N°168 FÉVRIER 2014

GNU LINUX MAGAZINE / FRANCE

SECU / SMARTCARD

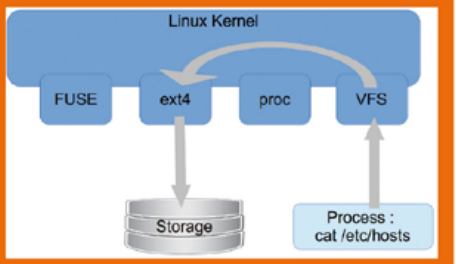
Découvrez l'OpenPGP Card V2.0 et les différentes solutions pour lire et utiliser cette smartcard

p.43

ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

INTRO / FS

Tout ce que vous avez toujours voulu savoir sur les systèmes de fichiers p.20



GIT / WEB

GitLab : Une solution open source pour avoir votre GitHub à vous tout seul sur votre serveur dédié p.47

CODE / FUSE

Faites briller d'admiration les yeux de vos collègues en codant votre système de fichiers avec python-fuse p.76

L 19275 - 168 - F: 7,90 € - RD

LATEX / LIVRE

Oubliez les suites bureautiques et apprenez à utiliser un véritable outil de composition de texte ! p.04

SURVEILLANCE / RIPOSTE

Prism, StellarWind, XKeyscore, Turmoil...

Maintenant, vous n'avez plus le choix...

CHIFFREZ !

...avec GnuPG p.30

1. Fonctionnement et principes de base
2. Création avancée de votre trousseau
3. Sécurisation par utilisation de sous-clés
4. Signature, vérification, chiffrement, déchiffrement

PHP / STREAM

Maîtrisez la gestion des flux et développez vos propres wrappers pour prendre en charge des schemes non natifs p.62

C++ / STL

Continuez et terminez votre exploration de la bibliothèque standard : conteneurs associatifs, itérateurs, multimap, bitset, etc. p.66

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD

ET SI VOUS PASSIEZ À LA PUISSANCE CLOUD ?

VAR

Distinguez-vous de vos concurrents en valorisant votre offre.

SSII

Profitez d'infrastructures IaaS déployées en un clin d'œil.

ÉDITEURS

Passez au SaaS en vous appuyant sur notre savoir-faire.

PROGRAMME PARTENAIRE ARUBA CLOUD

Contactez-nous

- 2 niveaux de marque blanche disponibles.
- Gestion simple, souple et performante de l'infrastructure (publique, privée ou hybride).
- Modélisation et activation immédiate de votre datacenter virtuel dans le pays de votre choix.
- Interface client totalement personnalisable.
- Facturation en " Pay as you go ".
- Finesse dans la gestion des droits utilisateurs.

Aruba, le bon partenaire pour bénéficier de la puissance d'un acteur majeur qui considère que chaque client, dans chaque pays, est unique. **MY COUNTRY. MY CLOUD.**

aruba CLOUD

arubacloud.fr | TÉL : 0810 710 300
(COÛT D'UN APPEL LOCAL)

SOMMAIRE N°168

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax** : 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – boutique.ed-diamond.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Réalisation graphique : Jérémy Gall

Responsable publicité : Valérie Fréchard, Tél. : 03 67 10 00 27
v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, 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

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, N° **ISSN** : 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

MIXTE Papier issu de sources responsables FSC® C015136 www.fsc.org

Membre April www.april.org

ÉDITORIAL

Connaissez-vous Litmus ?

Non, ce n'est pas une typo, je n'ai pas dit Linus (désolé par ailleurs pour la monstrueuse et néanmoins classique ambiguïté de clavier Linux/linus dans le dernier numéro). Litmus est une solution propriétaire (berk) de test, d'analyse et de tracking de mail (re-berk) permettant de profiler au mieux vos messages commerciaux HTML (re-re-berk). Mais ce n'est pas là l'important. Ce qui l'est, c'est la statistique de répartition des clients mail que diffuse régulièrement cette société (en CC BY-NC-SA, ça c'est sympa : emailclientmarketshare.com).

En janvier dernier, Litmus annonçait ainsi que 51% des mails ouverts l'étaient sur une plateforme mobile (iOS et Android), contre 29% sur un client desktop et 20% sur un webmail. Ne cherchez pas un client GNU/Linux ou *BSD dans le top 10, c'est illusoire.

51% ! C'est tout simplement énorme et surtout le signe annonciateur de ce que beaucoup avaient prophétisé ces derniers temps. L'informatique personnelle bascule vers le tout-mobile ou, en d'autres termes, l'informatique personnelle disparaît et avec elle, la maîtrise des outils à la disposition des utilisateurs. Cette même maîtrise qui est au cœur du logiciel libre, pour ainsi dire son moteur et son but, car maîtriser c'est avoir la liberté d'utiliser, modifier et partager. Sans connaissances, point de libertés et sans nécessité, envie ou possibilité de connaître, point de connaissances...

Le monde de l'informatique utilisateur glisse, sinon dérive, doucement vers l'abstraction absolue. Il bascule inéluctablement de la technologie à la magie (cf. Arthur C. Clarke) et le mouvement s'accélère.

Le 11 février prochain sera officiellement « The Day We Fight Back » (thedaywefightback.org) en opposition au régime d'espionnage de masse et à la surveillance globale d'Internet. Nous savons ce que cela signifie, nous savons ce qu'il se passe actuellement, mais on ne peut que s'inquiéter, non pas de la portée de ce genre d'action, mais de la compréhension limitée de la part de la vaste majorité des utilisateurs. À l'instar de l'absence de mots pour définir une idée, l'absence de connaissances sur le fonctionnement d'une technologie annihile l'évaluation du risque (et donc, dans l'esprit des gens, le risque lui-même). Dans un monde magique où les mails ne sont guère plus que des parchemins enchantés qui se téléportent de tablette en tablette, difficile d'expliquer les dangers des connexions non chiffrées...

Est-on réellement dépossédé de notre espace de créative liberté qu'est Internet par des entités souhaitant à tout prix réguler et contrôler, ou ne sont-elles que les symptômes d'une maladie plus grave, un cancer appelé paresse intellectuelle chronique ? « L'ignorance, c'est la force », n'est-ce pas ?

Tant pis, soyons faibles ! Apprenons, expérimentons, jouons, découvrons... et chiffrons !

Denis Bodor

CRÉATION D'UN MODÈLE LATEX POUR UN LIVRE

par Tristan Colombo

LaTeX permet d'écrire très simplement des formules mathématiques, mais pas seulement. En passant un peu de temps à la création d'un modèle vous obtiendrez exactement la présentation que vous aviez en tête avec une qualité bien supérieure à ce qu'il est possible de faire avec LibreOffice Writer (et bien sûr Microsoft Word).

La présentation d'un document, même si son contenu est fascinant, va influencer le lecteur. Prenons un exemple concret : deux étudiants doivent rendre un mémoire qui sera corrigé par un même professeur. Les deux étudiants travaillent sur le même sujet et nous allons considérer que le professeur est très distrait et ne s'aperçoit pas que le contenu des deux mémoires est identique à la virgule près (si certains de mes étudiants me lisent, ne rêvez pas, même d'une année sur l'autre je détecte les doublons...). Une seule chose va différencier les deux documents : la présentation. Dans le premier cas le style aura été travaillé et dans le second pas du tout. La note sera forcément différente : le lecteur aura un apriori négatif en lisant le second mémoire et sera donc moins indulgent, plus enclin à pénaliser les erreurs. Ici la sanction est une note mais cet exemple pourrait être transposé dans de nombreux autres cas : rapport technique, document de présentation d'un projet, etc. Le temps passé à élaborer un modèle n'est donc pas du temps perdu, loin de là !

Dans cet article je considérerai que vous avez déjà manipulé LaTeX et que vous en connaissez donc les principes de base. Je commencerai tout de même par quelques rappels de manière à ce que les novices puissent suivre en effectuant quelques recherches (deux livres **[1][2]** en particulier donnent de très bonnes informations). Il nous faudra détailler ensuite les éléments constituant un livre pour lesquels nous écrirons un style particulier. Ce travail est long mais s'il est bien fait, on ne le modifie que très rarement.

1 Quelques rappels pour se remettre dans le bain

1.1 La structure d'un livre

Tout le monde connaît la structure générale d'un livre mais nous allons néanmoins revoir celle-ci pour établir une liste des éléments que nous devrons incorporer dans notre modèle.

Lorsque nous prenons un livre, la première chose que nous voyons est la couverture qui contient éventuellement une illustration et à minima le titre du livre et le nom de son auteur. Au verso de cette page nous n'avons rien.

La page suivante est un rappel du titre et de l'auteur comportant éventuellement des mentions légales. Le recto peut contenir des mentions légales ou rester vide.

Nous arrivons ensuite sur la table des matières répertoriant les chapitres et les sections tout en indiquant leur numéro de page. Les numéros de page peuvent être indiqués de diverses manières : centrés en bas de page, en haut de page et à gauche pour les verso de pages et à droite pour les recto de pages avec éventuellement un rappel du titre de la section de la page, etc. Il faut donc gérer différemment le recto et le verso des pages, ce qui est d'autant plus important au niveau des marges pour laisser une place pour la reliure.

Les chapitres suivent ensuite le style de numérotation de la table des matières et seule la première page peut être plus ou moins travaillée. Je retiendrai le style consistant à inclure une image et à indiquer le titre dans un cadre. Il faut penser qu'un chapitre commence toujours au recto d'une page.

En fin de livre on peut trouver éventuellement une bibliographie et un index des termes importants cités dans les pages précédentes. Leur style sera lié au style des chapitres.

La dernière page est le verso de la couverture qui va contenir un résumé du contenu du livre ou qui sera laissé blanc.

Pour récapituler, nous allons donc devoir définir des styles pour :

- une couverture contenant une illustration,
- une page de rappel,
- une table des matières,
- des chapitres,

- une bibliographie,
- un index.

La définition de ces styles va passer par des commandes LaTeX.

1.2 Commandes LaTeX

Pour écrire du code LaTeX un simple éditeur de code suffit. Vous pouvez donc utiliser Vim, Emacs, etc, et compiler votre code par des commandes dans le shell. Si vous préférez utiliser une interface graphique, il en existe quelques unes mais je trouve deux d'entre elles particulièrement bien pensées : Kile et Latexila. Je décrirai ici une utilisation en « mode shell » qui pourra être transposée au niveau des interfaces graphiques en retrouvant les boutons qui effectuent les mêmes actions.

Tout au long de cet article nous utiliserons des extensions supplémentaires. Nous les installerons au moment où j'y ferai référence. Nous aurons toutefois besoin de LaTeX pour commencer et j'utiliserai TeXLive sur une distribution basée sur Debian :

```
$ sudo aptitude install texlive texlive-latex3 texlive-latex-extra
```

Cela suffit pour créer un premier document.

1.2.1 La structure de base

Un document est toujours construit en suivant le même structure :

```
01: \documentclass{article}
02:
03: \begin{document}
04:     Le texte que l'on souhaite écrire
05: \end{document}
```

La ligne 1 indique la classe de document que nous souhaitons utiliser. Il s'agit d'un modèle de style qui peut éventuellement être paramétré à l'aide d'options qui seront indiquées entre crochets avant le nom de la classe de document : **`\documentclass[option_1, option_2, ...]{type_de_document}`**. Les lignes 3 et 5 définissent le début et la fin du document. En LaTeX on dit qu'il s'agit d'un environnement.

Si le document a été écrit dans un fichier **`structure_base.tex`**, la compilation se fera par :

```
$ pdflatex structure_base.tex
```

Vous obtiendrez de nombreuses informations à l'écran ainsi que plusieurs fichiers dont un fichier **`pdf`** correspondant à notre document et un fichier de log contenant des informations sur le déroulement de la compilation. En ouvrant le fichier **`structure_base.pdf`** vous verrez un document avec des marges exagérément grandes et un petit texte : « Le texte que l'on souhaite ecrire ». Vous avez bien lu, il ne s'agit pas d'une erreur d'impression, il manque l'accent. La gestion des polices accentuées peut se faire de diverses manières, la plus simple étant l'utilisation d'une extension.

1.2.2 Utilisation d'une extension

Le chargement d'une extension se fait à l'aide de la commande **`\usepackage`** suivie éventuellement d'options et du nom de l'extension. On indique la liste des extensions après la commande **`\documentclass`** :

```
01: \documentclass{article}
02:
03: \usepackage[utf8]{inputenc}
04:
05: \begin{document}
06:     ...
```

L'extension utilisée ici en exemple permet de prendre en compte les lettres accentuées. Pour ceux d'entre vous n'ayant pas (encore) configuré leur machine en utf-8, il faudra utiliser l'option **`latin-1`**... à moins, bien sûr, que vous ne travailliez en mandarin.

1.2.3 Les commentaires

On peut commenter le code LaTeX à n'importe quel endroit en utilisant le caractère **`%`** et le code ne sera plus lu jusqu'à la fin de la ligne :

```
...
04: % Début du document
05: \begin{document} % Commentaire en fin de ligne
06:     ...
```

1.2.4 Définition d'un chapitre ou d'une section

Pour définir un nouveau chapitre ou une nouvelle section on utilise les commandes **`\chapter`** (pour certaines classes de documents) ou **`\section`** (et **`\subsection`** ou **`\subsubsection`** pour les différentes sous-sections). Ces commandes calculent automatiquement le numéro du chapitre ou de la section et prennent seulement en paramètre le titre à afficher :

```
01: \documentclass{book}
02:
03: \usepackage[utf8]{inputenc}
04:
05: \begin{document}
06:     \chapter{Premier chapitre}
07:     \section{Première section}
08:     \subsection{Première sous-section}
09:     Texte...
10: \end{document}
```

Ici avec le style par défaut de la classe de documents **`book`**, vous obtiendrez l'affichage de la figure 1. Il y aura donc du travail à effectuer sur la commande **`\chapter`** puisqu'il est hors de question de laisser cet affichage en anglais qui ne correspond pas à notre souhait de départ.

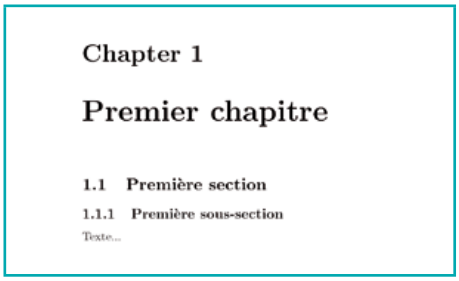


Fig. 1 : Affichage par défaut d'un chapitre d'un document utilisant la classe « book ».

1.2.5 Création d'une commande

Une commande LaTeX, encore appelée macro-commande, peut être créée à l'aide de l'instruction **`\newcommand`**

ou redéfinie à l'aide de **\renewcommand** (pour modifier le comportement d'une commande existante). La syntaxe de ces instructions sera la suivante :

```
\newcommand{\nom_de_la_commande}[nombre_
arguments][défaut_1]{suite_commandes}
```

Le nombre d'arguments doit être compris entre **0** et **9** et la référence à un argument dans la suite de commandes se fait à l'aide du caractère **#** suivi de la position de l'argument. On peut indiquer la valeur par défaut pour le premier paramètre (considéré alors comme optionnel) et on doit ensuite spécifier la liste des commandes. Voici un exemple où nous créons la commande **\etatcivil** qui affiche le nom et le prénom d'une personne avec la civilité en gras :

```
\newcommand{\etatcivil}[3]{
    \textbf{#1} #2 #3
}
```

L'appel se fait alors par :

```
\etatcivil{M.}{Richard}{Stallman}
```

Nous pouvons simplifier l'appel en utilisant le premier paramètre comme paramètre optionnel et en lui allouant la valeur par défaut « M. » :

```
\newcommand{\etatcivil}[3][M.]{
    \textbf{#1} #2 #3
}
```

L'appel est alors modifié :

```
\etatcivil{Richard}{Stallman}
\etatcivil[Mme]{Steffi}{Graff}
```

1.2.6 Création d'un environnement

Comme avec les commandes, nous pouvons créer ou modifier un environnement existant. La syntaxe est sensiblement la même à la différence prêt qu'il faut indiquer les séquences de début et de fin d'environnement :

```
\newenvironment{nom_de_l_environnement}
[nombre_arguments][defaut_par_1]{sequence_
debut}{sequence_fin}
```

Il faut faire attention ici à ne pas préfixer le nom de l'environnement par un caractère ****, et à bien refermer les environnements qui auraient été ouverts dans la séquence de début. Voici un exemple qui affiche du texte au centre de la page avec une sorte de titre en gras :

```
\newenvironment{mon_environnement}[1]{\
begin{center}\textbf{#1}\\}{\end{center}}
```

Pour utiliser cet environnement il faut indiquer un titre et le texte à afficher :

```
\begin{mon_environnement}{Environnement spécial}
    Texte dans mon environnement
\end{mon_environnement}
```

C'est exactement comme si nous avions écrit directement :

```
\begin{center}
    \textbf{Environnement spécial}\\
    Texte dans mon environnement
\end{center}
```

Le résultat de ce code est visible en figure 2.

Environnement spécial
Texte dans mon environnement

Fig. 2 : Exemple d'utilisation de notre environnement personnalisé « mon_environnement ».

1.2.7 Création d'une classe de documents

Une classe de documents est simplement un fichier qui va s'appuyer sur une classe existante et y ajouter des définitions de commandes, d'environnements et charger des extensions. Elle commence par la commande **\ProvidesClass** qui indique le nom de la classe puis **\LoadClass** qui charge la classe que l'on va modifier et les lignes suivantes contiennent le code de modification. L'extension des fichiers de classes est **.cls**. Voici un exemple d'entête d'un tel fichier :

```
01: \ProvidesClass{ma_classe}
02:
03: \LoadClass[a4paper,12pt]{article}
04:
05: % Chargement des extensions
06: \usepackage[utf8]{inputenc}
07: ...
08:
09: % Définition des commandes et environnements
10: ...
```

L'utilisation de cette classe dans un document LaTeX se fera ensuite par **\documentclass{ma_classe}**. Pour que le code compile il faudra que vous vous assuriez que le fichier **ma_classe.cls** se trouve dans le même répertoire que le fichier **.tex** ou dans un répertoire accessible par LaTeX (par exemple dans **~/.texmf/tex/latex/maclasse/** ou alors dans **/usr/local/share/texmf/tex/latex/maclasse/** pour que tous les utilisateurs y aient accès mais pensez à exécuter **sudo texhash** pour remettre la base à jour).

2 Le modèle

Pour notre modèle de livre nous allons partir de la classe de documents « book » que nous allons modifier. Nous allons donc écrire une nouvelle classe qui nous permettra de définir toute la structure de notre livre.

2.1 Définition du format des pages et des polices employées

Nous partons de la classe de documents **book** et nous pouvons commencer par spécifier quelques options. Il est par exemple possible de formater le texte sur deux colonnes avec l'option **twocolumn**, de changer la taille de la police ou du papier, etc. Nous allons simplement indiquer que nous souhaitons une police en 11pt et que si des équations sont affichées nous souhaitons les voir alignées à gauche plutôt que centrées (comportement par défaut). Voici donc les premières lignes de notre fichier de classe **mybook.cls** :

```
01: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
02: % Mybook template
03: %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
04:
05: \ProvidesClass{mybook}
06:
07: \LoadClass[11pt,fleqn]{book}
```

En ligne 5 nous spécifions le nom de notre classe et en ligne 7 nous chargeons la classe **book** sur laquelle nous nous appuyons en spécifiant nos options. Pour l'instant notre classe n'est qu'une encapsulation de la classe **book** avec des paramètres fixés.

Pour fixer la taille du papier et la taille des marges nous allons utiliser l'extension **geometry**. Cette extension permet la configuration de nombreux paramètres, résumés par le schéma de la figure 3. Pour les pages de notre livre nous avions indiqué qu'il fallait faire attention à la taille des marges suivant que l'on se trouve sur un recto ou un verso pour permettre l'ajout d'une reliure. Nous allons tenir compte de cette contrainte dans notre paramétrage :

```
09: %%%%
10: % Page settings
11: \usepackage[a4paper,headsep=10pt,top=3cm,bottom=3cm,inner=3.2cm,
outer=2.2cm]{geometry}
```

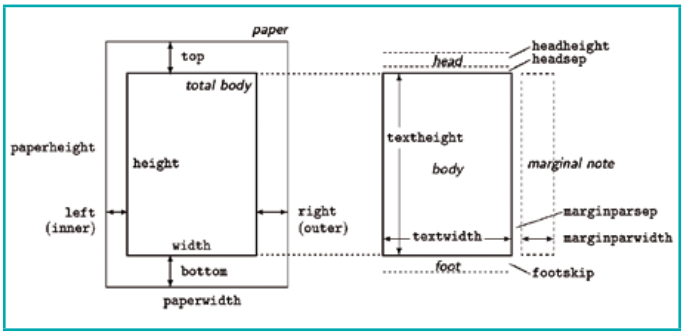


Fig. 3 : Paramètres de la page configurables à l'aide de l'extension geometry.

Pour raffiner nous pouvons charger des polices particulières. Le plus simple est de faire son choix dans le catalogue du site http://www.tug.dk/FontCatalogue/. Toutes les polices présentées sont des polices hébergées sur le CTAN (Comprehensive TeX Archive Network, le dépôt des extensions LaTeX). Les polices sont classées par catégories et un code d'exemple d'utilisation vous est donné pour chacune d'elles. Il faut choisir une police pour les titres et une autre pour le texte courant :

```
13: %%%%
14: % Font settings
15: % Use site http://www.tug.dk/FontCatalogue/ to select your fonts
16: \usepackage{microtype}      % Improves general appearance of text
17: \usepackage[utf8]{inputenc} % Letters with accentuation
18: \usepackage[T1]{fontenc}    % T1 characters encoding
19: \usepackage{emerald}        % Augie font for headings
20: \usepackage{mathptmx}       % Times Roman as default font
```

En dehors de la possibilité d'écrire des caractères accentués (ligne 17) et du chargement de nos deux polices de caractères qui sont ici Augie et Times Roman (lignes 19 et 20), nous avons également activé l'encodage de caractères T1 (ligne 18) et le chargement de **microtype** (ligne 16). Le codage T1 est la nouvelle norme LaTeX qui n'est pas activée par défaut pour des raisons de compatibilité avec d'anciens documents LaTeX et **microtype** permet à l'aide de différentes techniques **[3]** d'améliorer l'apparence du texte.

S'il vous manque une police, il faudra aller la chercher sur le site du CTAN http://www.ctan.org et télécharger l'extension au format **.zip** (pour Augie il faut se rendre sur http://www.ctan.org/tex-archive/fonts/emerald). Il faut ensuite extraire le contenu de ce fichier dans votre répertoire **~/.texmf** :

```
$ cd ~/.texmf
$ wget http://mirrors.ctan.org/fonts/emerald.zip
$ unzip emerald.zip
$ rm emerald.zip README
$ cp -R emerald/* .
$ rm -R emerald
$ cd fonts/map/dvips
$ updmap --enable Map=emerald.map
$ texhash
```

Méfiez-vous toutefois des extensions que vous installez de la sorte, certaines ne respectant pas le format TDS (TeX Directory Structure). Par exemple c'est le cas de la police Nimbus Sans Serif où il faut déplacer un à un tous les répertoires pour que l'installation soit correcte.

2.2 La page de titre

Nous savons que la page de titre contiendra une illustration en arrière plan. Il faut donc charger l'extension qui nous permettra l'insertion de cette image :

```
22: %%%%
23: % Graphics
24: \usepackage{graphicx}       % Enables images inclusion
25: \graphicspath{{./images/}}  % Directory containing images to use
26: \usepackage{eso-pic}        % To add background image
```

Notez qu'en ligne 25 nous indiquons dans quel répertoire seront stockées les images.

Nous pouvons ensuite définir une nouvelle commande qui insérera la page de titre, le verso de cette page et la page de rappel du titre. Cette commande acceptera quatre paramètres : une image à mettre en arrière plan, un titre, un sous-titre qui pourra éventuellement rester vide, le nom de l'auteur, et éventuellement des informations additionnelles à afficher :

```
28: %%%%
29: % General
30: \usepackage{ifthen}
31:
32:
33: %%%%
34: % Title Page
35: % [#1] : Background image
36: %  #2  : Title
37: %  #3  : Subtitle
38: %  #4  : Author
39: %  #5  : Additional informations (leave blank if not needed)
40: \newcommand{\titlePage}[5][backgroundImage]{
```

```
41:     \begingroup
42:         \thispagestyle{empty}
43:         % Background image
44:         \AddToShipoutPicture*{
45:             \put(0,0){\
includegraphics[scale=1]{#1}}
46:         }
47:         \centering
48:         \vspace*{7.5cm}
49:         \par\fontsize{40}{40}
50:         \ECFAugie
51:         % Title
52:         #2\par
53:         \vspace*{0.5cm}
54:         % Subtitle
55:         {\Huge #3}\par
56:         \vspace*{1cm}
57:         % Author
58:         {\LARGE #4}\par
59:     \endgroup
60:
61:     \begingroup
62:         \newpage
63:         ~\vfill
64:         \thispagestyle{empty}
65:         % Title
66:         \noindent #2 - #3\\
67:         % Author
68:         \noindent #4\\
69:         % Additional informations
70:         \ifthenelse{\equal{#5}{}}{
71:         }
72:         {
73:             \noindent #5\\
74:         }
75:     \endgroup
76:
77:    \begingroup
78:        \newpage
79:        \thispagestyle{empty}
80:        \centering
81:        \vspace*{7.5cm}
82:        \par\fontsize{40}{20}
83:        \ECFAugie
84:        % Title
85:        #2\par
86:        \vspace*{0.5cm}
87:        % Subtitle
88:        {\Huge #3}\par
89:        \vspace*{1cm}
90:        % Author
91:        {\LARGE #4}\par
92:        \newpage
93:         \thispagestyle{empty}
94:     \endgroup
95: }
```

Il y a beaucoup de choses à commenter dans ce code. Tout d'abord en ligne 30 nous chargeons l'extension **ifthen** qui permet d'effectuer des tests conditionnels. Ce mécanisme nous servira pour les informations additionnelles du verso de la couverture. La commande **\titlePage** proprement dite commence en ligne 40 et elle est composée de trois sections encadrées par des **\begingroup ... \endgroup** permettant de créer des zones de visibilité pour les configurations de style (et aussi les variables) :

- lignes 41 à 59 : description de la couverture. La ligne 42 indique que nous ne souhaitons pas d'affichage du numéro de page. Les lignes 44 à 46 indiquent quelle image nous souhaitons insérer en arrière plan (commande issue de l'extension **eso-pic**). Les coordonnées (0, 0) indiquent la position du coin supérieur gauche de l'image sur la page. En ligne 47 nous passons en mode « centré », en ligne 48 nous déplaçons le curseur d'écriture de 7,5cm vers le bas et en ligne 49 nous déterminons une taille de police de 40pt avec un espace interligne de 20pt. La commande **\par** indique la fin d'un paragraphe et permet à LaTeX de mieux mettre en forme le texte. La ligne 83 indique que nous souhaitons utiliser la police Augie et les lignes suivantes affichent les différents paramètres transmis à la commande.
- lignes 61 à 75 : le verso de la couverture. En ligne 62 nous passons à la page suivante puis nous indiquons en ligne 63 que nous allons écrire au bas de la page (**~\vfill** « pousse » le texte en bas de page). La commande **\noindent** utilisée à partir de la ligne 66 indique que nous désactivons l'indentation automatique sur un début de nouveau paragraphe. Les lignes 70 à 74 sont particulièrement intéressantes : il s'agit d'une structure conditionnelle où le premier paramètre de **\ifthenelse** est un test, le deuxième paramètre correspond au code exécuté si le test est vrai et le troisième contient le code a exécuté si le test est faux. Ici si le cinquième paramètre ne contient aucune donnée on ne fait rien (ligne 71) sinon on l'affiche (ligne 73).- lignes 77 à 94 : le rappel du titre. Il s'agit du rappel de l'affichage du texte de la couverture sans image d'arrière plan. Par soucis de lisibilité ce code est un copié/collé des lignes 47 à 58, dans une vraie classe il faudrait bien sûr utiliser une commande annexe qui serait utilisée deux fois. Les lignes 92 et 93 permettent de désactiver l'affichage du numéro de page sur le verso.

Pour créer l'image de fond vous pouvez utiliser Inkscape ou Gimp avec un format A4 et l'enregistrer en format PostScript encapsulé (**.eps**) qui sera ensuite converti en pdf lors de la première compilation avec **pdflatex** ou utilisé directement en compilant avec **latex**. Nous pouvons dès à présent créer un petit fichier LaTeX pour tester notre classe. Ici mon image se nomme **couverture.eps** et se trouve dans le répertoire **./image** :

```
01: \documentclass{mybook}
02:
03: \begin{document}
04: \titlePage[couverture]{GNU Linux Magazine}
{Un modèle de livre en \LaTeX}{Tristan Colombo}
05: \end{document}
```

Fig. 4 : Aperçu de la couverture.

La figure 4 montre le résultat obtenu, résultat dépendant évidemment de la qualité de l'image de fond que vous utiliserez.

2.3 Les chapitres

Nous devrions normalement nous occuper maintenant de la table des matières... mais pour les tests, comme nous n'avons encore aucun chapitre ni section, nous n'aurions rien obtenu. Autant donc commencer par les chapitres.

Pour dessiner et ajouter une image à notre titre, nous allons utiliser l'extension TikZ **[4]** qu'il faut donc commencer par charger :

```
96: \usepackage{tikz}          % To draw figures
```

Nous dessinerons un cadre autour du texte et il faudra lui allouer une couleur. Pour cela nous allons charger l'extension **xcolor** et créer des couleurs à partir d'un code RGB (vous pouvez utiliser le menu de choix de couleur de Gimp pour obtenir ces codes). Les couleurs sont exprimées entre 0 et 1 ou entre 0 et 255 :

```
97: \usepackage{xcolor}
98: \definecolor{defaultColor}{rgb}{0, 0.56, 1} % Default color
99: \definecolor{black}{rgb}{0, 0, 0}           % Black color
```

Une image va illustrer l'en-tête du chapitre. Nous pouvons choisir une image par défaut qui servira pour tous les chapitres ou changer d'images à chaque chapitre. Le moyen le plus simple de mettre en place ce mécanisme est de créer une variable dont le contenu peut être modifié à l'aide d'une commande :

```
100: %%%%
101: % Variables
102: \newcommand{\theChapterImage}{}
103: \newcommand{\chapterImage}[1]{\renewcommand{\theChapterImage}{#1}}
104: % Default chapter image
105: \chapterImage{chapter_header}
```

Par défaut LaTeX recherchera l'image **./images/chapter_header.pdf** (ou **.eps** qu'il convertira). Si l'utilisateur souhaite changer d'image il n'aura qu'à appeler la commande **\chapterImage** en précisant un autre fichier.

Pour modifier l'affichage de l'en-tête d'un chapitre il faut modifier la macro **\@makechapterhead** (pour **\chapter**) ou **\makeschapterhead** (pour **\chapter***). Nous nous contenterons dans un premier temps de la première :

```
108: %%%%
109: % Chapter header (\chapter)
110: \def\@makechapterhead#1{
111:     \thispagestyle{empty}
112:     {
113:         \begin{tikzpicture}[remember picture,overlay]
114:             \node at (current page.north west){
115:                 \begin{tikzpicture}[remember picture,overlay]
116:                     \node[anchor=north west,inner sep=0] at (0, 0){
117:                         \includegraphics[width=\paperwidth]{\theChapterImage}
118:                     };
119:                     \draw[anchor=west] (5cm,-13cm) node [rounded corners=25pt,
120:                     text opacity=1, draw=defaultColor, draw opacity=1,
121:                     line width=2pt, inner sep=15pt]{
122:                         \ECFAugie
123:                         \huge\textcolor{black}{
124:                             \thechapter\ ---\ #1\makebox[20cm]{}
125:                         }
126:                     };
127:                 \end{tikzpicture}
128:             };
129:         \end{tikzpicture}
130:     }
131:     \par\vspace*{320pt}
132: }
```

La première page d'un chapitre ne doit pas afficher le numéro de page (ligne 111). Nous définissons ensuite un bloc de commandes à l'aide des accolades des lignes 112 et 130. Ces accolades sont équivalentes à **\begingroup** et **\endgroup** et peuvent s'utiliser dans les macros. Nous utilisons ensuite TikZ pour inclure l'image d'en-tête (ligne 117) enfouie dans des environnements **tikzpicture** permettant de positionner l'image (les coordonnées (0, 0) ont la même signification que précédemment). Les lignes 119 à 126 déterminent le cadre dans lequel se trouve le titre : à 13cm du haut de la page et 5cm de la gauche, un cadre arrondi (**rounded corners** ligne 119), tracé en bleu (**draw** ligne 120) avec une épaisseur de 2pt, un espacement de 15pt (ligne 121) et une longueur de 20cm qui sort de la page (ligne 124). La police utilisée pour afficher le titre est Augie (ligne 122) en noir et en 25pt (ligne 123). Le numéro du chapitre est donné par **\thechapter** et son nom par **#1** (voir ligne 124). Enfin, on laisse un espace de 320pt avant d'écrire le texte de manière à ne pas recouvrir l'image (ligne 131). La figure 5 montre le résultat obtenu.

Fig. 5 : Aperçu d'une première page de chapitre.

Si vous souhaitez raffiner un peu, vous pouvez faire remonter le cartouche contenant le texte sur l'image et rendre le fond transparent. Il faut alors remplacer les lignes 119 à 125 par :

```
119: \draw[anchor=west] (5cm,-9cm) node [rounded corners=25pt,
120:                    fill=white,fill opacity=.6,text opacity=1,
121:                    draw=black,draw opacity=1,line width=2pt,
122:                    inner sep=15pt]{
123:                        \ECFAugie
124:                        \huge\textcolor{black}{
125:                            \thechapter\ ---\ #1\makebox[20cm]{}
126:                        }
```

L'option **fill** détermine la couleur de fond et **fill opacity** indique la transparence. Il faut également modifier la ligne 131 pour ne laisser qu'un espace de 230pt. Le résultat obtenu est alors celui présenté en figure 6.

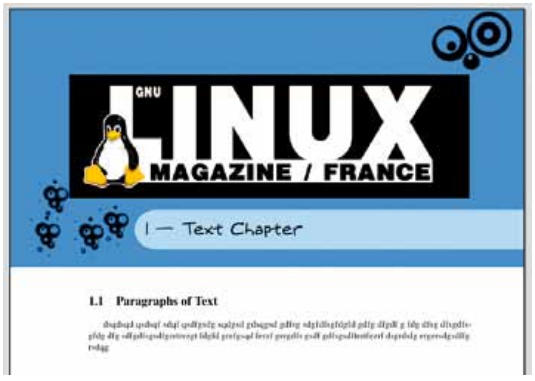


Fig. 6 : Aperçu de la première page de chapitre modifiée avec le texte sur l'image.

2.4 Les sections

Pour les sections nous allons simplement afficher le numéro de section en couleur et le titre en gras. Nous allons devoir modifier encore une macro et une commande :

```
128: %%%%
129: % Section header
130: \renewcommand{\@seccntformat}[1]{
131:     \llap{
132:         \textcolor{defaultColor}{\csname the#1\endcsname}
133:         \hspace{1em}
134:     }
135: }
136: \renewcommand{\section}{
137:     \@startsection{section}{1}{0pt}
138:     {-4ex \@plus -1ex \@minus -.4ex}
139:     {1ex \@plus.2ex }
140:     {\normalfont\large\bfseries}
141: }
```

Dans les lignes 130 à 135 nous modifions la macro **\@seccntformat**. La commande **\llap** indique que nous nous plaçons à gauche de la marge (**\rlap** se positionne bien sûr à droite) pour écrire en couleur le numéro de section (ligne 132) et définir un espace par rapport à la marge de **1em**, soit la taille qu'occuperait un caractère (pour une police en 11pt, **1em** vaut 11pt, pour une police en 20pt, **1em** vaut 20pt, etc.).

Dans les lignes 136 à 141 nous redéfinissons la commande **\section** en indiquant à la macro **\@startsection** six paramètres :

- le nom de la section : ici tout simplement **section** (ligne 137) ;
- le niveau associé qui sera utilisé pour générer la table des matières (ligne 137) ;
- l'indentation ou espace entre la marge et le titre (ligne 137) : ici **0pt** (on peut aussi utiliser **\z@** qui est une sorte de 0 générique)
- l'espace vertical qui précède le titre de la section (ligne 138). Cet espace à une valeur élastique de manière à ne pas laisser un titre tout seul en bas de page (ajout/retrait de valeurs). L'unité **ex** désigne la hauteur des plus petites lettres de la police en cours d'utilisation ;
- l'espace vertical qui suit le titre : une valeur positive indique un espacement vertical et une valeur négative indique un espacement horizontal (ligne 139) ;
- le style (ligne 140).

Si vous souhaitez que le titre de la section apparaisse coloré, il faut ajouter une instruction **\textcolor{defaultColor}** après **\bfseries** ligne 140.

2.5 Les en-têtes de pages

Pour modifier les en-têtes de page il existe une extension très pratique nommée **fancyhdr** (pour *fancy header* soit en-tête de luxe) :

```
143:  \usepackage{fancyhdr}
```

Il faut ensuite indiquer que nous souhaitons utiliser le style **fancy** pour les en-têtes et pieds de page et que nous ne voulons plus de l'affichage par défaut :

```
144: \pagestyle{fancy}
145: \fancyhf{}
```

Voici maintenant quelques valeurs que nous aurons à manipuler :

- **\leftmark** contient le nom du chapitre courant en majuscules (défini par la commande **\chaptermark**) ;
- **\rightmark** contient le nom de la section courante en majuscules (défini par **\sectionmark**) ;
- **\markboth** contient le nom du chapitre courant tel qu'il apparaît dans la table des matières ;
- **\markright** est le pendant de **\markboth** pour la section courante.

Nous souhaitons obtenir, comme dans la plupart des livres, le nom du chapitre sur la page de gauche et le nom de la section courante sur la page de droite. Les numéros de pages se positionneront sur la marge extérieure des pages. Pour indiquer le contenu de l'en-tête des pages de gauche on peut utiliser la commande **\lhead** et pour les pages de droite **\rhead** :

Abonnez-vous !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Consultez l'ensemble de nos offres sur : boutique.ed-diamond.com !

11 Numéros de GNU/Linux Magazine

60€*

au lieu de 86,90 €* en kiosque

Économie 26,90€

Économisez plus de 30%*

* Sur le prix de vente unitaire France Métropolitaine

Nouveauté 2014
TOUS LES HORS-SÉRIES PASSENT EN GUIDE !

Un nouveau format avec une reliure de luxe pour ces Guides de référence de 128 pages à conserver dans votre bibliothèque !

Découvrez tous les Guides déjà parus sur

boutique.ed-diamond.com

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE. Pour les tarifs hors France Métropolitaine, consultez notre site : boutique.ed-diamond.com

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 26,90 €/an ! (soit plus de 3 magazines offerts !)

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur boutique.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles de nos partenaires.

GNU LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

Abonnez-vous !

➡ Tous les abonnements incluant GNU/Linux Magazine :

offre LM

60€* au lieu de **86,90€**** en kiosque

Économie 26,90€

INCLUS : GNU/Linux Magazine (11nos)

offre LM+

+

115€* au lieu de **164,30€**** en kiosque

Économie 49,30€

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides

NOUVEAUTÉ 2014
TOUS LES HORS-SÉRIES PASSENT EN **GUIDE !**

offre T

198€* au lieu de **277,10€**** en kiosque

Économie 79,10€

INCLUS : GNU/Linux Magazine (11nos), Open Silicium (4nos), MISC (6nos), Linux Pratique (6nos) et Linux Essentiel (6nos)

offre T+

299€* au lieu de **411,20€**** en kiosque

Économie 112,20€

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides, Linux Pratique (6nos) + ses 3 Guides, MISC (6nos) + ses 2 Hors-Séries, Open Silicium (4nos) et Linux Essentiel (6nos)

* Tarifs France Métro (F) ** Base tarifs kiosque zone France Métro (F)

Consultez l'ensemble de nos offres d'abonnements sur :

boutique.ed-diamond.com

Vous pouvez également commander par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

➡ Nos Tarifs s'entendent TTC et en euros

		F	OM1	OM2	E	RM
		France Métro	Outre-Mer		Europe	Reste du Monde
LM	Abonnement GLMF	60 €	75 €	96 €	83 €	90 €
LM+	Abonnement GLMF + GLMF HS (6 Guides)	115 €	147 €	190 €	160 €	173 €
T	Abonnement GLMF + MISC + OS + LP + LE	198 €	253 €	325 €	276 €	300 €
T+	Abonnement GLMF + GLMF HS (6 Guides) + MISC + MISC HS + OS + LP + LP HS (3 Guides) + LE	299 €	382 €	491 €	415 €	448 €

• OM1 : Guadeloupe, Guyane française, Martinique, Réunion, St Pierre et Miquelon, Mayotte

• OM2 : Nouvelle Calédonie, Polynésie française, Wallis et Futuna, Terres Australes et Antarctiques française

MA FORMULE D'ABONNEMENT :

Offre	Zone	Tarif
☐ LM		
☐ LM+		
☐ T		
☐ T+		

J'indique la somme due : (Total) €

Exemple :
Je souhaite m'abonner à l'ensemble des magazines + tous les Hors-séries/Guides et je vis en Belgique. Je coche donc l'offre **T+** (la totale avec tous les Hors-Séries/Guides) puis ma zone (E), le montant sera donc 415 euros.

Je choisis de régler par :

☐ Chèque bancaire ou postal à l'ordre des Éditions Diamond

☐ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

```
147: %%%%
148: % Header style
149: \renewcommand{\chaptermark}[1]{
150:     \markboth{\bsc{\chaptername~\thechapter{} :} #1}{}
151: }
152: \renewcommand{\sectionmark}[1]{
153:     \markright{\thesection{} #1}
154: }
155: % Left header page
156: \lhead[\textbf{\textcolor{defaultColor}\thepage}]{
157:     \textsl{\rightmark}
158: }
159: % Right header page
160: \rhead[\textsl{\leftmark}]{
161:     \textbf{\textcolor{defaultColor}\thepage}
162: }
```

Il existe un autre format permettant de définir l'en-tête : la commande **\fancyhead** qui prend en option la position et le côté de page à modifier et en paramètre le texte à insérer. Au niveau de la position on peut indiquer gauche (**L**), droite (**R**) ou centré (**C**) et pour les pages se sera **E** pour les pages paires (côté gauche) et **O** pour les pages impaires (côté droit). Les lignes 156 à 158 seraient par exemple traduites en :

```
\fancyhead[LE]{\textbf{\textcolor{defaultColor}\thepage}}
\fancyhead[RE]{\textsl{\rightmark}}
```

Il reste maintenant des détails à régler. L'espacement entre l'en-tête et le corps du document est fixé par défaut à 13pt. Il faut modifier la valeur de **\headsep** pour agrandir ou diminuer cet espace :

```
\setlength{\headsep}{16pt}
```

Pour modifier l'épaisseur du trait séparant l'en-tête du corps du document c'est la valeur de **\headrulewidth** qu'il faudra régler, quitte à la mettre à zéro pour désactiver la ligne :

```
\renewcommand{\headrulewidth}{0pt}
```

Si votre en-tête est trop petit vous pourrez régler sa hauteur en modifiant **\headheight** :

```
\setlength{\headheight}{3cm}
```

Dernier réglage : supprimer l'en-tête sur les pages blanches de fin de chapitre (pages gauches ou paires). Pour cela nous redéfinissons la commande **\cleardoublepage** en effaçant le contenu si la page est paire (*odd* signifiant impair et *even* pair) :

```
163: \renewcommand{\cleardoublepage}{
164:     \clearpage
165:     \ifodd
166:         \c@page
167:     \else
168:         \hbox{}
169:         \vspace*{\fill}
170:         \thispagestyle{empty}
171:         \newpage
172:     \fi
173: }
```

Les figures 7a et 7b illustrent le résultat obtenu grâce au code précédent.

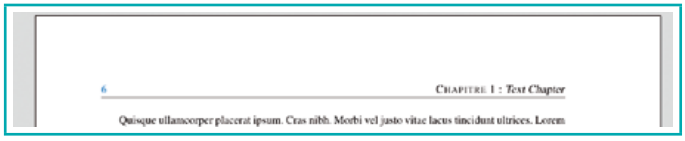


Fig. 7a : En-tête des pages paires (pages de gauche).

Fig. 7b : En-tête des pages impaires (pages de droite).

Si vous désirez modifier également les pieds de page, les commandes seront les mêmes que celles des en-têtes en modifiant **head** par **foot**. Ainsi **\fancyhead** devient **\fancyfoot**, etc.

