheureusement, l'algorithme utilisé par SOUNDEX() utilise les règles de la prononciation anglaise (qui en doutait ?). Et cette fonction n'est pas « localisée » dans la langue de Molière.

Exemples :

- en anglais :

```
mysql> SELECT SOUNDEX("steer"),SOUNDEX("stir");
+-----------------+----------------+
| SOUNDEX("steer") | SOUNDEX("stir") |
+-----------------+----------------+
| S360             | S360           |
+-----------------+----------------+
```

- en français :

```
mysql> SELECT SOUNDEX("seau"),SOUNDEX("sot");
+----------------+---------------+
| SOUNDEX("seau") | SOUNDEX("sot") |
+----------------+---------------+
| S000           | S300          |
+----------------+---------------+
```

Nous décidons alors d'écrire notre propre fonction qui s'appellera PHONEX et qui sera adaptée à notre langue.

Conception de la fonction PHONEX

Notre fonction accepte 2 paramètres :

- une chaîne dont il faut calculer la valeur « phonétique » ;
- un drapeau capable d'influer sur l'algorithme implémenté afin de conférer un peu de souplesse à la fonction.

La fonction retourne une valeur réelle (REAL). Ce choix est la conséquence d'un besoin de performances : il est plus rapide de comparer des réels que des chaînes – sans compter le gain en termes de stockage !

Mais il ne tiendra qu'à vous de la modifier pour qu'elle retourne une chaîne !

Exemple d'utilisation :

Soit la table suivante :

```
CREATE TABLE Personne (
    nom CHAR(64) NOT NULL DEFAULT "",
    prenom CHAR(32) NOT NULL DEFAULT "",
    matricule CHAR(8) NOT NULL DEFAULT ""
)
```

Puisque nous allons rechercher des noms ou des prénoms grâce à une approximation phonétique, nous allons précalculer la valeur phonétique associée à chacun des champs sur lesquels vont porter les recherches.

Nous ajoutons donc 2 champs et la table devient :

```
CREATE TABLE Personne (
    nom CHAR(64) NOT NULL DEFAULT "",
    pex_nom REAL NOT NULL,
    prenom CHAR(32) NOT NULL DEFAULT "",
    pex_prenom REAL NOT NULL,
    matricule CHAR(8) NOT NULL DEFAULT ""
)
```

L'insertion d'une ligne dans la table s'écrit alors :

```
INSERT INTO Personne VALUES ("Doe",PHONEX("Doe"),"John
",PHONEX("John"),"matr0001")
```

La recherche d'une personne dans la table peut alors s'écrire :

```
SELECT * FROM Personne WHERE pex_prenom =
PHONEX("Jean")
```

Règles de reconnaissance phonétique

Probablement la partie la plus difficile, cette étape consiste à élaborer les règles qui permettent de réduire un mot en un réel en se basant sur sa prononciation phonétique.

Pour écrire ces règles, je me suis inspiré de l'article **[BROUARD]** ainsi que de mes souvenirs de grammaire française ! Et rien ne vaudra des tests concrets pour vérifier et affiner le fonctionnement de notre algorithme.

L'idée consiste, grâce à des passes multiples (une quinzaine) à simplifier le mot original pour le réduire à ses sons essentiels. Puis, on applique une formule de conversion de la chaîne finale en un réel.

L'implémentation est faite en langage C, sans utilisation de recherches de sous-chaînes dans des chaînes ni d'expressions régulières pour obtenir un maximum d'efficacité.

Les règles de calcul pour associer une « valeur » phonétique à une chaîne sont basées sur la similitude des sons. Il nous faut donc simplifier les chaînes pour en extraire une représentation abstraite d'un point de vue sonore. Si vous êtes un adepte du langage « SMS », vous savez ce que retranscription phonétique signifie !

