CVS : Concurrent Versions System

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 mars 1999

https://www.bortzmeyer.org/cvs.html

CVS est un outil de gestion de sources. Tout projet qui manipule des fichiers texte, que ce soit la gestion d'un serveur Web, l'écriture d'un livre avec LaTeX ou bien la réalisation d'un programme peut profiter de CVS. CVS garde trace automatiquement des différentes versions d'un même fichier et permet de récupérer les vieilles versions.

En outre, contrairement à beaucoup d'outils de gestion de sources, CVS est vraiment multi-utilisateurs : pas besoin de pénibles réservations lorsque vous voulez modifier un fichier, fusion automatique des changements (si cela est possible : autrement, CVS vous prévient du conflit et vous demande de le résoudre à la main). Et CVS fonctionne en client/serveur, ce qui vous permet de travailler sur des machines différentes, chcun avec son environnement. Cela autorise aussi le travail depuis une machine non-Unix, par exemple un Macintosh.

CVS a de nombreuses fonctions utiles pour les développeurs dispersés sur Internet : on peut par exemple lui demander d'envoyer un message à une adresse pré-définie à chaque fois qu'une modification est enregistrée (on dit "*commited*").

Voyons quelques exemples d'utilisation de CVS.

D'abord un exemple simple avec des développeurs travaillant sur la même machine. CVS va utiliser les protections Unix (pas de "*setuid*").

```
# On crée le répertoire où CVS va stocker ses fichiers
/var/tmp/essais-CVS> mkdir Repository

/var/tmp/essais-CVS/Work> setenv CVSROOT /var/tmp/essais-CVS/Repository
/var/tmp/essais-CVS/Work> cvs init

# On crée un répertoire de travail, ici pour le projet "toto"
/var/tmp/essais-CVS/Work> mkdir $CVSROOT/toto

# On lui donne les bonnes protections
```

```
/var/tmp/essais-CVS/Work> chgrp -R toto $CVSROOT/toto
/var/tmp/essais-CVS/Work> chmod g+w -R $CVSROOT/toto

# Maintenant, le premier développeur va pouvoir travailler. Il commence
# par sortir ("co" = check out) le répertoire.
/var/tmp/essais-CVS/Work> cvs co toto
cvs checkout: Updating toto
/var/tmp/essais-CVS/Work> cd toto/

# Puis, il crée des fichiers, qu'il peut ajouter à CVS et enregistrer
# ("commit") quant il en est satisfait.
/var/tmp/essais-CVS/Work/toto> emacs main.c
/var/tmp/essais-CVS/Work/toto> cvs add main.c
cvs add: scheduling file `main.c' for addition
cvs add: use 'cvs commit' to add this file permanently
/var/tmp/essais-CVS/Work/toto> cvs commit -m "Premiere version"
cvs commit: Examining .
RCS file: /var/tmp/essais-CVS/Repository/toto/main.c,v
done
Checking in main.c;
/var/tmp/essais-CVS/Repository/toto/main.c,v  <--  main.c
initial revision: 1.1
done

# Un autre développeur en fait autant. Pour se synchroniser, le
# premier dévelopeur met à jour ("update") son répertoire :
/var/tmp/essais-CVS/Work/toto> cvs update
cvs update: Updating .
U main.c

# Il voit que main.c a été modifié. Il peut regarder l'historique :
/var/tmp/essais-CVS/Work/toto> cvs log main.c
RCS file: /var/tmp/essais-CVS/Repository/toto/main.c,v
Working file:
main.c head: 1.2 branch: locks: strict access list: symbolic names:
keyword substitution: kv total revisions: 2; selected revisions: 2
description: ---------------------------- revision 1.2 date:
1999/03/20 20:10:38; author: eve; state: Exp; lines: +1 -0 Appel de
toto() ajoute ---------------------------- revision 1.1 date:
1999/03/20 20:07:21; author: stephane; state: Exp; Premiere version
=============================================================================

# Puis il le change, mais l'autre développeur en fait autant en même
# temps. cette fois, il y a conflit :
/var/tmp/essais-CVS/Work/toto> cvs commit -m "Declaration de i"
cvs commit: Examining .
cvs commit: Up-to-date check failed for `main.c'
cvs [commit aborted]: correct above errors first!

/var/tmp/essais-CVS/Work/toto> cvs update
cvs update: Updating .
RCS file: /var/tmp/essais-CVS/Repository/toto/main.c,v
retrieving revision 1.2
retrieving revision 1.3
Merging differences between 1.2 and 1.3 into main.c
M main.c

# Ici, CVS a réussi à résoudre le conflit seul. Il faut parfois
# l'aider.
```