2.6 La table des matières

Maintenant que nous avons des chapitres et des sections nous pouvons définir l'apparence de la table des matières. Là encore il existe une extension qui simplifie les manipulations : **titletoc** **[5]**.

```
175: %%%%
176: % Table of contents
177: \usepackage{titletoc}
```

Les différents styles utilisés ensuite pour indiquer les pages des chapitres, sections, etc. sont configurés à l'aide de la commande **\titlecontents** qui attend les six paramètres suivants :

- le nom de la section ;
- la position par rapport à la marge de gauche ;
- le style général ;
- la définition du contenu précédant l'entrée textuelle (le numéro de section) ;
- la définition du contenu précédant l'entrée textuelle s'il n'y a pas d'étiquette (inutile dans notre cas, ce paramètre restera vide);
- la définition de la composition de l'espace séparant le titre de la section du numéro de page (le numéro de page **\thecontentspage** est inclus dans cette définition).

La définition des différents styles est alors très simple :

```
178: % No margin
179: \contentsmargin{0cm} % Removes the default margin
180:
181: % Chapter text style in table of contents
182: \titlecontents{chapter}[1.25cm]
183: {\addvspace{15pt}\large\bfseries}
```

```
184: {\color{defaultColor!60}\contentslabel[\Large\thecontentslabel]
{1.25cm}\color{defaultColor}}
185: {}
186: {\color{defaultColor!60}\normalsize\bfseries \titlerule*[.5pc]
{.} \thecontentspage}
187:
188: % Section text style in table of contents
189: \titlecontents{section}[1.25cm]
190: {\addvspace{5pt}\bfseries}
191: {\contentslabel[\thecontentslabel]{1.25cm}}
192: {}
193: {\hfill\color{black}\thecontentspage}
194:
195: % Subsection text style in table of contents
196: \titlecontents{subsection}[1.25cm]
197: {\addvspace{1pt}\small}
198: {\contentslabel[\thecontentslabel]{1.25cm}}
199: {}
200: {\titlerule*[.5pc]{.} \thecontentspage}
```

Dans les lignes 184, 191 et 198, **\contentslabel** crée l'espace nécessaire à l'écriture du numéro de section **\thecontentslabel** et l'affiche. La seule autre difficulté de ce code est la présence de la commande **\titlerule*** qui définit l'espacement et les caractères représentant les lignes qui relient le titre d'une section à son numéro de page.

Nous avons défini un nouveau style pour l'affichage des données de la table des matières... mais nous n'avons pas encore inséré d'image pour être conforme à la mise en page des chapitres. Pour cela il faut modifier la commande **\tableofcontents** à laquelle nous allons ajouter un paramètre optionnel qui nous permettra le cas échéant de spécifier une image différente de l'image par défaut :

```
201: %%%%
202: % \tableofcontents replacement
203: % [#1] : name of the image to display
204: \renewcommand{\tableofcontents}[1][]{
205:     % Left table of contents header page
206:     \lhead[\textbf{\textcolor{defaultColor}\thepage}]{
207:         \textsl{\contentsname}
208:     }
209:     % Right table of contents header page
210:     \rhead[\textsl{\contentsname}]{
211:         \textbf{\textcolor{defaultColor}\thepage}
212:     }
213:     % Initialize number of page to 1
214:     % Change style of number of page to roman for table of contents
215:     \pagenumbering{roman}
216:     \if@twocolumn
217:         \@restonecoltrue\onecolumn
218:     \else
219:         \@restonecolfalse
220:     \fi
221:     \ifthenelse{\equal{#1}{}}{
222:     }
223:     {
224:         \chapterImage{#1}
225:     }
226:
227:     \chapter*{
228:         \contentsname
229:         \@mkboth{\MakeUppercase\contentsname}{\MakeUppercase\contentsname}
230:     }
231:     \@starttoc{toc}
232:     \if@restonecol
233:         \twocolumn
234:     \fi
235:     \cleardoublepage
236:     \pagestyle{fancy}
237:     % Initialize number of page to 1
238:     % Change style of number of page to arabic
239:     \pagenumbering{arabic}
240:     % Reactivation of default header
241:     \defineDefaultHeader{}
242: }
```

Dans ce code vous noterez que nous modifions le style de l'en-tête dans les lignes 205 à 215. Les pages utiliseront une numérotation romaine (i, ii, etc.) et nous afficherons **\contentsname** c'est-à-dire « Table des matières » puisque nous avons configuré le document en français. Le retour au style normal se fait dans les lignes 235 à 241. En ligne 241 justement apparaît une fonction **\defineDefaultHeader**. Il s'agit d'une encapsulation des lignes 147 à 162 de manière à ne pas faire de copier/coller. Je n'ai pas écrit directement cette fonction puisqu'au moment où nous avons défini l'en-tête nous n'en avions pas l'utilité. Les lignes 221 à 225 permettent de modifier l'image à afficher si un paramètre est passé à la commande. Lignes 227 à 230 nous affichons le titre et l'image. Ici nous utilisons **\chapter*** que je n'ai pas défini précédemment mais qui peut être créé très simplement à partir des indications de la section « Les chapitres » (définition de la commande **\makeschapterhead** qui est sensiblement la même que **\makechapterhead**). Enfin en ligne 231 nous affichons le contenu de la table des matières dont vous pouvez voir un aperçu en figure 8.

Fig. 8 : La table des matières.

Si vos documents sont également destinés à une utilisation sur support numérique, vous pouvez ajouter des hyperliens dans la table des matières en chargeant l'extension **hyperref**. Par défaut les liens se trouvent dans d'immondes boîtes rouge, pensez donc à activer l'option **hidelinks** :

```
\usepackage[linktoc=all,hidelinks]{hyperref}
```

Si vous utilisez une version de TeXLive inférieure à 2012 l'option **hidelinks** ne fonctionnera pas. Vous pourrez toutefois cacher les cadres en utilisant l'option **pdfborder={0 0 0}**.

2.7 L'index

Pour afficher l'index on doit intégrer une nouvelle extension. Nous ajouterons ici l'extension **calc** qui nous permettra de réaliser des calculs plus simplement :

```
244: %%%%
245: % Index
246: \usepackage{calc}
246: \usepackage{makeidx}
247: \makeindex
```

L'affichage de l'index proprement dit se fera grâce à une nouvelle commande que nous allons créer :

```
248: \newcommand{\displayIndex}{
249:     \cleardoublepage
250:     \setlength{\columnsep}{0.75cm}
251:     \addcontentsline{toc}{chapter}{\textcolor
{defaultColor}{\indexname}}
252:     \printindex
253: }
```

Il n'y a rien de bien compliqué ici. En ligne 251 nous ajoutons une ligne à la table des matières pour référencer l'index et en ligne 252 nous affichons celui-ci.

Dans le fichier **main.tex**, à chaque fois que vous voudrez insérer une entrée dans l'index il faudra utiliser la commande **\index** :

```
Article \LaTeX pour GNU Linux Magazine\index{GNU Linux Magazine}
```

À la fin du fichier il faudra bien entendu demander à ce que l'index soit affiché :

```
\displayIndex{}
```

La compilation comprendra une étape supplémentaire :

```
$ pdflatex main.tex
$ makeindex main.idx
$ pdflatex main.tex
```

Comme vous pouvez le voir sur la figure 9, nous obtenons un affichage correct de notre page d'index mais le style laisse encore un peu à désirer. Nous pouvons modifier cela en créant un fichier de style pour l'index. Ce fichier, que nous nommerons **styleIndex.ist** permet de spécifier le code LaTeX qui sera inséré avant et après la lettre de titre (**heading_prefix** et **heading_suffix**), de déterminer l'espacement entre un mot et le numéro de page (**delim_0**, **delim_1** et **delim_2**), etc :

```
01: heading_prefix "\\vspace*{0.5cm}\\nopagebreak\n\\tikz\\node
at (0pt,0pt) [rounded corners=5pt,draw=defaultColor,fill=defaul
tColor!10,line width=1pt,inner sep=5pt]{\\parbox{\\linewidth-2\\
fboxsep-2\\fboxrule-2pt}{\\centering\\large\\sffamily\\bfseries\\
textcolor{black}{"
02:
03: heading_suffix "}}};\\vspace*{0.2cm}\\nopagebreak\n"
04:
05: headings_flag 1
06:
07: delim_0 "\\dotfill\ "
08: delim_1 "\\dotfill\ "
09: delim_2 "\\dotfill\ "
```

Prenez garde ici à bien laisser le code de chaque ligne sur une unique ligne et à protéger les antislashes en les doublant. Il faudra préciser lors de la compilation quel est le fichier de style employé :

```
$ makeindex -s styleIndex.ist main.idx
```

Le style de l'index sera légèrement amélioré comme le montre la figure 10.

En utilisant l'extension **lettrine** on peut aussi imaginer un style un peu plus moderne où les lettres de référence se trouveront sous le texte comme le montre la figure 11. Le code permettant de réaliser cela est le suivant (n'oubliez pas de charger l'extension **lettrine** dans votre fichier de classe) :

```
01: heading_prefix "{\\vspace\{1.5cm\}\\bfseries\\lettrine\[lines=4,
lraise=-0.2, findent=-2cm\]\{\\color{defaultColor!40}"
02:
03: heading_suffix "\}\\hfill}\\nopagebreak\n"
04:
05:
06: headings_flag 1
07:
08: delim_0 "\\dotfill\ "
09: delim_1 "\\dotfill\ "
10: delim_2 "\\dotfill\ "
```

2.8 La bibliographie

Nous arrivons au dernier élément essentiel de notre style : la bibliographie. Nous utiliserons ici l'extension **biblatex** et non **biber** bien que plus récente. Ce choix provient simplement du fait que suivant les distributions il n'existe pas de paquetage pour **biber** ou alors il est distribué dans une version incompatible avec la version de perl installée sur la machine. Plutôt que d'installer manuellement cette extension qui se greffe sur **biblatex**, autant utiliser **biblatex** :

```
255: %%%%
256: % Bibliography
257: \usepackage[style=alphabetic,sorting=nyt,sortcites=true,autopunct=true,
258:            babel=hyphen,hyperref=true,abbreviate=false,backref=true]{biblatex}
259: \bibliography{bibliography}
260: \defbibheading{bibempty}{}
```

Nous avons signalé ici que notre fichier de références bibliographiques s'appelle **bibliography.bib** (ligne 259). Pour rappel un fichier de bibliographie à la forme suivante :

```
01: @article{latex,
02:  publisher = {Les éditions Diamond},
03:  author = {Colombo, Tristan},
04:  title = {Un modèle de livre en \LaTeX},
05:  year = {2013},
06: }
```

Comme avec l'index nous allons créer une commande spéciale qui sera d'ailleurs fort semblable :

```
261: \newcommand{\displayBibliography}{
262:     \cleardoublepage
263:     \chapter*{\bibname}
264:     \addcontentsline{toc}{chapter}{\textcolor{defaultColor}{\
bibname}}
265:     \printbibliography[heading=bibempty]
266: }
```

En ajoutant un appel à cette commande dans notre fichier **main.tex** nous obtiendrons l'affichage de la bibliographie... à condition de la compiler ! Ce qui rajoute une étape à la compilation du document :

```
$ pdflatex main.tex
$ makeindex main.idx
$ bibtex main.aux
$ pdflatex main.tex
```

Si vous souhaitez créer votre propre style de références, l'extension **custom-bib** **[6]** vous sera d'un grand secours.

Fig. 9 : Index avec le style par défaut.

Fig. 10 : Index avec un style personnalisé.

Fig. 11 : Index avec un second style personnalisé.

2.9 Un petit plus pour l'insertion d'un listing

Et si nous souhaitons insérer du code dans notre document ? En général on emploie l'environnement **listings** qui est adapté à ce genre d'affichage et numérote automatiquement les lignes. Voici un petit exemple saisi dans le fichier **main.tex** et dont le résultat est visible en figure 12 :

```
\usepackage{listings}
...

\lstset{
    language=Python,
    numbers=left,
    numberstyle=\footnotesize,
    breaklines=true
}
\begin{lstlisting}
#!/usr/bin/python

def maFonction():
    print("Hello World !")

if __name__ == "__main__":
    maFonction()
\end{lstlisting}
```

```
1 #!/usr/bin/python
2
3 def maFonction():
4     print("Hello World !")
5
6 if __name__ == "__main__":
7     maFonction()
```

Fig. 12 : Affichage d'un listing de code à l'aide de l'extension listings.

Ici nous n'avons configuré que peu de valeurs grâce à la commande **\lstset**, de nombreuses autres options sont disponibles. Le résultat est intéressant... mais on peut faire mieux, beaucoup mieux ! Ça ne passera pas par un usage intensif des autres options mais par une autre extension : **minted** qui utilise Pygments pour produire un code où la coloration syntaxique sera activée.

Il va falloir installer cette extension, à commencer par Pygments sur lequel elle s'appuie. Ce dernier est disponible dans les dépôts Debian et s'installe donc très simplement :

```
$ sudo aptitude install python-pygments
```

Occupons nous maintenant de **minted**. Rendez-vous sur la page http://tug.ctan.org/tex-archive/macros/latex/contrib/minted/ et récupérez les fichiers **Makefile**, **minted.dtx** et **minted.ins**. Exécutez ensuite les commandes suivantes :

```
$ make
$ mkdir ~/.texmf/tex/latex/minted
$ mv minted.sty ~/.texmf/tex/latex/minted
```

Dans le code LaTeX il faudra seulement modifier le nom de l'extension et de l'environnement :

```
\usepackage{minted}
...


\begin{minted}[[linenos=true]{python}
#!/usr/bin/python

def maFonction():
    print("Hello World !")

if __name__ == "__main__":
    maFonction()
\end{minted}
```

Lors de la compilation vous devrez obligatoirement ajouter l'option **-shell-escape** :

```
$ pdflatex -shell-escape main.tex
```

Le résultat (voir figure 13) se passe de commentaires... surtout que nous n'avons utilisé que très peu des options (**showspaces** affiche les espaces, etc) qui sont listées dans la documentation **[7]**.

```
1 #!/usr/bin/python
2
3 def maFonction():
4     print("Hello World !")
5
6 if __name__ == "__main__":
7     maFonction()
```

Fig. 13 : Affichage d'un listing de code avec coloration syntaxique à l'aide de l'extension minted.

2.10 Des cadres de mise en valeur

Pour terminer je vous propose de créer une série de cadres de mise en valeur. Ce seront des boîtes dont la couleur de fond mettra en valeur le contenu et qui posséderont un logo indiquant clairement la nature de l'information. Nous développerons un environnement générique qui sera ensuite utilisé pour des remarques, avertissements et exercices. L'extension **mdframed**, spécialisée dans la réalisation de boîtes, nous permettra de réaliser cette tâche simplement. Elle est disponible à l'adresse http://mirrors.ctan.org/macros/latex/contrib/mdframed.zip et son installation est simplifiée :

```
$ wget http://mirrors.ctan.org/macros/latex/
contrib/mdframed.zip
$ unzip mdframed.zip
$ rm mdframed.zip
$ cd mdframed
$ make localinstall
$ cd ..
$ rm -R mdframed
```

Nous associerons les dessins à l'extension TikZ que nous avons déjà utilisée :

```
268: \usepackage[framemethod=TikZ]{mdframed}
% To draw boxes
```

Nous commençons par définir des styles de boîtes à l'intérieur d'une commande dont le second paramètre permettra d'identifier le type de boîte que nous souhaitons. Nous aurons ainsi deux styles : **#2** et **internal#2** qui sera une boîte que nous utiliserons à l'intérieur de la première pour un effet de style :

```
269: %%%%
270: % Generic box definition
271: % [#1] : Color of the box
272: %  #2  : Name of the box
273: \newcommand{\defineBox}[2][defaultColor]{
274:     \newmdenv[
275:         backgroundcolor=#1!10,
276:         innerleftmargin=5pt,
277:         innerrightmargin=5pt,
278:         innertopmargin=5pt,
279:         innerbottommargin=5pt,
280:         leftmargin=0cm,
281:         rightmargin=0cm,
282:         linecolor=#1,
283:         outerlinewidth=.1pt,
284:         roundcorner=4pt,
285:         skipabove=\baselineskip,
286:         skipbelow=\baselineskip]{#2}
287:
288:     \newmdenv[
289:         skipabove=7pt,
290:         skipbelow=7pt,
291:         rightline=false,
292:         leftline=true,
293:         topline=false,
294:         bottomline=false,
```

```
295:        backgroundcolor=#1!10,
296:        innerleftmargin=5pt,
297:        innerrightmargin=5pt,
298:        innertopmargin=5pt,
299:        innerbottommargin=5pt,
300:        leftmargin=0cm,
301:        rightmargin=0cm,
302:        linewidth=4pt,
303:        linecolor=#1,
304:        skipabove=\baselineskip,
305:        skipbelow=\baselineskip]{internal#2}
306: }
```

C'est la commande **\newmdenv** (lignes 274 à 286 et 288 à 305) qui permet de définir le style des boîtes. On détermine ainsi la taille des marges internes et externes (296 à 301), la couleur de fond (ligne 295), l'épaisseur du trait (ligne 302) et sa couleur (ligne 303), quels traits constituant la bordure de la boîte doivent être tracés (lignes 291 à 294), etc... Nous définissons ensuite une nouvelle commande chargée d'appeler **\defineBox** pour définir un nouvel environnement de boîte en lui associant une couleur, un titre, un logo et éventuellement un compteur :

```
308: %%%%
309: % Environment box definition
310: % [#1] : Color of the box
311: %  #2  : Name of the environment
312: %  #3  : Label
313: %  #4  : Logo
314: %  #5  : Enable counter (true or false)
315: \newcommand{\defineEnvBox}[5][defaultColor]{
316:     \defineBox[#1]{box#2}
317:     \ifthenelse{\equal{#5}{true}}{
318:         \newcounter{c#2}
319:         \setcounter{c#2}{1}
320:     }
321:     {
322:     }
323:     \newenvironment{#2}{
324:         \vspace*{.5em}
325:         \begin{box#2}
326:             \vspace*{-1em}
327:             \includegraphics[width=1em]{#4}
328:             ~\\\textbf{#3}
329:             \ifthenelse{\equal{#5}{true}}{
330:                 ~\textbf{\expandafter\csname thec#2\endcsname}
331:                 \refstepcounter{c#2}
332:             }
333:             {}
334:             ~\vspace*{-1em}
335:             \begin{internalbox#2}
336:     }{
337:                 \hfill{
338:                     \color{#1}\tiny\ensuremath{\blacksquare}
339:                 }
340:             \end{internalbox#2}
341:         \end{box#2}
342:     }
343: }
```

Trois éléments sont à noter sur ce code :

- le logo est toujours au format **eps** dans le répertoire **./images** ;
- il faut charger l'extension **amssymb** pour pouvoir utiliser le symbole « carré noir » **\blacksquare** en ligne 338 ;

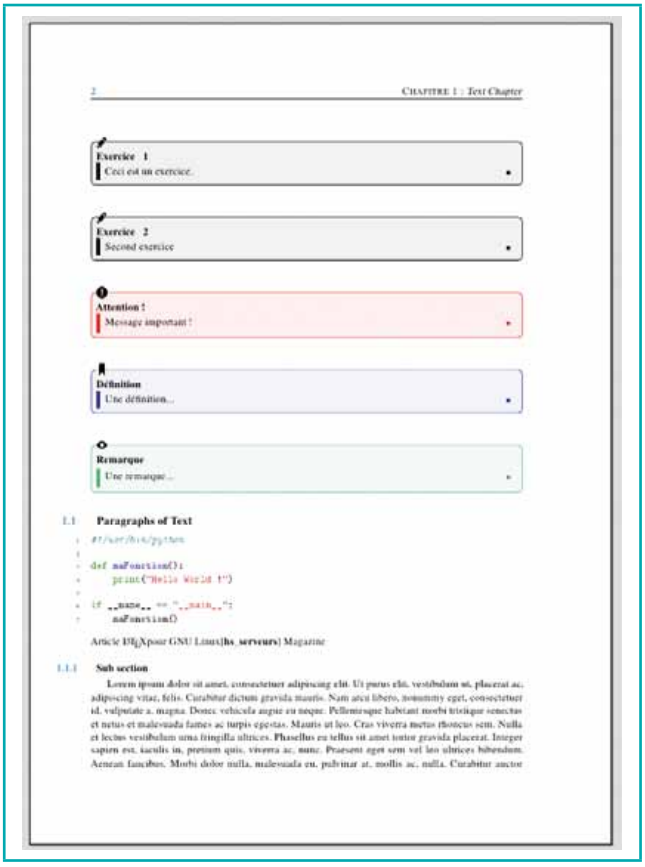


Fig. 14 : Boîtes de mise en valeur.

- Pour pouvoir afficher le compteur que nous définissons en ligne 318 à partir de la valeur contenue dans le deuxième paramètre, et que nous initialisons à **1** en ligne 319, il faut utiliser une syntaxe très particulière en ligne 313. En effet, pour rappel, la valeur numérique d'un compteur **monCompteur** s'obtient par **\valeur{monCompteur}** mais sa valeur littérale pour affichage s'obtient par **\themonCompteur**. Donc si le nom du compteur est défini par un paramètre **#2** il faut que celui-ci soit extrait avant d'exécuter **\the#2**. La solution consiste à indiquer implicitement à LaTeX qu'il doit le faire grâce à la séquence **\expandafter\csname...\endcsname**.

Il est maintenant très simple de créer autant d'environnements que ce que l'on souhaite :

```
345: %%%%
346: % Boxes environments
347: \defineEnvBox[black]{exercise}{Exercice}{exercise}{true}
348: \defineEnvBox[remarkColor]{remark}{Remarque}{remark}{false}
349: \defineEnvBox[definitionColor]{definition}{Définition}
{definition}{false}
350: \defineEnvBox[warningColor]{warning}{Attention !}{warning}
{false}
```

Pour afficher des boîtes dans le code du document LaTeX il faut utiliser les environnements qui ont été créés précédemment :

```
\begin{exercise}
    Ceci est un exercice.
\end{exercise}

\begin{warning}
    Message important!
\end{warning}
```

La figure 14 montre quelques boîtes obtenues à l'aide de cette technique.

Conclusion

Les possibilités avec LaTeX sont infinies. Il est vrai qu'il faut accepter de passer du temps pour développer un modèle un peu sophistiqué mais grâce à lui vous obtiendrez une mise en page impeccable et vous pourrez vous concentrer sur le contenu.

Pour vous aider, de nombreuses ressources sont disponibles sur internet et vous pourrez trouver des bouts de code un petit peu partout sur la toile. Pour cet article j'ai par exemple adapté un modèle de Mathias Legrand sous licence CC BY-NC-SA (http://www.latextemplates.com/template/the-legrand-orange-book). Gardez un œil critique sur ces codes, ces exemples qui sont fournis par des personnes de bonne volonté... bien souvent au mépris de toute logique informatique. Vous trouverez sans aucun doute des réponses aux questions que vous vous posez mais ne réutilisez pas les exemples bêtement et directement, organisez votre code correctement : ce sont des instructions pour mettre en forme des documents mais ce que vous écrivez n'en reste pas moins du code informatique ! ■

Bibliographie

[1] Rolland (Christian), « LaTeX par la pratique », éditions O'Reilly, 1999.

[2] Chevalier (Céline) et al., « LaTeX pour l'impatient », éditions MiniMax, 2009.

[3] Documentation de l'extension microtype : http://ftp.oleane.net/pub/CTAN/macros/latex/contrib/microtype/microtype.pdf

[4] Tisseau (Gérard) et Duma (Jacques), « TikZ pour l'impatient », http://math.et.info.free.fr/TikZ/bdd/TikZ-Impatient.pdf, 2012.

[5] Bezos (Javier), « The titlesec and titletoc Packages », http://www.ctex.org/documents/packages/layout/titlesec.pdf, 2000.

[6] Daly (Patrick), « Customizing Bibliographic Style Files », http://mirrors.ircam.fr/pub/CTAN/macros/latex/contrib/custom-bib/makebst.pdf, 2003.

[7] Rudolph (Konrad), « The minted package : Highlighted source code in LaTeX », http://tug.ctan.org/tex-archive/macros/latex/contrib/minted/minted.pdf, 2011.

LA MAGIE DES FILESYSTEMS : 1- TOUR D'HORIZON

par Étienne Dublé [Ingénieur de Recherche CNRS au LIG]

Le filesystem... Si notre OS préféré a plusieurs modules intéressants, c'est peut-être le plus sympa ! Cette première partie vous propose de le rencontrer.

1 Le filesystem au début de son histoire

Historiquement, un système de fichiers n'était guère plus qu'un simple « gestionnaire de disques ».

Ce module de l'OS a été conçu comme un passage obligé pour tous les processus qui souhaitent écrire ou lire sur le disque. Avec cette position centrale, on pouvait assurer une cohérence entre les zones allouées pour ces différents processus.

On a alors appelé ces « zones du disque » des « fichiers » et donc, ce module est devenu un « système de fichiers », au sens classique du terme.

Pour repérer les fichiers sur le disque, on a aussi introduit les répertoires, ce qui a permis d'obtenir une vision hiérarchique (arborescente) des données.

2 Les filesystems dits « virtuels »

Cette approche ayant eu un succès certain, on a commencé à étendre l'utilisation des filesystems bien au-delà de cette approche historique. À tel point qu'on dit, depuis un certain temps déjà, que « tout est fichier ».

Ce qu'on a gardé de l'approche classique du système de fichiers, c'est la représentation arborescente. Par contre, pourquoi ne pas représenter autre chose que des zones allouées dans un espace de stockage ?

Par exemple, pourquoi ne pas utiliser un filesystem pour représenter les informations sur les processus en cours d'exécution, avec un répertoire par PID ? C'est ce que fait (entre autres fonctionnalités) le filesystem monté sur `/proc`.

Et les périphériques connectés ? C'est ce qu'on monte sur `/dev`.

Et pourquoi ne pas revenir à la notion historique des filesystems et donc, faire de la gestion d'espace, mais en mémoire plutôt que sur disque ? Pour des fichiers temporaires, cela pourrait grandement accélérer les choses ! Eh bien c'est ce que fait `tmpfs`.

Je vais m'arrêter là, mais il est clair que les exemples ne manquent pas.

Ce qu'il faut bien voir, c'est qu'avec chaque nouveau filesystem de ce genre, on étend d'avantage le concept du « tout est fichier ». Et j'espère bien qu'on continuera à l'étendre. Par exemple, sous Linux, les interfaces réseau ne sont pas (encore) vues comme un fichier. Cela serait pourtant utile pour envoyer ou lire des paquets en mode « raw », en utilisant simplement nos outils UNIX classiques [1]... Cela dit, rien n'empêche [2] de coder un filesystem adéquat, qui remplisse cette fonction et d'autres, et qu'on pourrait monter sur `/net`...

3 FUSE : codage en espace utilisateur

En réalité, cette explosion de filesystems en tous genres n'en était qu'à ses débuts. Il était clair que le concept de filesystem pouvait être utilisé à des fins très diverses et la demande était

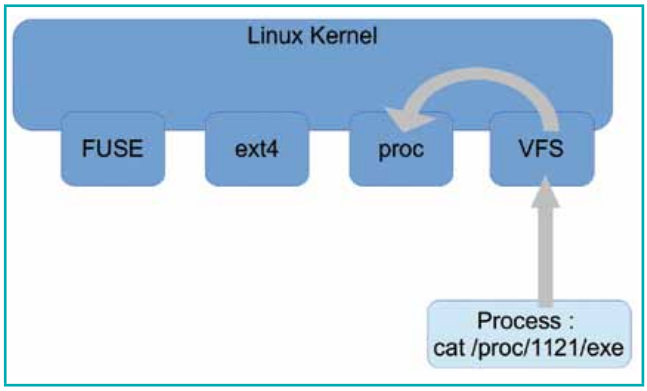


Fig. 1 : Architecture de procfs

[1] La solution actuelle est de coder des programmes ad-hoc avec des interfaces TAP et des bridges, ce qui est relativement laborieux.

[2] Sauf que si je me mets à faire ça et 3 ou 4 autres trucs du même genre, je ne vois plus ma femme...

bien là. Mais l'offre était finalement limitée, car tout le monde n'a pas l'habitude de la programmation noyau.

Jetons un œil sur l'architecture sur laquelle reposent les filesystems.

Le cas le plus simple est sans doute celui d'un filesystem comme **procfs** (Fig. 1).

Sur cette figure, la commande **cat /proc/1121/exe** est exécutée. Sous Linux, le processus d'analyse des chemins est effectué par le module VFS (*Virtual File System*) du noyau. Celui-ci analyse donc le chemin **/proc/1121/exe** à partir de la racine jusqu'au dernier élément. Comme un filesystem de type **procfs** est monté sur **/proc**, il transmet la requête de lecture au module dédié à la gestion de ce filesystem (module **proc** indiqué sur la figure), qui renverra le résultat.

Le cas de **ext4** n'est pas vraiment plus complexe (Fig. 2).

Cette fois, c'est la commande **cat /etc/hosts** qu'on exécute. À nouveau, le chemin est analysé par VFS. Comme un filesystem de type **ext4** est monté sur **/**, il transmet la requête de lecture au module dédié à la gestion de ce filesystem (module **ext4** indiqué sur la figure). Ce module fera alors une lecture sur le disque (sauf si les données sont en cache) et pourra retourner le résultat.

Avec l'apparition du module FUSE dans le noyau, une nouvelle architecture est apparue (voir figure suivante) et avec elle, une nouvelle méthode pour implémenter les systèmes de fichiers (Fig. 3).

Cette fois, la commande que l'on a exécutée est : **cat /tmp/mountpoint/stock**. On considère ici qu'un filesystem « charpentefs », utilisé pour gérer les stocks d'un charpentier, est monté sur **/tmp/mountpoint**. Charpentefs étant un filesystem basé sur FUSE, c'est vers ce module que la requête sera redirigée. Ce module fait alors office de passerelle vers le programme chargé de la gestion de ce filesystem. Ce qui est fondamental ici, c'est que ce programme tourne en espace utilisateur et non dans le noyau...[3]

Développer un filesystem en utilisant FUSE présente ainsi un avantage important : le programmeur n'a pas à écrire du code noyau. Et, s'agissant de code utilisateur, les risques pour la stabilité du système sont moindres, donc le développement peut se faire sans trop de précautions.

En revanche, cette indirection supplémentaire provoque évidemment une baisse de performances. Linus Torvalds a déclaré un jour à ce sujet : « *People who think that userspace filesystems are realistic for anything but toys are just misguided* »[4]. Les avis tranchés de Linus sont bien connus... ;) A mon avis, FUSE est parfait pour un prototype ou même, si on n'a pas besoin de hautes performances, pour un filesystem finalisé. Et même en matière de performances, il faut faire la part des choses. Par exemple, si les données qu'affiche votre filesystem sont stockées sur une autre machine du réseau, alors le temps d'accès à ces données aura sans doute d'avantage d'impact que l'architecture (FUSE ou noyau) que vous choisirez.

4 Les live CD : une arborescence de filesystems

Comme nous allons le voir dans cette partie, les live CD constituent un cas d'utilisation relativement avancée des filesystems.

4.1 Les premiers live CD : ramdisk + filesystem

Le côté magique des filesystems me fascine depuis plusieurs années. Vers 2004, j'ai étudié le fonctionnement de la distribution *Damn Small Linux* et j'ai fini par la modifier pour les besoins de l'entreprise où je travaillais. Ce qui m'intéressait le plus, c'était l'aspect « live ». Comment peut-on, à partir d'un support en lecture seule (CD), booter un système qui nous autorise à créer ou modifier des fichiers ??

En réalité, dans Damn Small Linux, on ne pouvait pas modifier les fichiers, mais on pouvait déjà en créer. En effet, **/home/<user>** était le point de montage

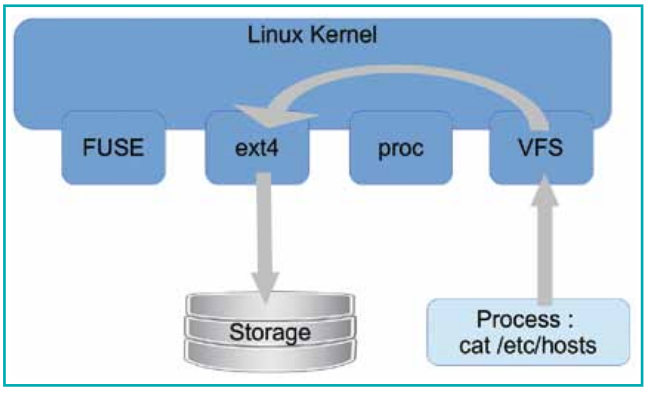


Fig. 2 : Architecture de ext4

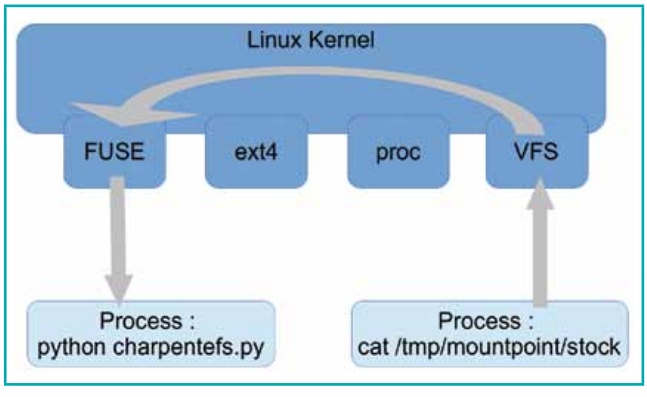


Fig. 3 : Architecture de FUSE

[3] En réalité, il y a bien du code noyau : il s'agit du module FUSE en lui-même. Mais le programmeur de charpentefs n'a pas eu à l'écrire...

[4] En français : « Les gens qui pensent que les filesystems en espace utilisateur sont utiles pour autre chose que des jouets sont mal avisés. ».

d'un filesystem créé dans un « ramdisk » (espace réservé en mémoire vive). Les fichiers créés dans ce répertoire étaient ainsi stockés en mémoire. Mais cette technique était relativement limitée : il était impossible d'installer (ou de mettre à jour) un programme, sauf en l'installant dans ce répertoire, tout le reste de l'arborescence étant en lecture seule.

En fait, il restait bien un moyen pour rendre des fichiers existants modifiables : au démarrage de l'OS, on les lit sur le CD et on les copie sur le ramdisk. Mais ce n'est pas très optimal : avec une quantité de mémoire réduite, on ne peut pas tous les copier de cette manière. Et cette copie prend du temps au démarrage (car lire le CD prend du temps).

4.2 tmpfs : stockage de fichiers en RAM

Aujourd'hui, plutôt que d'empiler un filesystem (de son choix, par exemple **ext2**...) dans un ramdisk, on utilise plutôt **tmpfs**. Il s'agit d'un filesystem qui stocke les fichiers en mémoire et qui permet donc d'obtenir un résultat comparable en une seule opération. Par exemple :

```
$ mount -t tmpfs no_device /tmp
```

À partir de là, ce qu'on fera dans **/tmp** sera fait en mémoire, ce qui permet d'obtenir une réactivité qu'on n'aura jamais avec un disque (même SSD). (Bien sûr, il faut garder à l'esprit qu'on est limité par la taille de nos barrettes et que, si votre fils appuie sur l'interrupteur de la multiprise, vous perdez votre travail).

Dans cet exemple, on voit aussi que le prototype de la commande **mount** nous oblige à spécifier un *device*. Il s'agit d'un héritage de l'époque où les filesystems étaient utilisés uniquement pour les disques. Cela ne s'applique pas à notre cas, j'ai donc mis **no_device**, mais on peut en réalité mettre n'importe quelle chaîne.[5]

4.3 Les systèmes de fichiers « union »

Les live CD ont pris une autre dimension avec l'apparition des systèmes de fichiers « union ». Un de leurs premiers représentants fut **unionfs**, puis **aufs** (les deux ont la même syntaxe). Voici un exemple qui illustre cette notion d'« union » :

```
# ls dir1
subdir
# ls dir1/subdir
file1
# ls dir2
subdir
# ls dir2/subdir
file2
# mkdir dir_union
# mount -t aufs -o dirs=dir1:dir2 no_device dir_union
# ls dir_union
subdir
# ls dir_union/subdir
file1 file2
#
```

Note **Faire tourner MySQL en mémoire...**

Au labo, j'ai par exemple utilisé **tmpfs** pour faire tourner MySQL en mémoire, pour les besoins d'un prototype qui simulait plusieurs clients, tous connectés à la même base MySQL et qui faisaient des requêtes assez lourdes.

Sur une Ubuntu, il suffit de créer un script de démarrage (lancé avant MySQL) qui sauvegarde le contenu de **/usr/share/mysql**, monte ce répertoire avec **tmpfs**, puis restaure son contenu.

(En fait, l'un des moteurs disponibles pour MySQL est conçu pour tourner en mémoire. Mais nous héritions d'un code largement incompatible avec ce moteur.)

Voilà pour le petit exemple amusant. Mais ces systèmes de fichiers en « union » peuvent faire bien plus que ça...

Commençons par remonter l'union en précisant que **dir2** ne doit pas être modifié :

```
# umount dir_union
# mount -t aufs -o dirs=dir1=rw:dir2=ro
no_device dir_union
```

Donc, logiquement, il n'est plus possible de supprimer **file2**, car il est dans l'arborescence de **dir2**, qui est en lecture seule...

```
# rm dir_union/subdir/file2
#
```

Pas d'erreur ??

Vérifions que la suppression a vraiment été effectuée :

```
# ls dir_union/subdir/
file1
#
```

Apparemment oui, donc l'union **file2** n'est plus visible.

Est-ce que ce fichier aurait été supprimé dans l'arborescence de **dir2** malgré l'option de montage **dir2=ro** ??

```
# ls dir2/subdir/
file2
#
```

Non, il est toujours là !

Au cas où, voyons ce qu'on a dans **dir1**, qui lui, pouvait être modifié...

```
# ls -a dir1/subdir/
.  ..  file1  .wh.file2
#
```

Voilà donc toute la subtilité... Comme il n'est pas autorisé à supprimer un fichier dans l'arborescence de **dir2**, **aufs** a créé ce fichier **.wh.file2**[6] dans la branche où il peut écrire, celle de **dir1**.

Au prochain démontage-remontage, **aufs** saura interpréter l'existence de ce fichier « spécial » et donc, **file2** ne sera toujours pas visible dans l'arborescence du point de montage.

[5] Par exemple on peut mettre « stockage_long_terme » pour tromper l'ennemi ;).

[6] Le préfixe « wh » vient de l'anglais « whiteout » (on « rend un fichier invisible »).

Revenons à nos moutons... Et imaginons maintenant les montages suivants :

```
# mount -o ro /dev/cdrom /mnt/cdrom
# mount -t tmpfs no_device /mnt/rw
# mount -t aufs -o dirs=/mnt/rw=rw:/mnt/cdrom=ro no_device /mnt/union
```

S'agissant d'une union, dans **/mnt/union** on voit, au départ, tous les fichiers du cdrom. Si ensuite des fichiers sont créés dans ce point de montage, ils seront écrits dans la branche en lecture-écriture, celle de **tmpfs**, donc en mémoire. Et si on supprime un fichier, cette suppression sera gérée via la création d'un fichier **.wh.<file>** dans l'arborescence de **tmpfs**.

On a donc, en quelque sorte, créé une couche (*overlay*) **tmpfs**, permettant de rendre l'arborescence de notre cdrom modifiable à souhait !

Il ne reste plus qu'à remplir le CD avec une arborescence d'OS et, si on fait booter l'OS de cette manière, on pourra tout modifier... Voilà donc la base des distributions live CD modernes...

Note **Les filesystems en union aujourd'hui**

On trouve en général un des filesystems-union suivants dans les distributions « live » d'aujourd'hui : **aufs** ou **overlayfs**. Cependant, malgré le fait qu'ils soient beaucoup utilisés dans ce contexte, aucun de ces filesystems n'a encore été intégré dans le noyau Linux. Les distributions doivent donc maintenir des patchs noyaux.

Il existe un autre moyen pour implémenter une fonctionnalité d'union : celle d'autoriser le montage de plusieurs filesystems sur le même point de montage. Pour cela, l'utilisateur préciserait une option **--union** lors du montage. Cette approche a également fait l'objet d'une implémentation préliminaire dans le noyau et, à l'époque, a remporté l'adhésion des décisionnaires du noyau Linux.

Malheureusement, des défauts se cachent dans ses diverses implémentations et les développeurs n'ont pas encore réussi à trouver des solutions qui fassent l'unanimité.

4.4 chroot : changeons de référentiel

Je vous raconte des trucs, mais peut-être que vous commencez à vous dire que j'ai tout imaginé depuis le début...

En effet, si je démarre une distribution live CD, disons une Ubuntu, je devrais donc m'attendre à ce que tous les chemins soient préfixés par le point de montage de l'union, non ?

Par exemple, pour lancer bash, il faudrait taper quelque chose comme **/mnt/union/bin/bash**... Et ce n'est pas le cas : je lance juste **/bin/bash**. Il y a donc une autre étape...

Cette autre étape, c'est d'indiquer à **/mnt/union/bin/bash** que la racine de son système de fichiers n'est pas **/** mais plutôt **/mnt/union**. C'est ce que fait la commande **chroot**[7]. En voici la preuve :

```
# cd union/
# pwd
/mnt/union
# chroot . pwd
/
#
```

Au 1er appel de la commande **pwd**, celle-ci nous indique qu'elle est lancée dans le répertoire **/mnt/union**. Au 2ème appel, **chroot** nous permet de lui indiquer que le répertoire courant (donc **/mnt/union**) est la racine de son système de fichiers ; elle nous renvoie donc **/**.[8]

Ce qu'il faut bien comprendre, c'est que, au final, *la racine du système de fichiers n'est qu'un paramètre de chaque processus...*

De plus, ce paramètre est transmis aux processus fils. Donc... il suffit de faire un **chroot** dans le 1er processus du système (**/mnt/union/sbin/init** a priori) pour que **tous** les processus considèrent que **/mnt/union** est leur racine !

Voilà donc pourquoi **/mnt/union** n'est pas visible une fois l'OS lancé.

4.5 squashfs : ajoutons de la compression

En dehors de la gestion du périphérique en lecture seule, les distributions basées sur un live CD doivent prendre en compte l'espace réduit disponible sur un CD.

Et là, vous commencez à deviner la solution : une autre couche de filesystem !

Certains filesystems proposent en effet de compresser les données (**btrfs** par exemple). Le plus simple à implémenter est de compresser le contenu de chaque fichier. Certains sont également capables de compresser les métadonnées (propriétaire, groupe, permissions, etc.).

Le filesystem **squashfs**[9] est capable de compresser encore plus. En effet, plutôt que de compresser chaque fichier indépendamment des autres, on peut

[7] Les OS modernes font aussi usage de pivot_root, qui permet de garder trace de l'ancienne racine.

[8] En fait, je simplifie un tout petit peu : pour être précis, ce n'est pas la même commande pwd qui est exécutée dans les 2 cas. La 1ère fois, c'est /bin/pwd et la 2ème fois, c'est /mnt/union/bin/pwd.

[9] Et cloop (pour « compressed loop ») avant lui...

tirer partie des corrélations entre eux. Mais ce niveau de compression est à peu près ingérable dès qu'on autorise la modification des fichiers... C'est pourquoi **squashfs** est un système de fichiers en lecture seule. Mais, sur un live CD, où est le problème ? Le CD est déjà en lecture seule...

Voilà donc pourquoi, quand on installe une distribution à partir d'un live CD de 700 Mo, on se retrouve avec un filesystem installé qui prend dans les 2 Go ou plus...

4.6 Mise en pratique : modification d'un live CD Ubuntu

Imaginons que vous vouliez déclarer un truc à une fille qui vous a demandé de réparer son PC. Déjà, vous avez prévu de lui installer un bon vieux système GNU/Linux, histoire de la libérer de l'emprise d'une multinationale. Et vous avez prévu le plan diabolique suivant :

- Vous allez modifier l'image ISO de l'OS à l'avance, pour ajouter un fichier **secret.txt** sur le bureau ;
- Vous partirez aux toilettes avant que le bureau ne s'affiche pour qu'elle ait envie d'ouvrir ce fichier.

Allez, c'est parti ! (J'ai une image d'Ubuntu 12.04 (« Precise ») sous la main, mais ce serait à peu près la même chose avec les versions qui suivent...).

Voyons ce qu'on a dans cette ISO :

```
# cd /tmp
# mkdir iso
# mount -o ro [...]/precise-desktop-i386.iso iso
# ls iso/
autorun.inf  dists     md5sum.txt  preseed            wubi.exe
boot         install   pics        README.diskdefines
casper       isolinux  pool        ubuntu
# ls iso/casper/
filesystem.manifest         filesystem.squashfs
filesystem.manifest-remove  initrd.lz
filesystem.size             vmlinuz
#
```

Ce fichier **filesystem.squashfs** a un nom bien explicite : c'est une image de filesystem compressée par **squashfs**. Il nous faut un montage de plus...

```
# mkdir filesystem
# mount iso/casper/filesystem.squashfs filesystem
#
```

Voilà.