Les règles adoptées sont les suivantes (et j'engage tout le monde à les améliorer !) :

Règle	Description	Exemples
R1	- substitution des caractères accentués par les voyelles « simples » - remplacement du « ç » par « ss » - remplacement du « y » par un « i » - tous les caractères sont convertis en minuscules	Pyrénées -> pirenees hameçon -> hamesson
R2	remplacement du son « ph » par un simple « f »	éléphant -> elefant
R3	suppression des « h » muets, c'est-à-dire ceux qui ne sont ni précédés d'un « c » ni d'un « s »	hache -> ache
R4	remplacement du « g » par « k » devant « an/am/ain/aim »	gamin -> kamin
R5	- remplacement de aina,eina,aima,eima par yna - remplacement de aine,eine,aime,eime par yne - remplacement de aini,eini,aimi,eimi par yni - remplacement de aino,eino,aimo,eimo par yno - remplacement de ainu,einu,aimu,eimu par ynu	chaine -> chyne rainure -> rynure
R6	- remplacement de eau par o - remplacement de oua par 2 - remplacement de ein par 4 - remplacement de ain par 4	seau -> so rein -> r4 bain -> b4 couac ->c2c
R7	- remplacement de ai par y - remplacement de ei par y - remplacement de ee par y - remplacement de er par yr - remplacement de ess par yss - remplacement de et par yt - remplacement de ez par yz	altesse -> altysse nez -> nyz
R8	suppression des lettres doublées	pelle -> pele benne -> bene
R9	- remplacement de « an » par « 1 » - remplacement de « am » par « 1 » - remplacement de « en » par « 1 » - remplacement de « em » par « 1 » - remplacement de « in » par « 4 » à condition que ces modèles ne soient ni suivis d'une voyelle ni d'un son « 1 » à « 4 »	patient -> pati1t incorrect -> 1correct

R10	remplacement du « z » par « s » s'il est en tête de mot ou précédé et suivi d'une voyelle ou d'un son « 1 » à « 4 »	zebu -> zebu azteque -> azteque bizarre -> bisarre
R11	- remplacement de « oe »par « e » - remplacement de « eu » par « e » - remplacement de « au » par «o» - remplacement de « oi » par « 2 » - remplacement de « ou » par « 3 »	heureux -> erex paul -> pol roue -> r3e
R12	- remplacement de « ch » par « 5 » - remplacement de « sch » par « 5 » - remplacement de « sh » par « 5 » - remplacement de « ss » par « s » - remplacement de « sc » par « s » si suivi d'un « i » ou d'un « e »	chat -> 5at scarlatine -> scarlatine scie -> sie
R13	remplacement du « c » par « s » s'il est suivi d'un « e » ou d'un « i »	car -> car innocence -> inosense
R14	- remplacement de « c » par « k » - remplacement de « q » par « k » - remplacement de « qu » par « k » - remplacement de « gu » par « k » - remplacement de « ga » par « ka » - remplacement de « go » par « ko »	car -> kar gateau -> kato
R15	- remplacement de « a » par « o » - remplacement de « d » et « p » par « t » - remplacement de « j » par « g » - remplacement de « b » et « v » par « f » - remplacement de « m » par « n »	depart -> tetord homme -> one
R16 (optionnelle)	remplacement des « y » (les sons « é ») par « e »	nez -> nyz -> nez
R17	- suppression des finales « t », « x », « s », « z » - (optionnelle) suppression du « e » final après une consonne (fait en deux fois pour éliminer les finales « ez » par exemple)	heureux -> ere tetard -> tetor
R18	suppression des lettres doubles (pour la 2ème fois)	arrivee -> arive
R19	retourne une chaîne d'une taille maximale de 16 caractères et ne contenant que les caractères autorisés. Au vu des traductions décrites, les caractères obtenus font partie de l'alphabet réduit : `12345efghiklnorstuwxyz` Les caractères qui n'appartiennent pas à cet ensemble sont supprimés.	

Altération de l'algorithme

La fonction `phonex2()` accepte, outre la chaîne à traiter, un 2ème paramètre qui modifie légèrement l'algorithme de traduction phonétique précédent. La valeur du paramètre est la somme de plusieurs drapeaux dont les significations sont :

- 0x1 : ne supprime pas les « e » finaux (par exemple, la chaîne « arrivée » devient « orife » et non pas « orif »).
- 0x2 : ne convertit pas les « é » en « e » (par exemple, la chaîne « arrivez » devient « orify » et non pas « orif »).

