Gérer ses fichiers de configuration personnels avec Subversion

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 mars 2008

https://www.bortzmeyer.org/home-in-subversion.html

Je travaille sur de nombreuses machines Unix, à la maison, au bureau, chez des hébergeurs divers, et maintenir une configuration cohérente sur toutes ces machines n'est pas facile. Par exemple, je voudrais bien avoir la même invite zsh partout. Subversion fournit une solution élégante pour cela.

Sur Unix, les réglages personnels se font par des fichiers situés dans le répertoire connexion de l'utilisateur, identifié par la variable d'environnement HOME (également notée). Ces fichiers ont traditionnellement un nom qui commence par un point d'où leur surnom de "*dotfiles*" (voici pourquoi l'excellent site qui permet de partage de tels fichiers se nomme <http://www.dotfiles.org/>).

Pour avoir le même environnement sur plusieurs machines Unix, il « suffit » donc de recopier ces fichiers depuis un site de référence. Mais je n'aime guère cette méthode car la copie remplace les éventuelles modifications locales. Cette copie depuis un site central nécessite une bonne discipline : il faut penser à transférer les modifications locales sur le site central régulièrement, sinon elles seront détruites.

Ce genre de problèmes est bien connu depuis longtemps dans le monde du développement logiciel et est résolu par les systèmes de gestion de versions. Ils gardent trace de l'historique complet des fichiers (jamais de perte d'information) et copient intelligemment : lorsqu'on met à jour une machine, les nouveautés en provenance du dépôt central sont **fusionnées** avec les données locales, elles ne les remplacent pas aveuglément. D'où l'idée d'utiliser un de ces logiciels de gestion de versions, Subversion.

Avant d'expliquer la configuration du dépôt, voyons d'abord le travail quotidien. Si je veux mettre à jour une machine, je fais juste :

```
% svn update
U    .procmailrc
Updated to revision 948.
%
```

Ici, le fichier `.procmailrc` a été mis à jour. Mais si j'avais fait des changements locaux ?

```
% svn update
G    .procmailrc
Updated to revision 948.
%
```

Le **G** (pour "*merGed*") indique que les nouveautés ont été fusionnées avec le changement fait localement. Celui-ci n'a pas été perdu.

Si j'ai modifié un fichier de configuration et que je veux transmettre ces changements aux autres machines, j'enregistre :

```
% svn commit -m 'Liste CSS' .procmailrc
Sending        .procmailrc
Transmitting file data .
Committed revision 949.
```

Donc, pour résumer, cette solution, simple pour le développeur (qui est déjà habitué à ces outils), permet de ne pas obliger à une stricte discipline.

Comment configure t-on le dépôt central ? Et comment fait-on pour la première installation ? Comme ce service est strictement personnel, mon dépôt central est accessible en SSH uniquement. C'est le dépôt sur lequel je stocke des fichiers personnels, et il contient un répertoire `dotfiles`. Pour toute nouvelle machine, j'ai juste à installer Subversion (qu'on trouve désormais sur tous les Unix) et à faire, dans le répertoire de connexion :

```
% svn co svn+ssh://svn.bortzmeyer.org/home/stephane/Subversion-Personal-Repository/trunk/dotfiles .
```

Est-ce possible d'avoir **exactement** le même fichier de configuration sur toutes les machines ? Évidemment pas. Les machines en question ne sont pas équivalentes, elles n'ont pas la même version de tous les logiciels, elles n'ont pas les données au même endroit, etc. Les fichiers de configuration doivent donc s'adapter.

Commençons par le cas le plus simple, où ces fichiers sont écrits dans un langage de programmation complet. C'est le cas du `.zshrc`, qui configure le shell ou du `.emacs` qui configure l'éditeur Emacs (et qui est écrit en Lisp). On peut alors tester l'environnement et s'y adapter. Par exemple, dans mon `.zshrc`, on trouve :

```
# Tester si /local existe
if [ -d /local ]; then
   LOCAL=/local
else
   LOCAL=/usr/local
fi
PATH=~/bin:$LOCAL/sbin:$LOCAL/bin: ...

...

# Est-ce une Debian ?
if [ -e /etc/debian_version ]; then
```

```
    debian=1
else
    debian=0
fi

# Créer un alias seulement pour les Debian
if [ $debian != 0 ]; then
    if [ -x /usr/bin/nsgmls ]; then
alias nsgmlsxml='nsgmls -s -wxml /usr/share/sgml/declaration/xml.dcl'
    elif [ -x /usr/bin/onsgmls ]; then
    alias nsgmlsxml='onsgmls -s -wxml /usr/share/sgml/declaration/xml.dcl'
    fi
fi

#
case $uname in
...
   NetBSD)
    # Définir l'endroit où se trouvent les paquetages binaires en
    # fonction de la version de NetBSD installée
    export NETBSD_SERVER=ftp.fr.netbsd.org
    export PKG_PATH=ftp://$NETBSD_SERVER/pub/NetBSD/packages/`uname -r|cut -d _ -f 1`/`uname -p`/All
```