Pour notre modification, il y a une chose à savoir. En fait, quand on lance Ubuntu en mode *live*, l'utilisateur de la session *live* est créé dynamiquement (via une commande **adduser** ou un truc du genre). Si vous vous rappelez bien vos cours de système, quand on crée un utilisateur, le contenu de **/etc/skel** (pour *skeleton*) est copié dans le répertoire **/home/<user>** nouvellement créé. Cela permet de mettre en place un environnement minimal automatiquement pour chaque nouvel utilisateur créé. Donc, a priori, si on met notre fichier à l'emplacement **/etc/skel/Desktop/secret.txt**, il devrait se retrouver sur le bureau... Allons-y :

```
# mkdir filesystem/etc/skel/Desktop
mkdir: impossible de créer le répertoire "filesystem/etc/skel/
Desktop": Système de fichiers accessible en lecture seulement
#
```

Ah mince ! Évidemment, on est en lecture seule. A la fois l'ISO et l'image **squashfs** d'ailleurs...

Pour effectuer notre modification, il va falloir adopter une méthode différente. Voici celle qu'on trouve sur Internet :

1) **extraire** le contenu de l'ISO,
2) **extraire** le contenu de l'image **squashfs**,
3) modifier les fichiers comme on veut,
4) recompresser l'image **squashfs**,
5) reformater l'ISO,
6) supprimer les fichiers temporaires.

Vous voyez ce que je vois ?? On traite les images de filesystem comme de vulgaires archives !! Clairement, je ne le permettrai pas. Surtout dans cet article ! ;)

En fait, pour toucher du doigt la puissance des filesystems, on adoptera plutôt cette méthode-là :

1a) **monter** l'ISO,
1b) **monter** une union « union_iso » avec un *overlay* pour pouvoir modifier les fichiers de l'image ISO,
2a) **monter** l'image **squashfs**,
2b) **monter** une union « union_filesystem » avec un *overlay* pour pouvoir modifier les fichiers de l'image **squashfs**,
3) modifier les fichiers comme on veut,
4) recompresser l'image **squashfs** à l'emplacement adéquat de « union_iso »,
5) reformater l'ISO,
6) démonter.

En fait, ce n'est pas vraiment plus compliqué. D'ailleurs, on a toujours 6 étapes ;) Le principe, c'est qu'on a remplacé les 2 phases d'extraction par, à chaque fois, un montage et la mise en place d'une « union » pour gérer les modifications.

Outre le plaisir d'ajouter des niveaux d'abstraction, il y a clairement 2 avantages :

- Les extractions prennent du temps alors que les montages sont quasi-instantanés ;
- Ces deux phases d'extraction demandent beaucoup d'espace disque (temporaire), plusieurs gigas a priori (vu

qu'on extrait tous les fichiers et que ceux-ci ne seront plus compressés). Alors que nos « unions » vont stocker uniquement ce qu'on modifie.

Pour être honnête, si on a pas mal d'espace disque, on aurait presque pu envisager la première méthode. Mais cela reste non optimal, donc désagréable pour l'esprit. :) Et si un jour vous envisagez d'industrialiser vos modifications d'image ISO, pour, par exemple, publier et maintenir une distribution maison (dérivée d'Ubuntu), alors il faudra appliquer ces modifs dans un script. Et là, il n'y a plus qu'un seul choix raisonnable, à mon avis.

Allons-y. En fait, plus haut, on a déjà fait les étapes 1a et 2a (montage ISO et image **squashfs**). On enchaîne donc avec les unions :

```
# mkdir union_iso union_filesystem rw_iso rw_filesystem
# mount -t tmpfs no_device rw_iso
# mount -t tmpfs no_device rw_filesystem
# mount -t aufs -o dirs=rw_iso=rw:iso=ro no_device union_iso
# mount -t aufs -o dirs=rw_filesystem=rw:filesystem=ro no_device
union_filesystem
```

Le montage de **rw_iso** et **rw_filesystem** en mémoire (**tmpfs**) est optionnel. Il faut d'ailleurs se méfier, car si on a beaucoup de modifications à faire, on peut être limité par la quantité de mémoire sur la machine. Ici, on va en particulier mettre à jour le fichier **casper/filesystem.squashfs** dans **union_iso** et ce fichier fait quasiment 700 Mo. Sur une machine limitée en mémoire, il faudrait donc éviter ce montage.

On peut maintenant revenir à la modification que l'on voulait réaliser au départ :

```
# mkdir union_filesystem/etc/skel/Desktop
# cp [...]/secret.txt union_filesystem/etc/skel/Desktop/
```

Et voilà.

Notez qu'à cette étape on peut entreprendre des modifications de bien plus grande envergure, comme par exemple ajouter ou supprimer des paquets. Le principe est grosso-modo le suivant :

```
# [preparation de l'environnement]
# cd union_filesystem
# chroot .     # par défaut le binaire exécuté est /bin/sh
-in_chroot-# apt-get [...]
-in_chroot-# [autres modifs...]
-in_chroot-# exit  # sortie du chroot
# [nettoyage]
```

L'astuce est ainsi d'utiliser **chroot** pour interagir au sein de l'arborescence du filesystem.[10]

Une fois les modifications effectuées, on peut maintenant reformater l'image **squashfs** :

```
$ mksquashfs union_filesystem union_iso/casper/filesystem.squashfs
```

Et reformater l'ISO [11]:

```
$ cd union_iso
$ mkisofs [...tout un tas d'options...] -o [...]/custom.iso .
```

Note **Et les live USB ?**

Un live USB peut en théorie être géré beaucoup plus simplement qu'un live CD, car on n'a pas la problématique du périphérique en lecture seule. Et en réalité, contrairement à un CD, une clé USB a la même structure qu'un disque dur (on peut mettre plusieurs partitions, etc.). Le plus logique serait donc d'installer un OS sur une clé de la même façon qu'on l'installe sur un disque.

Mais, dans les faits, le fonctionnement des live USB est calqué sur celui des live CD. La différence étant que, pour permettre de sauvegarder de manière pérenne les modifications effectuées, le filesystem en *overlay* ne travaille pas en mémoire (comme **tmpfs**), mais plutôt dans un espace réservé de la clé USB (partition dédiée ou fichier dédié).

L'autre avantage du fonctionnement façon live CD vient de la couche de compression, qui est assez optimale, car basée sur un filesystem en lecture seule. Jusqu'à récemment, la taille des clés USB était un peu juste pour installer un OS comme sur un disque. Il est donc possible que le fonctionnement calqué sur les live CD se soit imposé parce qu'il permet une meilleure optimisation de l'espace disponible.

Cependant, l'optimisation d'espace n'est effective qu'au départ. Quand on met à jour des fichiers (essayez un **apt-get upgrade** sur votre live USB...), ces nouvelles versions de fichiers viennent remplir l'*overlay*. On se retrouve ainsi avec la nouvelle version du fichier, non compressée, dans l'*overlay*, mais aussi la version originale, qui est conservée *ad vitam aeternam* dans le filesystem compressé... Ce mécanisme de compression est donc très contre-productif sur le moyen terme.

A mon avis, il est donc possible que l'on revienne à un fonctionnement plus classique dans le futur, calqué sur celui des disques. On y trouvera sans doute toujours un filesystem compressé, mais un de ceux qui supportent les modifications (par exemple **btrfs**).

[10] La phase de « préparation de l'environnement » consiste à pallier le fait que, dans cette arborescence, l'OS n'a pas été démarré. Par exemple, dans cette arborescence, /proc n'est pas monté, il faut donc le monter. On peut trouver ces étapes de préparation sur la page https://help.ubuntu.com/community/LiveCDCustomization. Et le nettoyage est le pendant de cette phase de préparation.

[11] En réalité le formatage correct de l'ISO requiert un peu plus d'opérations (par exemple, il faut mettre à jour un fichier qui contient la taille de filesystem.squashfs) et j'ai omis les options à préciser à mkisofs (pour que l'image soit bootable, etc.), de façon à ne pas trop s'éloigner du thème de l'article. Le lecteur intéressé pourra obtenir ces détails en consultant l'URL donnée plus haut.

La dernière étape consiste à démonter les filesystems créés jusqu'ici, du dernier créé au premier créé (c'est trivial, je ne détaille pas).

Et c'est tout bon, il ne vous reste plus qu'à tenter votre chance avec cette fille et moi à me réjouir de participer à la lutte contre le célibat...

Note **Scripter la création d'un live USB**

Un live CD contient uniquement un filesystem, normalement de type ISO9660. Par contre, un live USB étant comme un disque, il contient des partitions, chacune formatée avec un filesystem (souvent il n'y a qu'une partition de type FAT, mais cela peut être différent). Un live CD est donc analogue à une partition de disque, alors qu'un live USB contient un schéma de partitionnement et une ou plusieurs partition(s).

Si vous copiez un CD-ROM avec la commande **dd**, vous obtiendrez donc une image ISO que vous pourrez monter directement avec la commande **mount**. Par contre, si vous faites de même avec une clé USB, vous obtiendrez une image disque. Vous pourrez ainsi observer son partitionnement avec une commande comme **fdisk <image_cle_usb>** (ou **gdisk** en cas de partitionnement de type GPT).

Scripter la création de systèmes live est donc (légèrement) plus compliqué dans le cas d'un live USB. Pour obtenir un accès aux partitions de l'image USB, je vous invite à consulter la page de manuel de la commande **kpartx** (installable par le paquet du même nom). Et pour que le noyau prenne en compte une modification du partitionnement, vous pourrez utiliser **partprobe** (du paquet **parted**).

5 Boot réseau et hébergement de filesystem

5.1 Contexte

Dans mon labo, on veut mettre en place un banc de test pour les réseaux de capteurs. Dans ce cadre, on a un ensemble de cartes Raspberry Pi qui bootent en réseau. Et sur certains points, cette problématique se rapproche... de celle des live CD[12] !

Les cartes Raspberry Pi disposent d'un lecteur de carte SD. De ce fait, en général, on installe leur OS sur la carte SD. Dans notre cas, afin de faciliter l'administration du système, nous avons préféré réduire au strict minimum le contenu de cette carte (on n'y met que le bootloader, u-Boot), et les faire booter via le réseau, en utilisant un serveur central DHCP, TFTP & NFS. Et de ce fait, nous utilisons la carte SD uniquement en lecture. Nous pouvons donc en espérer une durée de vie bien supérieure.

5.2 Création du filesystem des Raspberry Pi

Sur le serveur, nous avons donc un répertoire **/nfs/debian-fs** qui contient l'arborescence de l'OS utilisé par les Raspberry Pi. Ce filesystem a été créé en utilisant la commande habituelle pour initialiser une arborescence de système Debian :

```
$ debootstrap --arch=armel [..autres options..]
sid /nfs/debian-fs <url_ftp_debian>
```

Dans notre contexte, les clients NFS (les Raspberry Pi) n'ont pas la même architecture que le serveur, il faut donc bien préciser l'option **--arch** adéquate.

Après cette initialisation, nous devons bien évidemment adapter cette arborescence à notre problème, installer les outils nécessaires pour la gestion du banc de test, etc.

Dans ce genre de cas, vous devez commencer à connaître la technique :

```
# [preparation de l'environnement]
# cd /nfs/debian-fs
# chroot .
-in_chroot-# apt-get [...]
-in_chroot-# [autres modifs...]
-in_chroot-# exit
# [nettoyage]
```

Essayons :

```
# cd /nfs/debian-fs
# chroot .
chroot: impossible d'exécuter la commande
" /bin/bash ": Exec format error
#
```

Ah oui, évidemment... L'architecture est différente... Notre serveur sera bien incapable d'exécuter les binaires de cette arborescence (tels que **/nfs/debian-fs/bin/sh** ou **/nfs/debian-fs/usr/bin/apt-get**), s'agissant de binaires pour architecture ARM !

Eh bien en fait si, il en sera capable. Juste après ça :

```
# apt-get install qemu-user-static
[...]
# cp /usr/bin/qemu-arm-static /nfs/debian-fs/
usr/bin/qemu-arm-static
#
```

Et c'est tout ? Eh bien oui. La preuve :

```
# chroot .
-in_chroot-# echo 'je suis dedans !!!'
je suis dedans !!!
-in_chroot-# exit
#
```

[12] Avouez que vous ne vous y attendiez pas !

Voilà qui mérite une petite explication...

Pour faire tourner un binaire d'une architecture différente, il existe bien une solution : utiliser un émulateur, tel que qemu. Il est donc logique d'installer qemu. Et comme on va faire des indirections, à coups de **chroot**, on a choisi la version compilée en statique (sinon c'est plus compliqué à cause des chemins des librairies partagées).

Le paquet nous a ainsi installé le binaire **/usr/bin/qemu-arm-static**. Voilà donc de quoi émuler des binaires ARM. On peut tester avec un binaire ARM téléchargé sur Internet :

```
# file ./hello_world-arm-static
./hello_world-arm-static: ELF 32-bit LSB executable, ARM, version 1,
statically linked, for GNU/Linux 2.6.9, stripped
# qemu-arm-static ./hello_world-arm-static
Hello world!
#
```

Voilà pour la vérification. Mais vous allez me dire que, pour le **chroot**, je n'ai pas utilisé **qemu-arm-static** et ça marchait quand même.

Eh bien en fait, là non plus, je ne suis plus obligé de le préciser :

```
# ./hello_world-arm-static
Hello world!
#
```

Voilà qui vous en bouche un coin... On s'attendait plutôt à un « *Exec format error* » comme tout à l'heure, non ?

L'astuce, c'est que l'installation du paquet **qemu-user-static** a fait plus de choses que ce à quoi on s'attendait : elle a aussi enregistré tout un tas de nouvelles associations dans le système **binfmt_misc** du noyau :

```
# cd /proc/sys/fs/binfmt_misc/
# ls
jar  python3.3  qemu-arm  qemu-cris  qemu-microblaze [...]  register
#
```

Celle qui nous intéresse ici est bien évidemment **qemu-arm** :

```
# cat qemu-arm
enabled
interpreter /usr/bin/qemu-arm-static
flags: OC
offset 0
magic 7f454c4601010100000000000000000002002800
mask ffffffffffffff00fffffffffffffffffeffffff
#
```

Voilà donc la clé de l'énigme : grâce à cette association, quand on demande au noyau d'exécuter un fichier qui correspond aux champs **magic** et **mask**, il va lancer l'interpréteur indiqué (**/usr/bin/qemu-arm-static** ici) et cela de manière totalement transparente pour l'utilisateur !

Puissant ce système **binfmt_misc**, non ?

Note **Plus d'info sur binfmt_misc**

Le système **binfmt_misc** est utilisé dans bien d'autres cas. Par exemple, si j'installe **wine** pour pouvoir lancer un jeu Windows, je verrai apparaître l'association adéquate. Je peux même créer une association à la main en écrivant ce qu'il faut dans le fichier virtuel **register**.

J'ai dit « fichier virtuel » parce que, bien évidemment, l'interface de configuration de **binfmt_misc** est un système de fichiers :

```
# mount | grep binfmt_misc
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc
(rw,noexec,nosuid,nodev)
#
```

Revenons à nos moutons.

Nous avons donc tout ce qu'il faut pour customiser le système que vont booter nos Raspberry Pi :

```
# cd /nfs/debian-fs
# chroot .
-in_chroot-# ls
bin   contiki  etc   lib    mnt  proc  run   srv  tmp  var
boot  dev      home  media  opt  root  sbin  sys  usr
-in_chroot-# apt-get update
0% [Working]qemu: Unsupported syscall: 374
Get:1 http://ftp.fr.debian.org unstable InRelease [198 kB]
Get:2 [...]
[...]
-in_chroot-# apt-get install [...]
[...]
-in_chroot-#
```

L'émulation est assez transparente pour l'utilisateur, sauf concernant 2 choses :

- c'est beaucoup plus lent, évidemment. Si vous voulez recompiler le noyau linux, alors cet environnement peut paraître pratique (pas de cross-compilation à configurer car le gcc de cet environnement va générer nativement des binaires ARM...) ; mais le temps de compilation va sans doute vous décourager !
- on peut rencontrer des erreurs si qemu ne supporte pas certaines fonctionnalités. La plupart du temps, on peut ignorer ces erreurs (comme celle reportée ici), mais il peut arriver que cela soit bloquant, pour certaines tâches bas-niveau.

Dans l'ensemble cela reste très pratique et fonctionnel.

5.3 Gestion du partage NFS en lecture seule

Bon, maintenant qu'on a préparé l'OS qui doit booter par le réseau, il reste un souci à gérer. Si on autorise chaque nœud Raspberry Pi à accéder en lecture-écriture à ce

partage NFS, on va sans doute avoir des soucis, avec toutes les lectures & écritures concurrentes.

Il faut donc configurer le partage NFS en lecture seule. L'ennui, c'est que la distribution Debian que l'on a installée et configurée sur ce partage NFS n'est pas adaptée pour tourner sur un système de fichiers en lecture seule.

La solution, c'est que chaque Raspberry Pi suive le fonctionnement suivant :

1) Récupération (via TFTP) et boot du noyau Linux,
2) Montage du partage NFS (via l'option adéquate du noyau) en lecture seule,
3) Montage d'une union avec un overlay **tmpfs** pour stocker en mémoire les fichiers créés ou modifiés,
4) Lancement de l'OS sur l'union (via **chroot**).

Quand je vous disais que cette problématique se rapproche de celle des live CD !

Les étapes 1 et 2 sont l'affaire du bootloader et du noyau, je ne vais pas les détailler ici. Reste à imaginer comment on va insérer l'étape 3 avant le lancement de l'OS.

En fait, une fois le noyau lancé et le système de fichiers racine monté (ici le partage NFS), le noyau démarre l'OS en exécutant le premier processus, celui qui lance tous les autres. Sauf option noyau spécifique, ce premier processus est lancé en exécutant le fichier **/sbin/init**.

Voilà donc comment on peut insérer cette étape 3 :

```
$ cd /nfs/debian-fs
$ mv sbin/init sbin/init.orig
$ mv [...]/init-customise sbin/init
$
```

Et voilà. Le noyau va maintenant appeler notre **init** customisé à la place de l'ancien. Ce nouvel **init** devra ainsi créer l'union puis lancer **/sbin/init.orig** sur cette union.

Pour l'union, le script va avoir besoin de 2 points de montage. Le filesystem NFS sera en lecture seule, donc on ne pourra pas les créer dans le script ! Donc on le fait maintenant :

```
$ mkdir -p mnt/fs_rw mnt/fs_union
$
```

Et voilà ce script **init** customisé [13] :

```
#!/bin/sh

mount_and_run_startup_init()
{
        cd /mnt

        # on cree l'union
        # /: le montage nfs (lecture seule)
        # /mnt/fs_rw: l'overlay pour stocker
les modifs
        # /mnt/fs_union: l'union
        mount -t tmpfs no_device fs_rw
        mount -t aufs -o dirs=fs_rw=rw:/=ro
no_device fs_union

        # on demarre le script d'init original
        # en lui donnant l'union pour racine
        cd fs_union
        exec chroot . sbin/init.orig
}

if [ $$ = 1 ]
then
        mount_and_run_startup_init
else
        /sbin/init.orig $*
fi
```

La fonction **mount_and_run_startup_init()** ne doit pas trop vous surprendre...

Le **if** à la fin est moins évident. En fait, historiquement, les scripts **init** doivent être capables de fournir 2 fonctionnalités :

F1) Initialiser l'OS, donc faire office de 1er processus et lancer tous les autres ;
F2) Fournir une interface à ce processus d'initialisation.

Vous est-il déjà arrivé de taper la commande **init 0** pour arrêter l'OS ou **init 6** pour rebooter ? Cela correspond à F2. Dans ce cas, la commande **init** doit envoyer un message au 1er processus (celui qui a été créé par F1...), pour lui demander de passer dans le *runlevel* indiqué.

La méthode pour déterminer la fonctionnalité à remplir (F1 ou F2) consiste à vérifier si on est le 1er processus ou non. La variable spéciale **$$** renvoie le PID du processus en cours d'exécution... Ce qui explique ce **if** à la fin du script. Si on est effectivement le 1er processus, on lance l'OS en appelant **mount_and_run_startup_init()**. Sinon, on délègue directement la requête à **/sbin/init.orig**, en lui passant le même argument que celui qu'on a reçu ; **/sbin/init.orig** n'aura pas non plus un PID égal à 1, donc il saura qu'il doit remplir la fonctionnalité F2.

Il y a quand même une subtilité dans **mount_and_run_startup_init()**. Avez-vous remarqué le **exec** avant le **chroot** ? Sans lui, la commande **chroot . sbin/init.orig** serait lancée dans un nouveau processus, donc avec un PID différent de 1. Donc **init.orig** croirait qu'il doit remplir la fonctionnalité F2 au lieu de F1 et c'est le *kernel panic* assuré. Au contraire, en préfixant avec **exec**, le processus en cours d'exécution est « remplacé » par la commande indiquée et de ce fait le PID (égal à 1 dans ce cas) est conservé.

Une fois ce script d'init mis en place, nos Raspberry Pi peuvent cohabiter sur le même partage NFS, en stockant les écritures en mémoire... Mission accomplie !

Pour conclure

J'espère vous avoir convaincu de la puissance des systèmes de fichiers. En particulier, le fait de pouvoir les empiler permet d'obtenir des fonctionnalités très évoluées tout en gardant des briques relativement simples. C'est donc un bon exemple de la philosophie UNIX...

Dans une seconde partie, je vais vous montrer que ces « briques » ne sont pas seulement faciles à utiliser, mais aussi faciles à concevoir... ■

[13] En réalité, pour être plus cohérent avec le reste de l'article, ce script est légèrement différent de celui que nous utilisons. Si un lecteur veut faire booter plusieurs Raspberry Pi sur un partage réseau, qu'il n'hésite pas à me contacter (http://tinyurl.com/eduble) pour que je lui transmette le vrai, avec 2 ou 3 infos utiles...

Complétez votre collection d'anciens numéros !

boutique.ed-diamond.com

numerique.ed-diamond.com

GNU PRIVACY GUARD OU « MAINTENANT ON N'A PLUS LE CHOIX, IL FAUT CHIFFRER ! »

par Denis Bodor

878D 735E A07A AD58 45DD 3F26 A1B2 2ECD 43C4 B3C8

PRISM, StellarWind, Snowden, EvilOlive, XKeyscore, Turmoil, accord Lustre, LPM article 20... Voici autant de noms et de termes qui sont maintenant dans l'esprit de bon nombre de sysadmins/codeurs/utilisateurs et peut-être même de gens ordinaires. Tous ces termes gravitent autour d'une seule et même chose : l'existence avérée d'un système global et international de surveillance numérique et massive. À cela, à défaut d'avoir une solution à la fois politique et technique, il n'y a qu'une seule réponse à cette date : le chiffrement !

Aujourd'hui et plus que jamais, après tous les scandales qui n'auront finalement fait que révéler ce que la plupart n'osaient soupçonner ou clamer, la question consistant à se demander si oui ou non il existe un système de captage, de surveillance, de collecte, d'analyse et de stockage de toutes communications Internet, n'a plus à être posée. Peu importe comment on nomme la chose, qui la contrôle, qui en sont les acteurs, et quels pays s'en partagent les avantages, c'est un fait, elle est bel et bien là, en activité. Les pires craintes se sont matérialisées sous nos yeux, petit à petit, et avec l'approbation, sinon l'insistance, de ceux qui auraient normalement dû nous en protéger.

On pourrait en parler longtemps, éplucher la réalité des événements couche par couche et reconstruire les étapes nous ayant conduits d'un réseau des réseaux idéalement libre à un lieu placé sous surveillance constante, où la notion même de vie privée disparaît de manière aussi constante que certaine. Mais nous choisirons ici une vision plus pragmatique selon un raisonnement simple : ce qui circule est lu ; nous ne voulons pas que cela soit lu ; nous le rendrons donc illisible à ceux à qui cela n'est pas destiné. En d'autres termes, nous allons rendre ce contenu illisible par le moyen le plus évident : le chiffrement.

1 Le chiffrement asymétrique, un peu de théorie (ou presque)

Inutile d'être un mathématicien pour assimiler les bases du chiffrement et les utiliser pour correctement faire usage d'un outil comme GnuPG (*GNU Privacy Guard*). Chiffrer un texte ou un document consiste à le rendre illisible par quiconque ne connaît pas la clé de déchiffrement. Des procédés mathématiques et algorithmiques permettent ces opérations. On distingue principalement deux types de chiffrement : symétrique et asymétrique. Lorsque la même clé est utilisée à la fois pour chiffrer et déchiffrer, le chiffrement est symétrique. Lorsque deux clés différentes entrent en jeu, l'une pour la première opération et l'autre pour la seconde, il s'agit de chiffrement asymétrique ou chiffrement à clé publique. En raison de cette asymétrie, on parle alors de clé privée, qui reste secrète et de clé publique, qui peut être connue de tous. Le principe fondamental de ce type particulier de chiffrement repose sur l'impossibilité de déduire, en un temps et avec des moyens réalistes, la clé privée de la clé publique.

Plutôt que de devoir choisir entre l'une ou l'autre solution, les outils et systèmes modernes utilisent généralement les deux de concert. En effet,

le chiffrement symétrique (DES, Blowfish, AES, etc.) est généralement peu gourmand en ressources, mais le chiffrement asymétrique est plus pratique puisqu'il n'oblige pas à partager la clé secrète unique entre les intervenants par un canal sécurisé avant de communiquer. La solution est donc généralement d'utiliser le chiffrement asymétrique pour échanger la clé servant au chiffrement symétrique effectivement utilisé pour communiquer. C'est ce que fait OpenSSH en utilisant des certificats, afin d'à la fois partager une clé de chiffrement de session et s'assurer de l'identité à chaque extrémité de l'échange.

Le chiffrement asymétrique permet également la signature électronique, puisqu'un message ou un document qui peut être déchiffré avec une clé publique ne peut avoir été chiffré qu'avec la clé privée correspondante et aucune autre. Pour signer un tel document, il suffit donc de l'accompagner d'un hash ou d'une empreinte, qui sera chiffré via la clé secrète de l'émetteur. Si le hash est déchiffré par la clé publique correspondante, c'est qu'il a bien été chiffré par la clé secrète et si le hash du document reçu est identique à celui obtenu après déchiffrement, c'est que le document est intègre. Ceci permet donc également de vérifier l'intégrité d'un document, car s'il est modifié lors du transit, son empreinte sera différente.

Attention cependant, cette signature et vérification d'intégrité ne se rapportent qu'aux clés elles-mêmes. En effet, si Alice envoie à Bernard un message ainsi signé et que Carole fait de même grâce à sa propre clé privée (ou secrète) en jurant d'être Alice, comment Bernard peut-il savoir qui est qui ? Les deux messages pourront être vérifiés sans le moindre problème, respectivement avec la clé publique d'Alice et celle de Carole se faisant passer pour Alice. Techniquement, tout fonctionne. Ceci est important et je me permets d'insister (tout autant que GnuPG le fait), si vous, ou quelqu'un en qui vous avez confiance, ne vous êtes pas assuré que la clé publique d'Alice est la sienne, vous ne pouvez avoir aucune certitude sur l'identité des personnes. Il ne s'agira que de clés, rien de plus. Nous verrons plus loin qu'il existe deux façons de régler ce problème précis et que GnuPG utilise l'une d'entre elles, indépendante d'une autorité supérieure.

2 PGP, GnuPG, versions, etc.

L'un des outils utilisant le chiffrement asymétrique afin de vous permettre de protéger vos mails, vos archives, vos fichiers, ou encore de signer et vérifier l'authenticité d'un document, est *GNU Privacy Guard* également appelé GnuPG ou tantôt GPG. Cet utilitaire respecte le standard OpenPGP (RFC 4880) découlant initialement de la création d'un autre logiciel : *Pretty Good Privacy* ou PGP, créé par Phil Zimmermann en 1991. La création de PGP a valu un certain nombre de problèmes à Zimmermann en raison, à l'époque, d'un certain nombre de lois concernant l'exportation de produits cryptographiques, interdisant en principe sa diffusion mondiale. Le fait que PGP utilise RSA, un algorithme de chiffrement asymétrique breveté, a également causé quelques problèmes avant l'expiration du brevet en 2000.

Comme PGP n'a jamais été et n'est toujours pas véritablement un logiciel libre, la *Free Software Foundation*, via le projet GNU, a donc créé une implémentation basée sur les spécifications OpenPGP : *GNU Privacy Guard*. À l'heure actuelle, les versions récentes de PGP et GnuPG sont interopérables avec les autres implémentations compatibles OpenPGP. Il n'y a donc pas d'inquiétude à avoir sur la possibilité d'échanger des messages signés ou chiffrés avec des utilisateurs se servant de PGP, par exemple.

Notez qu'il existe deux versions majeures de GnuPG, 1.x et 2.x, toutes deux maintenues en parallèle. La version 2.0.0 a été annoncée fin 2006, avec une refonte de l'architecture interne du logiciel davantage axé sur la modularisation des fonctionnalités, ajoutant un certain nombre de fonctions comme le support S/MIME, un démon unifiant l'accès aux *smartcards*, ou encore le fait que **`gpg-agent`** puisse intégralement remplacer le démon **`ssh-agent`**. La version 1.x (actuellement 1.4.16) est plus légère, plus facile à compiler (moins de dépendances), mais se limite aux fonctionnalités liées à OpenPGP, avec un support limité pour les smartcards (avec ou sans *middleware* PC/SC).

Les deux versions de GnuPG peuvent être installées de concert sur une même machine (paquet **`gnupg`** et **`gnupg2`** sous Debian GNU/Linux) et fonctionner sans conflit. La version 1.x est même capable d'utiliser le démon de mise en cache de la phrase de passe de la version 2.x. Dans bien des cas, la version 1.x vous sera suffisante, sauf si vous comptez utiliser une carte à puce (smartcard) récente pour y stocker vos paires de clés, mais nous y reviendrons dans un article spécifique.

3 Le réseau de confiance et la signature de clé

Comme nous l'avons évoqué précédemment, la principale problématique liée au chiffrement asymétrique et à l'utilisation de paires de clés (privée/publique) tient dans l'association avec l'identité d'une personne. La méthode choisie par PGP/GnuPG est celle de « la toile de confiance » ou « du réseau de confiance » (*web of trust* en anglais). Il s'agit d'un système décentralisé de modèle de confiance.

Le fonctionnement est relativement simple. Plutôt que de s'en remettre à une autorité supérieure garante de la correspondance entre un individu et une clé publique, cette vérification est couverte par les utilisateurs eux-mêmes.

Avant toute chose, il est important de relever que, bien qu'on parle de clé privée et de clé publique, il s'agit de bien plus que cela. En effet, la clé publique d'un utilisateur ne se limite pas à cette seule information mathématique, mais est accompagnée d'un certain nombre de métadonnées et d'informations pour former un certificat (exactement comme une certification SSL/TLS). Ces certificats, que nous continuerons cependant à appeler clés publiques, peuvent ainsi être accompagnés de signatures d'autres utilisateurs de PGP/GnuPG.

Ainsi, ce sont les utilisateurs qui servent d'autorité de contrôle et confirment, en signant votre clé publique, que vous êtes bien associé à cette dernière. Pour maximiser ces interactions, des *key signing parties* sont organisées, afin que les utilisateurs puissent renforcer le réseau de confiance. La démarche est généralement simple, puisqu'il suffit de vérifier l'identité d'une personne (via son passeport ou sa carte d'identité) et noter l'empreinte de sa clé publique (courte chaîne de caractères permettant d'identifier la clé), pour ensuite signer la clé publique avec votre clé privée. Vous devenez donc ainsi l'un des garants de l'identité numérique de cette personne.

Bien entendu, il n'est pas possible de rencontrer physiquement toutes les personnes avec qui l'on souhaite correspondre dans le futur de manière authentifiée et éventuellement chiffrée. Il nous faut alors faire confiance, dans une certaine mesure, aux signatures des autres sur les clés publiques des utilisateurs. Ainsi, on peut considérer une clé publique comme digne de confiance (« valide » dans le jargon GnuPG) si :

- on a soi-même procédé à la signature de cette clé et donc, on a nous-même vérifié l'identité de la personne et la correspondance avec l'empreinte de la clé,
- la clé est signée par une personne en qui on a complètement (pleinement) confiance (*fully trusted*) quant à cette vérification,
- la clé est signée par trois personnes à qui on accorde une confiance partielle (*marginally trusted*).

Tout cela à condition que :

- le chemin inverse de signature vers la clé de l'utilisateur en question fasse 5 sauts ou moins.

Pour clarifier ce dernier point, très important, il faut comprendre que des chemins se forment lors des signatures de clés. En effet, si vous signez la clé de votre voisin Bernard, celui-ci peut également signer celle de son amie Claire. Elle-même peut signer celle de son cousin David, et ainsi de suite. Il existe donc un chemin entre votre clé et celle de David.

Les règles permettant de considérer une clé publique comme valide peuvent être changées dans GnuPG. Celles définies ci-avant sont celles par défaut provenant du standard OpenPGP et sont communément utilisées. Cependant, rien ne vous empêche de les changer, selon vos préférences et votre niveau de méfiance (ou de paranoïa). Notez également que tout ce mécanisme n'est valable que si vous-même signez des clés et donc rencontrez des utilisateurs pour vérifier leur identité. En l'absence de signatures, aucun chemin ne sera formé et vous ne pourrez avoir confiance en la validité d'une quelconque clé publique... Oui, il faut un peu sortir de chez soi.

4 Allons-y ! Générer ses paires de clés

Précisons d'entrée de jeu que nous allons parler ici de ce que certains pourraient considérer comme une utilisation avancée de GnuPG. En effet, dans les grandes lignes, les documentations et tutoriels parlent généralement de paires de clés générées par l'utilisateur et utilisées directement. Nous rentrerons davantage dans le détail du fonctionnement de GnuPG en ne nous limitant pas à la simple vision de la fameuse « paire de clés ».

Pour correctement asseoir nos propos, nous allons nous placer dans la peau d'un nouvel utilisateur de GnuPG. De ce fait, nous allons devoir générer une nouvelle paire de clés. Notez cependant que cette génération nécessite la présence d'une grande quantité de nombres aléatoires et donc d'une entropie importante dans le système. On conseille généralement de lancer une commande gourmande en ressources afin de générer l'entropie nécessaire. Il existe toutefois une solution plus simple : installer le paquet **`rng-tools`** (sous Debian) et configurer **`/etc/default/rng-tools`** en intégrant la ligne **`HRNGDEVICE=/dev/urandom`**. Il vous suffira ensuite d'un petit **`sudo /etc/init.d/rng-tools restart`** pour disposer de suffisamment d'entropie en permanence, à la fois pour GnuPG et pour toutes les autres applications ayant besoin d'un grand nombre de valeurs aléatoires en stock.

On lancera ensuite la génération avec :

```
% gpg --gen-key
gpg (GnuPG) 1.4.14; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: répertoire " /home/denis/.gnupg " créé
gpg: nouveau fichier de configuration " /home/denis/.gnupg/gpg.conf " créé
gpg: Attention : les options de " /home/denis/.gnupg/gpg.conf " ne sont pas encore actives cette fois
gpg: le porte-clefs " /home/denis/.gnupg/secring.gpg " a été créé
gpg: le porte-clefs " /home/denis/.gnupg/pubring.gpg " a été créé
```

```
Sélectionnez le type de clef désiré :
   (1) RSA et RSA (par défaut)
   (2) DSA et Elgamal
   (3) DSA (signature seule)
   (4) RSA (signature seule)
Quel est votre choix ?
```

RSA, DSA et Elgamal sont trois algorithmes de chiffrement asymétrique différents. DSA et Elgamal reposent sur un problème mathématique dit de logarithme discret. RSA, pour sa part, repose sur la difficulté de mise en facteurs premiers de grands entiers (le fait de retrouver les deux facteurs premiers d'un produit). Le choix entre l'une ou l'autre option est largement discutable (RSA plus courant, génération DSA plus rapide, vérification de signature RSA plus rapide, signature DSA plus rapide, etc.). De manière générale, l'étape suivante, qui consiste à choisir la taille de clé, est plus importante. On lit souvent que 4096 bits est une valeur conseillée pour RSA et 2048 pour DSA. Nous conserverons ici la valeur proposée par défaut (« RSA et RSA ») et passerons à l'étape suivante :

```
Quel est votre choix ? 1
les clefs RSA peuvent faire entre 1024 et 4096 bits de longueur.
Quelle taille de clef désirez-vous ? (2048) 4096
La taille demandée est 4096 bits
```

Nous choisissons ici la valeur la plus importante disponible, soit 4096. Nous y reviendrons par la suite, mais ce que nous générons ici est une paire de clés que nous appellerons clés maîtresses, qui ne seront utilisées que rarement. Une fois la taille définie, nous passons à l'étape suivante consistant à déterminer la durée de validité de la paire. Comme il s'agit ici de notre paire maîtresse, nous décidons qu'elle sera valable indéfiniment :

```
Veuillez indiquer le temps pendant lequel cette clef devrait être valable.
         0 = la clef n'expire pas
      <n>  = la clef expire dans n jours
      <n>w = la clef expire dans n semaines
      <n>m = la clef expire dans n mois
      <n>y = la clef expire dans n ans
Pendant combien de temps la clef est-elle valable ? (0) 0
La clef n'expire pas du tout
Est-ce correct ? (o/N) o
```

Nous allons maintenant ajouter des métadonnées à notre paire de clés, en spécifiant nos informations personnelles qui sont notre nom et adresse mail :

```
Une identité est nécessaire à la clef ; le programme la construit à partir
du nom réel, d'un commentaire et d'une adresse électronique de cette façon :
   " Heinrich Heine (le poète) <heinrichh@duesseldorf.de> "

Nom réel : Duane Barry
Adresse électronique : duane@petitsgris.org
Commentaire :
Vous avez sélectionné cette identité :
    " Duane Barry <duane@petitsgris.org> "

Faut-il modifier le (N)om, le (C)ommentaire, l'(A)dresse électronique
ou (O)ui/(Q)uitter ? O
```

Notez que cette information n'est qu'une UID ou, en d'autres termes, un identifiant rattaché à la paire de clés. Vous pouvez, par la suite, ajouter d'autres UID afin de spécifier, par exemple, d'autres adresses mail.

L'étape suivante consistera à la génération des clés, mais avant cela, on vous demandera une phrase de passe permettant de protéger la clé secrète. Attention, tout le fonctionnement et la sécurité de ce mécanisme de protection et de signature repose sur le fait que votre clé privée ne tombe JAMAIS entre de mauvaises mains (donc entre les mains de PERSONNE si ce n'est les vôtres). En cas de compromission de votre système ou du support sur lequel se trouve le fichier contenant la clé secrète (**secring.gpg**), le dernier rempart est le chiffrement protégeant ces données et donc, la phrase de passe permettant le déchiffrement. Ce n'est pas simplement important, c'est capital ! Oubliez donc toute forme de phrase de passe intelligible, trop courte, basée sur des informations qui puissent être déduites d'autres mots de passe ou de vos goûts et préférences... Dans la suite de cet article, nous prendrons soin de mettre en sécurité cette clé secrète maîtresse, mais cela ne vous dispense en aucun cas de choisir une phrase de passe robuste, sûre et longue !

```
Une phrase de passe est nécessaire pour protéger votre clef secrète.

De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
.......+++++
...+++++
De nombreux octets aléatoires doivent être générés. Vous devriez faire
autre chose (taper au clavier, déplacer la souris, utiliser les disques)
pendant la génération de nombres premiers ; cela donne au générateur de
nombres aléatoires une meilleure chance d'obtenir suffisamment d'entropie.
...+++++
.....+++++
```

Les symboles « **.** » et « **+** » permettent de suivre la progression de la génération des clés (surtout sur des machines peu puissantes). Nous avons choisi ici de conserver la sortie de la commande intégralement pour deux raisons :

- Comme le précise le message, la génération des clés nécessite de l'entropie dans le système. Plutôt que de lancer des commandes faisant « travailler » le système, nous vous recommandons d'utiliser **rng-tools** comme précisé précédemment. Ceci vous évitera un éventuel message vous signifiant que les clés ne peuvent être générées faute de valeurs aléatoires en quantité suffisante.
- Le message apparaît deux fois... parce que DEUX paires de clés sont générées (cf. plus loin dans l'article).

Au terme de la génération, GnuPG va construire automatiquement une première version de la base de données permettant de déduire la confiance qu'on peut accorder dans les clés publiques à notre disposition :

```
gpg: /home/denis/.gnupg/trustdb.gpg : base de confiance créée
gpg: clef 480318BB marquée de confiance ultime.
les clefs publique et secrète ont été créées et signées.
```

```
gpg: vérification de la base de confiance
gpg: 3 marginale(s) nécessaire(s), 1 complète(s) nécessaire(s),
     modèle de confiance PGP
gpg: profondeur : 0  valables :   1  signées :   0
     confiance : 0 i., 0 n.d., 0 j., 0 m., 0 t., 1 u.
```

Les clés sont identifiées par un identifiant hexadécimal. **480318BB** est notre clé maîtresse et nous lui accordons une confiance ultime (au-delà donc de « complète » et « marginale »), puisqu'il s'agit de notre propre clé. GnuPG rappelle également les règles (le modèle de confiance) utilisées par défaut pour considérer une clé publique comme valide (une clé doit être signée par un utilisateur à qui on accorde une confiance complète, ou trois utilisateurs à qui on accorde une confiance marginale). Pour l'heure, la seule clé publique en notre possession est celle, fraîchement créée, qui nous est rattachée. Elle est donc considérée comme valable (valide), puisqu'on lui accorde une confiance ultime (**1 u** sur la dernière ligne). Le niveau de confiance est précisé avec la nomenclature suivante :

- **i.** : inconnu (non calculable),
- **n.d.** : non décidé, pas d'avis, je sais pas,
- **j.** : aucune, pas confiance,
- **m.** : confiance marginale,
- **t.** : confiance totale/complète,
- **u.** : confiance ultime.

Vous pourrez afficher à nouveau ce résumé en utilisant l'option **--check-trustdb**, ou en mettant à jour la base de confiance avec **--update-trustdb**. Notez qu'il est tantôt plus lisible d'afficher ces informations en désactivant la localisation et donc avec une sortie en anglais :

```
% LANG="" gpg --check-trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 1 signed: 0  trust: 0-, 0q, 0n, 0m, 0f, 1u
```

Enfin, dans les dernières lignes de la sortie affichée en console, nous obtenons un résumé des informations sur ce que nous venons de générer :

```
pub   4096R/480318BB 2014-01-07
Empreinte de la clef = DF1D D553 BE15 5AB9 1E65  71CE 769C 2DDD 4803 18BB
uid                  Duane Barry <duane@petitsgris.org>
sub   4096R/8B374F5D 2014-01-07
```

Le lecteur assidu remarquera que nous n'avons pas une clé, mais deux : **480318BB** et **8B374F5D**. Toutes deux sont en RSA 4096 (**4096/R**), mais **8B374F5D** est une sous-clé de **480318BB** (notée **sub**). L'utilisation de la commande suivante, permettant d'éditer une clé, nous explique pourquoi :

```
% gpg --edit-key  480318BB
gpg (GnuPG) 1.4.14; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

La clef secrète est disponible.

pub  4096R/480318BB  créé : 2014-01-07  expire : jamais      utilisation : SC
                     confiance : ultime       validité : ultime
sub  4096R/8B374F5D  créé : 2014-01-07  expire : jamais      utilisation : E
[  ultime ] (1). Duane Barry <duane@petitsgris.org>

gpg>
```

Le champ **utilisation** précise l'usage qu'il est possible d'avoir de cette (paire) de clés :

- **C** : *certificate* ou signer la clé publique d'un utilisateur,
- **S** : signer un document (mail, fichier, etc.),
- **E** : chiffrer un document,
- **A** : authentification (SSH, TLS, etc.).

GnuPG, lorsque nous utilisons **--gen-key**, ne crée donc pas une paire de clés, mais deux : la première est utilisable pour signer d'autres clés, ainsi que des documents, mais c'est une sous-clé (*subkey* ou clé subordonnée) à part, qui est utilisée pour chiffrer. Pourquoi ? Pour des raisons non seulement de sécurité, mais également de praticité. En effet, tout le fonctionnement du réseau de confiance repose sur la signature des clés publiques des utilisateurs. Ceci est une tâche ponctuelle (à moins de croiser des gens souhaitant voir leur clé signée et signer la vôtre à longueur de journée) et souvent peu amusante. Si votre clé de chiffrement est corrompue parce qu'un attaquant souhaite lire vos messages et documents et réussit dans cet exercice, vous pouvez révoquer cette clé et en créer une nouvelle. Celle-ci sera rattachée à votre clé maîtresse qui n'aura pas été corrompue (peut-être) et vous ne serez pas obligé de re-collecter des signatures sur votre clé publique, détruisant par la même occasion votre réseau de confiance mais aussi, en partie, celui des utilisateurs concernés.