Libre à vous d'ajouter de nouvelles valeurs et de modifier le comportement de l'algorithme initial !

Codage de notre fonction

Nous codons alors la fonction `phonex2()` ainsi (extrait) :

```
/* liste des caractères autorisés dans la conversion en phonex: */
static unsigned char *finals = "12345efghiklnorstuwxyz";

static void phonex2(unsigned char *copy, unsigned int flag)
{
  unsigned char *p,achar;
  unsigned char *voyelles1 = "aeiouy";
  unsigned char *voyelles2 = "aeiouy1234";

  /* Cas trivial de la chaine vide: */
  if (!*copy) return;

  /* R1:
   * - remplace les accents et autres caracteres "bizarres".
   * - le Y/y devient i
   * - la cédille devient 'ss'
   * - les autres cars. sont mis en minuscules
   */
  p = copy;
```

```
  while (*p) {
    if ((*p >= 0xE0 && *p <= 0xE6) || (*p >= 0xC0 && *p <= 0xC6)) { *p++ =
a'; continue; }
    if ((*p >= 0xE8 && *p <= 0xEB) || (*p >= 0xC8 && *p <= 0xCB)) { *p++ =
e'; continue; }
    if ((*p >= 0xD2 && *p <= 0xD6) || (*p >= 0xF2 && *p <= 0xF6)) { *p++ =
o'; continue; }
    if ((*p >= 0xCC && *p <= 0xCF) || (*p >= 0xEC && *p <= 0xEF)) { *p++ =
i'; continue; }
    if ((*p >= 0xF9 && *p <= 0xFC) || (*p >= 0xD9 && *p <= 0xDC)) { *p++ =
u'; continue; }
    if (*p == 'Y' || *p == 'y') { *p++ = 'i'; continue; }
    if (*p == 0xC7 || *p == 0xE7) {
      *p++ = 's';
      memmove(p,p+1,strlen(p));
      *p++ = 's';
      continue;
    }
    *p++ = tolower(*p);
  }

  /* R2: Remplace 'ph' par 'f': */
  p = copy;
  while (*p) {
    if (*p == 'p' && *(p+1) == 'h') {
      *p = 'f';
      memmove(p+1,p+2,strlen(p+1));
    }
    *p++;
  }

  /* .... code manquant ...
   * code complet disponible sur le Net */

  /* FIN: ne retourne que les caractères autorisés avec un
  * maximum de 16 car. dans la chaine:
  */
  p = copy;
  while (*p && (p-copy) < PHONEX_MAXSZ) {
    if (!strchr(finals,*p)) {
      memmove(p,p+1,strlen(p));
      continue;
    }
    p++;
  }
  *p = '\0';
}
```

Le code complet de la fonction est disponible à l'adresse suivante : http://www.trickytools.com/php/phonex.php.

Conversion du résultat en nombre réel

Le mode de conversion est inspiré de **[BROUARD]** : comme l'alphabet final comporte 22 symboles, nous allons utiliser les puissances de 22. Pour chaque caractère du résultat final, nous obtenons le rang de ce caractère dans l'ensemble des symboles définis par la variable `finals` (voir ci-dessus, l'extrait du code de la fonction `phonex2()`. Puis nous multiplierons ce rang par la puissance « i ** 22 », où « i » est le rang du caractère dans la chaîne.

En fait, le résultat est un « grand nombre », non décimal, qui est obtenu par le code suivant :

```
/* chiffrement numerique en ignorant les car. non autorises: */
float phonex = 0; /* contient la valeur finale */
p = copy;
indice = 0;
while (*p) {
  unsigned char *rank = strchr(finals,*p);
  /* normalement tous les car. sont corrects */
  phonex += (rank-finals) * powf(indice,base);
  indice++; p++;
}
```

Intégration dans MySQL

Maintenant que nous disposons de notre fonction de calcul `phonex2()`, nous devons l'intégrer à MySQL.