Dans mon `.emacs`, il y a des constructions comme :

```
; Supprimer cet ennuyeux message, apparu avec Emacs 22. Le test fait
; que les vieilles versions d'Emacs (qui n'ont pas la variable
; inhibit-startup-message) ne râleront pas.
(if (>= emacs-major-version 22)
    (setq inhibit-startup-message t))
...
(if (not (featurep 'xemacs)) ; Not supported in Xemacs
    (global-font-lock-mode t))
...
; Ici, on va tester l'existence d'une bibliothèque avec
; locate-library, avant de la charger.
; Trouver le meilleur mode pour XSL
(if (locate-library "xslide")
    ; If possible, use xslide
    (progn
      (autoload 'xsl-mode "xslide" "Major mode for XSL stylesheets." t)
      (add-hook 'xsl-mode-hook
'turn-on-font-lock)
      (setq auto-mode-alist
    (append
     '(
      ("\\.fo" . xsl-mode)
      ("\\.xsl" . xsl-mode))
     auto-mode-alist))
      )
  ; else try nxml
  (if (locate-library "nxml-mode")
      (setq auto-mode-alist
    (append
     '(
      ("\\.xsl" . nxml-mode))
     auto-mode-alist))
    ; else normal XML mode
    (setq auto-mode-alist
  (append
   '(
    ("\\.xsl" . xml-mode))
   auto-mode-alist))
    )
  )
```

Mettre tous ces `if` et ces `case` n'est pas toujours très amusant. Une autre solution est de placer les choses purement locales à une machine dans un fichier qu'on inclus. Dans le `.zshrc`, cela se fait :

```
if [ -f ~/.zshrc_local ]; then
   source ~/.zshrc_local
fi
```

Ce `.zshrc_local` ne sera **pas** géré par Subversion et contiendra des instructions qui n'ont de sens que pour une seule machine. Même système pour un `.xsession` :

```
if [ -x $HOME/.xsession-local ]; then
   . ./.xsession-local
fi
```

Dans le `.emacs`, cela sera :

```
(if (file-exists-p "~/.emacs_local")
    (load-file "~/.emacs_local"))
; On peut même charger un fichier du répertoire courant, pour une
; adaptation par répertoire, mais ATTENTION, il faut alors vérifier
; son propriétaire (pensez à /tmp)
(if (and
     ; Does it exist?
     (file-exists-p "./myemacs.el")
     ; Is it mine?
     (= (nth 2 (file-attributes "./myemacs.el")) (user-uid)))
    (load-file "./myemacs.el"))
```

Les programmes comme mutt ou procmail sont plus contrariants car leur fichier de configuration n'utilise pas un langage de programmation complet. Il ne reste donc que la solution du fichier local mais attention, comme on ne peut pas tester son existence, il faut s'assurer qu'il existe bien, même vide. Dans un `.muttrc` :

```
source $HOME/.muttrc_local
```

et dans un `.procmailrc` :

```
INCLUDERC=$HOME/.procmailrclocalstart
```

Pour mutt, une autre possibilité d'adaptation est d'appeler des commandes externes qui, elles, pourront être écrites dans un langage de Turing :

```
`$HOME/bin/alternates-mutt` 'bortz(meyer)?(\+[^@]+)?@nic.fr|bortz(meyer)?(\+[^@]+)?@(generic-nic|gennic)\.ne
```

avec `$HOME/bin/alternates-mutt` qui teste la version de mutt pour utiliser la bonne syntaxe :

```
# Without TERM=dumb, mutt sends curses control characters :-(

version=`TERM=dumb mutt -v | head -n 1 | cut -d' ' -f2 | cut -d. -f1,2`

if [ `expr $version ">=" 1.5` = 1 ]; then
    echo "alternates "
else
    echo "set alternates="
fi
```

Enfin, lorsque le fichier de configuration n'offre aucune de ces possibilités (cas du `.ssh/config` de SSH), il reste à le faire traiter par un autre programme. J'utilise M4, qu'on trouve sur toutes les machines Unix. Le fichier géré par Subversion contient des instructions M4 :

```
define(SOURCES_FW,bortzmeyer.example.net)

ifdef(`MAISON',
`
',
`Host preston.sources.org
      ProxyCommand ssh -l stephane SOURCES_FW nc %h %p
')
```

qui se lisent : si je suis à la maison, rien de particulier, sinon, pour atteindre `preston.sources.org`, passer par le relais `SOURCES_FW` (défini plus haut). Le fichier est ensuite traité par M4 à chaque connexion. Le fichier `.zshrc_local` définit les variables M4 :

```
SSH_M4_MACROS="-DMAISON"
```

et le `.zshrc` contient les instructions de traitement :

```
if [ -f ~/.ssh/config.m4 ]; then
   m4 ${SSH_M4_MACROS} ~/.ssh/config.m4 > ~/.ssh/config
fi
```

Joey Hess a décrit une configuration encore plus élaborée (puisque tout son répertoire personnel, pas uniquement les fichiers de configuration, est géré par Subversion) dans "*Subverting your homedir, or keeping your life in svn*" <`http://kitenet.net/~joey/svnhome/`>.