GnuPG fait cela par défaut (avec les options 1 et 2 lors de l'utilisation de **--gen-key** et du choix du type de clé), mais nous pouvons utiliser ce mécanisme pour nous protéger davantage. Notez également que les options 3 et 4 de la génération de clés permettent de créer une seule et unique paire maîtresse, qui ne peut alors être utilisée QUE pour la signature.

Enfin, lors de l'utilisation de **--edit-key**, l'interface interactive de GnuPG nous présente les informations sur les clés publiques. Pour basculer sur les clés privées associées, il suffit d'utiliser la commande :

```
gpg> toggle

sec  4096R/480318BB  créé : 2014-01-07  expire : jamais
ssb  4096R/8B374F5D  créé : 2014-01-07  expire : jamais
(1)  Duane Barry <duane@petitsgris.org>
```

Là, la clé secrète maîtresse est désignée par **sec** et la sous-clé par **ssb**. Une seconde utilisation de **toggle** nous repasse en mode clés publiques.

5 La révocation

Juste après avoir généré ses paires de clés et en particulier la paire maîtresse, il est impératif de faire quelque chose de très important : créer un certificat de révocation.

En chiffrement asymétrique, on ne supprime pas les clés (sauf pour les protéger), mais on les révoque. En d'autres termes, on signale le fait qu'elles ne peuvent plus et ne doivent plus être utilisées. Ceci permet de continuer à déchiffrer les anciens documents et, aux destinataires de nos anciens messages et documents, de pouvoir toujours vérifier nos anciennes signatures.

Dans cette situation, la paire maîtresse nécessite une attention toute particulière. De sa sécurité dépend la validité de toutes les sous-clés. Il vous est possible de créer autant de sous-clés que vous souhaitez. Vous le faites en utilisant la clé secrète maîtresse. Il en va de même pour la révocation d'une sous-clé, la clé secrète maîtresse est indispensable.

En revanche, la paire de clés maîtresse elle, ne dispose pas de clé « supérieure » permettant de la contrôler, elle ne peut être révoquée que par l'utilisation de la clé privée correspondante. De ce fait, si vous perdez votre clé secrète ou êtes incapable de vous souvenir de votre phrase de passe qui la protège, vous êtes dans l'impossibilité de la révoquer et donc, de dire au monde entier qu'il ne faut plus s'en servir. Pour éviter ce genre de situation, vous pouvez préparer le terrain et produire par avance un certificat de révocation. Autrement dit, vous prenez les devants, tant que vous avez la maîtrise de la situation, pour créer un « objet » qui vous permettra de révoquer votre paire maîtresse sans le mot de passe de la clé secrète. Vous programmez une future révocation en quelque sorte, qu'il vous suffira ensuite d'appliquer.

Pour générer ce certificat, nous procédons comme suit :

```
% gpg --output revoke.asc --gen-revoke 480318BB

sec  4096R/480318BB 2014-01-07 Duane Barry <duane@petitsgris.org>

Faut-il créer un certificat de révocation pour cette clef ? (o/N) o
choisissez la cause de la révocation :
  0 = Aucune raison indiquée
  1 = La clef a été compromise
  2 = La clef a été remplacée
  3 = La clef n'est plus utilisée
  Q = Annuler
(Vous devriez sûrement sélectionner 1 ici)
Quelle est votre décision ? 0
Entrez une description facultative, en terminant par une ligne vide :
>
Cause de révocation : Aucune raison indiquée
(Aucune description donnée)
Est-ce d'accord ? (o/N) o

Une phrase de passe est nécessaire pour déverrouiller la clef secrète de
l'utilisateur : " Duane Barry <duane@petitsgris.org> "
clef RSA de 4096 bits, identifiant 480318BB, créée le 2014-01-07

sortie forcée avec armure ASCII.
Certificat de révocation créé.

Veuillez le déplacer sur un support que vous pouvez cacher ; toute personne
accédant à ce certificat peut l'utiliser pour rendre votre clef inutilisable.
Imprimer ce certificat et le stocker ailleurs est une bonne idée, au cas où le
support devienne illisible. Attention quand même : le système d'impression
utilisé pourrait stocker ces données et les rendre accessibles à d'autres.
```

Notez :

- que la phrase de passe est, bien entendu, exigée, puisque la clé secrète est utilisée pour signer le certificat,
- que vous pouvez produire plusieurs certificats selon les futures raisons de la révocation,
- qu'il vous faut prendre pour habitude de confirmer plusieurs fois certaines commandes sensibles avec GnuPG,
- et que le commentaire en fin de sortie est suffisamment explicite pour ne pas avoir à être paraphrasé.

Révoquer une sous-clé ne nécessite pas, en revanche, de certificat de révocation. Vous pouvez le faire directement depuis l'interface interactive de GnuPG avec **`gpg --edit-key`**, suivi de l'identifiant de la clé maîtresse. On utilise alors la commande **`key`** pour sélectionner la sous-clé :

```
gpg> key 1

pub  4096R/480318BB  créé : 2014-01-07  expire : jamais      utilisation : SC
                     confiance : ultime       validité : ultime
sub* 4096R/8B374F5D  créé : 2014-01-07  expire : jamais      utilisation : E
[  ultime ] (1). Duane Barry <duane@petitsgris.org>
```

Notez qu'on commence à compter à partir de 0 (le numéro de la clé maîtresse) et que le caractère **`*`** placé sur la ligne de la sous-clé, après la commande, indique que celle-ci est bien sélectionnée. On utilise ensuite :

```
gpg> revkey
Voulez-vous vraiment révoquer cette sous-clef ? (o/N) o
choisissez la cause de la révocation :
  0 = Aucune raison indiquée
  1 = La clef a été compromise
  2 = La clef a été remplacée
  3 = La clef n'est plus utilisée
  Q = Annuler
Quelle est votre décision ? 0
Entrez une description facultative, en terminant par une ligne vide :
>
Cause de révocation : Aucune raison indiquée
(Aucune description donnée)
Est-ce d'accord ? (o/N) o

Une phrase de passe est nécessaire pour déverrouiller la clef secrète de
l'utilisateur : " Duane Barry <duane@petitsgris.org> "
clef RSA de 4096 bits, identifiant 480318BB, créée le 2014-01-07
```

Le résultat est immédiat, ou presque :

```
pub  4096R/480318BB  créé : 2014-01-07  expire : jamais      utilisation : SC
                     confiance : ultime       validité : ultime
Cette clef a été révoquée le 2014-01-08 par la clef RSA 480318BB Duane Barry
<duane@petitsgris.org>
sub  4096R/8B374F5D  créé : 2014-01-07  révoquée : 2014-01-08  utilisation : E
[  ultime ] (1). Duane Barry <duane@petitsgris.org>
```

La sous-clé **8B374F5D** est révoquée, mais GnuPG vous demandera s'il doit enregistrer les changements dès lors que vous utiliserez **quit** (ou CTRL+D) pour terminer la session interactive.

Remarquez que vous pouvez également révoquer, de cette manière, la clé maîtresse. Ceci aura pour conséquence, et il en va de même pour l'utilisation du certificat de révocation avec **gpg --import revoke.asc** (ATTENTION, il n'y a pas de confirmation demandée pour cette commande), de révoquer la clé maîtresse AINSI QUE TOUTES LES SOUS-CLÉS ! C'est logique, puisque la clé maîtresse permet de manipuler à souhait les sous-clés. Si elle est corrompue, tout le reste s'effondre. Soyez donc très prudent avec ces commandes et faites une copie de votre répertoire **~/.gnupg** avant toute manipulation dangereuse (copie que vous effacerez ensuite de manière sécurisée une fois la manipulation opérée avec succès).

6 Les sous-clés, un élément important trop souvent sous-utilisé

Dans l'état, notre utilisateur Duane dispose d'une paire de clés maîtresse et d'une sous-clé de chiffrement. L'idée est ici de créer une autre sous-clé de signature, afin de ne plus avoir besoin d'utiliser la clé maîtresse pour signer des documents. Attention cependant, la clé maîtresse est toujours celle utilisée pour signer d'autres clés publiques. C'est une tâche qui ne peut être déléguée à une sous-clé. Ceci fait, nous pourrons retirer la clé secrète de la paire de clés maîtresse, après l'avoir mise en sûreté sur un support amovible (à enterrer dans un endroit secret et protégé).

Nous allons donc commencer par éditer notre clé pour ajouter une sous-clé de signature avec **gpg --edit-key 480318BB**. Dans ce mode interactif, nous utilisons ensuite **addkey** comme suit :

```
gpg> addkey
La clef est protégée.

Une phrase de passe est nécessaire pour déverrouiller la clef secrète de
l'utilisateur : " Duane Barry <duane@petitsgris.org> "
clef RSA de 4096 bits, identifiant 480318BB, créée le 2014-01-07

Sélectionnez le type de clef désiré :
   (3) DSA (signature seule)
   (4) RSA (signature seule)
   (5) Elgamal (chiffrement seul)
   (6) RSA (chiffrement seul)
Quel est votre choix ? 4
les clefs RSA peuvent faire entre 1024 et 4096 bits de longueur.
Quelle taille de clef désirez-vous ? (2048)
La taille demandée est 2048 bits
Veuillez indiquer le temps pendant lequel cette clef devrait être valable.
         0 = la clef n'expire pas
      <n>  = la clef expire dans n jours
      <n>w = la clef expire dans n semaines
      <n>m = la clef expire dans n mois
      <n>y = la clef expire dans n ans
Pendant combien de temps la clef est-elle valable ? (0) 6m
La clef expire le mer. 09 juil. 2014 10:55:38 CEST
Est-ce correct ? (o/N) o
Faut-il vraiment la créer ? (o/N) o
```

On nous demande, bien entendu, la phrase de passe de la clé secrète maîtresse, puis le type de clé que nous souhaitons ajouter. Ici, RSA 2048 (par défaut) fera l'affaire. contrairement à la génération de la clé maîtresse, nous choisissons de faire expirer la sous-clé dans un certain temps (**6m** = 6 mois). Après plusieurs confirmations (je vous ai dit qu'il fallait en prendre l'habitude), la clé est générée. Ceci est clairement affiché lors du listage des informations :

```
gpg> list

pub  4096R/480318BB  créé : 2014-01-07  expire : jamais      utilisation : SC
                     confiance : ultime       validité : ultime
sub  4096R/8B374F5D  créé : 2014-01-07  expire : jamais      utilisation : E
sub  2048R/D4EDDEF6  créé : 2014-01-10  expire : 2014-07-09  utilisation : S
[  ultime ] (1). Duane Barry <duane@petitsgris.org>
```

Nous venons de créer **D4EDDEF6**, dédiée à la signature (**S**). Si vous changez d'avis concernant l'expiration de la sous-clé, ce n'est pas un problème. Il vous suffit de la sélectionner (ici avec **key 2**) et d'utiliser la commande **expire**. Le nouveau délai d'expiration vous sera alors demandé, ainsi que la phrase de passe de la clé maîtresse pour appliquer les changements. Vous pouvez d'ailleurs utiliser cette commande pour déterminer un délai d'expiration pour la sous-clé de chiffrement par la même occasion.

À ce stade, nous vous recommandons très fortement de faire une sauvegarde de votre **~/.gnupg**, car nous allons supprimer des éléments et une erreur de manipulation ne pardonnera pas.

Nous allons ensuite exporter les clés dans des fichiers :

```
% gpg --armor --export 480318BB > clepub.asc
% gpg --armor --export-secret-key 480318BB > clesec.asc
% gpg --armor --export-secret-subkeys 480318BB > sclepub.asc
```

Nous exportons respectivement, les clés publiques (**clepub.asc**), la clé secrète de la paire maîtresse (**clesec.asc**) et les clés secrètes des sous-clés (**sclepub.asc**). Notez l'utilisation de **--armor** afin de produire des fichiers ASCII. Ce n'est pas capital, mais cela vous permettra de placer ces éléments sur des supports imprimés. Remarquez que les clés secrètes sont exportées sans demande de phrase de passe, car elles sont exportées « en l'état » et donc, chiffrées.

Voici maintenant l'étape dangereuse, nous allons supprimer les clés secrètes :

```
% gpg --delete-secret-keys 480318BB
[...]
sec  4096R/480318BB 2014-01-07 Duane Barry <duane@petitsgris.org>
Faut-il supprimer cette clef du porte-clefs ? (o/N) o
C'est une clef secrète – faut-il vraiment la supprimer ? (o/N) o
```

Nous pouvons immédiatement constater leur suppression effective avec **gpg --list-secret-keys** (ou **gpg -K**), qui ne nous retourne plus rien. Pour pouvoir utiliser à nouveau notre configuration GnuPG, il nous faut les clés secrètes correspondant aux sous-clés de chiffrement et de signature (**8B374F5D** et **D4EDDEF6**). Il nous suffit alors de réimporter ces dernières :

```
% gpg --import sclepub.asc
gpg: clef 480318BB : clef secrète importée
gpg: clef 480318BB : " Duane Barry <duane@petitsgris.org> " n'est pas modifiée
gpg:       Quantité totale traitée : 1
gpg:                 non modifiées : 1
gpg:           clefs secrètes lues : 1
gpg:      clefs secrètes importées : 1
```

La sortie est un peu déroutante, puisque contrairement à ce qui est affiché, ce n'est pas la clé secrète de la paire **480318BB** qui est importée, mais celle des sous-clés. Si nous affichons à nouveau la liste des clés secrètes à notre disposition, nous constatons :

```
% gpg --list-secret-keys
/home/denis/.gnupg/secring.gpg
------------------------------
sec#  4096R/480318BB 2014-01-07
uid                  Duane Barry <duane@petitsgris.org>
ssb   4096R/8B374F5D 2014-01-07
ssb   2048R/D4EDDEF6 2014-01-10
```

Notez le **#** après le **sec** sur la ligne **4096R/480318BB**. Ceci signale que la clé secrète n'est pas disponible. En revanche, nous avons bien nos deux **ssb**. Vous pouvez dès lors utiliser la configuration pour chiffrer et signer des documents. En éditant la clé, en revanche, nous constatons qu'une partie des fonctionnalités ne sont plus disponibles en raison de l'absence de la clé secrète maîtresse. Exemple, ajouter une sous-clé :

```
gpg> addkey
Les parties secrètes de la clef principale ne sont pas disponibles.
gpg: Échec de génération de la clef : la clef secrète n'est pas disponible
```

Afin d'éviter de devoir manipuler trop souvent les fichiers (import/export) et simplifier les opérations de signature de clés, nous pouvons créer, sur un support amovible, une autre configuration de GnuPG :

```
% gpg --home=/mnt/clefUSB --import clepub.asc clesec.asc
gpg: Attention : les droits du répertoire personnel " /mnt/clefUSB "
     ne sont pas sûrs
gpg: le porte-clefs " /home/denis/securerep/secring.gpg " a été créé
gpg: le porte-clefs " /home/denis/securerep/pubring.gpg " a été créé
gpg: /home/denis/securerep/trustdb.gpg : base de confiance créée
gpg: clef 480318BB : clef publique " Duane Barry <duane@petitsgris.org> " importée
gpg: clef 480318BB : clef secrète importée
gpg: clef 480318BB : " Duane Barry <duane@petitsgris.org> " n'est pas modifiée
gpg:       Quantité totale traitée : 2
gpg:                   importées : 1  (RSA: 1)
gpg:                 non modifiées : 1
gpg:           clefs secrètes lues : 1
gpg:      clefs secrètes importées : 1
```

Notez les différents messages de GnuPG en raison de la présence de l'option **--home** précisant un répertoire de configuration et de travail différent de **~/.gnupg**. Les fichiers sont créés tout comme lors de notre toute première utilisation de la commande **gpg** avec, en plus, un petit avertissement de sécurité. En effet, le répertoire **~/.gnupg** est normalement exécutable, inscriptible et lisible UNIQUEMENT par son propriétaire. Pour nous débarrasser de cet avertissement, qui serait sinon présent à chaque commande impliquant le répertoire **/mnt/clefUSB**, il nous suffit de définir les permissions adéquates : **chmod 700 /mnt/clefUSB**.

Dès lors, nous avons un répertoire **/mnt/clefUSB** qui peut être utilisé comme un répertoire **~/.gnupg**, mais mieux encore, il est possible de l'utiliser AVEC le **~/.gnupg** actuel qui, je le rappelle, n'a pas de clé secrète maîtresse.

Pour ce type d'utilisation, si l'on souhaite modifier sa clé maîtresse (ajouter une sous-clé) ou signer la clé d'un autre utilisateur, il suffit de monter le support USB dans **/mnt/clefUSB** et d'utiliser, par exemple (édition de sa propre clé) :

```
% gpg --home=/mnt/clefUSB --keyring ~/.gnupg/pubring.gpg \
--secret-keyring ~/.gnupg/secring.gpg --trustdb-name \
~/.gnupg/trustdb.gpg --edit-key 480318BB
```

Toutes les opérations nécessitant la clé privée maîtresse deviendront alors possibles. Mais attention toutefois, si vous modifiez votre clé maîtresse, ces changements se feront dans **/mnt/clefUSB**. Ainsi, l'ajout d'une sous-clé devra être répercuté manuellement dans l'installation d'usage courant :

```
% gpg --home=/mnt/clefUSB --armor --export 480318BB > maj.asc

% gpg --import maj.asc
gpg: clef 480318BB : " Duane Barry <duane@petitsgris.org> " 1 nouvelle signature
gpg: clef 480318BB : " Duane Barry <duane@petitsgris.org> " 1 nouvelle sous-clef
gpg:       Quantité totale traitée : 1
gpg:          nouvelles sous-clefs : 1
gpg:          nouvelles signatures : 1
```

Ce n'est pas tout, contrairement à ce qu'on peut lire dans quelques documentations faisant usage des sous-clés ! Cette commande ne mettra à jour que la partie publique des paires de clés. Il faudra faire de même pour les clés secrètes des nouvelles sous-clés. Cependant, il existe un problème de fusion de ces informations. Souvenez-vous du message qui semblait faux précédemment... En réalité, il n'y a pas de mise à jour des sous-clés secrètes et plusieurs techniques sont utilisables. Personnellement, je trouve celle-ci plus efficace : on efface à nouveau toutes les clés secrètes de notre version courante (sans clé secrète maîtresse), on exporte les sous-clés depuis les données GnuPG du support USB, et on importe cela dans la version courante :

```
% gpg --delete-secret-keys 480318BB
[...]
% gpg --home=/mnt/clefUSB --armor \
--export-secret-subkeys 480318BB > ssbmaj.asc

% gpg --import ssbmaj.asc
gpg: clef 480318BB : clef secrète importée
```

```
gpg: clef 480318BB : " Duane Barry <duane@petitsgris.org> " n'est pas modifiée
gpg:      Quantité totale traitée : 1
gpg:                non modifiées : 1
gpg:          clefs secrètes lues : 1
gpg:      clefs secrètes importées : 1
```

Là, si on a créé une multitude de sous-clés, elles sont toutes présentes :

```
% gpg --list-keys Duane Barry
pub   4096R/480318BB 2014-01-07
uid                  Duane Barry <duane@petitsgris.org>
sub   4096R/8B374F5D 2014-01-07 [expire : 2014-05-09]
sub   2048R/221F3016 2014-01-09 [expire : 2014-05-09]
sub   2048g/3B19B168 2014-01-09
sub   2048R/F8B8A882 2014-01-09
sub   1024R/5A1BC2B6 2014-01-09
sub   1024R/BEEDBD1B 2014-01-09

% gpg -K
/home/denis/.gnupg/secring.gpg
------------------------------
sec#  4096R/480318BB 2014-01-07
uid                  Duane Barry <duane@petitsgris.org>
ssb   4096R/8B374F5D 2014-01-07
ssb   2048R/221F3016 2014-01-09
ssb   2048g/3B19B168 2014-01-09
ssb   2048R/F8B8A882 2014-01-09
ssb   1024R/5A1BC2B6 2014-01-09
ssb   1024R/BEEDBD1B 2014-01-09
```

Fort heureusement, la génération de sous-clés n'est pas chose courante et l'ajout d'UID, même si c'est sans doute tout aussi rare, ne nécessite que l'export/import de la clé publique et non le passage par l'inquiétante case « suppression ».

7 Publication des clés

Pour qu'une clé publique puisse être connue, avec toutes les informations qu'elle contient (mention des sous-clés et UID), afin d'être signée par d'autres utilisateurs, utilisée pour vous envoyer des documents/mails chiffrés ou vérifier la signature sur ceux que vous émettez, il faut qu'elle soit accessible publiquement. Pour ce faire, il existe des serveurs de clés publiques avec lesquels GnuPG (et d'autres implémentations d'OpenPGP) est en mesure de communiquer. keyserver.pgp.com, keys.gnupg.net ou encore pgp.mit.edu sont quelques-uns d'entre eux et il est très certainement fait mention dans votre configuration GnuPG (**~/.gnupg/gpg.conf**) d'un serveur par défaut. Sachez également qu'en dehors de quelques exceptions, ces serveurs se synchronisent entre eux régulièrement, afin de maintenir une version la plus à jour possible des informations de tous les utilisateurs.

Obtenir une clé d'un utilisateur pour l'utiliser ou la signer se fera très simplement avec **gpg --recv-keys**, suivi d'un identifiant de clé. Le plus souvent, les utilisateurs, lors des *key signing parties*, communiquent l'empreinte de leur clé :

```
% gpg --fingerprint  Duane Barry
pub   4096R/480318BB 2014-01-07
Empreinte de la clef = DF1D D553 BE15 5AB9 1E65  71CE 769C 2DDD 4803 18BB
uid                  Duane Barry <duane@petitsgris.org>
sub   4096R/8B374F5D 2014-01-07
```

Remarquez que les huit derniers caractères de l'empreinte correspondent à l'identifiant de la clé correspondante.

Lors d'une modification d'une clé publique, qu'il s'agisse de la vôtre (ajout de sous-clés ou d'UID) ou de celle d'un autre utilisateur (signature de clé), vous utiliserez l'option inverse pour republier la clé : **gpg --send-keys** avec l'identifiant concerné.

Vous pouvez également placer votre clé sur un serveur HTTP qui vous est propre, après l'avoir exportée avec **gpg --export --armor** suivi de l'identifiant, pour obtenir une version ASCII/texte (par opposition à une version binaire compressée en l'absence de **--armor**). Vous pouvez également échanger la clé par mail (ou même en version papier), mais cela devient alors rapidement difficile à gérer.

Conclusion : LE problème

Nous arrêterons là cet article déjà bien volumineux. La masse d'explications qui s'y trouvent détaille une utilisation avancée et, je pense, plus sûre, de GnuPG mais c'est également ce qui me pousse à conclure en parlant du principal problème de GnuPG et PGP : ce n'est finalement pas simple.

En effet, si l'on se place en tant qu'utilisateur GNU/Linux, voire de développeur ou de sysadmin ayant déjà un certain nombre d'heures de vol à son crédit, ceci n'est pas grand-chose (une demi-journée tout au plus)... Mais si l'on se place comme utilisateur lambda d'un outil informatique quelconque, c'est un véritable défi. Il faut comprendre le principe du chiffrement asymétrique, celui des signatures, assimiler le fonctionnement et la logique de GnuPG, apprécier (sous la contrainte parfois) l'utilisation de la ligne de commandes, etc.

Certes, il existe des interfaces graphiques comme Seahorse intégrée au bureau GNOME, mais cette couche d'abstraction est-elle bien rassurante ? Personnellement, je n'ai déjà pas confiance en un agent qui « traîne » en mémoire gardant en cache ma phrase de passe, alors que dire d'une multitude de couches entre GnuPG et une fenêtre qui me demande cette même phrase ? Il n'y a pas de GUI véritablement simple et légère permettant un usage simplifié. La plupart des projets sont à l'arrêt ou se sont transformés en monstres, intégrés dans des choses plus monstrueuses encore, aux dépendances tentaculaires...

Comme l'a parfaitement résumé Glenn Greenwald lors de sa *keynote* au 30c3 en décembre dernier à Hambourg, il n'est pas simple pour un utilisateur sans bagage technique de faire usage des bons outils permettant de protéger sa vie privée. Très certainement parce que la technicité est déjà importante mais, en plus, en raison du fait que rien n'est finalement accompli pour simplifier les choses pour les non technophiles... ■

GNUPG : QUELQUES COMMANDES BIEN PRATIQUES

par Denis Bodor

878D 735E A07A AD58 45DD 3F26 A1B2 2ECD 43C4 B3C8

Nous venons de voir comment configurer GnuPG pour disposer d'un maximum de sécurité, aussi bien pour notre propre gestion de clés que pour nos futures communications. Il est maintenant temps d'utiliser réellement GnuPG et découvrir les commandes de base que vous utiliserez au quotidien.

Cette courte liste d'exemples de commandes courantes pour GnuPG se base sur la configuration que nous avons présentée précédemment. Dans ce sens, un certain nombre d'options sont utilisées pour faire référence aux fichiers permettant d'utiliser la clé secrète maîtresse. Il sera laissé à la discrétion du lecteur le soin d'éventuellement adapter ces commandes si la méthode décrite dans le précédent article n'a pas été suivie afin de protéger au mieux la clé secrète.

1 Gérer ses identités

L'une des choses qu'un utilisateur GnuPG est obligé de faire tantôt est d'ajouter une nouvelle « identité » associée à sa clé publique. En d'autres termes, il doit ajouter une ou plusieurs UID, généralement de manière à faire apparaître une nouvelle adresse mail. Une telle modification nécessite un accès à la clé privée maîtresse et donc, l'utilisation des options **`--home=/mnt/clefUSB --keyring ~/.gnupg/pubring.gpg --secret-keyring ~/.gnupg/secring.gpg --trustdb-name ~/.gnupg/trustdb.gp --edit-key`**, après montage du fameux support USB contenant la configuration GnuPG disposant de la clé secrète (cf. article précédent).

On utilisera ensuite la commande **`adduid`** comme suit :

```
gpg> adduid
Nom réel : Duane Barry
Adresse électronique : duane@noyfb.org
Commentaire :
Vous avez sélectionné cette identité :
    " Duane Barry <duane@noyfb.org> "

Faut-il modifier le (N)om, le (C)ommentaire, l'(A)dresse électronique
ou (O)ui/(Q)uitter ? o

Une phrase de passe est nécessaire pour déverrouiller la clef secrète de
l'utilisateur : " Duane Barry <duane@petitsgris.org> "
clef RSA de 4096 bits, identifiant 480318BB, créée le 2014-01-07
```

Ce qui aura pour effet, d'obtenir la clé suivante :

```
pub  4096R/480318BB  créé : 2014-01-07  expire : jamais      utilisation : SC
                     confiance : ultime       validité : ultime
sub  4096R/8B374F5D  créé : 2014-01-07  expire : jamais      utilisation : E
sub  2048R/D4EDDEF6  créé : 2014-01-10  expire : 2014-07-09  utilisation : S
[  ultime ] (1)  Duane Barry <duane@petitsgris.org>
[ inconnue] (2). Duane Barry <duane@noyfb.org>
```

Notez qu'il sera ensuite nécessaire de répercuter cette modification de la clé publique dans l'installation « normale » de GnuPG avec une exportation via **`gpg --home=/mnt/clefUSB --armor --export 480318BB > ~/tmp/maj.asc`**, puis importation avec :

```
% gpg --import ~/tmp/maj.asc
gpg: clef 480318BB : " Duane Barry <duane@noyfb.org> " 1 nouvelle identité
gpg: clef 480318BB : " Duane Barry <duane@noyfb.org> " 1 nouvelle signature
gpg:       Quantité totale traitée : 1
gpg:           nouvelles identités : 1
gpg:         nouvelles signatures : 1
```

Il sera également nécessaire de reconstruire la base de confiance avec **`gpg --update-trustdb`** (cf. le **`inconnue`** à côté de la nouvelle UID créée).

En dehors de ces méta-informations, qui peuvent vous faciliter la vie et surtout celle des personnes qui souhaitent communiquer avec vous, certains utilisateurs apprécient d'intégrer leur photo dans leur clé publique. GnuPG, comme PGP, supporte en effet ce genre de fonctionnalités, qu'on peut allègrement considérer comme bien plus amusantes qu'utiles. Pour intégrer une photo, on éditera la clé comme précédemment et nous utiliserons la commande **`addphoto`**. Rappelons, tant est qu'il soit nécessaire de le faire, que le fait de voir la photo d'un utilisateur et reconnaître ce dernier n'est EN AUCUN CAS un motif légitime pour signer sa clé. On ne signe une clé que lorsqu'on peut réellement vérifier son identité et la correspondance entre lui et l'empreinte de sa clé !

La photo est intégrée dans la clé publique et fera donc augmenter sa taille proportionnellement. GnuPG conseille une image d'une taille de 240×288 pixels, mais si celle-ci fait plus de 6 ko un avertissement sera affiché (vous pourrez passer outre). Il est donc de bon ton d'utiliser un JPEG avec un niveau de compression important (qualité réglée sur 20% ou 30%) et, pourquoi pas, en niveau de gris. Après avoir spécifié le chemin vers le JPEG, l'image sera affichée avec l'un des outils disponibles sur votre plateforme avant que vous puissiez confirmer et enfin, entrer votre phrase de passe pour intégrer l'image dans la clé.

L'intégration de cette image apparaît en listant les clés :

```
% gpg --list-key Duane Barry
pub   4096R/480318BB 2014-01-07
uid                  Duane Barry <duane@noyfb.org>
uid                  Duane Barry <duane@petitsgris.org>
uid                  [jpeg image of size 6048]
sub   4096R/8B374F5D 2014-01-07
sub   2048R/D4EDDEF6 2014-01-10 [expire : 2014-07-09]
```

Vous pouvez très simplement afficher une image présente dans une clé avec **`gpg --list-options show-photo --list-keys Duane Barry`**, par exemple. Notez que les commandes d'affichage comme **`--show-photo`** ne devraient plus être utilisées (*deprecated options*) au bénéfice des « options d'affichage » et des « options de listing ». Cette approche différente permet de considérer clairement l'affichage d'images comme une option dans le cadre d'une action autre. Cette précédente commande liste les clés (**`--list-keys`**) et nous demandons, en plus, l'affichage des photos. Ceci prend tout son intérêt avec **`--verify-options show-photo`** permettant ainsi d'afficher l'image en guise d'option pour la vérification d'une signature sur un document (**`--verify`**, cf. plus loin dans l'article).

L'utilitaire servant à l'affichage est précisé dans votre **`~/.gnupg/gpg.conf`** après la directive **`photo-viewer`**. Par défaut, en l'absence de précision explicite, la version GNU/Linux de GnuPG tentera d'utiliser **`xloadimage`** (X utils), **`eog`** (GNOME) ou **`display`** (ImageMagick).

2 Signer et vérifier une signature

L'un des principaux usages de GnuPG est de prouver l'authenticité d'un document ou d'un fichier grâce à une signature électronique. Pour ce faire, il existe plusieurs manières de procéder, en fonction du document et du résultat qu'on souhaite obtenir. Bien entendu, dans tous les cas, les principes de la cryptographie à clé publique impliquent l'utilisation, pour une telle signature, de la clé privée de l'expéditeur. La clé publique, connue de tous, est utilisée pour vérifier cette signature et la valider.

Pour un fichier texte que nous voulons conserver lisible mais complété de notre signature, la méthode la plus efficace est la signature en clair. Le fichier est alors lu, signé et un nouveau fichier est produit combinant les deux (message et signature) :

```
% cat fichier1
Bonjour, ceci est le fichier 1Bonjour, ceci est le fichier 1

% gpg --clearsign fichier1
Une phrase de passe est nécessaire pour déverrouiller
la clef secrète de l'utilisateur :
" Duane Barry <duane@noyfb.org> "
clef RSA de 2048 bits, identifiant D4EDDEF6, créée
le 2014-01-10 (identifiant de clef principale 480318BB)

% cat fichier1.asc
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Bonjour, ceci est le fichier 1
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.14 (GNU/Linux)

iQEcBAEBAgAGBQJS0QdsAAoJENtLPQ/U7d72cjgH/3hiUV7Y09xWfuuSQCtqh58U
cXl1PDRYtgtVPtf7s2Ndef2DxA+a+h7au8pbT2tEzu/wUxMtgbaQ2zqIle0z/QSk
AZabvG6GS8tkCUMW3FikIQfV6D8oOfGnWdivdqq5hGUvyLLNOS4hQfpRpQilWzUd
VGOHl6J/Tt+y5u2vkJ64gX8EAv06dq8o5nIDZlvbO5RijIUoYXZjzcUD+y7oIoBj
31b7vy0tAtYXvr0/MP/62lCfGArKKnYXFwSMVcQJF93tZIjbbjXcZr6pnztV54T0
ErlL0eA4e6vzSAaVz4sh8BM3AClSZeZRJZu076LfV/3kw/nukPiOPo091nqVRmQ=
=WbhK
-----END PGP SIGNATURE-----
```

Le fichier **`fichier1.asc`** est automatiquement créé contenant le résultat. Il est cependant possible de spécifier le fichier de sortie avec **`--output`**. Pour vérifier cette signature, un utilisateur doit disposer de notre clé publique. Ceci étant, il lui suffira d'utiliser l'option **`--verify`** :

```
% gpg --verify fichier1.asc
gpg: Signature faite le sam. 11 janv. 2014 09:57:16 CET
  avec la clef RSA d'identifiant D4EDDEF6
gpg: Bonne signature de " Duane Barry <duane@noyfb.org> "
gpg:                 alias " Duane Barry <duane@petitsgris.org> "
gpg:                 alias " [jpeg image of size 6048] "
```

Notez qu'à la signature, comme au chiffrement, la sous-clé que nous avons créée à cet effet est utilisée (**`D4EDDEF6`**) et non la clé maîtresse (**`480318BB`**) dont nous n'avons pas besoin pour cette opération.

Il nous est également possible de produire un fichier plus compact et binaire pour cette opération de signature avec :

```
% gpg --sign fichier2
fichier2.gpg

% file fichier2.gpg
fichier2.gpg: data
```

Là, comme précédemment, le contenu du fichier d'origine est fusionné avec la signature, mais le résultat (**`fichier2.gpg`** par défaut) ne sera pas lisible directement (via un **`cat`** par

exemple). C'est la vérification du message qui permettra de récupérer les données d'origine signées :

```
% gpg --decrypt fichier2.gpg
Bonjour, ceci est le fichier 2
gpg: Signature faite le sam. 11 janv. 2014 10:09:00
CET avec la clef RSA d'identifiant D4EDDEF6
gpg: Bonne signature de " Duane Barry <duane@noyfb.org> "
gpg:                 alias " Duane Barry <duane@petitsgris.org> "
gpg:                 alias " [jpeg image of size 6048] "
```

L'utilisation de l'option **--decrypt** n'est pas très intuitive puisque, techniquement, le fichier **fichier2.gpg** n'est pas chiffré. Là encore, on pourra utiliser **--output** pour spécifier un fichier de destination pour les données traitées. Il existe toutefois une solution et une option plus pratique pour les fichiers binaires en particulier, qu'il est peu judicieux de modifier, ainsi :

```
% gpg --detach-sign fichier3

% file fichier3.sig
fichier3.sig: data
```

Ici, le fichier **fichier3.sig** produit ne contient que la signature (par défaut en binaire) et aucune donnée signée. Il faut donc transmettre le fichier original et le fichier de signature au destinataire, qui pourra alors tout vérifier comme précédemment. Le fichier **fichier3.sig** contient le nom du fichier original qui devra être présent, faute de quoi le destinataire obtiendra une erreur :

```
% gpg --verify fichier3.sig
gpg: pas de données signées
gpg: can't hash datafile: erreur d'ouverture de fichier
```

Il est cependant possible de spécifier le fichier des données signées à la vérification :

```
% mv fichier3 toto12
% gpg --verify fichier3.sig toto12
gpg: Signature faite le sam. 11 janv. 2014 10:15:13 CET
  avec la clef RSA d'identifiant D4EDDEF6
gpg: Bonne signature de " Duane Barry <duane@noyfb.org> "
gpg:                 alias " Duane Barry <duane@petitsgris.org> "
gpg:                 alias " [jpeg image of size 6048] "
```

Enfin, si vous préférez obtenir une signature sous la forme d'un fichier ASCII, ajoutez simplement l'option **--armor**. Le fichier de signature sera par défaut suffixé **.asc** et non **.sig**.

Enfin, notez qu'en l'absence de fichier spécifié en ligne de commandes, ce sont l'entrée et la sortie standard qui sont utilisées. De ce fait, en utilisant **gpg --clearsign** seul, il vous est possible de saisir directement du texte qui sera signé. Lorsque vous marquerez la fin de fichier (avec CTRL+D), GnuPG produira la signature en clair sur **STDOUT**. De la même manière, vous pouvez vous servir de **gpg --verify** (ou **cat | gpg --verify** si vous voulez) et copier/coller votre message signé pour, après avoir utilisé CTRL+D, que GnuPG puisse vérifier les données entrées.

3 Chiffrer et déchiffrer

L'autre utilisation principale de GnuPG consiste dans le chiffrement de messages, documents et fichiers. Pour envoyer un élément chiffré à un destinataire, on utilise sa clé publique (ou plus exactement, l'une de ses sous-clés de chiffrement). Chiffrer est encore plus simple que signer. Ainsi, pour envoyer un document chiffré à un destinataire, il nous suffit de faire usage de l'option **--encrypt** :

```
% gpg --encrypt --armor --recipient 480318BB fichier1
% file fichier1.asc
fichier1.asc: PGP message
```

Le fichier obtenu, **fichier1.asc** dans le cas d'une version ASCII (option **--armor**) et **fichier1.pgp** sinon, est chiffré avec la clé publique de **480318BB**. Nous pouvons cependant également utiliser tout ou partie d'une UID du destinataire, comme **duane** par exemple. En l'absence de l'option **--recipient**, GnuPG vous demandera de manière interactive à qui est destiné le fichier. Bien entendu, l'option **--output** nous permet, là encore, de spécifier un fichier de sortie.

Mieux encore, nous pouvons, en une seule opération, chiffrer le document à l'attention de notre destinataire et le signer avec notre clé privée, afin de prouver que nous sommes effectivement l'expéditeur. Il suffit pour cela de combiner la présente commande avec l'une de celles de signature vues précédemment.

À réception, inutile de spécifier **--verify**, la simple utilisation de **--decrypt** affichera les informations sur la signature et, si tout est normal, sa validité.

4 Signer une clé

Enfin, l'une des dernières autres opérations plus ou moins courantes consiste en la signature de clés publiques d'utilisateurs dont on aura vérifié l'identité. La manière dont se passe normalement les choses est la suivante :

- Vous rencontrez quelqu'un qui souhaite faire signer sa clé,
- celui-ci vous transmet l'empreinte de sa clé (souvent sous une forme imprimée sur un petit bout de papier ou une carte de visite),
- et vous prouve (dans la mesure du possible) son identité avec une pièce d'identité,
- une fois en mesure d'accéder à Internet, vous récupérez sa clé publique (via un serveur de clés, ou éventuellement un mail transmis par la personne elle-même),
- vous vous assurez de la correspondance entre l'empreinte transmise et celle de la clé réceptionnée,

- vous signez la clé avec votre clé secrète,
- et enfin, vous rendez cette version à jour disponible via un serveur de clés (ou éventuellement en la retournant à l'utilisateur).

Les huit caractères de l'empreinte d'une clé correspondent à l'identifiant d'une clé. Cependant, par souci de prudence, mieux vaut, lors d'une signature de clé, utiliser l'empreinte complète afin d'éviter les erreurs de manipulation. En guise d'exemple, notre utilisateur fictif Duane Berry a rencontré un certain Alex Krycek lors d'un événement quelconque (« pas de bol » diront certains). Celui-ci lui a transmis l'empreinte suivante : **002DE2B6 5BAA226C ECFFF38C 43929A96 58F828D7**. Nous récupérons donc la clé correspondante, puis affichons l'empreinte correspondante avec :

```
% gpg --fingerprint alex
pub   2048R/58F828D7 2014-01-13
Empreinte de la clef = 002D E2B6 5BAA 226C ECFF  F38C 4392 9A96 58F8 28D7
uid                  Alex Krycek <alex@toungouska.ru>
sub   2048R/46602929 2014-01-13
```

Les données correspondent, c'est parfait. Nous n'avons donc plus qu'à éditer la clé en question après avoir monté notre précieuse clé USB contenant notre clé secrète maîtresse indispensable pour la signature de clé :

```
% gpg --home=/mnt/clefUSB --keyring ~/.gnupg/pubring.gpg
--secret-keyring ~/.gnupg/secring.gpg \
--trustdb-name ~/.gnupg/trustdb.gpg \
--edit-key 002DE2B65BAA226CECFFF38C43929A9658F828D7
```

Il ne nous reste plus qu'à utiliser la commande **sign** en mode interactif pour déclencher la signature de la clé éditée :

```
Voulez-vous vraiment signer cette clef avec votre
clef " Duane Barry <duane@noyfb.org> " (480318BB)
Voulez-vous vraiment signer ? (o/N) o
Une phrase de passe est nécessaire pour déverrouiller la clef secrète de
l'utilisateur : " Duane Barry <duane@noyfb.org> "
clef RSA de 4096 bits, identifiant 480318BB, créée le 2014-01-07
```

Nous en profitons également pour décider du niveau de confiance que nous attribuons à ce Alex Krycek. Méfiants, nous décidons de n'accorder qu'une confiance marginale quant à son sérieux dans la vérification des identités et la signature d'autres clés utilisateurs :

```
gpg> trust
[...]
  1 = je ne sais pas ou n'ai pas d'avis
  2 = je ne fais PAS confiance
  3 = je fais très légèrement confiance
  4 = je fais entièrement confiance
  5 = j'attribue une confiance ultime
  m = retour au menu principal

Quelle est votre décision ? 3
```

Nous terminons par **save** afin d'enregistrer les changements en quittant le mode interactif. Nous pouvons vérifier la bonne marche des choses en listant simplement les signatures sur la clé d'Alex Krycek :

```
% gpg --list-sigs alex
pub   2048R/58F828D7 2014-01-13
uid                  Alex Krycek <alex@toungouska.ru>
sig 3        58F828D7 2014-01-13  Alex Krycek <alex@toungouska.ru>
sig          480318BB 2014-01-13  Duane Barry <duane@noyfb.org>
sub   2048R/46602929 2014-01-13
sig          58F828D7 2014-01-13  Alex Krycek <alex@toungouska.ru>
```

Il nous suffira ensuite de re-publier la clé sur un serveur de clés avec **--send-keys**. Dès lors, tout le monde saura que nous avons signé et donc, en principe, vérifié l'identité de cet utilisateur.

5 Pour finir

Nous venons de faire le tour des commandes courantes de GnuPG et nous terminerons avec un petit commentaire sur la notion de vie privée quant à l'utilisation d'un réseau de confiance. Comme le montre les sorties de commandes dans cet article, la signature d'une clé est datée et découle normalement d'une rencontre physique. Il paraît donc évident qu'il est possible, en collectant simplement les signatures sur une clé ainsi que celles qu'un utilisateur aura faite, d'en déduire les rencontres et éventuellement les déplacements de cet utilisateur. Il existe d'ailleurs des outils permettant, dans une certaine mesure, de faire cela sans trop d'effort. Ainsi, en utilisant les paquets Debian **signing-party** et **graphviz**, il est possible de dessiner votre réseau de confiance avec les relations de signatures entre clés. On peut donc considérer qu'il s'agit là d'une fuite d'informations potentiellement problématique. Vous pouvez cependant vous abstenir de publier les signatures sur un réseau public et fonctionner de la même manière en comité restreint et discret, mais vous perdrez alors proportionnellement tout l'intérêt d'un réseau de confiance public.

Enfin, rappelons que la signature de clés découle d'une vérification d'identité. Il est légitime de se demander dans quelle mesure on peut soi-même se considérer comme qualifié pour une telle démarche. Personnellement, je ne me considère pas comme un expert en la matière et même s'il existe pour les cartes d'identité françaises différentes démarches et vérifications, quid des cartes étrangères ? Seriez-vous capable de reconnaître des faux-papiers présentés par un utilisateur ? Ceci dit, les vérifications faites pour un certificat utilisateur SSL ne sont pas forcément plus efficaces. Au final, comme vous le savez, on ne peut faire confiance à personne... ■

UTILISATION DE LA CARTE GNUPG V2

par Denis Bodor

878D 735E A07A AD58 45DD 3F26 A1B2 2ECD 43C4 B3C8

L'utilisation de GnuPG implique un certain niveau de paranoïa afin de garantir au mieux la sécurité des clés privées. La méthode exposée dans ce numéro améliore grandement la sécurité de la clé privée maîtresse. Si vous considérez que cela n'est pas suffisant, il faudra alors vous tourner vers une solution encore plus sécurisée : la smartcard GnuPG.