Pour cela, nous allons ajouter trois fonctions à notre code :

- `phonex_init()` : cette fonction optionnelle est appelée par MySQL avant l'appel à `phonex()` décrite ci-dessous. Son prototype est :

```
my_bool phonex_init(UDF_INIT *initid, UDF_ARGS *args,
char *message);
```

Le paramètre `initid` est utilisé par les fonctions pour se passer une structure « opaque » pour MySQL.

Nos besoins sont simples. Nous n'avons besoin de passer ni valeurs ni paramètres.

Nous nous contenterons d'indiquer à MySQL que notre fonction `phonex()` ne retourne jamais `NULL`. Ce que nous écrirons ainsi :

```
initid->maybe_null = 0;
```

Le paramètre `args` est plus utile : il permet de vérifier le nombre et le type des paramètres spécifiés par l'utilisateur lors de l'appel à `phonex()`.

La structure `UDF_ARGS` permet de connaître le nombre de paramètres via le champ `argc_count`, puis chacun des paramètres est défini par les champs suivants :

- `arg_type[i]` : donne le type du paramètre. Les valeurs possibles sont `STRING_RESULT`, `INT_RESULT` ou `REAL_RESULT`.
- `arg->args[i]` : donne la valeur du paramètre. Ce pointeur doit ensuite être interprété suivant son type.
- ▶ Si le type est `INT_RESULT`, on obtient la valeur du paramètre ainsi :

```
long long int_val;
int_val = *((long long *)args->arg[i]);
```

- ▶ Si le type est `REAL_RESULT`, on obtient de manière similaire la valeur du paramètre `double real_val;`

```
real_val = *((double *)args->arg[i]);
```

- ▶ Enfin, si le type est `STRING_RESULT`, on ne doit pas considérer que l'argument pointe vers une chaîne terminée par un caractère nul (`'\0'`). La longueur de la chaîne est donnée par le champ `lengths[i]`.

Notre fonction `phonex_init()` doit donc s'assurer des points suivants :

- `phonex()` est appelée avec deux paramètres. Le premier doit être une chaîne, le deuxième doit être un entier. En cas d'erreur, la fonction d'initialisation a la possibilité de retourner un message d'erreur à MySQL, par l'intermédiaire du paramètre `message`. Le message d'erreur ne doit pas dépasser `MYSQL_ERRMSG_SIZE`, soit 80 caractères (la largeur de l'écran). Le code de `phonex_init()` est alors trivial et devient :

```
my_bool phonex_init(UDF_INIT *initid, UDF_ARGS *args, char
*message)
{
  /* ne retourne jamais NULL !*/
  initid->maybe_null = 0;

  /* on améliore en rendant le 2ème paramètre optionnel ! */
  if (args->arg_count < 1 || args->arg_count > 2) {
    strcpy(message,"PHONEX syntax is PHONEX(string[,flag])");
    return 1;
  }

  /* le 1er parm doit être une chaine ! */
  if (args->arg_type[0] != STRING_RESULT) {
    strcpy(message,"PHONEX needs a STRING Parameter");
    return 1;  }

  /* ...et le 2ème un entier ! */
  if (args->arg_count >= 2 &&args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"Second Parameter of PHONEX must be an INT
(the flag)");
    return 1;
  }

  /* pas d'erreur */
  return 0;
}
```

- `phonex_deinit()` : cette fonction optionnelle est appelée par MySQL après l'appel à `phonex()`. Son prototype est :

```
void phonex_deinit(UDF_INIT *initid);
```

Cette fonction peut désallouer la mémoire que `phonex_init()` aurait allouée. Dans notre cas, comme il n'y pas eu d'appel aux fonctions de la famille `malloc/calloc()`, la fonction `phonex_deinit()` a un corps vide.