Le principal problème d'ordre purement pratique qui se pose lors de l'utilisation de GnuPG se révèle lors de son usage systématique et quotidien. En effet, peu d'entre nous ne disposent que d'une seule et même station de travail. Dans le meilleur des cas on accède, en effet, à une machine professionnelle et une machine personnelle (dans le pire, le nombre dépasse souvent 3 ou 4, voire bien plus dans certains cas). Ainsi, en plus du fait qu'il faille fréquemment mettre à jour les clés publiques en raison des signatures des utilisateurs (mais les serveurs de clés sont là pour cela), on se rend compte que les clés privées sont disséminées un peu partout avec, pour seule protection, votre phrase de passe. Ce n'est guère rassurant...

Bien sûr, il est possible d'imaginer l'utilisation d'une clé USB et de l'option `--home` de manière systématique. Là encore, en dehors des manipulations fastidieuses que cela implique, l'inquiétude de voir ce support amovible perdu ou volé est omniprésent. Faut-il alors chiffrer le support ? Avec quelle méthode ? N'existe-t-il pas une solution plus simple ? Oui, elle existe : utiliser une carte à puce sécurisée pouvant contenir les clés et toutes les informations utiles.

1 La carte

Cette carte appelée OpenPGP Card, actuellement en version 2.0, présente les spécifications suivantes :

- intègre 3 clés RSA/2048 indépendantes (signature, chiffrement, authentification),
- longueur de clés configurable de 1024 à 3072 bits (pour la clé de signature),
- génération des clés directement sur la carte ou importation de clé générée par logiciel,
- compteur de signatures,
- enregistrement d'une URL vers la clé publique complète,
- enregistrement du nom du porteur,
- enregistrement de données de login,
- enregistrement de certificat X.509,
- configuration du code PIN de 6 à 32 caractères,
- utilisation du protocole T=1 compatible avec la plupart des lecteurs,
- version ID-000 disponible (découpe au format carte SIM),
- réinitialisation aux valeurs d'usine possible,
- spécifications techniques disponibles et publiques.

La précédente version de la carte (1.1) était bien plus limitée : pas de stockage de certificat X.509, clés limitées à 1024 bits, pas de *reset* possible, etc. Les deux cartes, ancienne et nouvelle, sont le fruit des développements de la société g10code, une entité commerciale supportant massivement le développement de GnuPG. La carte n'est cependant pas commercialisée par g10code directement, mais par Kernel Concepts (DE) au prix d'environ 17€. Soyons clairs, la boutique en ligne de Kernel Concepts est un peu... déroutante (bon ok, c'est carrément bordélique). Mais malgré des arrondis fantaisistes et quelques problèmes d'affichage de tarifs, il semble s'agir de gens parfaitement sérieux. J'ai acquis deux cartes V2 avec option « découpe SIM » (plus une V1.1 il y a quelques années) et ai été livré sans le moindre problème pour un coût total, port compris, de 38,85€ TTC.

Une autre manière de disposer de cette carte est d'adhérer à la FSFE (*Free Software Foundation Europe*). En tant que

fellow, vous recevrez la « carte à puce de la Fellowship », qui est une OpenPGP Card, en plus de l'adresse mail **@fsfe.org** et du compte IM/Wiki/Blog.

Les deux versions de l'OpenPGP Card de g10code ; en haut la 2.0, en bas l'ancienne 1.1.

2 Le lecteur

Disposer d'une carte à puce est une chose, encore faut-il pouvoir y accéder. Un lecteur de carte à puce USB comme le SCM Microsystems (maintenant Identive Group) SCR335 vous en coûtera entre 30 et 40€ neuf ou via un site d'enchères en ligne. C'est un montant non négligeable par rapport au coût de la carte elle-même, mais cela sera sans doute l'occasion pour vous d'explorer le monde des cartes à puce, puisque ce type de matériel vous donnera accès également aux cartes SIM, CB (EMV) et bien d'autres choses.

Il apparaît également que, si vous êtes client (« usager » devrait-on dire paraît-il) de la SNCF et disposez d'un abonnement TER, il est possible que, dans votre région, vous puissiez avoir accès à une solution de renouvellement de votre abonnement mensuel en ligne. Ainsi, en Alsace, les abonnements TER par carte Alséo peuvent être renouvelés depuis chez soi (service Dom'TER) après achat d'un lecteur à 8€. Il s'avère que ce lecteur économique, fabriqué par Xiring, est basé sur un contrôleur SCR35xx très similaire aux lecteurs de smartcard « classiques ». Celui-ci fonctionnera parfaitement avec votre OpenPGP Card, puisqu'il s'agit d'un lecteur compatible CCID (classe *Integrated Circuit Cards Interface*). A priori, un autre lecteur existe pour la Bretagne (KorriGo), un autre pour la région Rhône-Alpes (OùRA!), un autre pour la Lorraine (SimpliCité), etc. Nous ne les avons, bien entendu, pas tous testés, mais la probabilité que les cartes d'abonnement soient toutes basées sur la même technologie (carte Navigo) et donc les lecteurs intercompatibles, est très grande.

Une dernière solution, plus pratique et à un prix qui reste raisonnable, est d'opter pour la carte OpenPGP en version « découpe SIM » (supplément de 0,60€) et un lecteur adapté comme le Gemalto USB Shell Token v2 (alias IDBridge K30) également vendu chez Kernel Concepts à 18€ (ou, comme moi, être mal réveillé, ne pas voir le produit et acheter un Omnikey 6121USB/CardMan6121 à 20 GPB + le port sur eBay, qui arrivera en retard pour cause d'inondations dans la région d'Oxford). Quoi qu'il en soit, la solution de la clé USB SIM est très certainement la plus pratique en termes d'encombrement et de transport.

En termes de support logiciel, GnuPG est en mesure d'utiliser deux biais pour accéder à la carte via le lecteur :

- PC/SC Lite. Il s'agit d'un *middleware* prenant la forme d'un service **pcscd** fonctionnant sur votre système et unifiant les méthodes d'accès aux smartcards, mais également aux cartes de type RFID/NFC. L'avantage de cette méthode est de disposer d'un support dans tout le système pour un ou plusieurs lecteur(s). Les applications clientes, comme GnuPG, se connectent au middleware pour interroger le support carte sans nécessiter de pilotes spécifiques. Le revers de la médaille est, en revanche, l'impossibilité d'accéder au périphérique en question par un autre moyen (libUSB, libNFC, etc.), puisque le démon PC/SC Lite accapare le matériel. Si l'on est un rien explorateur, il faut ainsi jongler pour ses expérimentations et tantôt stopper le service.
- Support GnuPG. C'est un accès direct au lecteur par GnuPG. Ceci vous dispense de l'installation de middleware, mais nécessite que l'utilisateur exécutant **gpg** ou **gpg2**/**scdaemon** dispose des permissions adéquates sur le périphérique (ajout de règle **udev**). De plus, certains lecteurs, et cela ne se limite pas aux lecteurs « à contact », ne respectent pas complètement les spécifications CCID, ce qui conduit tantôt à certains dysfonctionnements.

Personnellement, bien qu'ayant un service **pcscd** installé sur la plupart

de mes systèmes d'expérimentation, celui-ci est par défaut à l'arrêt. Je préfère largement éviter un maximum d'intermédiaires, afin de plus efficacement tracer la source des bugs. Notez cependant que certains périphériques non compatibles CCID sont uniquement pris en charge par PC/SC Lite. Il en va de même pour certaines applications reposant uniquement sur ce middleware (comme le fantastique CardPeek par exemple).

Enfin, et c'est important de le préciser, l'affirmation selon laquelle l'OpenPGP Card V2.0 ne fonctionne pas avec GnuPG 1.x mais uniquement avec la 2.x est FAUSSE ! On retrouve ce type d'avertissement un peu partout, mais cela n'est plus du tout exact. La carte V2.0 fonctionne parfaitement, aussi bien avec GnuPG 1.4.14 et 2.0.22. Il semblerait que la prise en charge de cette carte, en premier lieu dans la version 2.x puis, plus tardivement, dans la 1.x, soit à l'origine de la confusion. Il ne vous est donc pas absolument nécessaire d'installer GnuPG 2.0.22 pour profiter de la carte et vous pouvez vous contenter de la simplicité de la 1.4.14 (absence d'agent et de **scdaemon**, le démon d'accès smartcard dédié).

3 Mise en œuvre

Trêve de discussion, passons à la pratique. Dès réception de votre carte, la première manipulation permettant de s'assurer du bon fonctionnement de l'installation, avec ou sans PC/SC, consiste à connecter le lecteur USB, selon le modèle y glisser la carte et se plier d'un :

```
% gpg --card-status
Application ID ...: D2760001240102000005000001FF10000
Version ..........: 2.0
Manufacturer .....: ZeitControl
Serial number ....: 00001FF1
Name of cardholder: [non positionné]
Language prefs ...: de
Sex ..............: non indiqué
URL of public key : [non positionné]
Login data .......: [non positionné]
Private DO 1 .....: [non positionné]
Private DO 2 .....: [non positionné]
Signature PIN ....: forcé
Key attributes ...: 2048R 2048R 2048R
Max. PIN lengths .: 32 32 32
PIN retry counter : 3 0 3
Signature counter : 0
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
General key info..: [none]
```

La première étape de configuration sera de personnaliser les informations sur le porteur avec **gpg --card-edit**. Ces mêmes informations vous seront alors présentées et vous arriverez sur le prompt du mode interactif. La personnalisation de la carte consiste en des manipulations dites d'administration et vous devrez, dans un premier temps, passer en mode administrateur via la commande **admin**. Là :

- **sex** vous permettra de choisir entre **M** et **F**,
- **name** pour définir, dans l'ordre, le nom et prénom,
- **lang** pour votre préférence linguistique (**fr** pour nous),
- **url** pour le chemin (URL) vers la clé publique (maîtresse) qui est la vôtre,
- **login** pour définir un identifiant de connexion si vous comptez utiliser la carte pour l'authentification.

Notez qu'à la première commande d'administration, le code PIN d'administrateur vous sera demandé. Vous avez dû recevoir celui-ci, ainsi que le code PIN utilisateur par défaut, sur un petit bout de papier : **12345678** pour l'administrateur et **123456** pour l'autre, pour mes cartes.

Ceci fait, il est temps de passer aux choses sérieuses et de placer des clés sur la carte. Il existe bien des méthodes pour cela :

- Générer les clés sur la carte,
- Générer les clés sur la carte et récupérer une copie de secours,
- Générer les clés ou utiliser celles déjà existantes et les placer sur la carte.

Notez qu'il n'est pas question d'utiliser la carte pour une clé maîtresse. La dépendance avec un matériel et l'infrastructure nécessaire font de cette éventualité une option généralement écartée d'office au bénéfice de l'utilisation de sous-clés. Ensuite, ne pas disposer de copie de secours est généralement une mauvaise idée. Il existe toujours une manière sûre de ne pas « laisser traîner » ses clés secrètes et de les mettre en sécurité, sans qu'il soit nécessaire de se passer d'un backup. La solution abordée ici consiste à produire les sous-clés via GnuPG, les importer sur la carte et faire exactement ce que nous avons fait pour la clé maîtresse. À savoir, éliminer la clé secrète du système.

La carte ne supporte que des clés RSA. Il conviendra donc, dans un premier temps, d'organiser tout cela et de révoquer les sous-clés incompatibles. Ainsi, nous arrivons à la situation suivante :

```
pub  4096R/480318BB  créé : 2014-01-07  expire : jamais      utilisation : SC
                     confiance : ultime       validité : ultime
Cette clef a été révoquée le 2014-01-15 par la clef RSA 480318BB Duane Barry
<duane@noyfb.org>
sub  4096R/8B374F5D  créé : 2014-01-07  révoquée : 2014-01-15  utilisation : E
```

```
sub  2048R/D4EDDEF6  créé : 2014-01-10  expire : jamais      utilisation : S
sub  2048R/FBE026F9  créé : 2014-01-15  expire : jamais      utilisation : E
[  ultime ] (1). Duane Barry <duane@noyfb.org>
[  ultime ] (2)  Duane Barry <duane@petitsgris.org>
[  ultime ] (3)  [jpeg image of size 6048]
```

Nous avons une clé de signature **D4EDDEF6** et une clé de chiffrement **FBE026F9** toutes deux en RSA/2048. La procédure est alors la suivante :

- Basculons sur le mode clés secrètes avec **toggle**,
- repérons nos sous-clés,
- sélectionnons la sous-clé de signature avec **key** suivi du numéro (**key 2** ici),
- utilisons **keytocard** pour transférer la sous-clé sur la carte,
- choisissons l'usage :

```
gpg> keytocard
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]

Veuillez sélectionner l'endroit où stocker la clef :
   (1) Clef de signature
   (3) Clef d'authentification
Quel est votre choix ? 1
```

- entrons la phrase de passe,
- entrons le code PIN administrateur,
- la sous-clé est copiée,
- dé-sélectionnons la sous-clé en réutilisant la même commande **key** que précédemment,
- sélectionnons la seconde sous-clé,
- utilisons à nouveau **keytocard** en choisissant l'emplacement de chiffrement cette fois, avec saisie de la phrase de passe :

```
gpg> keytocard
Signature key ....: 4975 B537 3457 DF0D ACC8  A3A3 DB4B 3D0F D4ED DEF6
Encryption key....: [none]
Authentication key: [none]

Veuillez sélectionner l'endroit où stocker la clef :
   (2) Clef de chiffrement
Quel est votre choix ? 2

Une phrase de passe est nécessaire pour déverrouiller la clef secrète de
l'utilisateur : " Duane Barry <duane@noyfb.org> "
clef RSA de 2048 bits, identifiant FBE026F9, créée le 2014-01-15

gpg: écriture d'une nouvelle clef
```

- GnuPG nous affiche finalement le résultat des opérations :

```
sec  4096R/480318BB  créé : 2014-01-07  expire : jamais
ssb  4096R/8B374F5D  créé : 2014-01-07  expire : jamais
ssb  2048R/D4EDDEF6  créé : 2014-01-10  expire : jamais
                     n° de carte : 0005 00001FF1
ssb* 2048R/FBE026F9  créé : 2014-01-15  expire : jamais
                     n° de carte : 0005 00001FF1
(1)  Duane Barry <duane@petitsgris.org>
(2)  Duane Barry <duane@noyfb.org>
(3)  [jpeg image of size 6048]
```

- quittons avec **save**.

En utilisant **gpg --card-status**, nous constatons que les clés sont bien sur la carte :

```
Signature counter : 0
Signature key ....: 4975 B537 3457 DF0D ACC8  A3A3 DB4B 3D0F D4ED DEF6
      created ....: 2014-01-10 08:55:38
Encryption key....: C4F2 2407 DC46 EC71 9211  977C A280 AC2C FBE0 26F9
      created ....: 2014-01-15 18:47:53
Authentication key: [none]
```

Il ne vous reste plus, à ce stade, qu'à faire le ménage comme précédemment (cf. premier article sur le sujet) et mettre la clé secrète en sécurité.

4 Pour finir

Le principal avantage de l'OpenPGP Card tient dans l'utilisation qui devient générique. Ainsi, si toutes ces manipulations ont été faites sur votre poste de travail principal à votre domicile, et que vous avez publié votre clé publique sur un serveur, tout ce que vous avez à avoir sur vous est votre carte (et éventuellement, un lecteur). Rendez-vous sur votre lieu de travail, récupérez votre clé publique avec **gpg --recv-key**, branchez votre lecteur et insérez votre carte. Même si vous n'avez jamais utilisé GnuPG sur cette machine auparavant, vous voici en mesure de signer des documents et de déchiffrer ceux qui vous sont destinés. Mieux encore, vous n'êtes pas même obligé d'avoir accès à un serveur de clés publiques. En renseignant l'URL vers la clé publique dans la carte, vous pouvez utiliser la commande **fetch** après un **gpg --edit-card** et celle-ci sera téléchargée.

Nous n'en avons pas fait mention dans cet article tant c'est logique, mais n'oubliez pas de changer le code PIN utilisateur et administrateur de votre carte (**gpg --change-pin**). À ce propos, chose qui peut être importante selon votre environnement de travail, si vous disposez d'un lecteur qui ne possède pas de pavé numérique pour la saisie des codes PIN, ceux-ci transiteront EN CLAIR, vers le lecteur. Il y a donc potentiellement un risque de récupération de l'information au niveau du support USB du noyau et/ou du matériel. Quoi qu'il en soit, changer régulièrement les codes PIN est une bonne habitude, comme celle de ne pas laisser sa carte traîner dans le lecteur après usage, ou encore d'explicitement menacer l'intégrité physique de toute personne qui s'approche de la carte... ■

GITLAB : GÉREZ VOS PROJETS OPEN SOURCE À GRANDE ÉCHELLE !

par Benjamin Zores [Architecte Logiciel @ Alcatel-Lucent]

Force est de constater que Git s'est aujourd'hui imposé de facto comme un standard pour la gestion collaborative du code source de nombreux projets open source. Des plateformes d'hébergement gratuites de projets comme GitHub (cf. [1]) ou Gitorious (cf. [2]) font désormais office de références pour qui cherche à créer son nouveau projet. Mais si vous êtes simplement adepte du DIY (ou « Do It Yourself »), ou ne faites pas confiance à ces plateformes publiques et souhaitez cependant en garder les fonctionnalités, le projet GitLab est fait pour vous.

Pour faire simple, on peut résumer le projet GitLab à une implémentation open source d'une plateforme telle que GitHub. GitLab est un projet de gestion de code collaboratif. Comme son grand frère propriétaire, le projet permet de créer et d'héberger des projets (et leurs dépôts Git associés), de créer et ajouter des contributeurs à vos projets, d'y gérer les droits, de gérer vos bogues, de créer un Wiki, ou encore de mettre en place des revues de code. Particulièrement utile pour les entreprises, GitLab vous permet de sécuriser votre code en hébergeant la plateforme sur vos propres serveurs, plutôt que sur une plateforme d'hébergement publique. Si vous êtes amené à faire du développement « privé » ou propriétaire, cela peut prendre tout son sens. GitLab est disponible sous licence MIT et propulsé par Ruby on Rails. Vous trouverez un aperçu de la plateforme sur le site officiel du projet (cf. **[3]**), ainsi que sur la figure 1.

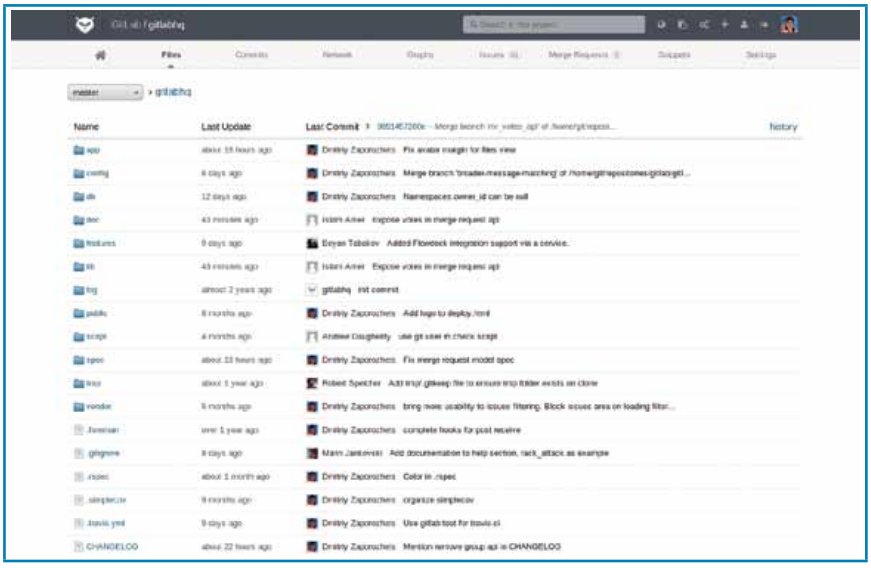


Figure 1 : Interface web de GitLab

1 Prérequis

GitLab s'installe très facilement sur les distributions Debian/Ubuntu et ne nécessite que très peu de dépendances : `git` (cela semble évident), Ruby (à partir de la version 1.9.3), Redis et un serveur de base de données (MySQL ou PostgreSQL). La configuration serveur nécessaire reste modeste, mais dépend très fortement de la charge attendue. En termes de CPU, la documentation officielle recommande une configuration classique de 1 à 2 cœurs jusqu'à 100 utilisateurs, 4 permettant d'en gérer 1000 et 8 pour aller jusqu'à 10000 utilisateurs.

Relativement à la mémoire, 1 à 4 Go de RAM seulement vous permettront de

supporter les seuils précédemment énoncés. L'espace disque constituera le nerf de la guerre. En effet, plus vous souhaiterez héberger de dépôts Git, plus vous nécessiterez d'espace disque (logique). Notez qu'il vous faudra au moins deux fois l'espace nécessaire à l'ensemble de vos dépôts Git. En effet, GitLab dispose de « satellites » qui constituent des environnements temporaires, copies des espaces de travail (*working trees*), nécessaires aux requêtes de fusion (*merge requests*), éditions de fichiers, etc.

En outre, mais cela est propre à chacun, je ne saurai que trop vous recommander de disposer également d'un espace de sauvegarde de vos dépôts Git sur un serveur ou disque distinct. Nous y reviendrons en fin d'article. L'espace disque étant probablement amené à croître au fur et à mesure de la vie de vos projets, je ne saurai que trop vous recommander d'utiliser une approche de type LVM ou, si vous en disposez, un montage externe par NFS/CIFS sur un NAS, un SAN, ou encore un volume de type Cinder ou EBS si vous hébergez votre GitLab sur un environnement de type OpenStack ou Amazon Web Services.

2 Déploiement

À ce jour, la dernière version stable de GitLab est la 6.3. C'est donc sur cette dernière que se basera ce tutoriel. Notre installation se fera donc sur une base Ubuntu Server LTS 12.04.3 en 64 bits. Commencez donc par installer les différentes dépendances requises :

```
sudo apt-get install -y build-essential zlib1g-dev
libyaml-dev libssl-dev libgdbm-dev libreadline-
dev libncurses5-dev libffi-dev curl openssh-server
redis-server checkinstall libxml2-dev libxslt-dev
libcurl4-openssl-dev libicu-dev logrotate
```

Assurez-vous ensuite de disposer de Python 2 (les versions 3.x ne sont pas supportées pour le moment) :

```
sudo apt-get install -y python
sudo apt-get install -y python2.7
sudo ln -s /usr/bin/python /usr/bin/python2
sudo apt-get install -y python-docutils
```

Et bien évidemment, Git vous sera nécessaire :

```
sudo apt-get install -y git-core
```

Enfin, nous allons installer un serveur mail SMTP. Non pas que nous souhaitions recevoir des mails, mais plutôt en envoyer. En effet, il est intéressant de proposer à la plateforme la capacité d'envoyer des mails pour notifier les administrateurs et utilisateurs de différentes choses (envoi du mot de passe original, de merge requests...). Cela se fait simplement en installant **postfix**. Lorsque la question vous sera posée, choisissez simplement « Internet Site » et renseignez votre nom de domaine :

```
sudo apt-get install -y postfix
```

L'installation de Ruby est plus problématique. Ubuntu 12.04 LTS ne dispose « que » de Ruby dans sa version 1.8. Or, comme précisé plus tôt, la version 1.9.3 (au minimum) est nécessaire à GitLab. Commencez donc par supprimer l'installation existante :

```
sudo apt-get remove ruby1.8
```

Et compilez/installez manuellement la version 2.0 (d'où l'installation précédente d'un certain nombre de paquetages de développement) :

```
mkdir /tmp/ruby && cd /tmp/ruby
curl --progress ftp://ftp.ruby-lang.org/pub/
ruby/2.0/ruby-2.0.0-p353.tar.gz | tar xz
cd ruby-2.0.0-p353
./configure -disable-install-rdoc
make
sudo make install
```

Ceci fait, installons maintenant **bundler**, qui va nous permettre de très facilement déployer des applications Ruby (et leurs dépendances) :

```
sudo gem install bundler --no-ri --no-rdoc
```

3 Shell restreint

Il nous est maintenant nécessaire de créer un utilisateur UNIX dédié à GitLab et de lui assigner un shell très particulier. Comme nous allons le voir, ceci est particulièrement astucieux. Cela va nous permettre, tout d'abord, de faire tourner GitLab dans un environnement à droits restreints par rapport au reste de votre système. En outre, cela va nous permettre d'utiliser un même et unique compte (UNIX) pour gérer l'ensemble des dépôts Git (et des droits associés). Mais voilà, GitLab nous permet de gérer des centaines d'utilisateurs, de projets, de groupes et de droits d'accès. Il semble donc absolument hors de question que quiconque ait le même niveau d'accès à tous les dépôts.

La gestion des utilisateurs se fait donc au niveau applicatif (GitLab) et pas au niveau système (UNIX). Les utilisateurs qui voudront pousser leur code (**commit** et **push**) devront donc être déclarés au niveau applicatif. Au niveau système, un seul et unique utilisateur UNIX effectuera réellement l'écriture sur disque. L'authentification des utilisateurs se fera donc, soit par un traditionnel couple login/mot de passe, par HTTP(S), soit par clé SSH. Les utilisateurs déclarés comme « ayants-droit » devront donc uploader leur clé SSH publique dans l'interface de GitLab, afin de pouvoir remonter leurs changements.

Mais place à la création de ce compte et shell restreint. La première chose à faire est donc de créer un compte que nous appellerons simplement **git** :

```
sudo adduser --disabled-login --gecos 'GitLab' git
```

Ceci fait, ajoutons le shell GitLab, utilisé pour l'accès SSH :

```
cd /home/git
sudo -u git -H git clone https://github.com/
gitlabhq/gitlab-shell.git
cd gitlab-shell
sudo -u git -H git checkout v1.8.0
sudo -u git -H cp config.yml.example config.yml
```

Il nous faut maintenant éditer notre fichier **config.yml** en conséquence. Vous trouverez ci-dessous la configuration type utilisée. Dans notre exemple, l'utilisateur UNIX se nomme **git**, les dépôts seront installés dans le répertoire **/repositories** (attention, il doit s'agir d'un chemin absolu et non pas d'un lien symbolique !) et notre GitLab sera accessible par l'URL http://git.mondomaine.com/.

```
# GitLab user. git by default
user: git
# Url to gitlab instance. Used for api calls.
Should end with a slash.
gitlab_url: "http://git.mondomaine.com/"
# Repositories path
repos_path: "/repositories"
# File used as authorized_keys for gitlab user
auth_file: "/home/git/.ssh/authorized_keys"
# Redis settings used for pushing commit notices
to gitlab
redis:
  bin: /usr/bin/redis-cli
  host: 127.0.0.1
```

```
  port: 6379
  # socket: /tmp/redis.socket # Only define this if
you want to use sockets
  namespace: resque:gitlab
# Log file.
log_file: "/home/git/gitlab-shell/gitlab-shell.log"
# Log level. INFO by default
log_level: INFO
# Audit usernames.
audit_usernames: false
```

Si tant est que vous souhaitiez sécuriser davantage la chose (chose que je vous recommande), songez HTTPS et remplacez les lignes suivantes de votre configuration (je passerai outre l'étape de génération d'un certificat SSL, qu'il soit signé par une autorité de certification, ou bien même auto-signé) :

```
# Url to gitlab instance. Used for api
calls. Should end with a slash.
gitlab_url: "https://git.mondomaine.com/"
http_settings:
  ca_file: /etc/ssl/git.mondomaine.com.crt
  ca_path: /etc/ssl/certs
  self_signed_cert: true
```

Enfin, la configuration terminée, procédez au déploiement de votre shell :

```
sudo -u git -H ./bin/install
```

4 Base de données

Place maintenant au déploiement et à la configuration de la base de données. MySQL et PostgreSQL sont tous deux supportés. Voyons ici pour le déploiement de MySQL. Commencez par installer le serveur et par sécuriser ce dernier :

```
sudo apt-get install -y mysql-server mysql-client
libmysqlclient-dev
sudo mysql_secure_installation
```

Connectez-vous ensuite en root (avec le mot de passe saisi dans l'étape précédente) et créez un utilisateur **gitlab** avec le mot de passe de votre choix (ici **mot_de_passe**) :

```
mysql -u root -p
mysql> CREATE USER 'gitlab'@'localhost' IDENTIFIED
BY 'mot_de_passe';
mysql> CREATE DATABASE IF NOT EXISTS `gitlabhq_
production` DEFAULT CHARACTER SET `utf8` COLLATE
`utf8_unicode_ci`;
mysql> GRANT SELECT, LOCK TABLES, INSERT, UPDATE,
DELETE, CREATE, DROP, INDEX, ALTER ON `gitlabhq_
production`.* TO 'gitlab'@'localhost';
```

5 Configuration de GitLab

Nous en avons fini avec les dépendances, place maintenant à l'installation de GitLab lui-même. Commençons par récupérer les sources de la version stable 6.3 :

```
cd /home/git
sudo -u git -H git clone https://github.com/gitlabhq/
gitlabhq.git gitlab
cd /home/git/gitlab
sudo -u git -H git checkout 6-3-stable
# Make sure GitLab can write to the log/ and tmp/
directories
sudo chown -R git log/
sudo chown -R git tmp/
sudo chmod -R u+rwX  log/
sudo chmod -R u+rwX  tmp/
# Create directory for satellites
sudo -u git -H mkdir /home/git/gitlab-satellites
# Create directories for sockets/pids and make sure
GitLab can write to them
sudo -u git -H mkdir tmp/pids/
sudo -u git -H mkdir tmp/sockets/
sudo chmod -R u+rwX  tmp/pids/
sudo chmod -R u+rwX  tmp/sockets/
# Create public/uploads directory otherwise backup
will fail
sudo -u git -H mkdir public/uploads
sudo chmod -R u+rwX  public/uploads
# Copy the example Rack attack config
sudo -u git -H cp config/initializers/rack_attack.
rb.example config/initializers/rack_attack.rb
# Configure Git global settings for git user, useful
when editing via web
# Edit user.email according to what is set in gitlab.
yml
sudo -u git -H git config --global user.name "GitLab"
sudo -u git -H git config --global user.email "gitlab@
localhost"
sudo -u git -H git config --global core.autocrlf input
```

Il nous faut maintenant configurer le fichier **gitlab.yml** en conséquence. Copiez donc le fichier depuis son modèle et modifiez-le :

```
sudo -u git -H cp config/gitlab.yml.example
config/gitlab.yml
```

GitLab supporte de multiples instances. Dans notre cas, seule celle de production nous intéresse et vous pouvez donc restreindre votre configuration à cette dernière.

```
production: &base
  gitlab:
    host: git.mondomaine.com
    port: 80
    https: false
    user: git
    email_from: git@mondomaine.com
    support_email: git@mondomaine.com
    ## User settings
    default_projects_limit: 50
    default_can_create_group: false
    username_changing_enabled: false
    signup_enabled: false
    ## Default project features settings
    default_projects_features:
      issues: false
      merge_requests: true
      wiki: false
      wall: false
      snippets: true
      public: false
```

La configuration précédente informe GitLab des informations propres à votre serveur et vous permet également de positionner un ensemble de paramètres relatifs à la gestion des utilisateurs. Dans notre exemple, nous empêchons ces derniers de s'auto-enregistrer sur le serveur, de changer leur nom d'utilisateur, ou encore de créer des groupes.

Il est également possible d'activer ou désactiver les fonctionnalités proposées par défaut dans vos projets, telles que l'outil de rapport de bogues (*issues tracker*), de requêtes de fusion, le wiki et bien d'autres encore. Sans vraiment s'y attarder, vous verrez au sein de ce fichier de configuration que GitLab supporte également l'authentification des utilisateurs par rapport à une base LDAP, ou par protocole OAuth via vos comptes Google, Twitter, LinkedIn et autres.

De la même façon, copiez le modèle de configuration d'Unicorn, le serveur web intégré à GitLab :

```
sudo -u git -H cp config/unicorn.rb.example
config/unicorn.rb
```

Et modifiez ce dernier pour paramétrer l'accès au serveur :

```
# Use at least one worker per core if you're
on a dedicated server,
worker_processes 3
working_directory "/home/git/gitlab" #
available in 0.94.0+
# listen on both a Unix domain socket and a
TCP port,
# we use a shorter backlog for quicker
failover when busy
listen "/home/git/gitlab/tmp/sockets/gitlab.
socket", :backlog => 64
listen "127.0.0.1:8080", :tcp_nopush => true
```

Enfin, configurez la base de données en fonction des paramètres saisis lors de l'installation de MySQL :

```
sudo -u git cp config/database.yml.mysql
config/database.yml
sudo -u git -H chmod o-rwx config/database.yml
```

Et faites-y figurer votre configuration :

```
production:
  adapter: mysql2
  encoding: utf8
  reconnect: false
  database: gitlabhq_production
  pool: 10
  username: gitlab
  password: "mot_de_passe"
```

Enfin, déployez votre installation de GitLab et initialisez la base de données :

```
cd /home/git/gitlab
sudo -u git -H bundle install --deployment
--without development test postgres aws
sudo -u git -H bundle exec rake gitlab:setup
RAILS_ENV=production
```

Finalisez enfin l'installation par le déploiement des scripts d'initialisation, permettant un démarrage automatisé :

```
sudo cp lib/support/init.d/gitlab /etc/
init.d/gitlab
sudo chmod +x /etc/init.d/gitlab
sudo update-rc.d gitlab defaults 21
sudo cp lib/support/logrotate/gitlab /etc/
logrotate.d/gitlab
```

Vérifiez le bon déploiement de GitLab :

```
sudo -u git -H bundle exec rake
gitlab:env:info RAILS_ENV=production
```

Et démarrez votre instance :

```
sudo service gitlab start
```

6 Accès web

Nous touchons au but ! GitLab est maintenant déployé et opérationnel. Reste à y adjoindre un serveur web, permettant de s'y connecter au travers de HTTP. Nous utiliserons pour cela Nginx, plus tendance, mais surtout plus performant qu'Apache.

```
sudo apt-get install -y nginx
sudo cp lib/support/nginx/gitlab /etc/nginx/
sites-available/gitlab
sudo ln -s /etc/nginx/sites-available/gitlab
/etc/nginx/sites-enabled/gitlab
```

Procédez enfin à la configuration de votre fichier **/etc/nginx/sites-enabled/gitlab**. Dans notre exemple, nous configurons Nginx pour forcer l'accès HTTPS. Toute requête sur le port HTTP (ou 80) sera ainsi redirigé vers le port HTTPS (443). Ce dernier est ensuite configuré comme proxy, transférant la plupart des requêtes (via **proxy_pass**) à Unicorn, le serveur web applicatif intégré à GitLab :

```
upstream gitlab {
  server unix:/home/git/gitlab/tmp/sockets/gitlab.
socket;
}
# This is a normal HTTP host which redirects all
traffic to the HTTPS host.
server {
    listen *:80;
    server_name git.mondomaine.com;
    server_tokens off;
    root /nowhere;
    rewrite ^ https://$server_name$request_uri
permanent;
}
server {
    listen 443 ssl;
    server_name git.mondomaine.com;
    server_tokens off;
    root /home/git/gitlab/public;
    ssl on;
    ssl_certificate /etc/ssl/git.mondomaine.com.crt;
    ssl_certificate_key /etc/ssl/git.mondomaine.
com.key;
    ssl_protocols  SSLv3 TLSv1 TLSv1.2;
    ssl_ciphers AES:HIGH:!ADH:!MD5;
    ssl_prefer_server_ciphers   on;
    # individual nginx logs for this gitlab vhost
    access_log  /var/log/nginx/gitlab_access.log;
    error_log   /var/log/nginx/gitlab_error.log;
    location / {
        # serve static files from defined root
folder;.
        # @gitlab is a named location for the
upstream fallback, see below
        try_files $uri $uri/index.html $uri.html @
gitlab;
    }
    # if a file, which is not found in the root
folder is requested,
    # then the proxy pass the request to the
upsteam (gitlab unicorn)
    location @gitlab {
        rewrite ^/gitlab(.*)$ $1 last;
        proxy_read_timeout    300; # https://
github.com/gitlabhq/gitlabhq/issues/694
        proxy_connect_timeout 300; # https://
github.com/gitlabhq/gitlabhq/issues/694
        proxy_redirect        off;
        proxy_set_header  X-Forwarded-Proto https;
        proxy_set_header  X-Forwarded-Ssl   on;
        proxy_set_header  Host              $http_
host;
        proxy_set_header  X-Real-IP
$remote_addr;
        proxy_pass http://gitlab;
    }
}
```

Ceci fait, il ne vous reste plus qu'à redémarrer votre serveur web :

```
sudo service nginx restart
```

Voilà, nous en avons terminé ! Vous pouvez maintenant vous connecter à votre serveur GitLab via https://git.mondomaine.com/ en tant qu'administrateur, créer des comptes utilisateurs, des groupes, des projets, des dépôts Git (ou en importer depuis des sources déjà existantes), etc.

7 Sauvegardes

Au fur et à mesure de l'utilisation de votre GitLab, l'occupation d'espace disque va aller croissant. Je ne saurai donc que trop vous recommander de penser dès à présent à sauvegarder vos données. Rien de plus simple : assurez-vous de disposer d'un simple NAS et de le monter via NFS/CIFS dans un répertoire **/backup**.

Je vous propose ensuite de créer un simple script shell **backup.sh**, qui vous permette de synchroniser les dépôts Git via **rsync**, ainsi qu'un dump de la base de données MySQL de GitLab :

```
#!/bin/sh
rsync -rptDvz --delete /repositories /backup
/usr/bin/mysqldump -u root -pmot_de_passe
gitlabhq_production | gzip > /home/git/sql_
dump/gitlab_db_`date +%y_%m_%d`.gz
rsync -atvz --delete /home/git/sql_dump /backup
```

Reste à lancer ce script toutes les nuits via une tâche Cron, et voilà :

```
ctrontab -e
0 23 * * * /home/git/backup.sh >> /home/git/
logs/rsync.log
```

Conclusion

Si vous avez suivi ces quelques étapes à la lettre, vous devriez désormais disposer d'un environnement collaboratif de gestion de projets et de sources, sécurisé, flexible et sauvegardé. Tout le nécessaire donc, pour permettre à votre entreprise et ses collaborateurs internes (ou externes) de travailler tous ensemble. GitLab n'est certes qu'une interface d'administration et de gestion de projets au-dessus de Git, mais il le fait bien. Lorsque vous êtes amené à gérer de multiples projets, avec des utilisateurs, groupes et droits d'accès différents, il révèle toute sa valeur. À vous donc, si cela n'était pas encore fait, de faire le pas et de passer à Git pour vos prochains projets d'entreprise. ■

Références

[1] http://www.github.com/
[2] http://www.gitorious.org/
[3] http://www.gitlab.org/

CRÉER UNE APPLICATION PERL AUTOUR DE MYSQL : MISE EN PLACE (1/3)

par Dominique Dumont

C'est l'histoire d'un hacker à qui on a demandé de créer des pages web pour le suivi de la production logicielle de la société. Historiquement, toutes les informations étaient gérées à la main, ce qui est une solution inacceptable pour tout hacker qui se respecte.

Le but de cette série d'articles est de vous montrer comment on peut utiliser Perl, MySQL, phpMyAdmin, les modules **DBIx::Class**, **DBIx::Class::Schema::Loader** et **HTML::Tiny** pour gérer ces informations d'une manière plus efficace. Ces articles vont donner une vision d'ensemble sur une solution mettant en jeu plusieurs technologies au lieu de se focaliser sur un domaine bien précis.

Ce premier article va expliquer les besoins de l'application, la structure de la base de données et comment créer cette structure dans une base MySQL avec phpMyAdmin. Au passage, on verra les principes de base des relations entre les tables et l'utilisation des index et des clés étrangères (*foreign keys*) pour que MySQL assure la cohérence des entités de l'application.

1 Introduction

Le problème à résoudre : comme toute société informatique qui se respecte, nous produisons plusieurs produits logiciels. Chaque produit évolue et cette évolution est traduite en termes de version. Et comme dans toute société, les noms de code et numéros de version utilisés pendant le développement correspondent rarement avec ceux décidés (tardivement) par le marketing. Devant cette profusion de noms de codes et de versions, il est difficile de s'y retrouver. J'ai donc été chargé d'écrire une application qui fournit des pages d'information résumant la situation.

Donc, pour générer ces pages, il faudra gérer la correspondance entre :

- un produit tel qu'il est vu par le marketing (et par le client) avec :
  - → un nom de produit en général évocateur de simplicité et de performances pour appâter le client,
  - → quelques informations supplémentaires (*release notes*) ;
- une version de produit dont le numéro est en général décidé tard dans le cycle de développement ;
- Cette version de produit est déclinée en fonction de l'environnement cible : distribution (RHEL 5 ou RHEL 6) et architecture (i386 ou x86_64) et est livrée dans un fichier **tar**.
- Un ensemble de paquets RPM contenus dans ces fichiers **tar**.

En termes de relations, on va avoir :

- un produit va contenir plusieurs versions marketing,
- chaque version marketing va contenir plusieurs fichiers **tar**,
- chaque fichier **tar** va contenir plusieurs paquets (RPM dans ce cas de figure),
- un paquet peut faire partie de plusieurs fichiers **tar**.

Une fois qu'on a tout ça en place, le but est de remplir les données des versions avec Perl et **DBIx::Class**, puis de produire des pages HTML décrivant chaque release et ses différents composants.

Pour ne pas compliquer cet article, on va ignorer le problème de suivi des bogues et des corrections associés à chaque version.

2 La base de données

2.1 Structure des données

On a vu dans l'introduction que nous avons affaire à des données structurées avec des relations entre les différentes entités :

- relation « un avec plusieurs » (*one-to-many*) entre les produits, versions marketing et fichiers **tar** ;

- relation « plusieurs avec plusieurs » (*many-to-many*) entre les fichiers **tar** et les paquets.

On verra comment mettre en place ces relations dans la base de données.

3 Installation de tout le bazar

Sous Debian, faites :

```
sudo aptitude install mysql-server phpmyadmin
```

Notez qu'un serveur web va être installé par le jeu des dépendances.

aptitude va vous proposer d'installer une base de données dédiée à phpMyAdmin. Celle-ci servira à stocker des informations relatives aux structures de vos bases de données et aux fonctions avancées de phpMyAdmin. Accepter cette option est conseillé.

Vérifiez aussi que l'extension de sécurité suhosin (paquet **php5-suhosin** sous Debian) n'est pas installée. Celle-ci pose des problèmes au bon fonctionnement de phpMyAdmin.

Une fois qu'**aptitude** a fini son boulot, votre phpMyAdmin tout neuf est accessible avec votre butineur favori à l'adresse : http://locahost/phpmyadmin.

3.1 Création de la base de données et de ses utilisateurs

Une fois dans l'interface de phpMyAdmin, cliquez sur ***Databases*** pour créer la base de données « Integ ».

Ensuite, il nous faut deux utilisateurs :

- **integ_user**, qui pourra lire les données et se connecter à la DB sans utiliser de mot de passe ;
- **integ_mgr**, qui pourra mettre à jour les données et devra utiliser un mot de passe pour se connecter.

Les utilisateurs sont ajoutés en deux étapes :

- Création avec l'onglet ***Privileges*** et ***Add a new user***. Si nécessaire, des privilèges globaux (c'est-à-dire pour toutes les bases du serveur) peuvent être attribués.
- Réglage fin des privilèges base par base. Ce réglage est accessible dans le tableau ***User overview*** dans l'onglet ***Privileges***.

Vous pouvez vérifier le résultat de la configuration des privilèges :

- Soit dans l'onglet ***Databases*** et le lien ***Check Privileges*** de la ligne ***integ***. Dans le tableau ***Users having access to "Integ"***, ***integ_user*** doit avoir juste un privilège « SELECT ».
- Soit avec la commande :

```
$ mysql -u integ_user Integ
[ ... ]
mysql>
```

La création de l'utilisateur **integ_mgr** se fera avec une procédure similaire, mais en mettant un mot de passe et en attribuant les privilèges : « SELECT », « INSERT », « UPDATE » et « DELETE » pour la base « Integ ».