- Nous devons maintenant réaliser l'interface entre MySQL et notre vraie fonction `phonex2()`. Pour cela, nous devons écrire une fonction `phonex()` qui doit se conformer à la spécification suivante :

```
double phonex(
  UDF_INIT *initid,
  UDF_ARGS *args,
  char *is_null,
  char *error)
```

En effet, suivant le type de la valeur retournée par notre fonction, MySQL impose un prototype particulier. Ici, notre fonction doit retourner une valeur réelle (d'où le type `double`), et elle reçoit en paramètre `initid` et `args` décrits précédemment, ainsi que deux paramètres supplémentaires :

- `is_null` : doit recevoir la valeur 1 si la valeur retournée par la fonction est le `NULL` de SQL.
- `error` : doit recevoir la valeur 1 si la fonction retourne une erreur.

Notre fonction `phonex()` doit donc appeler la fonction `phonex2()` codée dans le paragraphe précédent ; mais elle doit au préalable recopier le mot à examiner, passé sous forme de chaîne, dans un buffer temporaire, car rappelez-vous que MySQL n'assure pas que ce paramètre soit terminé par un `'\0'`.

Notez que le fait de dissocier `phonex()` de `phonex2()` permet d'invoquer notre fonction `phonex2()` sans passer par MySQL !

Le code de `phonex()` est alors :

```
double phonex(
  UDF_INIT *initid,
  UDF_ARGS *args,
  char *is_null,
  char *error)
{
  unsigned int flag = 0;
  float phonex = 0; /* résultat final */
  int indice = 0;
  int base = strlen(finals);

  /* copie la chaine: */
  unsigned int len = args->lengths[0];
  unsigned char *p,*copy = (unsigned char *)malloc(len+1);
  memcpy(copy,args->args[0],len);
  copy[len] = '\0';

  /* le 2ème parm est optionnel ! */
  if (args->arg_count > 1) {
    flag = *((long long *)args->args[1]);
  }

  /* appel de notre "fameuse fonction": */
  phonex2(copy,flag);

  /* pas de NULL, pas d'erreur */
  *is_null = *error = 0;

  /* chiffrement numerique: */
  p = copy;
  indice = 0;
  while (*p) {
  unsigned char *rank = strchr(finals,*p);
    /* normalement tous les car. sont corrects */
    phonex += (rank-finals) * powf(indice,base);
    indice++; p++;
  }

  free(copy);

  return phonex;
}
```

Finalisation de l'intégration :

Rien de plus simple :

- Compilons enfin notre programme avec la commande :

```
gcc -shared -I/usr/include/mysql -o phonex.so phonex.c
```

Vous constatez que vous avez besoin des en-têtes pour le développement MySQL. Suivant votre système, ces fichiers peuvent se trouver ailleurs que sous le répertoire `/usr/include/mysql`.

L'option `-shared` indique au compilateur de créer une bibliothèque dynamique. Le nom de cette bibliothèque peut être quelconque – on pourrait même y placer plusieurs fonctions UDF pour MySQL.

- Il faut rendre la bibliothèque accessible pour MySQL : le fichier `phonex.so` doit normalement être placé dans un des répertoires listés dans le fichier `/etc/ld.so.conf`. En pratique, nous avons constaté que MySQL, suivant ses versions, n'utilisait pas ces répertoires pour rechercher notre bibliothèque. Pour éviter les problèmes lors d'un premier test, je vous suggère de copier la bibliothèque sous `/lib` ou sous `/usr/lib` :

`cp phonex.so /lib` (il faut être *root* pour effectuer cette copie !)

- Pour que MySQL s'enrichisse de notre nouvelle fonction, lancez la commande `mysql` et déclarer la fonction ainsi :

```
mysql> CREATE FUNCTION phonex RETURNS REAL SONAME
"phonex.so";
Query OK, 0 rows affected (0.00 sec)
mysql>
```

Attention, respectez bien la casse des caractères pour le nom de la fonction et le nom de la bibliothèque !

Mais vous pourrez ensuite invoquer cette nouvelle fonction en utilisant son nom en minuscules ou majuscules.

- Nous pouvons alors tester notre fonction :

```
mysql> SELECT PHONEX("seau"),PHONEX("sot");
+---------------+--------------+
| PHONEX("seau") | PHONEX("sot") |
+---------------+--------------+
|            13 |           13 |
+---------------+--------------+
1 row in set (0.00 sec)
```

Victoire ! Ca marche !

- Si vous voulez modifier le code de la fonction, il faudra soit arrêter MySQL pour remplacer la bibliothèque `phonex.so`, soit supprimer l'extension par la commande suivante :

```
mysql> DROP FUNCTION phonex;
```

Conclusion

J'espère que cet aperçu sur les possibilités d'extension de MySQL vous a convaincu de l'intérêt et de la simplicité de cet énorme potentiel.

Toutefois, nous n'avons évoqué, ici, que les fonctions « simples », par opposition aux fonctions d'agrégation (comme `SUM()`, `MAX()`, `AVG()`...) qui travaillent sur des lignes multiples. L'écriture de telles fonctions utilise une interface légèrement différente dont vous trouverez les spécifications dans **[MYSQL2]**.

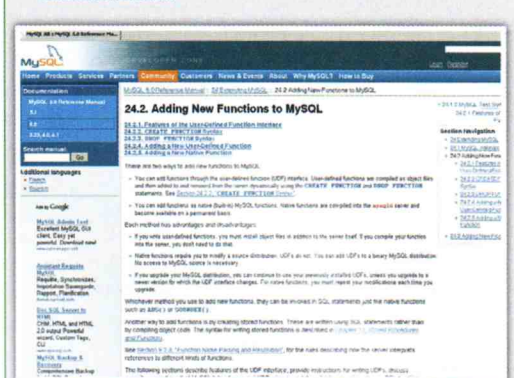
Sachez aussi que MySQL intègre, à partir de sa version 5.1, une nouvelle interface, appelée « plug-ins » qui est amenée à améliorer, puis remplacer, les fonctions UDF actuelles. Pour cela, une nouvelle API est proposée, mais qui ne permet que la création de plug-ins de recherches `FULL-TEXT`. La manipulation des plug-ins fait aussi intervenir de nouvelles commandes (`INSTALL PLUGIN`, `UNINSTALL PLUGIN` et `SHOW PLUGINS`).

L'adaptation de notre fonction `phonex()` au futur standard ne devrait toutefois pas poser de problème !

Vous n'avez plus qu'à implémenter votre extension de rêve pour MySQL...et à la faire partager à la communauté !

BIBLIOGRAPHIE

- **[MYSQL1]** String Comparison Functions, **http://dev.mysql.com/doc/refman/5.0/en/string-comparison-functions.html**
- **[BROUARD]** L'art des « Soundex », **http://sql.developpez.com/soundex/**
- **[MYSQL2]** Adding New Functions to MySQL, **http://dev.mysql.com/doc/refman/5.0/en/adding-functions.html**



Jérôme Delamarche,

jd@delamarche.com

Brèves de Perl

Sébastien Aperghis-Tramoni

Perl 6 Microgrants

Best Practical Solutions, la société de Jesse Vincent qui fournit les outils de *kewl kids* comme RT, Jifty et SVK, vient de donner 5 000 US$ à la Fondation Perl pour permettre le financement d'une dizaine de micro-projets à 500 US$ chacun. L'objectif est de donner un coup de fouet au développement de Perl 6, Pugs, Parrot, Perl 6-on-5 (l'exécution de Perl 6 sur l'interpréteur Perl 5 actuel), et tout autre projet lié à l'implémentation de Perl 6.

Comme il s'agit de projets courts, les organisateurs veulent favoriser des projets réalisables en 4 à 6 semaines. Les personnes intéressées peuvent envoyer une proposition à **perl6-microgrants@perl.org**.

Lien : **http://xrl.us/vg92 (news.perl-foundation.org)**

Un premier projet, proposé par Steve Peters, a déjà été accepté : il va travailler à l'amélioration du support de Parrot pour certains compilateurs et certaines plates-formes, en particulier GCC sur Cygwin et ICC (le compilateur d'Intel) sur Linux et Mac OS X.