4 Création des tables

4.1 La table « Products »

Une fois le lien « Integ » cliqué dans le menu de gauche, phpMyAdmin va vous montrer la structure (vide) de la base « Integ » et vous inviter à créer votre première table, « Products » avec 3 colonnes :

- La colonne « id », qui va être la clef primaire de cette table. « id » est de type entier (« INT »), avec un index « PRIMARY », avec « AUTO INCREMENT » sélectionné ;
- La colonne « name », pour le nom marketing du produit. « name » est de type « VARCHAR », de 32 caractères de longueur, index « UNIQUE ». Pourquoi cet index ? Le marketing ne sortira jamais simultanément deux produits différents ayant le même nom. Cet index « UNIQUE » permet de faire respecter cette règle par la DB. Ça vous permet aussi de ne pas vérifier cette contrainte dans votre code (ou dans toute autre application qui voudrait exploiter cette base) ;
- La colonne « home_page » pour les informations internes sur le produit. « home_page » est de type « VARCHAR », de 256 caractères de longueur, et potentiellement « NULL » (des fois que le produit soit créé dans la base avant que sa home page ne soit disponible).

Derniers points importants :

- Le module **DBIx::Class** utilisé dans la 2e partie de cet article ne fonctionne pas si le nom de la colonne contient des espaces. C'est pourquoi un souligné («_») est utilisé.
- Le « Storage engine » doit être « InnoDB ». (On verra dans cet article pourquoi).

Enfin, n'oubliez pas de cliquer sur ***Save*** en bas de page.

4.2 Mais pourquoi 2 colonnes ?

Vu que le nom du produit doit être unique, on pourrait être tenté de s'en servir comme clef primaire et de se débarrasser de la colonne **id**. C'est une solution, certes fonctionnelle, mais qui présente un inconvénient majeur : le nom du produit devra aussi être utilisé dans les tables qui le référencent. Pour cet article, il devrait ainsi être utilisé dans la table qui contiendra les versions de produit. Si jamais le nom du produit change, il faudra retrouver et modifier toutes ces utilisations du nom du produit.

4.3 La table « ProductVersions »

Cette table a besoin d'une clef primaire « id », d'un champ « version » et d'un champ « date ». La création de ces colonnes est similaire aux colonnes de la table « Products ».

4.4 Relier un produit avec ses versions

On a maintenant deux jolies tables, mais elles ne sont pas reliées. Il faut créer la relation entre le produit (singulier) et ses versions (pluriel). Il s'agit d'une relation « 1 vers n ». En termes de base de données, cette relation est établie en créant dans la table « ProductVersions » une colonne « product_id » qui contiendra l'« id » du produit correspondant (« product_id » doit être de type « INT » avec Index à « INDEX »). Dans cette table, plusieurs versions pourront référencer le même « id », donc le même produit. Mais une version ne peut référencer qu'un seul produit. On a bien établi une relation « 1 vers n ».

Mais, va-t-on devoir coder de quoi vérifier la cohérence entre « product_id » et les « id » disponibles dans la table « Products » ?

4.5 Mieux relier le produit et ses versions : les clefs étrangères

J'ai pratiqué dans une vie antérieure le codage de cette vérification. Eh bien, ce n'est guère passionnant. Heureusement, les bases de données modernes fournissent maintenant un mécanisme assurant cette cohérence : les clefs étrangères (foreign keys, dans la langue de Shakespeare).

On peut déclarer à la DB que la valeur de « product_id » doit être présente dans la colonne « id » de la table « Products ». Ceci fait, la cohérence est assurée. Mais il y a encore mieux pour la maintenance des données par la suite.

Qu'advient-il si un produit est supprimé du catalogue ? Devra-t-on supprimer une par une les versions de la DB avant de pouvoir supprimer le produit ? Eh bien, non. MySQL offre aussi la possibilité de paramétrer le comportement en cas de destruction d'une ligne produit : soit interdire, soit passer à « NULL » (mettre le « product_id » à 0, rendant la version orpheline), soit propager (toutes les versions du produit sont détruites).

C'est cette dernière option qui nous intéresse.

Notez que les foreign keys sont disponibles avec le « Storage engine » « InnoDB », mais pas avec « MyISAM », qui est proposé par défaut. Si vous avez oublié ce détail, ce n'est pas grave, vous pouvez le changer en allant modifier ce paramètre dans l'onglet ***Operations*** de la table restée coincée en « MyISAM » (cherchez « Storage Engine »).

Trêve de théorie, on peut ajouter cette relation dans la page « Structure » de la table « ProductVersions » avec le lien « Relation View » (si absent, voir le problème « InnoDB »). Si vous ne voyez pas de ligne « product_id », ajoutez un index sur cette colonne). Et là, mettez en place la « relation interne » (utilisé par phpMyAdmin) et la « Foreign key » (pour MySQL) avec l'option « ON DELETE: CASCADE ».

On peut aussi ajouter cette relation avec l'interface graphique « designer ». Cette interface est disponible en retournant dans la page de la base « Integ », onglet ***More***, bouton ***designer***.

Vous devriez voir les deux tables non reliées. Cliquez sur le bouton ***Create relation*** et suivez les instructions :

- Sélectionnez la clef référencée : « id » dans la table « Products »,
- Sélectionnez la clef étrangère : « product_id » de la table « ProductVersions »,
- Choisissez « cascade » pour l'option « on delete ».

Et devant vos yeux émerveillés, voyez ceci apparaître :

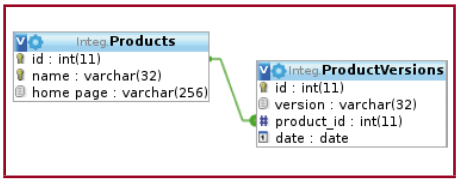


Fig. 1

(L'auteur admet ne pas aimer PHP, adorer phpMyAdmin et assumer ses contradictions...).

4.6 Blinder un peu plus

MySQL peut aussi assurer l'unicité de chaque couple produit-version. Il suffit de déclarer un index de type unique pour les deux colonnes « product_id » et « version ». Ça vous évitera de créer des lignes inutiles si votre code utilise au mauvais endroit une instruction de type « créer » au lieu du type « chercher ou créer ». (L'utilisation de ces fonctions sera expliquée plus en détails dans un article ultérieur).

4.7 La table « TarBalls »

Elle est créée de la même façon que la table « ProductVersions » avec une clef primaire « id », une « url » de type « varchar/128 », index « UNIQUE » et enfin, une clef étrangère « product_version_id » pointant sur l'id de la table « ProductVersions ».

4.8 La table « Packages » et la relation m-vers-n

Elle contient la colonne « id » et le nom du paquet (triplet **`nom-version.arch.rpm`**).

Mais là, ça se complique : un fichier **`tar`** peut contenir plusieurs paquets et chaque paquet peut faire partie de plusieurs fichiers **`tar`**. C'est une relation m-vers-n. Une seule colonne **`tarball_id`** dans la table « Packages » ne suffit pas.

La technique traditionnelle est d'utiliser une table intermédiaire pour relier tarball et paquet avec deux colonnes : une pour stocker l'id du tarball et une pour stocker l'id du paquet. Par exemple, si on crée un tarball (avec un id 42) contenant deux paquets (un déjà connu avec un id à 99 et un nouveau avec un id à 1024), on ajoutera dans la base :

- une ligne avec un id 42 dans la table « TarBalls »,
- une ligne avec un id 1024 pour le nouveau paquet dans la table « Packages »,
- deux lignes (42-99 et 42-1024) pour matérialiser les deux relations paquets-tarballs dans la table intermédiaire (nommée « TarballPackages »)

Cette table intermédiaire a :

- une colonne « tarball_id » de type entier, clef étrangère sur la colonne « id » de la table « TarBalls »,
- une colonne « package_id » de type entier, clef étrangère sur la colonne « id » de la table « Packages »,
- une clef primaire constituée des deux colonnes « tarball_id » et « package_id ».

Ce dernier point est important pour la suite avec le module Perl **`DBIx::Class::Schema::Loader`**. Ce module utilise des heuristiques assez contraignantes pour essayer d'interpréter correctement les relations entre les tables. Pour générer une relation « many-to-many » entre 2 classes **`DBIx::Class`**, la table de relation doit respecter ces règles.

Voici le schéma complet présenté par le « designer » de phpMyAdmin (les tables sont un peu maigres, mais c'est pour le bien de cet article) :

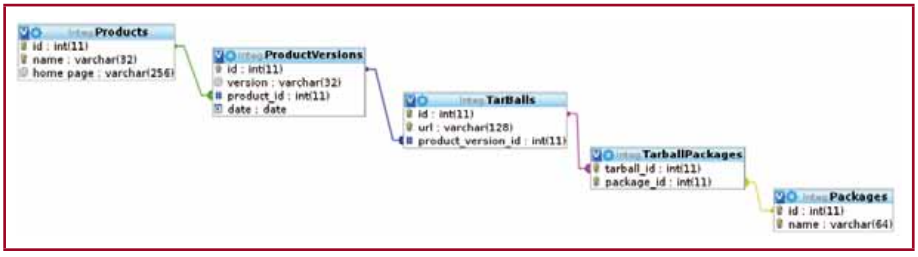


Fig. 2

5 Suite au prochain épisode

Maintenant que la persistance des données est assurée par la base de donnée « Integ », il reste à voir dans un prochain article :

- Comment utiliser **`dbcidump`** et **`DBIx::Class::Schema::Loader`** pour mettre en place une interface entre un programme Perl et la base de données en écrivant un minimum de code ;
- Le module **`DBIx::Class`** pour effectuer des requêtes sur la base de données et exploiter les relations mises en place dans cet article. ■

CRÉER UNE APPLICATION PERL AUTOUR DE MYSQL : DBIX::CLASS (2/3)

par Dominique Dumont

Reprenons l'histoire du hacker à qui on a demandé de créer des pages web pour le suivi de la production logicielle de la société. Comme tout hacker qui se respecte, il préfère que la machine fasse son boulot à sa place. Pour ce faire, il a mis en place une base de données MySQL. Reste maintenant à utiliser Perl et DBIx::Class pour accéder aux données de cette base.

Introduction

Petit rappel de l'épisode précédent : notre hacker favori a expliqué comment mettre en place dans une base de données MySQL les tables et relations qui assureront la persistance des données de production.

Pour rappel, voici un résumé des tables créées. Le code complet en SQL est disponible sur le GitHub de l'auteur **[GITHUBADOD]** :

- Table « Packages », avec « id » et « name »,
- Table « Products » avec « id », « name » et « home_page »,
- Table « ProductVersions » avec « id », « version », « product_id », « date », « log », « status »,
- Table « TarballPackages » avec « tarball_id » et « package_id »,
- Table « TarBalls » avec « id », « url » et « product_version_id ».

Chaque colonne finissant en « _id » est une clef étrangère sur la colonne « id » d'une autre table. Par exemple, « product_id » est une clef étrangère sur la colonne « id » de la table « Products ».

1 Accès DB avec Perl

On a maintenant une jolie base de données toute configurée pour recevoir les données de produits et leur myriade de versions. Mais, il s'agit maintenant d'y accéder à partir de notre application.

1.1 Les ORM ou comment ne pas se compliquer la vie ?

Un des premiers réflexes qu'on peut avoir est d'écrire une application générant tout le code SQL nécessaire pour gérer les données. (SQL est le langage de manipulation des bases de données). Au début, ça va suffisamment vite à coder pour rassurer le chef, mais ça finit souvent en un seau de code difficile à maintenir (c'est du vécu), car l'application mélange deux langages avec des paradigmes différents.

Bien des gens brillants se sont penchés sur le problème de relier les données d'un programme avec celles d'une base de données. Les ORM ou *Object Relational Mapper*, qu'on peut traduire en Passerelle Objet Relationnel (dommage que l'acronyme ne soit pas très vendeur...) permettent de minimiser le code à écrire (aussi bien en Perl qu'en SQL).

Avec un peu de chance, un ORM vous permettra d'écrire votre application sans avoir à écrire une seule ligne de SQL. Mais attention, des connaissances en SQL sont quand même fortement conseillées : les contraintes mises en place sur la base (c'est-à-dire index unique, clef étrangère) vont provoquer des erreurs lors de la mise au point. Ces erreurs sont montrées avec les instructions SQL rejetées par la DB. À vous d'interpréter ces erreurs SQL pour trouver les instructions Perl erronées dans votre programme.

Pour le projet, l'ORM Perl `DBIx::Class` a été choisi. Il contient toutes les notions requises pour faire de la programmation objet moderne. Le seul problème est qu'il faut lui spécifier la structure de la base de données pour qu'il puisse générer les classes et les méthodes qui feront l'interface avec la base.

Et là, miracle, un codeur fou (celui à qui personne n'a dit que c'était impossible) a créé une application qui interroge la base de données, en extrait le

schéma et génère toutes les déclarations qui vont bien pour **DBIx::Class** : que du bonheur ! Il s'agit de **dbicdump** fourni avec **DBIx::Class::Schema::Loader**.

1.2 Installation

Sous Debian, il faudra installer les paquets **libdbix-class-perl** et **libdbix-class-schema-loader-perl** (version >= 0.07015).

1.3 Génération des interfaces avec « dbicdump »

Générer l'interface Perl pour votre base de données se résume à cette commande :

```
dbicdump -o dump_directory=./lib -o use_moose=1 \
    -o components='["InflateColumn::DateTime"]' \
     Integ::Schema dbi:mysql:Integ root s3cr3t
```

Voici quelques explications sur les options :

- **use_moose=1** : **dbicdump** peut générer des classes Perl classiques ou des classes Moose (basées sur **MooseX:NonMoose**) ;
- **components='["InflateColumn::DateTime"]'** : cette option est fortement recommandée dans la documentation de **DBIx::Class::Schema::Loader**. Elle permet de gérer toute colonne de type **DATE** à travers un objet **DateTime**, ce qui est très pratique.
- Les 3 derniers arguments permettent à **dbicdump** de se connecter sur la base.

Une fois cette commande finie, on obtient :

```
$ find . -name *.pm
./lib/Integ/Schema.pm
./lib/Integ/Schema/Result/Package.pm
./lib/Integ/Schema/Result/ProductVersion.pm
./lib/Integ/Schema/Result/TarBall.pm
./lib/Integ/Schema/Result/TarballPackage.pm
./lib/Integ/Schema/Result/Product.pm
```

Chaque fichier contient la documentation en format Pod et toutes les instructions **DBIx::Class** pour créer la classe d'accès à votre base de données. Prenons par exemple la classe **Integ::Schema::Result::ProductVersion** et voyons les parties les plus importantes :

```
package Integ::Schema::Result::ProductVersion;
use Moose;
use MooseX::NonMoose;
use MooseX::MarkAsMethods autoclean => 1;
extends 'DBIx::Class::Core';
```

La classe d'interface est bien une classe Moose héritant de la classe **DBIx::Class::Core** (celle-ci n'est pas une classe Moose).

Le code suivant déclare la table et les composants chargés :

```
__PACKAGE__->table("ProductVersions");
__PACKAGE__->load_components("InflateColumn::DateTime");
```

Voici la déclaration des colonnes. **add_columns** va aussi ajouter les méthodes correspondantes (« accessors ») de façon à pouvoir écrire **print $obj->version;** ou **$obj->version(1.002)**.

```
__PACKAGE__->add_columns(
  "id",
  { data_type => "integer", is_auto_increment => 1, is_nullable => 0 },
  "version",
  { data_type => "varchar", is_nullable => 0, size => 32 },
  "product_id",
  { data_type => "integer", is_foreign_key => 1, is_nullable => 0 },
);
```

Déclaration de la relation entre **Products** et **ProductVersions** :

```
__PACKAGE__->belongs_to(
  "product",
  "Integ::Schema::Result::Product",
  { id => "product_id" },
  { is_deferrable => 1, on_delete => "CASCADE", on_update => "CASCADE" },
);
```

Voici la déclaration de la relation entre **Products** et **TarBalls**. Le code généré contient aussi des indications sur les clés étrangères et leur contraintes :

```
__PACKAGE__->has_many(
  "tar_balls",
  "Integ::Schema::Result::TarBall",
  { "foreign.product_version_id" => "self.id" },
  { cascade_copy => 0, cascade_delete => 0 },
);
```

Le commentaire suivant est important, car il définit la zone à partir de laquelle vous pouvez apporter vos propres modifications. Cette zone peut servir à définir de nouveaux attributs ou méthodes pour votre classe d'interface.

```
# Created by DBIx::Class::Schema::Loader v0.07015 @ 2012-02-07 20:31:16
# DO NOT MODIFY THIS OR ANYTHING ABOVE! md5sum:/R7o/BCNPW4v3IK8gUkIxQ
```

Si vous avez à modifier le schéma... Non, je reprends... Quand vous aurez à modifier le schéma de la DB pour suivre les besoins de vos clients, il suffira de relancer la commande **dbicdump** pour obtenir une nouvelle interface. La zone « libre » ne sera pas touchée. **dbicdump** ira même jusqu'à vérifier la syntaxe du fichier résultant.

1.4 Extension de la classe d'interface

En général, le premier réflexe est d'ajouter dans cette zone modifiable toutes les méthodes dont on a besoin. Mais c'est un mauvais réflexe. Pour pouvoir gérer correctement votre projet dans le temps et tester la logique métier indépendamment de la DB, il vaut mieux séparer clairement l'interface de la DB de la logique métier.

Le plus simple, quand on utilise Moose, est d'étendre la classe en ajoutant un rôle à la classe générée :

```
# Created by DBIx::Class::Schema::Loader v0.07015 @ 2012-01-31
15:49:49
# DO NOT MODIFY THIS OR ANYTHING ABOVE!
md5sum:NiaYh10AJwYDWMg5lwliVA

# contient mon code à moi pour créer les pages HTML
with 'Integ::HTML::ProductVersion' ;

__PACKAGE__->meta->make_immutable;
1;
```

Note

Dans le rôle Moose, on peut aussi ajouter des attributs pour enrichir la classe. Ces attributs ne seront pas stockés dans la base de données. Curieusement, les valeurs par défaut spécifiées dans les déclarations des attributs sont ignorées. Il faut utiliser **lazy_build => 1** (ou **lazy => 1, build => '_build_toto'**) et spécifier une valeur par défaut avec une fonction du genre **sub _build_toto { return "TOTO"; }**.

1.5 Connexion à la base

La connexion à la base se fait directement à partir de la classe générée par **dbicdump**. On prépare d'abord une variable pour spécifier à **DBI** (le gestionnaire de la connexion à la DB) où se connecter :

```
my $integ_dsn = 'dbi:mysql:database=Integ;host=localhost;port=3306';
```

Puis, on se connecte :

```
my $integ_schema = Integ::Schema->connect(
   $integ_dsn, 'integ_mgr', $password,
   { RaiseError => 1 }
);
```

Pour l'accès en lecture seule, le compte **integ_user** n'ayant pas de mot de passe, il suffira de passer une chaîne vide en lieu de mot de passe :

```
my $integ_schema = Integ::Schema->connect(
   $integ_dsn, 'integ_user', '',
   { RaiseError => 1, quote_names => 1 }
);
```

L'option **quote_names** est nécessaire, car la colonne **home page** contient un espace.

1.6 Lecture des données

Maintenant qu'on a une connexion toute neuve à la base, on va pouvoir passer aux choses sérieuses.

Avant de continuer, il faut préciser qu'un nom de produit fictif sera utilisé dans cet article. Toute ressemblance avec un produit existant ou ayant existé serait fortuite et involontaire. En effet, par les temps qui courent, il vaut mieux être prudent avec les marques déposées si on ne veut pas se prendre un procès sur la courge. Tiens, ça me donne une idée : notre produit va s'appeler « OpenCourge ». Avec ça, not' rédac-chef favori devrait dormir tranquille ;-)

Pour des raisons pédagogiques, on va d'abord se pencher sur l'extraction de données de la base avant de voir l'écriture. Comme la base est vide, on va ajouter en SQL un produit et une version pour pouvoir tester la lecture avec **DBIx::Class**. Je vous propose :

```
$ myqsl -u integ_mgr -p Integ
...
mysql> insert into Products set name="OpenCourge" ;
mysql> insert into ProductVersions set version=1.01, product_id=1, date = "2012-08-20" ;
mysql> insert into ProductVersions set version=1.02, product_id=1, date = "2012-09-27" ;
mysql> quit ;
```

Première étape pour lire dans la base, il faut trouver le produit OpenCourge. On va d'abord récupérer un objet **DBIx::Class::Resultset** qui donne accès à la table **Products** :

```
my $product_rs = $integ_schema ->resultset('Product');
```

Notez que le nom de la table est « Products » avec un 's', mais qu'on cherche « Product » (sans 's') dans le schéma. En effet, **DBIx::Class::Schema::Loader** passe au singulier le nom de la table (qui représente un ensemble de données) pour créer le nom de la classe d'interface (qui représente plutôt un type de données). Pour plus de détails, voir la doc de **DBIx::Class::Schema::Loader::Base**.

Une fois qu'on a accès à l'ensemble des produits à travers **$product_rs**, on peut rechercher un produit spécifique :

```
my $product_row = $product_rs ->find ( { name => 'OpenCourge' } );
```

La méthode **find** va renvoyer un objet **DBIx::Class::Row** référençant la ligne de la table **Products** ayant pour nom **OpenCourge**. Comme la colonne **Name** a un index **UNIQUE**, on est sûr de référencer la bonne ligne. Le SQL exécuté sera un simple **select * from Product where name='OpenCourge' ;**.

Surtout, n'oubliez pas les accolades, sinon **find** chercherait une clé primaire basée sur deux colonnes nommées « name » et « OpenCourge ».

1.7 Utilisation de la relation « 1-vers-n »

On a retrouvé notre produit favori dans la base, il s'agit maintenant de récupérer les versions disponibles. On rappelle que ces versions sont stockées dans leur propre table et liée à la table « Products » avec la colonne « product_id ». La documentation de la classe générée **Integ::Schema::Result::Product** indique que **dbicdump** a matérialisé cette relation avec la méthode « product_versions ».

Ainsi, la ligne suivante va renvoyer un objet **DBIx::Class::Resultset** contenant les versions attachées à notre produit :

```
my $product_version_rs = $product -> product_versions;
say $product_version_rs->count ; # 2
```

Dans cet ensemble de versions, on peut chercher une version spécifique :

```
my $v = $product_version_rs->find({version => '1.02'}) ;
say $v->date->ymd ; # 2012-09-27
```

Ou chercher des versions selon un critère :

```
my $v_rs = $product_version_rs->search({date => { like => '2012%' }}) ;
say join(' ', map { $_->version } $v_rs->all) ; # 1.01 1.02
```

Notez que la méthode **find** est utilisée si le critère renvoie une seule valeur et renvoie un objet **DBIx::Class::Row** relié à une ligne dans la table. La méthode **search** concerne des recherches plus floues et renvoie un objet **DBIx::Class::Resultset** représentant un ensemble de lignes.

1.8 Écriture des données

Maintenant, il est temps de créer de nouvelles versions du produit dans la base à travers **DBIx::Class**. Pour éviter les ennuis, les instructions de création de version doivent être idempotentes. Il faut donc vérifier si la version existe déjà avant de la créer. Heureusement, **DBIx::Class** fournit la méthode qui permet de faire ça d'une manière concise :

```
say $product_version_rs->count ; # 2

$product_version_rs->find_or_create( { version => '1.04' } );
say $product_version_rs->count ; # 3

$product_version_rs->find_or_create( { version => '1.04' } );
say $product_version_rs->count ; # 3, pas de changement
```

1.9 Modification des données

Mettre à jour les données dans une ligne est très simple avec les méthodes fournies :

```
my $product_version_row = $product_version_rs -> find( { version => '1.04'} ) ;
$product_version_row -> log( "c'est mûr" );
```

Mettre en place la date de publication de la version demande un peu plus de précautions si on ne veut pas changer cette date chaque fois que le programme est lancé :

```
my $d = $product_version_row->date ;
$product_version_row->date( DateTime->now ) unless $d;
```

Le composant **InflateColumn::DateTime** va se charger de traduire l'objet **DateTime** en une chaîne de caractères compréhensible par MySQL.

Et pour déclarer les tarballs générés, on fera :

```
my $tarball_obj = $product_version_row
   -> tarballs
   -> find_or_create ( {
       url => "http://cucurbitacée.com/repo/OpenCourge-i386.tar"
    } ) ;
# http://cucurbitacée.com/repo/OpenCourge-i386.tar
say $product_version_row->tar_balls->single->url ;
```

Mais les informations écrites sont pour l'instant dans la mémoire du programme. Il faut les transférer dans la base :

```
$product_version_row -> update;
```

1.10 Utilisation de la relation « m-vers-n »

Pour pouvoir gérer les paquets et les tarballs, on va aborder le domaine qui rebute souvent les débutants en base de données relationnelles : la relation « plusieurs vers plusieurs », qui implique une table de liaison intermédiaire. En SQL, retrouver ses petits dans cette relation nécessite l'utilisation de jointures.

Heureusement, la classe **DBIx::Class::Relationship** (héritée par **DBIx::Class::Core**) gère ces relations et permet de masquer au programmeur les détails de la jointure (sauf en cas d'erreur) avec la fonction **many_to_many**.

Pour que **dbicdump** déclare cette liaison **many_to_many**, il faut une table de liaison avec :

- deux colonnes configurées en clé étrangère,
- une clé primaire faite avec ces 2 colonnes.

Note

Attention : la génération de la relation **many_to_many** ne fonctionne qu'à partir de la version 0.07015 de **DBIx::Class::Schema::Loader**.

À défaut, vous pouvez ajouter vous-même cette relation dans la zone « libre » de la classe générée par **dbicdump**. Voir **[MANYTOMANY]** pour les détails.

Revenons-en au programme Perl. Il faut maintenant associer le tarball avec ses paquets dans la base de données. Plusieurs cas de figure peuvent se présenter :

- Vous êtes certain que le paquet n'existe pas dans la base. Dans ce cas, l'ajout est très simple :

```
$tarball->add_to_packages({ name => 'concombre-1.1.el5.i386.rpm'} ) ;
```

L'inconvénient de cette commande est qu'elle va chercher à créer une nouvelle ligne dans la table « Packages ». Si ce paquet est déjà connu, la base refusera l'insertion, car la colonne « name » a un index unique.

- Le paquet existe dans la base, mais la relation entre le tarball et le paquet n'existe pas. La méthode **find_or_create_related** est capable de gérer ce package qui existe peut-être :

```
$tarball_obj->find_or_create_related('tarball_packages',
      { package => { name => 'courgette-1.2.el5.i386.rpm' }}) ;
```

- Le paquet et la relation peuvent exister dans la base. C'est le cas à considérer si vous voulez obtenir un script idempotent. Dans ce cas, il faut toujours vérifier l'existence de la

relation et utiliser la méthode **find_or_create_related** comme avant :

```
my $pkg_name2 = 'courgette-1.2.el5.i386.rpm' ;
if (not $tarball_obj->packages->search({name => $pkg_name2})->count) { ;
    $tarball_obj->find_or_create_related('tarball_packages',
        { package => { name => $pkg_name2 }}) ;
}
```

Le premier cas de figure est le plus facile à traiter, mais le moins souple. À vous de choisir ce qui conviendra le mieux à votre application.

1.11 Et les jointures ?

Supposons que mon chef me demande maintenant de pourvoir lister les versions de produits qui contiennent des corrections de sécurité. Il paraît que c'est facile, les logs du paquet contiennent la mention du CVE corrigé. Il faudra, pour chaque version de produit, aller chercher le tarball associé, ses packages et chercher CVE dans le log.

Il faut donc :

- Modifier le schéma de la table « Packages » pour ajouter une colonne « log »,
- Régénérer les classes d'interface vers la base,
- Scanner les paquets pour récupérer leur log et le stocker dans la base,
- Enfin, construire une requête avec une jointure pour extraire l'information de la base.

Les trois premières étapes ne posant pas de problème particulier, on va se concentrer sur la dernière. Il faut utiliser une jointure basée sur les relations mises en place par **DBIx::Class** :

```
my $cve_rs = $integ_schema->resultset('ProductVersion')->search(
   {
       'package.log' => { like => '%CVE%'}
   },
   {
       join => { tar_balls => { tarball_packages => 'package' }}
   }
);
```

L'appel à **search** prend deux paramètres sous forme de « hash ref ». La jointure est spécifiée dans le deuxième « hash » : la première difficulté est que la jointure passe au travers d'une relation « many-to-many » (entre les tarballs et les paquets). **DBIx::Class** ne sait pas gérer la jointure en utilisant directement la relation « many-to-many » entre « TarBalls » et « Packages » (matérialisée par la méthode **packages** de la classe **Integ::Schema::Result::Tarball**).

La jointure est donc spécifiée avec les relations plus simples « has_many » et « belongs_to ». Chaque relation à traverser est représentée par un « hash ref » : il faut spécifier la jointure à partir de la table « ProductVersions » (**join => {**), trouver les tarballs associés (**tar_balls => {**), utiliser la relation **tarball_packages** vers la table de liaison et enfin, spécifier la dernière relation vers la table « Packages », (**tarball_packages => 'package'**).

Le premier paramètre du **search** contient juste le critère de recherche. Comme l'attribut **log** ne fait pas partie de la table « ProductVersions », mais de la table « Packages », il faut spécifier le nom de la relation utilisée dans la jointure et le nom de la colonne (**package.log**). Comme on recherche un log qui contient « CVE », un simple critère comme **package.log = 'CVE' >** ne suffit pas, car ce critère spécifie une égalité stricte. Il faut utiliser un hash ref pour spécifier un critère assez proche du SQL : **like = '%CVE%'>**. En SQL, le « % » signifie « n'importe quoi ». On recherche donc une chaîne quelconque contenant « CVE » quelque part.

D'autres formes de jointure sont possibles en jouant sur la structure de données passée au **join**. La doc **DBIx::Class::Manual::Joining** contient beaucoup d'autres possibilités.

2 « Show me the code ! »

Vous trouverez sur **[GITHUBADOD]** un petit script pour générer le schéma (**gen-schema.sh**) et un petit programme de test qui reprend tout le code de cet article (**test-db.pl**).

3 Oui, mais qu'est-ce que je montre à mon chef ?

Résumons : on a une base de données, de quoi y accéder en Perl sans trop se fatiguer. Avec ça, on peut écrire des programmes qui tournent dans un bon vieux terminal. Pour un hacker, c'est parfait. Mais ce n'est pas bien alléchant pour mon chef. Il faudrait lui montrer un truc à la mode, genre une application web qui permette de voir et manipuler les versions de produit.

Pour remédier à ça, je vous propose de découvrir dans un prochain article comment exploiter **DBIx::Class** dans une application web. ■

Remerciements

Les Mongueurs de Perl pour leur accueil et la relecture de cet article.

Liens

[DBICLASS] Class::DBI, un ORM pour Perl : Application à la gestion de DNS, GNU/Linux Magazine n° 96 (http://articles.mongueurs.net/magazines/linuxmag96.html)

[MANYTOMANY] http://search.cpan.org/dist/DBIx-Class/lib/DBIx/Class/Relationship.pm#many_to_many

[GITHUBADOD] https://github.com/dod38fr/glmf-article-dbix-class-web

R & D, Enseignants, voici une offre qui vous est spécialement destinée !

NOUVEAUTÉ 2014 !

L'accès à la base documentaire en ligne totale des Éditions Diamond vous permettra d'effectuer des recherches dans la majorité des articles parus dans nos magazines.

Vous pourrez ainsi trouver l'article indexé qu'il vous faut, puis, par exemple copier/coller les codes etc.

Ces articles seront disponibles avec un décalage de 6 mois après leur parution dans l'ensemble de nos titres.

UNE SOLUTION !

Il y a certainement la réponse dans

LA BASE DOCUMENTAIRE !

...ET ACCÉDEZ À + DE 3500 ARTICLES DE TOUS NOS MAGAZINES !

diamond.com

DES OFFRES BASE DOCUMENTAIRE PAR MAGAZINE SONT DISPONIBLES SUR : **boutique.ed-diamond.com**

CRÉEZ VOTRE PROPRE GESTIONNAIRE DE FLUX SOUS PHP

par Stéphane Mourey [Taohacker]

La gestion des flux est l'une des fonctionnalités avancées de PHP. Il est ainsi possible de configurer leur fonctionnement, de créer un nouveau type de flux, ou même de remplacer la gestion standard d'un type de flux natif par une classe que vous aurez écrite vous-même. Voyons comment procéder.

1 Les flux et leur gestion

Les flux sont une généralisation conceptuelle d'un grand nombre de ressources comme les fichiers, les sockets, les connexions réseau, les données compressées. Elles ont en commun un certain nombre d'opérations de lecture et d'écriture linéaire, ainsi que d'accès à des positions arbitraires dans leurs contenus. Cette généralisation permet à PHP d'utiliser les mêmes instructions pour effectuer les mêmes opérations sur des flux de nature radicalement différente. Ainsi, PHP pourra recourir à `file_get_contents()` pour lire un fichier du système de fichiers local, le contenu d'une page web, un fichier sur un serveur FTP... Selon le type de flux, PHP utilise le gestionnaire adapté.

Lorsque vous accédez à une ressource à l'aide de `fopen()` par exemple, vous lui passez une URL telle que http://www.unixgarden.com. Pour PHP, la partie de l'URL indiquant le protocole (`http`) de communication est généralisée dans le concept de *scheme* qui est alors le nom du gestionnaire ou *wrapper* que PHP utilisera pour accéder à la ressource. Ainsi le scheme `zip` permet, si le serveur le supporte (certains gestionnaires ne sont pas activés par défaut), d'accéder au contenu d'une archive au format ZIP avec les fonctions usuelles d'accès aux fichiers. Si le scheme n'est pas précisé, alors celui par défaut est utilisé, à savoir `file`, le gestionnaire d'accès aux fichiers locaux.

Il est possible de placer des filtres sur les flux pour contrôler le contenu, tant lu qu'écrit, par extension de la classe `php_user_filter` (http://fr2.php.net/manual/fr/class.php-user-filter.php). Vous pourrez paramétrer le comportement de vos flux selon le scheme à l'aide des contextes de flux (http://fr2.php.net/manual/fr/stream.contexts.php) : vous serez ainsi heureux d'apprendre que vous pourrez configurer un proxy pour vos connexions HTTP... Malheureusement, le contexte n'a pas de portée propre, il faut le passer en paramètre des fonctions qui le supportent comme `fopen()`... Ce qui est regrettable à double titre : tout d'abord, si vous avez besoin d'un proxy, ce sera sans doute toujours le cas, et il vous faudra quand même penser à ajouter le contexte à vos fonctions d'ouverture de fichiers ; ensuite, si vous utilisez une application développée par un tiers, vous n'aurez plus qu'à passer son code en revue pour déterminer à quels endroits il faut que vous ajoutiez votre contexte !

À moins de créer votre propre gestionnaire et de lui permettre d'aller lui-même chercher son contexte. Car voilà la possibilité la plus intéressante offerte par PHP en ce qui concerne la gestion des flux : celle d'écrire son propre gestionnaire. Celui-ci peut même remplacer le gestionnaire natif, tout en s'appuyant sur lui d'ailleurs, par exemple pour lui passer votre contexte sans qu'il y ait besoin de modifier le code autrement que pour la mise en place de ce nouveau gestionnaire.

2 Mon premier gestionnaire : le wrapper identité

D'un point de vue méthodologique, le meilleur moyen de découvrir le fonctionnement d'un gestionnaire est sans doute de commencer par en développer un qui remplace un des gestionnaires standards, de telle sorte que le comportement de PHP n'en soit en rien affecté. Cela ajoute une difficulté, il nous faudra également gérer le changement de gestionnaire *à l'intérieur de la classe*. Par la suite, ce gestionnaire nous fournira une base idéale pour construire de nouveaux gestionnaires : tout y sera déjà défini, nous n'aurons

plus qu'à *modifier* le comportement, et non l'inventer de toutes pièces.

Concrètement, qu'est-ce qu'un gestionnaire ? Il s'agit d'une classe comprenant un certain nombre de méthodes plus ou moins obligatoires. Ces méthodes sont utilisées par PHP pour effectuer les opérations demandées par les fonctions d'accès aux flux. Par exemple, lors d'un appel à **fwrite()**, PHP appellera la méthode **stream_write**. Le gestionnaire que nous souhaitons implémenter fera donc simplement le lien entre la méthode du gestionnaire appelée et la fonction standard correspondante. Ce lien sera la plupart du temps très simple à établir, mais dans certains cas, il faudra se montrer un peu plus subtil : les fonctions standards et les méthodes du gestionnaire n'ont pas toujours la même signature, ni le même type de valeur de retour. J'ai personnellement eu quelques difficultés à lier **fopen()** avec **stream_open()**, jusqu'au moment où j'ai réalisé que **fopen()** renvoie un identifiant de ressource pointant sur le fichier ouvert alors que **stream_open()** renvoie un booléen... PHP ne me renvoyait aucune erreur, mais le comportement des fonctions suivantes d'accès au fichier était pour le moins étrange. Il faut donc se montrer assez précautionneux.

Pour que notre gestionnaire ne provoque pas une boucle infinie, il faudra qu'il se désactive à l'intérieur de lui-même. Mais comme les instructions qui enregistrent et effacent un gestionnaire de flux ont une portée globale, il faut le faire à l'entrée et à la sortie de chaque méthode... Qui plus est, il nous faut prendre garde à restaurer le gestionnaire précédent, et même l'enregistrement successif de plusieurs gestionnaires... Le mieux nous a semblé de rassembler le code nécessaire à ces actions dans les méthodes **unwrap** et **wrap**, que nous appellerons respectivement à l'entrée et à la sortie de chaque méthode.

Si le cœur vous en dit, vous pouvez écrire cette classe vous-même, cela fera un excellent moyen de comprendre le fonctionnement des gestionnaires de flux.

Les plus malins d'entre vous auront recours à une méthode magique pour intercepter les appels à ces méthodes et factoriser mieux la suppression et le rétablissement du gestionnaire de la même façon que nous avions mis en place un cache de méthodes dans l'article *PHP5 : La Magie Continue* (GLMF n° 153). Nous ne le ferons pas ici, pour ne pas compliquer la compréhension.

Voici le code de la classe :

```
<?php

class core_taoIdentityStreamWrapper {

  protected static $protocol;
  protected static $nbWrap = 0;
  protected $handle;
  protected $path;
  public $context;

  static function wrap() {
    if (in_array(static::$protocol,stream_get_
wrappers()))
      stream_wrapper_unregister(static::$protocol);
    stream_wrapper_register(static::$protocol,
get_called_class());
    static::$nbWrap++;
  }

  static function unwrap() {
    if (static::$nbWrap) {
      stream_wrapper_restore(static::$protocol);
      stream_wrapper_unregister(static::$protocol);
      static::$nbWrap--;
    }
  }

  function dir_closedir() {
    static::unwrap(static::$protocol);

    closedir($this->handle);

    static::wrap(static::$protocol);
    return true;
  }

  function dir_opendir($path,$options) {
    static::unwrap(static::$protocol);

    $this->path = $path;
    $this->handle = opendir($path,$this->context);
    $t = is_resource($this->handle);
    static::wrap(static::$protocol);
    return $t;
  }

  function dir_readdir() {
    static::unwrap(static::$protocol);

    $t=readdir($this->handle);
    static::wrap(static::$protocol);
    return $t;
  }

  function dir_rewinddir() {
    static::unwrap(static::$protocol);

    rewinddir($this->handle);
    static::wrap(static::$protocol);
    return true;
}

  function mkdir($path,$mode,$options) {
    static::unwrap(static::$protocol);

    $this->path = $path;
    $recursive=(bool)($options & STREAM_MKDIR_
RECURSIVE);
    $t=mkdir($path,$mode,$recursive,$this-
>context);
    static::wrap(static::$protocol);
    return $t;
  }

  function rename($pathFrom,$pathTo) {
    static::unwrap(static::$protocol);

    $t=rename($pathFrom,$pathTo,$this->context);
    static::wrap(static::$protocol);
    return $t;
  }

  function rmdir($path,$options) {
    static::unwrap(static::$protocol);

    $t=rmdir($path,$this->context);
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_cast($cast_as) {
    static::unwrap(static::$protocol);

    $t=$this->handle;
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_close() {
    static::unwrap(static::$protocol);

    fclose($this->handle);
    static::wrap(static::$protocol);
}

  function stream_eof() {
    static::unwrap(static::$protocol);

    $t=feof($this->handle);
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_flush() {
    static::unwrap(static::$protocol);

    $t=fflush($this->handle);
```

```
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_lock($operation) {
    static::unwrap(static::$protocol);

    $t=flock($this->handle,$operation);
    return $t;
  }

  function stream_metadata($path, $option, $var) {
    static::unwrap(static::$protocol);

    switch ($option) {
      case STREAM_META_TOUCH:
        $t=touch($this->path,$var[0],$var[1]);
        break;
      case STREAM_META_OWNER_NAME:
        $t=chown($this->path,$var);
        break;
      case STREAM_META_OWNER:
        $t=chown($this->path,$var);
        break;
      case STREAM_META_GROUP_NAME:
        $t=chgrp($this->file,$var);
        break;
      case STREAM_META_GROUP:
        $t=chgrp($this->file,$var);
        break;
      case STREAM_META_ACCESS:
        $t=chmod($this->file,$var);
        break;
    }
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_open($path, $mode, $options,
&$opened_path) {
    static::unwrap(static::$protocol);

    $usePathOption = (bool)($options & STREAM_
USE_PATH);
    $this->path = $path;
    $this->handle = fopen($path,$mode,$usePathOpti
on,$this->context);

    if ($usePathOption)
      $opened_path = realpath($path);
    static::wrap(static::$protocol);
    return is_resource($this->handle);
  }

  function stream_read($count) {
    static::unwrap(static::$protocol);

    $t=fread($this->handle,$count);
    static::wrap(static::$protocol);
    return $t;
  }
  function stream_seek($offset, $whence) {
    static::unwrap(static::$protocol);

    $t=fseek($this->handle,$offset,$whence)===0?t
rue:false;
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_set_option($option,$arg1,$arg2) {
    static::unwrap(static::$protocol);

    switch ($option) {
      case STREAM_OPTION_BLOCKING:
        $t=stream_set_blocking($this->handle,
$arg2);
        break;
      case STREAM_OPTION_READ_TIMEOUT:
        $seconds = floor($arg2/1000);
        $microseconds = $arg2-$seconds;
        $t=stream_set_timeout($this->handle,$second
s,$microseconds);
        break;
      case STREAM_OPTION_WRITE_BUFFER:
        $t=stream_set_write_buffer($this->handle,
$arg2);
        break;
      default:
        $t=false;
    }

    static::wrap(static::$protocol);
    return $t;
  }

  function stream_stat() {
    static::unwrap(static::$protocol);

    $t=fstat($this->handle);
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_truncate($new_size) {
    static::unwrap(static::$protocol);

    $rftruncate($this->handle,$new_size);
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_tell() {
    static::unwrap(static::$protocol);

    $t=ftell($this->handle);
    static::wrap(static::$protocol);
    return $t;
  }

  function stream_write($data) {
    static::unwrap(static::$protocol);

    $t=fwrite($this->handle,$data);
    static::wrap(static::$protocol);
    return $t;
  }

  function unlink($path) {
    static::unwrap(static::$protocol);

    $t=unlink($path,$this->context);
    static::wrap(static::$protocol);
    return $t;
  }

  function url_stat($path,$flags) {
    static::unwrap(static::$protocol);

    $function = ($flags & STREAM_URL_STAT_
LINK)?'lstat':'stat';
    $t=$function($path);
    static::wrap(static::$protocol);
    return $t;
  }
}
```

Ouf ! Voilà un beau morceau de code pour obliger PHP à faire exactement la même chose que ce qu'il faisait sans lui...

Voyons maintenant un peu comment nous en servir.

3 Première utilisation : généralisation du contexte

Nous avons vu au début de cet article que PHP permet de définir des options sur les flux susceptibles de modifier leur comportement. Ces options sont regroupées au sein de ce qu'il est convenu d'appeler des contextes de flux. Ces contextes sont construits à partir de tableaux à l'aide de la fonction **`stream_context_create`** (et peuvent être éventuellement modifiés ensuite à l'aide de la fonction **`stream_context_set_option`**). Ainsi, vous pourrez indiquer à PHP d'utiliser un proxy HTTP à l'aide du code suivant :

```
<?php
$opts = array(
  'http' => array(
    'proxy' => 'tcp://mon_proxy:8080',
    'request_fulluri'=>true,
  )
);
$context = stream_context_create($opts);
// le contexte est créé, maintenant
// utilisons-le :
file_get_contents("http://www.example.com",
false,$context);
```

Remarquez que pour utiliser le contexte, nous avons besoin de le passer en option

aux fonctions qui l'acceptent. On peut y voir là une forte limitation à l'utilisation de cette fonctionnalité, car lorsque vous utilisez une application tierce, celle-ci n'est pas forcément consciente que vous voudrez lui faire utiliser un contexte. La solution immédiate consisterait à scanner entièrement le code pour découvrir toutes les occurrences des fonctions susceptibles d'utiliser un contexte et de les modifier selon le besoin...