Lien : **http://xrl.us/vg95 (news.perl-foundation.org)**

DBI2 Grants

Tim Bunce a rebondi sur l'annonce initiale de Jesse Vincent sur P5P en proposant d'utiliser les fonds de Développement de DBI (indépendants de la Fondation Perl et de l'argent donné par Best Practical Solutions) pour financer le développement de DBI version 2 (écrite en Perl 6), dont l'API doit suivre celle de JDBC, jugée comme étant la plus pratique. L'un des besoins serait donc d'écrire un outil capable de transformer l'API Java en code Perl 6 équivalent.

Lien : **http://xrl.us/vg2u (groups.google.com)**

LaTeX pour l'impatient

Denis Bodor

Voici un ouvrage qui cache bien son jeu. Derrière ce titre qui laisse présager des explications rapides et superficielles, se cache en réalité un guide pratique très complet pour l'utilisateur désireux de « se mettre » à LaTeX.

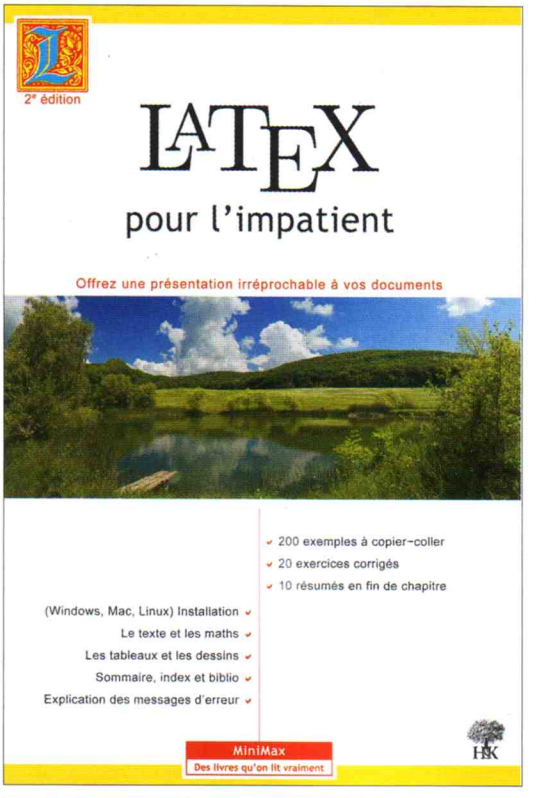
Composeur de texte par excellence, LaTeX est aussi d'une complexité déroutante pour le débutant, car il suit une logique particulière, mais néanmoins structurée.

L'ouvrage est conçu comme un guide et le lecteur prendra en main LaTeX progressivement, étape par étape, chapitre par chapitre. Chaque chapitre se conclut par quelques exercices sur les connaissances acquises. De quoi vérifier ce qu'on apprend et surtout se satisfaire de sa progression !

Deux points sont particulièrement remarquables dans cet ouvrage :

- On constate clairement que les auteurs ont un usage pratique de LaTeX. Il ne s'agit pas de résultats de recherches ou d'expérimentations théoriques sur l'art et la manière d'utiliser telle ou telle extension, mais bel et bien d'un guide pratique. On le remarque non seulement grâce aux explications elles-mêmes, mais surtout via les conseils, commentaires et astuces qui ponctuent la plupart des exemples.
- La mise en page du livre elle-même est une magnifique vitrine pour LaTeX d'une part, mais aussi pour les connaissances maîtrisées par les auteurs. Il est toujours très sympathique et rassurant de voir que les explications données ont une application directe. Lorsqu'on feuillette ce livre, on a réellement l'impression que les auteurs savent de quoi ils parlent et le mettent en pratique.

Voici un ouvrage d'apprentissage de LaTeX comme on aimerait en voir plus souvent : simple, progressif, clair, direct et pratique. Un livre fait par de vrais utilisateurs de LaTeX pour ceux qui souhaitent vraiment utiliser LaTeX. Bref, un ouvrage avec lequel j'aurais aimé débuter.



- Auteurs : W. Appel, C. Chevalier, E. Cornet, S. Desreux, J.J. Fleck et P. Pichaureau
- Éditeur : H & K
- ISBN : 978-2-35141-016-5
- 160 pages – 12,90 euros
- Format : 17cm x 25cm

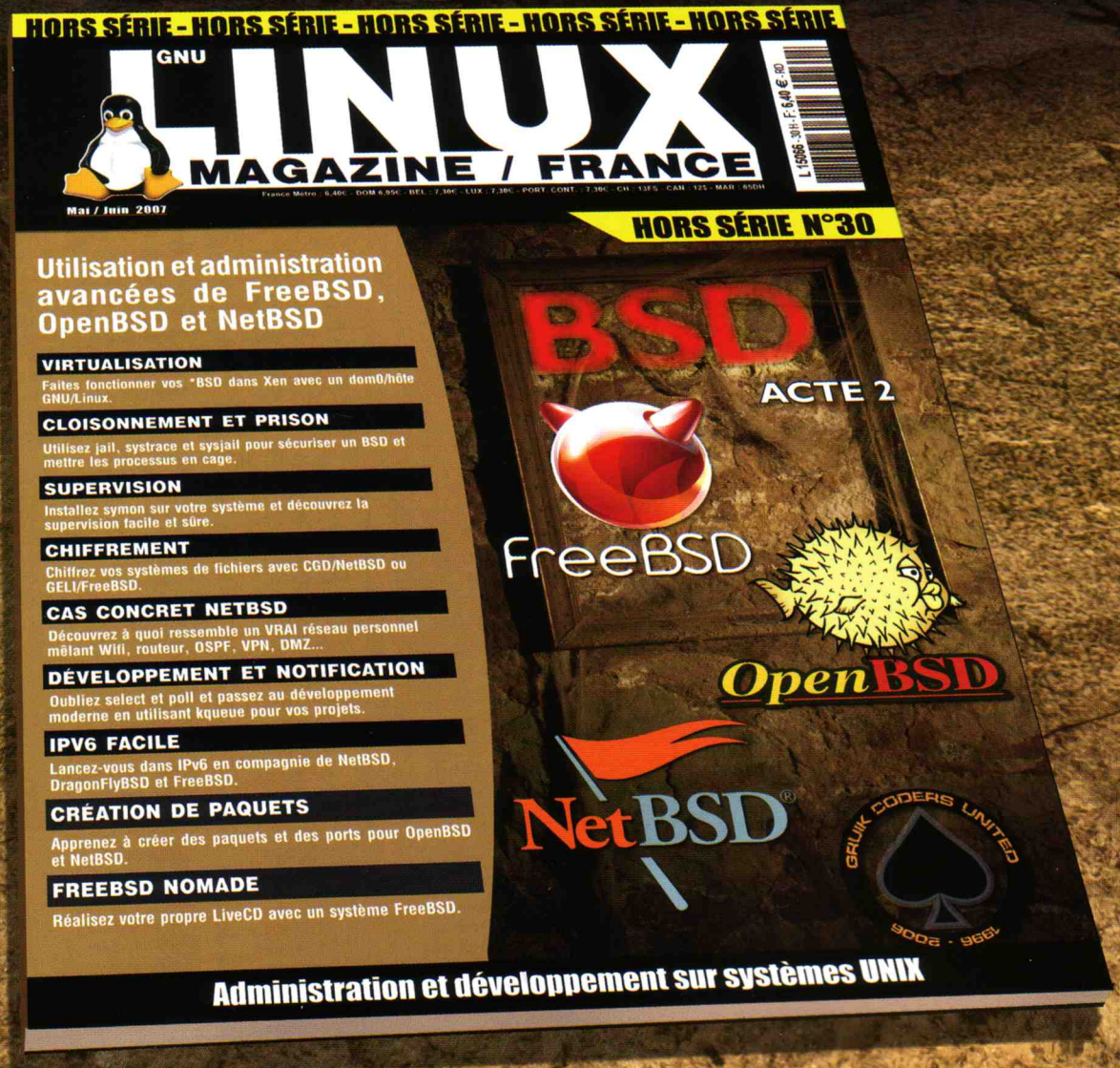
À paraître début juin

GNU LINUX MAGAZINE / FRANCE

HORS SERIE 30

En KIOSQUE

BSD acte II



INCLUS : encore plus de poils