L'autre solution consiste à définir le contexte sur un gestionnaire identité dédié aux flux HTTP :

```
class httpStreamWrapperWithProxy extends
core_taoIdentityStreamWrapper {
  protected static $protocol='http';
  protected static $contextOpts = array(
    'http' => array(
      'proxy' => 'tcp://localhost:8080',
      'request_fulluri'=>true,
    )
  );
  protected static $staticContext;

  function __construct() {
    if (!is_resource(static::$staticContext)
|| get_resource_type(static::$staticContext)
!='stream-context') {
      static::$staticContext = stream_
context_create(static::$contextOpts);
    }
    $this->context = &static::$staticContext;
  }
}
```

Pour tester la classe, vous pouvez utiliser ce morceau de code :

```
httpStreamWrapperWithProxy::wrap();
file_get_contents("http://www.google.fr","r");
```

Si vous n'avez pas de proxy sur votre port 8080, vous devrez alors obtenir une erreur d'échec lors de l'ouverture de la connexion, que vous ne devriez pas avoir en supprimant la ligne contenant le **`wrap()`**.

4 Mise sous surveillance du code

Dans notre article sur les espaces de noms (*PHP : Explorez les espaces de noms*, GLMF n° 158), nous avions évoqué la possibilité de les utiliser de manière détournée pour mettre le code sous surveillance. Nous avions alors regretté de ne pouvoir soumettre certaines instructions particulièrement sensibles à cette surveillance car il ne s'agissait pas à proprement parler de fonctions : il s'agissait, par exemple, de l'instruction **`include`**, particulièrement redoutable.

L'utilisation des gestionnaires de flux permet de combler cette brèche : il suffit d'intercepter les tentatives d'ouverture de fichiers et d'examiner ce qu'elles appellent et comment.

Il serait intéressant de généraliser notre classe **`core_taoIdentityStreamWrapper`** de sorte qu'elle puisse prendre en charge un nombre indéterminé de schemes et ainsi, étendre la surveillance facilement à tous les protocoles retournés par **`stream_get_wrappers()`**, tout en s'arrangeant pour que **`wrap()`** et **`unwrap()`** à l'entrée et à la sortie de chaque méthode n'agissent que sur le protocole concerné, afin de ne pas grever les performances, mais cela compliquerait inutilement les explications données ici...

Nous n'illustrerons cette méthode que pour le gestionnaire **`http`**.

```
class fileStreamWrapperForMonitoring extends
core_taoIdentityStreamWrapper {
  protected static $protocol='http';

  function stream_open($path,$mode,$option,&
$opened_path) {
    if (!static::checkCallerAndPath(debu
g_backtrace()[1]['function'],$path))
      return false;
    if (is_null($opened_path))
      return parent::stream_
open($path,$mode,$option);
    return parent::stream_open($path,$mode,$
option,$opened_path);
  }

  function checkCallerAndPath($caller,$pa
th) {
    if ($caller=='include' &&
stripos($path,'http')===0) {
      echo 'ALERTE!';
      //lancer les alertes nécessaires...
      return false;
    }
    return true;
  }
}

fileStreamWrapperForMonitoring::wrap();
include('http://localhost/test1.php');
```

Nous avons inclus dans notre classe une méthode **`checkCallerAndPath`** simplifiée autant que possible dans le but d'illustrer notre propos ; une véritable surveillance demandera naturellement un travail bien plus élaboré.

Notre méthode se contente de vérifier si la fonction appelante est **`include`** et si le chemin demandé commence par les caractères **`http`**. Si oui, il convient de lancer une alerte et de renvoyer **`false`**. La nouvelle méthode **`stream_open`**, quant à elle, détermine quelle est la fonction appelante (à l'aide de **`debug_backtrace()`** et demande à **`checkCallerAndPath()`** de vérifier si cet appel est légal. S'il ne l'est pas, l'ouverture du fichier ne se fait pas. S'il l'est, l'appel est transmis à la méthode homonyme de la classe parente, qui prendra en charge l'ouverture effective du fichier.

Notez que nous interrogeons la fonction **`debug_backtrace`** comme s'il s'agissait d'un tableau, alors que c'est *son résultat* qui est un tableau. Cette nouvelle syntaxe est autorisée depuis PHP 5.5.

Vous n'avez plus qu'à généraliser pour surveiller toutes les ouvertures de fichiers.

Conclusion

J'ai fait mentir le titre de mon article, puisque nous n'avons pas, à proprement parler, abordé la création d'un nouveau gestionnaire de flux. En réalité, nous avons utilisé cette possibilité offerte par PHP pour pallier certaines autres difficultés. Il n'en reste pas moins que, si vous avez tout suivi jusqu'ici, votre compréhension de ce mécanisme est maintenant telle qu'il sera pour vous trivial d'implémenter votre propre scheme et de donner à PHP un air de FUSE (*File system in User SpacE* : http://fuse.sourceforge.net)... ■

C++ STANDARD LIBRARY / STL REPARTONS SUR DE BONNES BASES (SUITE)

par Laurent Navarro
[Développeur/formateur freelance – Altidev Toulouse]

Cet article est le second d'une série de deux articles ayant pour but de vous aider à repartir sur de bonnes bases avec la librairie standard C++, aussi bien en C++ 98 qu'en C++ 11.

Dans GLMF n° 165, nous avons traité des chaînes de caractères (**string**) et des collections séquentielles (**vector**, **list**, **deque**), en passant par une petite explication sur les espaces de nommage (**namespace**). Je vous propose donc aujourd'hui de continuer notre visite de la librairie standard C++.

1 Les conteneurs associatifs

La STL met à notre disposition 2 conteneurs associatifs, les **map** et les **multimap** ; nous allons nous concentrer sur la **map** qui me paraît être la plus intéressante.

1.1 Présentation

La classe **map** est une classe qui implémente un tableau associatif générique pouvant être indexé par une clé générique elle aussi. En termes d'implémentation, c'est généralement un arbre (binaire, bicolore rouge/noir ou AVL) qui est utilisé, étant donné que la norme impose un accès aléatoire avec une complexité logarithmique et que les clés soient triées. Le type de la clé doit être ordonnable en implémentant l'opérateur **<**, sinon il faut fournir un comparateur au constructeur.

C++ 11 introduit des versions utilisant des tables de hachage (**unordered_map** et **unordered_multimap**) ; pour utiliser ces classes, la clé doit être hachable (les types entiers et **string** le sont).

Un des cas fréquents de l'utilisation d'une **map** est de créer un tableau associatif avec une chaîne de caractères, nous allons donc illustrer son utilisation ainsi :

```
#include <iostream>
#include<map>

using namespace std;

void TestMap()
{
  // Déclaration d'une map clé de type string,
  // valeur de type string
  map<string,string> Capitales;
  Capitales["France"]="Paris";
  Capitales["Belgique"]="Bruxelles";
  Capitales["Espagne"]="Madrid";
  Capitales["Angleterre"]="Londres";
  cout<<"Capitale de la France =
"<<Capitales["France"]<< endl;
}
```

Ce qui nous donnera la sortie suivante :

```
Capitale de la France = Paris
```

Je vois les yeux du développeur PHP qui est en vous briller.

Comme vous l'aurez compris, l'opérateur **[]** permet d'accéder à un élément par sa clé aussi bien en lecture qu'en écriture (en fait, il retourne une référence sur l'objet contenu).

Mais vous vous demandez déjà ce qu'il adviendra si la clé n'existe pas ? Eh bien, une entrée sera ajoutée et l'élément sera créé avec le constructeur par défaut. Ainsi, le code suivant :

```
cout<<"Capitale de l'Allemagne
="<<Capitales["Allemagne"]<< endl;
```

ne provoque pas d'erreur, mais affiche une chaîne vide à droite du signe égal comme ceci :

```
Capitale de l'Allemagne =
```

Si vous souhaitez vous prémunir de la création accidentelle d'éléments, utilisez la méthode **at**, qui comme pour **vector**, lèvera une exception **std::out_of_range** si la clé n'existe pas :

```
cout<<"Capitale de l'Allemagne ="<<Capitales.
at("Allemagne")<< endl;
```

Sinon, **at** retournera une référence comme l'opérateur crochet. Exemple d'affectation :

```
Capitales["Allemagne"]="TBD"; // Insère ou modifie
Capitales.at("Allemagne")="Berlin"; // Modifie seulement
```

Je sens que vous commencez à vous habituer à voir une méthode en *Left-Value*.

Attention, la méthode **at** ne peut pas être utilisée pour insérer une nouvelle

valeur. Elle n'est spécifiée qu'en C++ 11, mais est déjà disponible en C++ 98 dans GCC.

Si vous souhaitez vérifier la présence d'une clé dans la collection, la solution la plus simple sera d'utiliser la méthode **count**, qui vous retournera 0 ou 1 :

```
if(Capitales.count("Belgique"))
    cout<<"Belgique dans la liste"<< endl;
```

Notons que C++ 11 introduit la construction avec une liste d'initialisation, comme ceci :

```
map<string,string> Capitales{{"France","Paris"},{"Belgique","Bruxelles"}};
```

1.2 Les itérateurs

Tout comme les conteneurs séquentiels, les conteneurs associatifs mettent à notre disposition des itérateurs. Le fonctionnement est similaire, au petit détail près que cet itérateur nous permettra d'accéder à une paire de valeurs qui seront la clé et la valeur. Elle sont accessibles respectivement avec les attributs **first** et **second** (oui, c'est vrai, **key** et **value** auraient été plus intuitifs, mais l'itérateur retourne un **std::pair**).

```
map<string,string>::iterator it;
for(it=Capitales.begin();it!=Capitales.end();it++)
    cout<<it->first<<" : "<<it->second<< endl;
```

Nous donne donc :

```
Allemagne : Berlin
Angleterre : Londres
Belgique : Bruxelles
Espagne : Madrid
France : Paris
```

Vous remarquez que la liste est bien triée dans l'ordre des clés, alors que nous avons inséré les éléments dans le désordre.

Comme avec les autres conteneurs, **end()** retourne un itérateur invalide qui sert de valeur spéciale. Par exemple, la méthode **find** permet de chercher une clé et retourne **end()** si elle ne la trouve pas et sinon, un itérateur permettant d'accéder à la valeur.

```
it=Capitales.find("Belgique");
if(it!=Capitales.end())
    cout<<"Belgique dans la liste et sa capitale est "<<it->second<< endl;
```

Exemple de parcours avec C++ 11 en utilisant **auto** et la boucle **for** sur intervalle :

```
for(auto &C : Capitales)
    cout<<C.first<<" : "<<C.second<< endl;
```

Comme vous pouvez le constater, c'est un peu plus concis.

1.3 Manipulation des éléments

En plus de l'opérateur **[]**, la méthode **insert** permet d'insérer un élément qui n'existe pas déjà. S'il existe, l'insertion échoue de façon silencieuse (et donc, ne met pas à jour la valeur associée à la clé). La méthode **insert** attend une paire de valeurs dans un objet **std::pair**. Voilà deux façons de procéder à l'insertion :

```
Capitales.insert(map<string,string>::value_type("Italie","Rome"));
Capitales.insert(pair<string,string>("Italie","ROME"));
```

En C++ 11 c'est plus simple, grâce à la liste d'initialisation de la classe **pair** :

```
Capitales.insert({"Italie","Rome"});
```

Pour supprimer un élément avec la méthode **erase**, il vous faudra le désigner par son itérateur, ce qui est bien moins pratique que de le désigner par sa clé, j'en conviens ; la méthode **find** pourra vous y aider, mais vous devrez veiller à n'effacer l'élément que si vous le trouvez.

```
it=Capitales.find("Belgique");
if(it!=Capitales.end())
    Capitales.erase(it);
```

La méthode **clear()** est plus simple d'usage puisqu'elle vide le conteneur en entier.

La méthode **empty()** permet de savoir si le conteneur est vide, **size()** permet de connaître le nombre d'éléments dans le conteneur.

Pour récupérer les valeurs entre 2 bornes, **lower_bound** et **upper_bound** pourront vous aider :

```
for(it=Capitales.lower_bound("E");it!=Capitales.upper_bound("G");it++)
    cout<<"Entre E et G : "<<it->first<<" : "<<it->second<< endl;
```

```
Entre E et G : Espagne : Madrid
Entre E et G : France : Paris
```

La classe **map** étant doublement générique (clé et valeur), les écritures de déclaration sont souvent fastidieuses (surtout pour les itérateurs et les paires via **value_type**). En C++ 11, **auto** nous évitera ces notations dans certains cas, mais en C++ 98, on aura parfois intérêt à utiliser **typedef** pour simplifier l'écriture des cas les plus fréquemment utilisés, par exemple :

```
// Déclaration des types
typedef map<string,string> MapSS;
typedef map<string,string>::iterator MapSSit;
// Déclaration des variables
MapSS Capitales;
MapSSit it;
```

1.4 Manipulation de multimap

La classe **multimap** permet de gérer des collections de clés/valeurs ordonnées avec doublon sur les clés. Je n'y trouve pas un très grand intérêt, mais voilà tout de même un petit exemple :

```
#include <iostream>
#include<map>

using namespace std;
void TestMultiMap()
{
multimap<string,string>  Villes;
Villes.insert(pair<string,string>("France","Paris"));
Villes.insert(pair<string,string>("France","Toulouse"));
Villes.insert(pair<string,string>("Espagne","Madrid"));
Villes.insert(pair<string,string>("Espagne","Barcelonne"));
Villes.insert(pair<string,string>("Italie","Rome"));
Villes.insert(pair<string,string>("Italie","Naples"));

multimap<string,string>::iterator it;
for(it=Villes.begin();it!=Villes.end();it++)
    cout<<it->first<<" : "<<it->second<< endl;

cout<<"Il y a "<<Villes.count("France")<<" villes en France"<< endl;
```

```
for(it=Villes.lower_bound("France");it!=Villes.upper_bound("France");it++)
    cout<<"(bound)En France il y a "<<it->second<< endl;

it=Villes.find("France");
if(it!=Villes.end())
  for(;it!=Villes.upper_bound("France");it++)
    cout<<"(bound2)En France il y a "<<it->second<< endl;

pair<multimap<string,string>::iterator,multimap<string,string>::iterator> R;
R=Villes.equal_range("France");
for(it=R.first;it!=R.second;it++)
    cout<<"(equal) En France il y a "<<it->second<< endl;
}
```

Ce qui donne :

```
Espagne : Madrid
Espagne : Barcelonne
France : Paris
France : Toulouse
Italie : Rome
Italie : Naples
Il y a 2 villes en France
(bound)En France il y a Paris
(bound)En France il y a Toulouse
(bound2)En France il y a Paris
(bound2)En France il y a Toulouse
(equal) En France il y a Paris
(equal) En France il y a Toulouse
```

Pour l'insertion et l'affichage, ça ressemble beaucoup à une **map**, mais pour récupérer les valeurs d'une clé, il y a plusieurs options, mais aucune n'est franchement pratique.

lower_bound et **upper_bound** permettent de manipuler des intervalles autour de bornes.

find permet de rechercher une clé, mais ne retourne que le premier élément.

equal_range permet de retourner une paire d'itérateurs sur l'intervalle d'éléments ; avec **auto** en C++ 11 ça ira, mais en C++ 98, la déclaration de **R** est un peu à rallonge.

1.5 Utilisation des ensembles

La classe générique **set** permet de gérer un ensemble de valeurs uniques et ordonnées. L'utilisation est similaire aux clés d'une **map**, hormis que l'itérateur est simple. Petit exemple :

```
#include<set>
void TestSet()
{
set<string> noms;
noms.insert("Paul");
noms.insert("Pierre");
noms.insert("Jacques");
set<string>::iterator it;
for(it=noms.begin();it!=noms.end();it++)
    cout<<*it<< endl;
if(noms.find("Paul")!=noms.end())
    cout<<"Paul est dans la liste"<< endl;
}
```

Ce qui donne :

```
Jacques
Paul
Pierre
Paul est dans la liste
```

multiset est le croisement d'un **set** et d'un **multimap**, je vous laisse le soin de le découvrir par vous-même, car je ne lui trouve pas un très grand intérêt.

unordered_set et **unordered_multiset** sont les versions utilisant une table de hachage de **set** et **multiset**.

1.6 Les bitset

Je voudrais à présent vous parler d'un conteneur un peu spécial, puisqu'il ne peut contenir que des bits : il s'agit de **bitset**, qui est plutôt classé dans la catégorie des utilitaires.

Cette classe est générique pour spécifier sa taille, elle peut contenir autant de bits que vous le souhaitez et occupe une taille multiple de 32 bits sur mon PC 32 bits. Elle permet une conversion aisée depuis/vers des entiers 32 bits et même 64 bits depuis C++ 11, des chaînes de caractères où les symboles **0** et **1** sont personnalisables depuis C++ 11.

Vous pouvez accéder aux bits en lecture/écriture avec l'opérateur **[]**, ou avec les fonctions **set**, **reset** et **test**. Les opérateurs bit à bit sont disponibles entre **bitset** de même taille. Les numéros des bits sont indexés à partir de zéro.

L'essentiel est dit, un petit exemple pour illustrer tout ça :

```
#include <bitset>
// Initialisation d'un bitset 16 bits avec la valeur 54
bitset<16> bs1(54);
bs1[9]=1;     // Écriture du Bit 9
bs1.set(7);   // Écriture à 1 du Bit 7
bs1.reset(6); // Écriture à 0 du Bit 6
// Conversion vers texte
cout <<"bs1 as text : "<< bs1.to_string() <<endl;
// Conversion vers unsigned long
cout <<"bs1 as uint : "<< bs1.to_ulong()<<endl;
// Lecture d'un bit
cout <<"bs1 test b1 : "<< bs1.test(1) <<endl;
// ET bit à bit puis affichage dans une stream
cout <<"bs1 & 3     : "<< ( bs1&bitset<16>(3)) <<endl;
// Initialisation à partir d'une chaîne, cast explicite vers string obligatoire
bitset<16> bs2(string("010101"));
cout <<"bs2 0 & 1 = "<<bs2<<endl;
// Conversion en chaîne avec 0=_ et 1=.
cout <<"bs2 _ & . = "<<bs2.to_string('_','.')<<endl;
```

Ce qui nous donne ceci :

```
bs1 as text : 0000001010110110
bs1 as uint : 694
bs1 test b1 : 1
bs1 & 3     : 0000000000000010
bs2 0 & 1 = 0000000000010101
bs2 _ & . = __________._._.
```

2 Les algorithmes

Nous allons parler à présent des algorithmes génériques présents dans la STL. Ces algorithmes peuvent s'utiliser sur la plupart des conteneurs séquentiels que nous avons étudiés, ainsi que sur les tableaux C. Ils s'utilisent avec les itérateurs.

Nous allons commencer avec un algorithme qui ne requiert pas trop de paramètres dans son utilisation de base. L'algorithme **sort** permet de trier un conteneur avec l'opérateur **<** par défaut et s'utilise ainsi :

```
#include<vector>
#include<algorithm>
void TestAlgo()
{
    // Création d'un vecteur rempli
    const int Valeurs[]={5,3,65,40,25,87,4,3};
    vector<int> V1(Valeurs,Valeurs+sizeof(Valeurs)/
sizeof(*Valeurs));
    PrintContainerCsv("V1=",V1);
    // Tri et affichage du vecteur
    sort(V1.begin(),V1.end());
    PrintContainerCsv("V1=",V1);
}
```

Ce qui nous donne ceci :

```
V1= 5,3,65,40,25,87,4,3
V1= 3,3,4,5,25,40,65,87
```

Pour l'affichage du vecteur, qui va être une tâche récurrente, nous utilisons la fonction générique suivante :

```
template <class T>
void PrintContainerCsv(string Msg,T &Ctnr)
{
    cout<<Msg<<" ";
    typename T::iterator it;
    for(it=Ctnr.begin();it!=Ctnr.end();++it)
        cout<<(it==Ctnr.begin()?"":",")<<*it;
    cout<<endl;
}
```

Pour utiliser un algorithme, il faut l'invoquer en lui transmettant le début et la fin (plus exactement l'élément suivant la fin) de la partie sur laquelle nous souhaitons l'appliquer. Il sera commun de vouloir l'appliquer à tout le conteneur ; dans ce cas, il suffira d'utiliser **begin()** et **end()**.

Certains algorithmes vont prendre une valeur en paramètre, c'est par exemple le cas de **count**, qui compte le nombre de valeurs égales à la valeur passée en paramètre dans un conteneur.

```
cout<<"Nbr de 3 : "<<count(V1.begin(),V1.
end(),3)<<endl;
```

nous donne logiquement la ligne ci-dessous, puisque la valeur 3 est présente 2 fois dans le vecteur.

```
Nbr de 3 : 2
```

De nombreux algorithmes sont paramétrés, non pas par une simple valeur, mais par un objet fonction faisant office de prédicat ou de traitement.

Un objet fonction, est un bout de code appelable par l'algorithme, qui est passé en paramètre et peut être implémenté par différentes techniques. Nous allons commencer par la plus simple, à savoir le bon vieux pointeur sur fonction C. Nous allons l'illustrer avec l'algorithme **count_if** qui est un cousin de **count**, mais qui prend en paramètre un prédicat unaire, c'est-à-dire un objet fonction prenant un seul paramètre (c'est le côté unaire) et retournant un booléen, ou un entier égal ou différent de 0 (c'est le côté prédicat).

Exemple de prédicat testant si un entier est inférieur à 10 :

```
bool IntInf10(int x)
{
    return x<10;
}
```

qui est donc utilisé ainsi :

```
cout<<"Nbr de <10 (fct) : "<<count_if(V1.
begin(),V1.end(),IntInf10)<<endl;
```

On passe le pointeur de fonction (le nom de la fonction sans les parenthèses) en paramètre à **count_if**, qui va donc évaluer le prédicat en appelant la fonction pour chacune des valeurs comme le montre l'implémentation possible documentée sur **[cppreference]** :

```
template<class InputIt, class UnaryPredicate>
typename iterator_traits<InputIt>::difference_type
    count_if(InputIt first, InputIt last,
UnaryPredicate p)
{
    typename iterator_traits<InputIt>::difference_
type ret = 0;
    for (; first != last; ++first) {
        if (p(*first)) {
            ret++;
        }
    }
    return ret;
}
```

On voit que **first** est l'itérateur utilisé pour parcourir la collection dans la boucle **for** depuis laquelle on appelle le prédicat **p** successivement avec chaque valeur du conteneur ; si le résultat retourné par le prédicat **p** n'est pas nul, alors on incrémente le compteur.

De nombreux algorithmes de la STL ont une structure similaire à celui présenté ici. Si les paramètres sont des objets de taille significative, pensez à spécifier les paramètres de vos objets fonctions comme des références constantes **(const MyClass &obj)**.

L'utilisation d'un pointeur de fonction a l'avantage d'être simple à mettre en œuvre, mais a l'inconvénient que le prédicat sera toujours le même, à savoir, dans le cas présent, évaluer si un entier est inférieur à 10. Si on veut faire un prédicat testant si un entier est inférieur à 20, il nous faudra écrire une nouvelle fonction.

Une seconde technique va permettre de rendre le prédicat paramétrable. Il s'agit d'utiliser une classe implémentant l'opérateur **()**, on appelle aussi cela un *foncteur*.

```
struct ClassInf
{
    int Valeur;
    ClassInf(int _v){Valeur=_v;}
    bool operator ()(int x)
    {
      return x<Valeur;
    }
};
```

Cette classe, entièrement publique grâce à l'utilisation de **struct** au lieu de **class**, contient :

- Un attribut **Valeur** qui a pour rôle de mémoriser la valeur avec laquelle il faut se comparer ;
- Un constructeur qui stocke la valeur passée en paramètre dans l'attribut **Valeur** ;
- Une surcharge de l'opérateur **()** avec un paramètre qui retourne en booléen le résultat de la comparaison entre le paramètre **x** et l'attribut **Valeur**.

Cette classe est utilisable ainsi :

```
ClassInf classinf50(50);
cout<<"25 <50 : "<<classinf50(25)<<endl;
```

On déclare un objet **classinf50** en le construisant avec le paramètre **50** et on appelle l'opérateur **()** en passant une valeur avec laquelle comparer la valeur contenue dans l'objet. Cet objet peut être passé en paramètre de **count_if** :

```
cout<<"Nbr de <50 (obj) : "<<count_if(V1.begin(),
V1.end(),classinf50)<<endl;
```

Mais le plus souvent, on ne nommera pas cet objet et on passera en paramètre des algorithmes un objet non nommé en invoquant seulement le constructeur.

```
cout<<"Nbr de <10 (obj) : "<<count_if(V1.begin(),
V1.end(),ClassInf(10))<<endl;
```

L'utilisation de ce type de classe peut aussi être pratique si vous avez besoin de conserver un état entre 2 appels par l'algorithme. Ceci se fera en stockant cet état dans un attribut, comme le montre cet exemple d'utilisation de l'algorithme **generate**, qui remplit les éléments d'un conteneur à partir des données fournies par l'objet fonction qui donne une suite d'entiers en partant d'une valeur initiale :

```
struct Sequenceur{
    int NextValue;
    Sequenceur(int InitVal)
    {NextValue=InitVal;}
    int operator ()()
    { return NextValue++; }
};

vector<int> V2(10);
generate(V2.begin(),V2.end(), Sequenceur(10));
PrintContainerCsv("V2=",V2);
```

```
V2= 10,11,12,13,14,15,16,17,18,19
```

Sequenceur(10) crée un objet et initialise l'attribut **NextValue** à **10**, puis chaque appel effectué à l'opérateur **()** par **generate** incrémente **NextValue**.

Concernant les prédicats et opérations simples, il est pénible de devoir écrire une classe pour exprimer quelque chose d'aussi simple que « inférieur à une valeur » ou « somme de 2 valeurs ».

Afin de nous simplifier la tâche, il existe des classes dans le header **functional** qui implémentent les opérateurs des expressions du langage C au moyen de foncteurs. Par exemple, la classe **plus** implémente l'opérateur **+** ; on trouvera aussi **minus**, **multiplies**, **divides**, **modulus**, **negate** pour les opérations mathématiques, **equal_to**, **not_equal_to**, **greater**, **less**, **greater_equal**, **less_equal** pour les comparaisons, **logical_and**, **logical_or**, **logical_not** pour les opérations logiques, **bit_and**, **bit_or**, **bit_xor**, **bit_not** pour les opérations bit à bit. Ces classes implémentent une surcharge de l'opérateur **()** prenant en paramètres les 2 opérandes à manipuler.

Pour les prédicats qui requièrent 2 opérandes, comme le prédicat optionnel de **sort**, ces classes seront directement utilisables pour, par exemple, trier en ordre décroissant :

```
sort(V1.begin(),V1.end(),greater<int>());
    PrintContainerCsv("V1=",V1);
```

Ce qui affichera :

```
V1= 87,65,40,25,5,4,3,3
```

greater<int>() crée un foncteur implémentant l'opérateur **>** pour le type **int**.

Pour les prédicats qui requièrent un seul paramètre, il va falloir utiliser en plus un *binder* qui va nous permettre de fixer une des 2 valeurs.

Nous allons par exemple, utiliser **bind2nd** qui va nous permettre de transformer l'objet fonction binaire **less**, en objet fonction unaire, qui contient la valeur avec laquelle on souhaite faire la comparaison ; dit autrement, nous allons câbler le second paramètre à une valeur fixe :

```
cout<<"Nbr de <50 (binder) : "<<count_if(V1.
begin(),V1.end(),bind2nd(less<int>(),50))<<endl;
```

En C++11, cette fonction est dépréciée, il faut à présent utiliser le binder généralisé **bind**, qui prend en paramètres les positionnements des paramètres passés en fonction au moyen de marqueurs placeholders :

```
bind(less<int>(),placeholders::_1,50)
```

qui, si vous ajoutez la ligne suivante :

```
using namespace  std::placeholders;
```

s'écrit ainsi :

```
cout<<"Nbr de <50 (bind) : " <<count_if(V1.
begin(),V1.end(),bind(less<int>(),_1,50))<<endl;
```

On peut s'amuser ainsi à faire >10 et <50 :

```
cout<<"Nbr de >10 et <50 : "<<
    count_if(V1.begin(),V1.end(),
        bind(logical_and<bool>(),bind(less<int>(),
10,_1),bind(less<int>(),_1,50))
    )
    <<endl;
```

Vous voyez que ce n'est pas une façon très simple pour écrire des prédicats complexes.

C++ 11 va nous apporter, avec les expressions lambda, un moyen simple d'écrire un objet fonction pour nos algorithmes, en mettant le code directement dans l'appel de la fonction algorithme :

```
cout<<"Nbr de <50 (lambda) : "<<
   count_if(V1.begin(),V1.end(),[](int x) {
return x<50;})<<endl;
```

C'est comme une fonction C, mais sans le nom et inséré directement en paramètre (voir GLMF hors-série n° 55 pour plus de détails sur les expressions lambda).

2.1 Les algorithmes non-modifiants

Ces algorithmes permettent d'interroger le conteneur sans le modifier.

count et **count_if**, que nous avons précédemment utilisés, permettent de dénombrer les éléments égaux à une valeur ou répondant à un prédicat.

find, **find_if**, **find_if_not** permettent de rechercher une valeur dans la collection et de retourner l'itérateur y faisant référence.

```
vector<int>::iterator it;
it=find(V1.begin(),V1.end(),25);
if(it!=V1.end())
    cout<<"Valeur "<<*it<<" trouvée"<<endl;
else
    cout<<"Valeur non trouvée"<<endl;
```

adjacent_find recherche la première paire de valeurs égales consécutives.

search_n est similaire, mais recherche *n* valeurs consécutives égales.

search permet de rechercher une séquence particulière de valeurs dans le conteneur.

```
vector<int> SearchedValues{13,14};
it=search(V2.begin(),V2.end(),SearchedValues.
begin(),SearchedValues.end());
if(it!=V2.end())
    cout<<"Valeurs trouvées ici : "<<*it<<endl;
else
    cout<<"Valeur non trouvées"<<endl;
```

find_end est similaire à **search**, mais recherche la dernière occurrence de la séquence.

Toutes ces fonctions utilisent, par défaut, le test d'égalité, mais peuvent prendre un objet fonction pour, par exemple, évaluer l'égalité de certains attributs d'une classe seulement.

mismatch permet de rechercher la première différence de valeurs entre 2 séquences.

equals permet de tester si deux séquences sont égales.

min_element retourne un itérateur vers la plus petite valeur d'une séquence.

```
cout<<"Min(V1) = "<<*min_element(V1.begin(),V1.
end())<<endl;
```

max_element est similaire, mais retourne la plus grande valeur.

minmax_element (C++11) retourne une paire d'itérateurs sur les valeurs mini & maxi :

```
auto mm=minmax_element(V1.begin(),V1.end());
cout<<"Min(V1) = "<<*(mm.first)<<" Max(V1) =
"<<*(mm.second)<<endl;
```

```
Min(V1) = 3 Max(V1) = 87
```

min & **max** retournent la plus petite/plus grande valeur entre 2 valeurs.

```
cout<<"min(10,9) = "<<min(10,9)<<endl;
```

C++11 introduit une variante entre *n* valeurs grâce aux initialiseurs.

```
cout<<"min({10,9,14,5}) = "<<min({10,9,14,5})<<endl;
```

accumulate a pour objet de calculer la somme des éléments d'une séquence + une valeur initiale (zéro dans l'exemple ci-dessous) :

```
#include<numeric>
cout<<"Sum(V1) = "<<accumulate(V1.begin(),V1.
end(),0)<<endl;
```

```
Sum(V1) = 232
```

partial_sum calcule les valeurs cumulées d'un conteneur dans un autre :

```
vector<int> Vres(10);
partial_sum(V2.begin(),V2.end(),Vres.begin());
PrintContainerCsv("V2  =",V2);
PrintContainerCsv("Vres=",Vres);
```

Ce qui donne :

```
V2  = 10,11,12,13,14,15,16,17,18,19
Vres= 10,21,33,46,60,75,91,108,126,145
```

adjacent_difference calcule la différence entre chaque paire d'éléments successifs de la collection et la stocke dans une 2ème. C'est en fait l'opération inverse de **partial_sum**.

all_of, **any_of**, **none_of** vont nous permettre de vérifier si un prédicat est vrai pour toutes les valeurs d'une séquence, au moins une, ou aucune. Disponible à partir de C++ 11.

2.2 Les algorithmes de manipulation des éléments

copy permet de copier une séquence de valeurs dans une autre destination qui doit déjà avoir la capacité à la recevoir (ne copiez pas 50 valeurs dans un **vector** de 10, il ne vas pas s'agrandir !).

copy est parfois utilisé avec des **OutputIterator** tels que **ostream_iterator** pour faire un affichage, par exemple :

```
copy(V1.begin(),V1.end(),ostream_iterator<int>
(cout,";"));
cout<<endl;
```

```
87;65;40;25;5;4;3;3;
```

copy_if est équivalent, mais ne fait la copie que si la valeur répond à un prédicat.

fill permet d'affecter une valeur à une séquence.

fill_n affecte une valeur à *n* éléments d'une séquence spécifiée uniquement par son début et sa taille. Cette fonction peut être pratique si on la combine à un itérateur d'insertion qui se chargera de faire croître le conteneur :

```
vector<int> V4;
fill_n(back_inserter(V4),5,3);
PrintContainerCsv("V4 =",V4);
```

```
V4 = 3,3,3,3,3
```

generate, que nous avons déjà étudié, remplit une séquence avec un générateur ; **generate_n** est la version début+taille.

replace remplace les valeurs d'une séquence égale au paramètre par une autre valeur.

replace_if remplace les valeurs répondant à un prédicat par une autre valeur.

replace_copy et **replace_copy_if** effectuent une copie en remplaçant à la volée des valeurs de façon similaire à **replace** & **replace_if**.

for_each permet d'invoquer une fonction unaire pour chaque valeur d'une séquence. Si cette fonction accepte le paramètre par référence comme ici :

```
void Multiplicateur2(int &x)
{    x*=2; }
```

elle peut modifier les éléments de la séquence.

```
for_each(V2.begin(),V2.end(),Multiplicateur2);
PrintContainerCsv("V2  =",V2);
```

```
V2  = 20,22,24,26,28,30,32,34,36,38
```

Avec les expressions lambda, on se croirait presque en Python ;-)

```
for_each(V2.begin(),V2.end(),[](int &x){x*=2;});
```

transform permet de copier une séquence dans une autre en y appliquant une fonction unaire. Ici, un exemple qui soustrait 40 à chaque valeur de **V2** et stocke le résultat dans **V3** :

```
vector<int> V3(10);
transform(V2.begin(),V2.end(),V3.begin(),bin
d2nd(minus<int>(),40));
PrintContainerCsv("V3  =",V3);
```

```
V3  = -20,-18,-16,-14,-12,-10,-8,-6,-4,-2
```

Une seconde version de **transform** permet de combiner 2 séquences et stocke le résultat de la fonction binaire dans une troisième. Ici, une addition de **V2** et **V3** dans **Vres**.

```
transform(V2.begin(),V2.end(),V3.
begin(),Vres.begin(),plus<int>());
PrintContainerCsv("Vres  =",Vres);
```

```
Vres  = 0,4,8,12,16,20,24,28,32,36
```

random_shuffle mélange les éléments d'une collection de façon aléatoire.

```
random_shuffle(V3.begin(),V3.end());
PrintContainerCsv("V3  =",V3);
```

```
V3  = -4,-18,-2,-16,-20,-10,-6,-14,-12,-8
```

rotate effectue un décalage à gauche pour placer un élément désigné en premier.

```
rotate(V3.begin(),V3.begin()+3,V3.end());
```

```
V3  = -16,-20,-10,-6,-14,-12,-8,-4,-18,-2
```

reverse inverse l'ordre de la séquence.

```
reverse(V3.begin(),V3.end());
```

```
V3  = -2,-18,-4,-8,-12,-14,-6,-10,-20,-16
```

Nous allons aborder à présent un algorithme non-intuitif dans son utilisation, il s'agit de **remove_if** qui est pourtant incontournable quand on veut effacer des éléments répondant à un critère.

Essayons d'effacer les valeurs supérieures à -15 de **V3** ainsi :

```
remove_if(V3.begin(),V3.end(),bind2nd(greater<int>(),
-15));
PrintContainerCsv("V3 =",V3);
```

Ce qui donne le résultat suivant :

```
V3 = -18,-20,-16,-8,-12,-14,-6,-10,-20,-16
```

Ce résultat est incompréhensible au premier abord, car il y a toujours 10 valeurs. Certaines sont supérieures à -15, mais certaines ont bien disparu, alors que d'autres se sont dupliquées. La logique de **remove_if** est qu'il regroupe les valeurs à conserver au début de la liste et retourne un itérateur sur le début de la séquence qu'il faut détruire pour raccourcir la liste à l'aide de **erase** par exemple, mais il n'efface rien. Il faut donc écrire ceci :

```
V3.erase(remove_if(V3.begin(),V3.
end(),bind2nd(greater<int>(),-15)),V3.end());
PrintContainerCsv("V3 =",V3);
```

pour obtenir le bon résultat :

```
V3 = -18,-20,-16
```

Je n'ai pas listé tous les algorithmes. Il y a, entre autres, quelques algorithmes ensemblistes qui s'appliquent sur des séquences triées et qui seront souvent utilisés avec un itérateur d'insertion : **merge**, **inplace_merge**, **set_difference**, **set_intersection**, **set_union**.

Toujours sur les séquences triées :

- **lower_bound** & **upper_bound** cherchent les bornes haute et basse relativement à une valeur.
- **binary_search** effectue une recherche dichotomique d'une valeur.

3 Les exceptions

Les exceptions sont un mécanisme de gestion des erreurs répandues dans les langages modernes. Les exceptions sont des objets, certains *frameworks* proposent une hiérarchie d'exceptions assurant les services de base d'une classe d'exception. Si vous développez une librairie, vous voudrez probablement avoir une hiérarchie d'exceptions, mais plutôt que de partir de rien, il sera peut-être pertinent de dériver des exceptions de la librairie C++.

En haut de la hiérarchie des exceptions on trouve la classe **std::exception**, qui a pour principal rôle de définir la méthode :

```
virtual const char* what() const noexcept;
```

qui permet de récupérer une chaîne détaillant l'exception. Le message retourné va, suivant les cas, du nom de la classe d'exception, ou de quelque chose s'en approchant, à un message explicite circonstancié (je ne vous cache pas que dans la libC++ c'est plutôt le premier cas).

Voilà un exemple de capture d'une exception de la libC++ :

```
try{
 //Allocation mémoire impossible
 new int[-1];
}catch(exception &e)
{
    cout<<"Exception : what()="<<e.what()<<endl;
}
```

qui affichera ceci :

```
Exception : what()=std::bad_alloc
```

Dériver de la classe **exception** vous permettra d'être capturé par le **catch** ci-dessus, mais il faut reconnaître que le service rendu par la classe **exception** est minime ; en fait, elle sert surtout à définir une interface.

Le header **<stdexcept>** définit deux classes similaires héritières de **exception** :

logic_error et **runtime_error**, qui servent d'ancêtres à quelques exceptions correspondant à des erreurs génériques en programmation : **range_error**, **overflow_error**, **underflow_error**, **invalid_argument**, **domain_error**, **length_error**, **out_of_range**, que vous pourrez utiliser telles quelles dans vos programmes. Ces classes ont l'avantage de fournir un constructeur gérant le message qui sera retourné par **what()**.

```
if(NomFichier.length()==0)
    throw runtime_error("Nom du fichier trop court");
```

Vous pourrez aussi dériver de **runtime_error** pour fabriquer votre propre hiérarchie d'exceptions :

```
class MyFileError:public runtime_error
{ public:
    MyFileError(const std::string& Msg):runtime_
error(Msg)
    { }
};
```

que vous pourrez utiliser ainsi :

```
throw MyFileError(string("Le fichier
'")+NomFichier+"' n'existe pas");
```

4 Les flux (streams)

Je vous propose à présent de parler des flux que l'on nomme *streams*. Les streams sont un mécanisme générateur (**istream**) ou consommateur (**ostream**) d'octets, généralement utilisés dans le cadre des opérations d'entrée/sortie. La librairie standard utilise les flux pour les fichiers, la console, les tampons mémoire.

Les plus célèbres flux sont ceux relatifs à la console texte, qui sont des objets globaux. **cout**, **cerr**, **clog** , qui écrivent sur la sortie standard ou le flux d'erreurs et **cin**, qui lit sur l'entrée standard. Ils existent aussi en version *wide char* **wcout**, **wcerr**, **wlog**, **wcin**.

Les flux en sortie utilisent une surcharge de l'opérateur **<<** pour afficher les caractères. Il est possible de le surcharger pour être capable d'afficher vos propres types.

Ci-dessous, un exemple (générique, mais ce n'est pas obligatoire) qui remplace notre fonction **PrintContainerCsv** pour les vecteurs :

```
template <class T>
ostream& operator <<(ostream &os,const vector<T> &v)
{
    typename vector<T>::const_iterator it;
    for(it=v.cbegin();it!=v.cend();it++)
        os << (it==v.begin()?"":",") << *it;
    return os;
}
```

qui sera utilisé ainsi :

```
cout<<"V3 by stream ="<<V3<<endl;
```

et nous donnera :

```
V3 by stream = -18,-20,-16
```

On remarque l'utilisation de paramètres par référence pour éviter la duplication d'objet, le **const** pour garantir la non-altération de l'objet à afficher et le retour du flux (**os**) pour pouvoir enchaîner les **<<** à l'appel.

Les flux entrants (**istreams**), eux, surchargent l'opérateur **>>** qui est lui aussi surchargeable pour vos propres types par le même principe.

La plupart des flux héritent des classes **basic_istream** et **basic_ostream**, qui implémentent des méthodes pour lire/écrire le flux et s'y déplacer. On notera que la plupart des méthodes retournent une référence sur le flux afin de pouvoir enchaîner les appels sur la même instruction.

Vous pouvez ainsi afficher 5 caractères à l'aide de la méthode **write** ainsi :

```
(cout<<"test write : ").write("0123456",5)<<endl;
```

```
test write : 01234
```

Je vous épargne les techniques permettant de faire du formatage, car elles sont assez peu pratiques d'utilisation. On préférera utiliser la librairie **boost::format** pour se rapprocher du mode d'utilisation de **printf** qui, malgré ses défauts, est bien pratique.

4.1 Les fichiers

Pour manipuler des fichiers, le fonctionnement est similaire, il faut juste utiliser la classe **fstream** et associer l'objet flux au fichier. Il existe la classe **ifstream** si vous manipulez un fichier en lecture seule, **ofstream** pour un fichier en écriture seule, ou **wfstream**, **wistream**, **wostream** pour les versions *wide string*.

Commençons par un petit exemple qui écrit un fichier :

```
#include <fstream>
. . .
unsigned x=1234,y=5678;
ofstream ofs("Fichier.txt",ios::out);
ofs<<"Exemple de fichier"<<endl;
// écriture textuelle d'entiers
ofs<<"X = "<<x<<" ,y = "<<y<<endl;
// écriture binaire d'entiers
ofs.write((char*)&x,sizeof(x)).
write((char*)&y,sizeof(y));
```

Qui nous donne le contenu suivant dans le fichier **Fichier.txt** :

```
Exemple de fichier
X = 1234 ,y = 5678
Ô<EOT><NUL><NUL>.<SYN><NUL><NUL>
```

La dernière ligne correspond à la séquence binaire suivante d2 04 00 00 2e 16 00 00, c'est-à-dire la version binaire de 1234 alias 0x000004d2 en little endian (intel x86) et de 5678 alias 0x162e.

La déclaration de **ofs** nous permet d'ouvrir le fichier, mais nous aurions pu le faire en 2 instructions pour dissocier la déclaration de l'objet de l'ouverture du fichier :

```
ofstream ofs;

ofs.open("Fichier.txt",ios::out);
```

Dans les 2 cas, le second paramètre (optionnel) permet de spécifier le mode d'ouverture du fichier, de façon similaire à **fopen** mais avec un masque de bit sur des constantes combinable avec **|** au lieu d'une chaîne.

ios::in ouvre le fichier en lecture.

ios::out ouvre le fichier en écriture.

ios::app écrite systématiquement à la fin du fichier, quelle que soit la position courante du pointeur de fichier.

ios::ate se place à la fin du fichier à l'ouverture avec un seek.

ios::trunc vide le fichier à l'ouverture.

ios::binary ouvre le fichier en mode binaire. L'impact de ce flag est principalement au niveau du traitement du retour chariot. Si vous écrivez un entier avec l'opérateur **<<**, il sera écrit sous forme textuelle (ce flag n'a aucun impact sur ce point) ; si vous désirez écrire vos données sous forme binaire, vous devez utiliser **write** comme vous auriez utilisé **fwrite** en C.

Une chose pratique avec ces objets est leur capacité à lever des exceptions sur les opérations de manipulation du flux (ouverture, lecture, écriture). Par défaut, aucune exception n'est levée en cas d'erreur, il faut activer ce comportement avec l'appel de la méthode **exceptions** sur chaque objet.

```
ofs.exceptions(std::ifstream::failbit);
```

Pour la gestion de la position dans le fichier, **tellp** et **seekp** vous permettent respectivement de récupérer la position courante et de vous déplacer de façon relative ou absolue. **flush** vous permet de forcer la purge du tampon interne C++ vers l'OS, mais pas l'écriture sur disque.

close vous permet de fermer le fichier, mais sachez que cet appel peut être omis, car la destruction de l'objet implique la fermeture automatique du fichier (mais une clôture explicite permet parfois d'avoir un code plus clair).

Pour le reste, il n'y a rien d'exceptionnel, on retrouve un croisement entre les flux (type-safe, surchargeable) et la gestion des fichiers C ; en termes de performance, les deux approches sont comparables.

4.2 Gestion des flux en entrée

Au niveau des flux utilisés en entrée (sur les fichiers, mais aussi sur les autres flux tels que **cin**), sont mis à notre disposition des bits d'état (**eofbit**, **failbit**, **badbit**) accessibles à travers les méthodes booléennes **eof()**, **fail()**, **bad()**, ainsi que **good()** qui est vraie quand 3 autres sont fausses. Voilà une explication synthétique de ces états (voir **[CppReference]** section **iostate** pour plus de détails).

eofbit permet de signaler la fin du fichier.

failbit recouvre les erreurs d'ouverture de fichier et de décodage (voir plus bas).

badbit recouvre les erreurs irrécupérables.

good() permet, à la fois, d'évaluer que tout va bien et que la fin du fichier n'est pas atteinte. Si on évalue de façon booléenne l'objet flux (dans un **if** par exemple), il retourne le résultat de **good()**.

Ici un petit exemple qui lit un fichier, ligne par ligne, dans la variable **s** et affiche le nombre de lignes :

```
string s;
ifstream ifs("main.cpp");
unsigned NbrLigne=0;
while(ifs)
{
    getline(ifs,s);
    NbrLigne++;
}
cout<<"Il y a "<<NbrLigne<<" lignes"<<endl;
```

while(ifs) est équivalent à **while(ifs.good())** qui inclut la détection de fin de fichier, mais couvre aussi le fait que le fichier n'a pas été ouvert, ce qui aurait aussi pu être fait ainsi :

```
if (!ifs.is_open())
    cout<<"Erreur d'ouverture du fichier "<<endl;
```

Je vais faire ici un petit aparté pour expliquer un peu la gestion des erreurs de décodage en entrée en m'appuyant sur la console **cin**.

Pour lire un entier saisi sous forme textuelle, on écrit **cin>>x**, mais si le texte saisi sur la console n'est pas un entier, alors :

- **x** est inchangé,
- **failbit** est positionné à **true** pour signaler le problème,
- les caractères restent dans le buffer d'entrée.

Un exemple lisant un entier au clavier et contrôlant la lecture :

```
cout<<"saisir un entier : "<<endl;
cin>>x;
if(cin)
    cout<<"OK vous avez saisi : "<<x<<endl;
else
    cout<<"Erreur vous n'avez pas saisi un entier"<<endl;
```

Si vous décidez, suite au message d'erreur, de laisser une seconde chance à l'utilisateur en écrivant **cin>>x**, alors vous allez rencontrer plusieurs problèmes.

Le premier est que vous allez relire les mêmes données sans même laisser le temps à l'utilisateur de saisir une nouvelle valeur. La seconde est, que tant que le flux est en erreur, les lectures suivantes n'auront aucun effet. Il va donc vous falloir utiliser la méthode **clear()** pour faire un reset des indicateurs d'erreurs, puis purger les caractères en attente dans le buffer en utilisant la méthode **ignore** qui permet de purger *n* caractères ou jusqu'à un délimiteur :

```
// Purge jusqu'à \n
cin.ignore(numeric_limits<streamsize>::max(),'\n');
// Purge jusqu'à \n et au maximum 1000 caractères
cin.ignore(1000,'\n');
```

Pour le cas commun d'une saisie clavier, la seconde fait l'affaire et est plus rapide à écrire que la première qui, de surcroît, nécessite **#include <limits>**.

On peut aussi utiliser la fonction **getline** qui lit une ligne dans une **string**. Cela nécessite une variable de plus, mais ça permet d'afficher le texte saisi dans le message d'erreur. Exemple de solution alternative pour le **else** du **if(cin)** :

```
else{
    string BadText;
    cin.clear();
    getline(cin,BadText);
    cout<<"Erreur '"<<BadText<<"' n'est pas un entier "<<endl;
    }
```

```
Erreur 'MaChaine' n'est pas un entier
```

Comme vous le voyez, cette gestion des états n'est pas des plus pratiques à utiliser.

4.3 Les flux en mémoire

Le classe **stringstream** permet de gérer un flux en mémoire (**istringstream** et **ostringstream** en version input et ouput stream). On pourra initialiser les données à partir d'une **string** depuis le constructeur, ou avec la méthode **str(buffer)** et récupérer le buffer sous forme d'une **string** avec la méthode **str()**. Je vous rappelle qu'une **string** peut contenir des caractères **\0** contrairement à une chaîne C, ce qui permet à cette classe de manipuler un buffer binaire avec **read** et **write**.

Avant C++ 11, les **stringstream** pouvaient être utilisés pour faire des conversions entre les valeurs numériques des chaînes.

Exemple convertissant une chaîne en flottant :

```
#include <sstream>

string MaChaine("123.45");
float f;
// Initialisation avec un buffer
istringstream iss(MaChaine);
// Décodage du texte
iss>>f;
cout<<"f="<<f<<endl;
```

Ici un exemple illustrant l'équivalent du **sprintf** en C++ avec les flux :

```
ostringstream oss;
// On écrit dans le flux
oss<<"f="<<f<<", x="<<x;
// On récupère le buffer
MaChaine=oss.str();
cout<<"MaChaine="<<MaChaine<<endl;
```

Avec C++11, on utilisera plutôt **stoi**, **stol**, ... et **to_string**.

Conclusion

Au travers de ces deux articles, nous avons découvert les principales fonctionnalités de la librairie C++ dans le périmètre de la norme C++ 98 (celles introduites avec C++ 11 faisaient l'objet d'articles dans les GLMF hors-série n° 55 et n° 153).

J'espère que ces articles vous auront donné l'envie d'utiliser les concepts abordés ici et auront contribué à démystifier certains aspects de la STL.

La librairie C++ n'ayant pas l'ampleur de celles de Python, PHP ou Java, n'hésitez pas à explorer d'autres librairies. Boost contient dans sa diversité pas mal de choses, qui, il faut bien le reconnaître, ne sont pas toujours simples à aborder. Regardez aussi cette liste **[Librairies]**. ■

Références

- [CppReference] Site avec doc sur la librairie standard C++, avec la partie C++ 11 (et même C++14) mise en évidence. http://en.cppreference.com/
- [Librairies] http://en.cppreference.com/w/cpp/links/libs
- Livre The C++ Standard Library - A Tutorial and Reference, 2nd Edition (présentation ici : http://www.cppstdlib.com/)

LA MAGIE DES FILESYSTEMS : 2- CODEZ LE VÔTRE !

par Étienne Dublé [Ingénieur de Recherche CNRS au LIG]

« - Alors tu fais quoi en ce moment au labo ? - Je code un Filesystem. - Un Fa-Fa... un Fa-Fa... un Filesystem ??? » Ce n'est pas désagréable de lire une telle admiration dans les yeux d'un collègue, mais quand même, ce n'est pas comme si je recodais la moitié du noyau Linux ! Cela dit, avant, moi aussi je croyais que c'était compliqué. Mais ça, c'était avant. Car deux amis, un charpentier et un D.J., m'ont permis de démystifier tout ça... Voilà à peu près comment ça s'est passé.

1 Le problème du charpentier

1.1 Contexte

Mon ami charpentier achète souvent du bois pour ses charpentes. Un jour, il lui manquait des renforts que l'on dispose en diagonale, comme indiqué sur la figure.

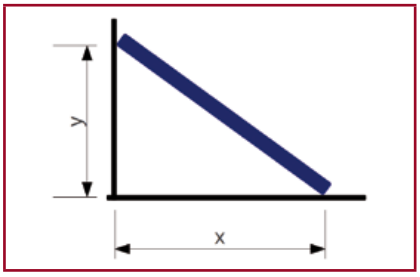


Fig. 1 : Schéma de pose d'un renfort

La pente est très variable suivant l'endroit où on doit poser un renfort : les dimensions **x** et **y** varient entre 0 et 10 cm, suivant les cas.

Ce jour-là, son fournisseur lui propose un lot de 40000 pièces en promo. Par contre, ces pièces font une longueur de 10 cm. C'est un peu court... En effet, comme on les pose en diagonale, il faut une longueur d'au moins **racine_carrée(x^2+y^2)**. Si le renfort est plus long, on le coupe. S'il est trop court par contre, c'est un souci. Au final, pour savoir si cette offre vaut le coup, mon ami devait donc savoir combien de pièces allaient faire l'affaire directement, plutôt que d'encombrer son entrepôt.

En fait, pour qu'un renfort soit de longueur suffisante, on doit donc avoir :

```
longueur_renfort >= racine_carrée(x^2+y^2)
```

Pour simplifier les calculs, on peut compter en unités de 10 cm. Cela donne :

```
longueur_renfort = 1
0 <= x <= 1
0 <= y <= 1
```

Et on simplifie donc la formule en :

```
racine_carrée(x^2+y^2) <= 1
```

Quand je suis arrivé chez lui, mon ami venait de terminer l'implémentation de sa solution. Et j'ai été plutôt surpris de constater que ce qu'il avait implémenté, c'était un système de fichiers !

1.2 Premier contact avec charpentefs

Ce filesystem est basé sur FUSE et plus précisément sur le binding **fuse-python**. Il est donc implémenté en espace utilisateur, via un fichier Python, **charpentefs.py**. On le monte de la façon suivante :

```
$ mkdir /tmp/mountpoint
$ ./charpentefs.py /tmp/mountpoint
```

La commande **mount** permet de s'assurer que le montage est bien effectif :

```
$ mount | grep charpente
charpentefs.py on /tmp/mountpoint type fuse.
charpentefs.py ([...options...])
$
```

On découvre alors que ce filesystem expose 3 fichiers « virtuels ». Deux d'entre eux, **stock** et **pieces_ok**, ne sont accessibles qu'en lecture et le troisième, **pose_renfort**, n'est accessible qu'en écriture :

```
$ ls -l /tmp/mountpoint/
total 8
-r-------- 1 etienne etienne 10 oct.  26 20:32 pieces_ok
--w------- 1 etienne etienne  0 oct.  26 20:32 pose_renfort
-r-------- 1 etienne etienne 10 oct.  26 20:32 stock
$
```

Et voici comment on les manipule :

```
$ cd /tmp/mountpoint/
$ cat stock
   40000
$ cat pieces_ok
       0
$ echo "0.5 0.5" > pose_renfort
$ cat stock
   39999
$ cat pieces_ok
       1
$ echo "0.9 0.9" > pose_renfort
$ cat stock
   39998
$ cat pieces_ok
       1
$
```

On simule donc l'utilisation d'une pièce en écrivant '**<x> <y>**' dans le fichier **pose_renfort**. À chaque fois, **stock** est décrémenté. En revanche, **pieces_ok** est incrémenté uniquement si la pièce de 10 cm est assez grande pour les valeurs **x** et **y** données.

Pour démonter le filesystem, on pourra utiliser **sudo umount <mount_point>** ou **fusermount -u <mount_point>**.

J'ai trouvé ça assez facile à utiliser, mais je lui ai quand même demandé pourquoi il était parti sur une implémentation de système de fichiers. Si j'ai bonne mémoire, ses explications tournaient autour des arguments suivants :

- Quand on implémente une fonctionnalité sous la forme d'un système de fichiers, alors n'importe quel programme pourra en profiter. En effet, lire et écrire dans des fichiers, c'est le B.A.-BA de n'importe quel langage de programmation. A partir de ce constat, tout est possible ; on peut par exemple interroger des services web avec un script bash... Et ce genre d'idée plaisait visiblement beaucoup à mon ami.
- Et par ailleurs, d'après lui, l'implémentation d'un filesystem n'est pas quelque chose de compliqué.

J'ai trouvé le premier argument assez convaincant. Effectivement, une fois la solution implémentée sous la forme d'un système de fichiers, on bénéficie d'une compatibilité inégalée avec tous les langages existants... (Et même, pour quelques décennies au moins, tous les langages futurs !) Par contre, pour vérifier son 2ème argument, je lui ai demandé de me montrer son code...

1.3 Le code de charpentefs

Voici en gros les explications qu'il m'a données sur son code [1].

1.3.1 Généralités : FUSE et code métier

Ce filesystem est principalement basé sur un composant appelé **FUSE** (*Filesystem in USErspace*). Si vous avez lu le 1er article de cette série, vous savez que FUSE permet d'implémenter un système de fichiers en espace utilisateur. Cela permet principalement de rendre plus accessible une telle implémentation, par rapport à une implémentation directe dans le noyau Linux...

FUSE est utilisable avec un certain nombre de langages de programmation. Comme on l'a vu, mon ami a choisi d'utiliser le « binding » Python, que l'on trouve dans toute distribution digne de ce nom (**`apt-get install python-fuse`** pour une Debian).

Son filesystem est donc implémenté dans le fichier Python **`charpentefs.py`**.

Ce fichier débute par des imports assez classiques, puis par la déclaration de quelques constantes :

```
FILES = [ "pose_renfort", "stock", "pieces_ok" ]
STOCK_INIT = 40000
READ_SIZE = 10
READ_STRING_FORMAT = '%' + str(READ_SIZE) + 's'
```

Les 2 premières se passent d'explications. Les 2 dernières permettront de simplifier l'implémentation, en fixant la taille des fichiers en lecture (**`/stock`** et **`/pieces_ok`**) à une valeur constante. En effet, ce qu'on doit lire dans ces fichiers est un entier compris entre **`0`** et **`40000`**, donc comportant entre 1 et 5 chiffres. Mettre à jour la taille de ces fichiers en fonction de l'entier qu'ils contiennent est une complication dont on peut se passer : on formatera ces entiers avec des espaces de « padding » de façon à avoir toujours la même longueur.

Ensuite, on trouve une classe **`StockEngine`** très simple qui implémente la partie « métier » de l'algorithme.

```
class StockEngine(object):
        def __init__(self):
                self.stock = STOCK_INIT
                self.pieces_ok = 0
        def pose_renfort(self, x, y):
                if self.stock == 0:
                        return False
                self.stock -= 1
               if math.sqrt(x*x+y*y) <= 1:
                        self.pieces_ok += 1
                return True
```

On a donc juste une initialisation dans le constructeur et une méthode **`pose_renfort()`**. Si l'opération de pose du renfort est valide (stock > 0), cette méthode décrémente le compteur **`self.stock`**, incrémente **`self.pieces_ok`** uniquement si la pièce convient (en fonction des paramètres **`x`** et **`y`**) et renvoie **`True`**. Si l'opération est invalide (stock déjà à 0) elle renvoie **`False`**.

Ensuite, l'implémentation de la classe de gestion du filesystem, **`CharpenteFS`**, est décrite. Voici comment elle débute :

```
class CharpenteFS(Fuse):
       def __init__(self, *args, **kw):
                """Constructor."""
                Fuse.__init__(self, *args, **kw)
                self.initSampleStatStructures()
                self.stock_engine = StockEngine()
```

Cette classe dérive de **`fuse.Fuse`**, comme pour toute implémentation utilisant **`python-fuse`**. Le constructeur crée une instance de la classe **`StockEngine`** et appelle la méthode **`initSampleStatStructures()`**, que nous allons décrire juste après.

1.3.2 Requête getattr()

La première requête à laquelle doit savoir répondre un système de fichiers est la requête **`getattr()`**. En effet, cette requête est utilisée pour deux choses très importantes :

- tester l'existence d'un fichier,
- récupérer les attributs de ce fichier.

Les attributs renseignent par exemple sur le propriétaire du fichier, son type (répertoire, lien symbolique, fichier régulier, etc.), les autorisations associées, les dates de dernier accès et de dernière modification.

Le résultat de cette requête **`getattr()`** (les attributs du fichier donc) est empaqueté. En C, il s'agit ainsi d'une structure **`struct stat`**. Dans notre cas, Python utilise un objet qui dépend du système d'exploitation (**`posix.stat_result`** dans mon cas), car la liste des attributs renvoyés varie légèrement en fonction du système d'exploitation.

La fonction **`lstat`** (voir **`man lstat`**) de la libc (ou **`os.lstat`** en Python) permet de déclencher ce genre de requête sur un système de fichiers. Un exemple évident d'utilisation de cette requête **`getattr()`** est la commande **`ls -l`**.

Cette requête **`getattr()`** est considérée comme la plus importante, car elle a de nombreuses implications. Par exemple, si on veux créer un répertoire dans l'arborescence de notre filesystem, une requête **`getattr()`** sera déclenchée pour tester s'il n'existe pas déjà un fichier (ou répertoire) portant ce nom et renvoyer une erreur le cas échéant. Une autre implication importante concerne les permissions de fichiers renvoyées par **`getattr()`**, qui doivent correspondre aux modes d'ouverture de fichier que l'on autorise ou pas.

Charpentefs étant un système de fichiers très simple, on a uniquement trois cas à gérer ;

[1] Vous pouvez trouver ce code à l'adresse : https://github.com/eduble/SimpleFilesystems

- requête **getattr()** sur un fichier en lecture seule (**/stock** ou **/pieces_ok**),
- requête **getattr()** sur un fichier en écriture (**/pose_renfort**),
- requête **getattr()** sur un répertoire (répertoire racine du filesystem).

De ce fait, il est rentable de préparer ces trois cas à l'avance. C'est le rôle de la méthode **initSampleStatStructures()** appelée dans le constructeur et décrite ci-après :

```
def initSampleStatStructures(self):
            self.sample_dir_stat = filestat.generateSampleDirStat()
            self.sample_wr_only_stat = filestat.generateSampleFileStat(
                    stat.S_IWUSR, # write-only
                    0
            )
            self.sample_rd_only_stat = filestat.generateSampleFileStat(
                    stat.S_IRUSR, # read-only
                    READ_SIZE
            )
```

On voit que c'est le module **filestat** qui fait à peu près tout le travail. Ce module a été implémenté par un autre ami, qui est D.J., et mon ami charpentier l'a réutilisé.

La méthode **filestat.generateSampleFileStat()** prend 2 arguments, le 1er correspondant au mode et le 2ème à la taille de fichier. On utilise donc ici notre constante **READ_SIZE** correspondant à la taille (fixe, voir plus haut) des fichiers en lecture.

Le code de gestion de la requête **getattr()** découle assez clairement de ces préparatifs :

```
    def getattr(self, path):
            if path == '/':
                    return self.sample_dir_stat
            if path.lstrip('/') not in FILES:
                    return -ENOENT  # no such file or dir
            if path == '/pose_renfort':
                    return self.sample_wr_only_stat
            else: # the 2 other files are read-only
                    return self.sample_rd_only_stat
```

Dans le code d'un filesystem, les chemins sont indiqués relativement au point de montage. Par exemple, si une requête **getattr()** est déclenchée sur un fichier **/chemin/du/point_de_montage/repertoire/fichier**, alors notre méthode **getattr()** sera appelée avec le paramètre **path** = **/repertoire/fichier**. Et une requête sur le point de montage sera reçue avec **path** = **/**.

Par ailleurs, la gestion des erreurs dans FUSE est un peu particulière : on doit retourner **-1*<numero_d_erreur>**. Par exemple, ici on retourne **-ENOENT** (« no such file or directory » voir le module python **errno**[2]) en cas de requête sur un fichier inexistant. Cette particularité n'est d'ailleurs pas spécifique au binding Python.

Pour le reste, on retourne bien évidemment, en fonction du fichier ou répertoire ciblé par la requête, un des 3 « paquets d'attributs » préparés dans la méthode **initSampleStatStructures()** décrite plus haut.

1.3.3 Requête readdir()

Une autre requête à gérer est la requête **readdir()**, qui permet de lister les entrées d'un répertoire.

```
    def readdir(self, path, offset):
            for e in FILES:
                    yield fuse.Direntry(e)
```

Je crois que celle-ci se passe quasiment de commentaires.

On pourra juste remarquer que l'argument **offset** n'est pas traité. Il s'agit d'une approximation, que l'on peut se permettre, dans la mesure où il est hautement improbable qu'on nous demande les 3 entrées de répertoire en plusieurs fois (si on en avait 300, ce serait autre chose).

L'argument **path** n'est pas non plus pris en compte parce qu'on a un seul répertoire (le répertoire racine) dans ce filesystem, donc c'est forcément de lui qu'il s'agit.

1.3.4 Requêtes d'ouverture et fermeture de fichier

Encore plus simple. Dans le cas de ce filesystem, on n'a rien besoin de faire au moment de l'ouverture ou de la fermeture d'un fichier. Donc :

```
    def open(self, path, flags):
            return 0
```

```
    def release(self, path, flags):
            return 0
```

1.3.5 Requête read()

Voilà qui est plus intéressant. Pour la gestion de la requête **read()** (il s'agit forcément d'un des 2 fichiers en lecture seule, **pieces_ok** ou **stock**), le code utilise une sous-méthode **read_from_int()**, qui permet de retourner une chaîne de caractères formatée à partir d'un entier, en l'occurrence **self.stock_engine.pieces_ok** ou **self.stock_engine.stock** suivant le cas :

```
    def read(self, path, size, offset):
            if path == '/pieces_ok':
                    return self.read_from_int(
                            self.stock_engine.pieces_ok, size, offset)
            else:   # /stock
                    return self.read_from_int(
                            self.stock_engine.stock, size, offset)
    def read_from_int(self, int_value, size, offset):
            s = READ_STRING_FORMAT % (str(int_value) + '\n')
            if offset < len(s):
                    if (offset + size) > len(s):
                            size = len(s) - offset
                    return s[offset:(offset + size)]
            else:
                    return ''
```

On utilise ici notre constante **READ_STRING_FORMAT** pour assurer le padding et obtenir une chaîne de longueur fixe. Cette chaîne de longueur fixe est vue comme le contenu du fichier lu. La méthode **read_from_int()** gère également le fait qu'on puisse lire une partie seulement de ce fichier (ou, du point de vue du filesystem, une partie de notre chaîne formatée), via les arguments **size** et **offset**.

[2] http://docs.python.org/2/library/errno.html

En réalité, étant donné la petite taille de ce fichier (**READ_SIZE** = **10**), on aurait presque pu simplifier la chose, un peu comme pour **readdir()**, et considérer **offset** toujours à zéro et **size** suffisamment grand... Mais cette fois-ci mon ami a semblé un peu plus courageux.

1.3.6 Requêtes d'écriture de fichier

Quand on écrit un fichier via, par exemple, une redirection de la sortie de commande **cat**, on fait en réalité 3 opérations :

- Si le fichier existe, on envoie une requête **truncate()** pour supprimer son contenu existant et passer sa taille à 0 ;
- On envoie la requête d'écriture : **write()** ;
- On envoie une requête pour mettre à jour la date de dernière modification du fichier : **utime()**.

Ce filesystem doit donc savoir gérer ces trois requêtes :

```
        def truncate(self, path, length):
                return 0
        def write(self, path, buf, offset):
                return self.pose_renfort_input(buf)
        def utime(self, path, times):
                return 0
```

Dans notre cas, on sait de quel fichier il s'agit, **pose_renfort**, car c'est le seul qui autorise une écriture. Par ailleurs, on n'a pas besoin de traiter les requêtes **truncate()** et **utime()** dans ce cas de filesystem.

La requête **write()** fait appel à une méthode **pose_renfort_input()** et, là aussi, l'offset est ignoré par simplification.

```
        def pose_renfort_input(self, buf):
                try:
                        t = tuple(float(f) for f in buf.split())
                except:
                        return -EINVAL
                if len(t) != 2:
                        return -EINVAL
                for x_or_y in t:
                        if x_or_y > 1 or x_or_y < 0:
                                return -EINVAL
                if self.stock_engine.pose_renfort(*t):
                        return len(buf)  # ok
                else:
                        return -EINVAL
```

Cette méthode **pose_renfort_input()** est assez triviale. Elle commence par vérifier la validité de ce qui est écrit : on attend une chaîne de la forme '**<x> <y>**' avec **0<=x<=1** et **0<=y<=1**. En cas de chaîne invalide, on retourne **-EINVAL** (« Invalid argument »). Si la chaîne est valide, on appelle **self.stock_engine.pose_renfort(x, y)**. Si cette méthode échoue (le stock était déjà à 0), cette méthode renvoie **False**, ce qu'on traduit à nouveau en **-EINVAL**. Sinon, tout est OK, donc on retourne **len(buf)** pour signifier à l'auteur de la requête d'écriture que tous les caractères ont été pris en compte.

1.3.7 Section « main »

Le module **charpentefs.py** se termine par une section « main » assez triviale.

```
if __name__ == '__main__':
        usage = "CharpenteFS: le filesystem du charpentier.\n" + \
                argv[0] + " <mount_point>"
        if len(argv) < 2:
                print usage
                exit()
        mount_point = argv[1]
        server = CharpenteFS(usage=usage)
        server.fuse_args.mountpoint = mount_point
        server.fuse_args.add('debug')
        server.multithreaded = False
        server.fuse_args.setmod('foreground')
        server.fuse_args.add('default_permissions')
        server.main()
```

Elle débute par une vérification sommaire des arguments de la ligne de commandes, puis crée un objet de type **CharpenteFS**, lui attribue quelques paramètres et appelle sa méthode **main()**, ce qui a pour effet de créer le montage.

Apparemment, la gestion des paramètres dans FUSE n'est pas la chose la plus jolie qu'il m'ait été donné de voir. Et malheureusement, contrairement à ce qu'on aurait pu espérer, le framework **python-fuse** ne rectifie pas le tir.

Deux des paramètres utilisés ici sont orientés « debugging » :

- **debug** permet de générer des traces pour chaque requête traitée,
- **foreground** évite que ce programme de gestion du filesystem se détache de notre terminal.

Deux autres paramètres permettent de largement simplifier l'implémentation :

- **multithreaded = False** permet d'éviter de se soucier des éventuelles requêtes concurrentes et des problèmes de réentrance de code. En effet, souvent on lance plusieurs programmes sur un même filesystem...
- **default_permissions** (paramètre très mal nommé à mon avis !) demande au noyau Linux de vérifier les permissions avant d'appeler nos gestionnaires de requêtes. Ainsi, si on tente de lire **pose_renfort** (qui n'est accessible qu'en écriture), le noyau va, au préalable, déclencher une requête **getattr()** pour obtenir les autorisations associées à ce fichier et finalement renvoyer une erreur d'autorisation ; notre gestionnaire **read()** ne sera donc jamais appelé. Comme on peut le voir dans cet exemple, **ce paramètre est très important**. Si on ne l'indique pas, alors il faut vérifier au début de chaque gestionnaire (**readdir**, **open**, **read**, **write**...) que le fichier existe et que l'opération est autorisée. A l'inverse, plusieurs hypothèses que j'ai formulées plus haut découlent directement de l'utilisation de ce paramètre. Par exemple : « on sait de quel fichier il s'agit, **pose_renfort**, car c'est le seul qui autorise une écriture ». Sans ce paramètre, le code serait donc clairement plus long.

1.4 Le script test_charpente.sh

Comme le filesystem paraissait fonctionnel, il ne restait plus qu'à l'utiliser pour faire un test statistique et avoir une idée du nombre de pièces utilisables directement dans un stock de 40000 pièces.

J'ai proposé à mon ami d'écrire le script bash avec lui, parce que j'ai l'habitude. Voilà ce qu'on a écrit :

```
$ cat test_charpente.sh
#!/bin/bash
PRECISION=5
MOUNTPOINT=$1
generate_float()
{
  echo -n "0."
  for i in $(seq $PRECISION)
  do
    echo -n $(($RANDOM % 10))
  done
  echo
}
while [ $(cat $MOUNTPOINT/stock) -gt 0 ]
do
  x=$(generate_float)
  y=$(generate_float)
  echo "$x $y" > $MOUNTPOINT/pose_renfort
done
$
```

En paramètre on attend le point de montage. C'est du code vite fait, on ne vérifie pas...

Ensuite, on a une fonction qui génère un nombre flottant du type **0.xxxxx** (donc entre 0 et 1, avec 5 décimales), en utilisant la fonction **RANDOM** de bash.

Et enfin, on boucle jusqu'à ce que le stock soit vide, en générant des valeurs x et y aléatoires à chaque itération pour poser un renfort.

Voilà ce que ça donne à l'utilisation :

```
$ ./test_charpente.sh /tmp/mountpoint
[...2 minutes plus tard...]
$ cat /tmp/mountpoint/pieces_ok
31410
$
```

Bon, ça fait donc 31410 pièces OK sur les 40000 du lot, un peu plus des trois quarts...

Tiens, c'est marrant, on dirait presque les décimales du nombre Pi !

C'est sans doute le hasard, la méthode étant basée sur des nombres aléatoires... Refaisons le test pour voir...

```
$ ./test_charpente.sh /tmp/mountpoint
[...]
$ cat /tmp/mountpoint/pieces_ok
31423
$ ./test_charpente.sh /tmp/mountpoint
[...]
$ cat /tmp/mountpoint/pieces_ok
31418
$
```

Bizarre ce truc ! On dirait bien qu'on reste proche des décimales de Pi !

Attendez, c'est pas possible... On tire des valeurs aléatoires et moyennant une petite formule toute bête, on obtient les décimales de Pi !! C'est quoi cette histoire ??

Bon... Tout cela est bien étrange. Et je dois avouer que je suis bien incapable d'expliquer ce phénomène...

Cela restera donc la « conjecture du charpentier »...

En attendant, l'histoire n'est pas finie. Parce que, pour avoir plus d'informations sur le module **filestat** utilisé par charpentefs, je suis allé voir son auteur original, mon ami D.J. Et celui-ci a commencé à m'expliquer le petit projet pour lequel il avait écrit ce module...

2 Le problème du D.J.

2.1 Contexte

Mon ami D.J. a accumulé au cours de sa carrière toute une collection de fichiers audio au format MP3. Ce qui l'ennuie, c'est que tous les logiciels actuels de gestion de collection multimédia se basent sur le tag ID3 pour afficher l'auteur ou l'album correspondant à un titre, alors que, pour toute une partie de sa collection, ces informations ne sont pas renseignées.

Les mettre à jour est une opération fastidieuse. Un jour, il décida donc de coder un petit filesystem pour faciliter cette opération.

Le but est le suivant : représenter la collection de fichiers MP3 sous la forme d'une arborescence de répertoires **/<artiste>/<album>/<fichier>**. Les noms de répertoires **<artiste>** et **<album>** refléteront les informations correspondantes du tag ID3 du fichier, s'il est renseigné. Dans le cas contraire, le chemin sera **/UnknownArtist/UnknownAlbum/<fichier>**. Pour simplifier l'implémentation, **<fichier>** sera simplement un lien symbolique vers l'emplacement original du fichier MP3 dans la collection.

Certaines opérations de modification, autorisées sur ce filesystem, permettront alors de mettre à jour les informations ID3 correspondantes. En particulier, déplacer un fichier **/UnknownArtist/UnknownAlbum/<fichier>** vers **/<artiste>/<album>/<fichier>** provoquera la mise à jour des informations **<artiste>** et **<album>** dans le tag ID3 du fichier. On autorisera également la création préalable de nouveaux artistes ou albums via la création du répertoire correspondant.

2.2 Premier contact avec taggerfs

Pour monter TaggerFS, il faut lui indiquer le chemin de notre collection de fichiers MP3, ainsi que le point de montage :

```
$ mkdir /tmp/mountpoint
$ ./taggerfs.py /mp3_collection /tmp/mountpoint
```

Ensuite, on peut explorer la collection via l'information contenue dans les tags ID3 :

```
$ cd /tmp/mountpoint/
$ ls
Ra  UnknownArtist  Whitesnake
$
```

Pour que l'explication soit plus claire, je n'ai copié dans **/mp3_collection** qu'un petit fragment de la collection de mon ami. Il n'est pas très éclectique (surtout pour un D.J. !) mais quand même un peu plus que ça. ;)

Ce résultat indique qu'il y a des MP3 où le tag artiste indique « Ra », d'autres indiquant « Whitesnake » et d'autres où le tag n'est pas renseigné.

Explorons plus précisément cette dernière catégorie.

```
$ ls UnknownArtist/
UnknownAlbum
$
```

Bon, dans ces fichiers, le tag album manque également, comme on pouvait s'y attendre.

```
$ ls -l UnknownArtist/UnknownAlbum/
[...]
lrwxrwxrwx [...] Sky.mp3 -> /mp3_collection/
HardRock/Ra/From One/Sky.mp3
lrwxrwxrwx [...] Rectifier.mp3 -> /mp3_collection/
HardRock/Ra/From One/Rectifier.mp3
$
```

Ah, voilà qui est plus intéressant. Car, même si ces fichiers n'avaient pas un tag ID3 renseigné, ils étaient par contre correctement classés dans la collection de mon ami. Et, comme taggerfs représente ces fichiers via un lien symbolique, qui pointe vers le fichier MP3 dans la collection, il est facile de deviner que ces fichiers sont du groupe de hard rock « Ra » et de l'album « From One ».

Nous allons donc maintenant mettre à jour le tag ID3 de ces fichiers.

Comme vu plus haut, « Ra » est déjà connu dans notre filesystem. En effet, il y a dans la collection d'autres fichiers MP3 de ce même artiste, mais d'un album différent (« Duality ») :

```
$ cd Ra
$ ls
Duality
$
```

Il faut donc « déclarer » cet album « From One », en créant le répertoire correspondant.

```
$ mkdir 'From One'
$ ls
Duality   From One
$
```

On peut alors déplacer dans ce répertoire nos fichiers (en fait, pour être plus précis, on déplace des liens symboliques) :

```
$ cd 'From One'
$ mv ../../UnknownArtist/UnknownAlbum/* .
$
```

Ceci a pour effet de mettre à jour les tags artiste et album de chacun des fichiers concernés.

Pour s'en assurer, on peut par exemple utiliser le programme en ligne de commandes **eyeD3** :

```
$ eyeD3 /mp3_collection/HardRock/Ra/From\ One/Sky.mp3
Sky.mp3   [ 4.52 MB ]
--------------------------------------------------------------------
Time: 04:56 MPEG1, Layer III [ 128 kb/s @ 44100 Hz - Joint stereo ]
--------------------------------------------------------------------
ID3 v2.4:
title:    artist: Ra
album: From Oneyear: None
track:
$
```

Le tag ID3 a donc bien été mis à jour. On remarque que les autres infos du tag ID3 sont bien évidemment restées vierges. On pourrait imaginer une version plus avancée de taggerfs pour gérer ces autres données...[3]

2.3 Le code de taggerfs

Le code de taggerfs est basé sur 3 modules codés par mon ami D.J. :

- **id3library.py**
- **taggerfs.py**
- **filestat.py**

2.3.1 id3library.py

Ce module permet de gérer les tags ID3 d'une collection de fichiers MP3. Il utilise en interne la librairie **python-mutagen** pour l'édition des tags.

Comme ce code est relativement éloigné de notre propos, je ne vais pas le détailler ici. Pour plus d'informations, vous pourrez consulter le fichier source **id3library.py**[4].

2.3.2 taggerfs.py

Le code de **taggerfs.py** présente des similitudes importantes avec **charpentefs.py**, du fait que tous deux décrivent un filesystem basé sur FUSE. Nous allons donc nous limiter à décrire les éléments qui diffèrent le plus.

En premier lieu, l'arborescence de charpentefs était plus simple, car on n'avait pas de sous-répertoires. Dans le code de taggerfs, il faut par contre analyser des chemins plus complexes. Ceux-ci doivent être de la forme **/[<artist>[/<album>[/<basename>]]]**, suivant qu'on s'intéresse aux artistes, aux albums d'un artiste ou aux fichiers d'un album.

La méthode **TaggerFS.analysePath()** permet de factoriser cette analyse :

```
class TaggerFS(Fuse):
  [...]
        def analysePath(self, path):
                if path == '/':
                        return None, None, None
                tokens = path.split("/")[1:]
                while len(tokens) < 3:
                        tokens.append(None)
                return tuple(tokens)    # (artist, album, mp3_basename)
  [...]
```

Cette méthode analyse donc **path** pour déduire chaque composante (optionnelle) **<artist>**, **<album>** et **<basename>** et les retourne sous la forme d'un tuple.

Ainsi :

- **'/'** retournera **(None, None, None)**
- '**/<artist>**' retournera **(<artist>, None, None)**
- **'/<artist>/<album>'** retournera **(<artist>, <album>, None)**
- **'/<artist>/<album>/<basename>'** retournera **(<artist>, <album>, <basename>)**

Une fois cette analyse faite via un appel à **analysePath()**, l'implémentation des gestionnaires de requêtes FUSE est relativement triviale.

Les filesystems charpentefs et taggerfs présentent une autre différence notable : la liste des opérations autorisées par chacun d'entre eux est différente. Dans le cas de **taggerfs.py**, on trouve ainsi de nouveaux gestionnaires de requêtes FUSE :

- **readlink(self, path)** doit retourner la cible du lien symbolique indiqué par **path**. Dans notre cas, on retourne donc le chemin du fichier MP3.
- **rename(self, old_path, new_path)** permet de gérer le renommage ou le déplacement d'un fichier. Dans notre cas on autorise uniquement le déplacement d'un lien symbolique **/<old_artist>/<old_album>/link_name** vers **/<new_artist>/<new_album>/link_name**.
- **mkdir(self, path, mode)** permet de gérer la création d'un répertoire. Dans notre cas, cette opération permet de « déclarer » un nouvel artiste ou un nouvel album via la création du répertoire correspondant.

[3] Mon ami D.J. ne compte pas poursuivre ce projet, car la fonctionnalité est suffisante pour son besoin. D'autre part, il s'est aperçu qu'il existait un autre projet, pytagfs, dont l'approche semble similaire et dont les fonctionnalités semblent plus complètes : http://www.pytagsfs.org/

[4] Voir : https://github.com/eduble/SimpleFilesystems

2.3.3 filestat.py

Ce module est utilisé à la fois par taggerfs et charpentefs. Il permet de créer des « paquets d'attributs de fichier » utiles pour répondre aux requêtes FUSE `getattr()`.

Il y a deux façons pour fabriquer ces « paquets d'attributs de fichier » :

- manipuler un objet de type `posix.stat_result` et spécifier chacun des attributs (type de fichier, taille du fichier, autorisations, propriétaire, etc.),
- créer un fichier temporaire adéquat et lire le paquet d'attributs via `os.lstat()`.

La deuxième méthode est sans doute bien plus portable. En effet, la première ne fonctionnera que sur les systèmes de type POSIX. D'autre part, avec la deuxième méthode, on n'a pas besoin de spécifier tous les attributs, car les valeurs par défaut utilisées pour la création du fichier temporaire conviennent souvent.

Mon ami D.J. a ainsi utilisé cette approche pour créer le module `filestat.py`.

Voici son code, qui est assez trivial :

```
import os, tempfile
def generateSampleDirStat():
  tmpdir = tempfile.mkdtemp()
  stat_info = os.stat(tmpdir)
  os.rmdir(tmpdir)
  return stat_info
def generateSampleFileStat(mode, size):
  fd, tmpfile = tempfile.mkstemp()
  if size > 0:
    os.write(fd, '-' * size);
  os.close(fd)
  os.chmod(tmpfile, mode)
  stat_info = os.stat(tmpfile)
  os.remove(tmpfile)
  return stat_info
def generateSampleSymlinkStat():
  tmpdir = tempfile.mkdtemp()
  tmplink = tmpdir + '/link'
  os.symlink(tmpdir, tmplink)
  stat_info = os.lstat(tmplink)
  os.remove(tmplink)
  os.rmdir(tmpdir)
  return stat_info
```

2.4 taggerfs : conclusion

Concernant taggerfs, je me suis vite rendu à l'évidence : l'approche filesystem est diablement efficace pour ce genre de problèmes !

Je ne sais pas si vous avez remarqué, mais tout à l'heure, on a mis à jour deux tags différents (artiste et album), sur tous les fichiers d'un album. Et tout cela en une seule commande `mv` triviale !!

Note **La conjecture du charpentier : le mystère dévoilé !**

Le problème du charpentier repose bien évidemment sur des hypothèses montées de toutes pièces ; et ceci dans le but d'obtenir le résultat « inattendu ». Et comme je suis un peu joueur, il fallait bien que je vous fasse mariner un peu avant de vous donner ces explications...

La figure ci-contre représente le placement aléatoire d'un point `P`, avec `x` et `y` choisis aléatoirement entre `0` et `1`. Suivant les cas, `P` peut se trouver dans la zone rouge clair ou dans la zone vert clair.

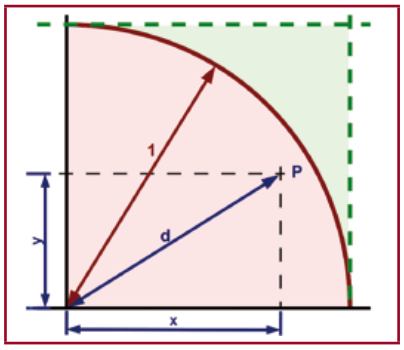


Fig. 2 : Explication sur l'obtention des décimales du nombre Pi

La distance `d` est égale à `racine_carree(x^2+y^2)`. La probabilité que `P` soit dans la zone rouge clair correspond donc à la probabilité que l'expression suivante soit vraie :

```
racine_carree(x^2+y^2) <= 1
```

Par ailleurs, comme `P` est placé de manière aléatoire dans le carré, la probabilité que `P` soit dans la zone rouge clair correspond aussi au rapport entre l'aire de la zone rouge clair sur l'aire du carré. La zone rouge clair étant un quart de disque de rayon 1, son aire est de `Pi/4`. Et l'aire du carré est bien évidemment `1`. Donc le rapport donne `Pi/4`.

Avec ces deux façons de calculer la même probabilité, on en déduit que la probabilité que l'expression `racine_carree(x^2+y^2) <= 1` soit vérifiée est de `Pi/4`.

Notre ami charpentier ayant testé `40000` fois cette expression, il est normal qu'il obtienne un nombre proche de `40000 * Pi/4 = 10000 * Pi`, d'où l'apparition des décimales...

Même si ces explications rendent évident le résultat obtenu, j'avoue avoir été surpris moi-même quand on me l'a montré : comment imaginer qu'on peut approcher la valeur de pi avec uniquement un grand nombre de valeurs aléatoires et une formule aussi simple ?

Et ce n'est pas tout : on pourra utiliser un gestionnaire de fichiers graphique pour sélectionner et déplacer les fichiers de manière ergonomique... Il ne faudra pas longtemps pour réorganiser la hiérarchie correctement et le filesystem va mettre à jour les tags ID3 en conséquence, de manière totalement transparente.

Le déplacement de fichiers est d'ailleurs la base de l'informatique, tout le monde sait faire ça (en tout cas de manière graphique). Donc, ce n'est pas forcément l'informaticien (que dis-je, le D.J.) qui a codé le filesystem qui s'en chargera...

3 Et toi, tu codes quoi comme filesystem ?

Moi ? Eh bien, en fait, je code un filesystem qui utilise des algorithmes de gestion de données avancés. Ce sont des algorithmes développés dans l'équipe de recherche où je travaille. Et j'espère donc obtenir des performances supérieures aux filesystems classiques (ce n'est pas gagné).

J'ai commencé par un prototype en C++ avec FUSE, et, suivant les résultats que j'obtiendrai, j'envisagerai ou non un codage noyau.

C'est un projet qui me plaît beaucoup, parce que c'est bien bas niveau. Même mon environnement de test était « amusant » à développer...

Je vous invite donc à vous essayer au codage de filesystem, c'est assez passionnant !

PS : Merci à Pierre-Henry pour la relecture. ■

Le CNLL

Conseil National du Logiciel Libre

et ses représentants régionaux

PRÉSENTENT

les RENCONTRES REGIONALES du LOGICIEL LIBRE & du SECTEUR PUBLIC

Du 04 Octobre 2013 au 17 Avril 2014

Venez rencontrer vos pairs
avec des problématiques communes !

Venez rencontrer
les prestataires locaux de l'OpenSource !

En savoir plus sur **www.rrll.fr**

Evénement soutenu par
ADOSIS, *Agence Digitale*, spécialiste OpenSource
www.adosis.com

adosis

TRAVAILLER DANS LE LIBRE : OUI

MAIS PAS À N'IMPORTE QUEL PRIX :) !

LINAGORA

REJOIGNEZ-NOUS !

www.linagora.com/-CARRIERE-