Maintenant, le cas d'un gros projet avec CVS anonyme. Beaucoup de projets de logiciels libres utilisent CVS, puisque ces projets rassemblent souvent de nombreux volontaires répartis sur une large zone géographique. Les développeurs ont un accès en écriture. Les autres peuvent récupérer des fichiers en « CVS anonyme ». Voyez par exemple les instructions pour accéder à FreeBSD <http://www.freebsd.org/handbook/anoncvs.html> (on peut aussi y voir un accès à CVS via le Web <http://www.freebsd.org/cgi/cvsweb.cgi>).

Autre exemple, cette fois pour des gens qui travaillent souvent seuls. Les développeurs de Debian/Linux se servent souvent de CVS <http://www.debian.org/devel/cvs_packages> pour gérer le fait qu'ils doivent modifier les programmes originaux pour les intégrer dans Debian. Le mécanisme des « branches » de CVS permet en effet de gérer facilement les deux fils : celui du développeur extérieur et celui du développeur Debian.

Enfin, dernier exemple, comment gérer un serveur Web avec CVS et l'outil WML <https://www.bortzmeyer.org/wml.html>. Cet exemple est plus complexe et moins pédagogique, mais c'est un exemple réel.

Principe : tout passe par CVS. Avantage : pas besoin d'être logué sur la machine et même pas besoin de connaitre Unix. Tout a été testé avec le client MacCVSPro sur Mac, qui est libre (voir plus loin).

Principe 2 : CVS gère les sources WML, pas leurs dérivés HTML. (Le même système marche si certains répertoires contiennent du HTML fait par une autre méthode que WML, ce qui était le cas pour www.freenix.org.)

Le fichier loginfo contient (src est le module CVS où se trouvent les sources, htdocs aurait été plus malin) :

```
^src (echo ""; echo ""; date; tee /tmp/cvs-$USER.log ;
      (sleep 2; cd /var/www-fubar/src; cvs -q update -P -d;
       make recursive;
       /opt/lib/cvs/log.pl -m webmaster@fubar.com -f /dev/null %s <
             /tmp/cvs-$USER.log;
       rm /tmp/cvs-$USER.log) &) >> $CVSROOT/CVSROOT/updatelog 2>&1
```

Inconvénient : c'est asynchrone, à cause des verrous de CVS. Il y a donc un léger délai entre le "*commit*" et le moment où la modification apparait dans un client Web.

Autre inconvénient : si on n'a pas WML sur sa machine (cas du Mac ci-dessous), il faut "*commiter*" pour voir. Si on l'a, on peut toujours cvs update dans un répertoire à soi, et make recursive.

Autrement, cela fournit tous les avantages de CVS : la nouvelle version est enregistrée, webmaster est prévenu et make va appeler WML.

/var/www-fubar/src est le DocumentRoot d'Apache. Les sources WML s'y trouvent donc (voir plus loin).

Pour travailler à plusieurs, on utilise les groupes Unix avec un compte par personne (c'est plus facile à gérer pour un Unixien) :

```
/var/www-fubar> ls -alt
total 7
drwxrwxr-x   6 stephane fubar        1024 Mar 18 17:13 src
drwxrwxr-x   6 stephane fubar        1024 Feb  6 14:04 .
drwxrwxr-x   2 stephane fubar        1024 Feb  6 11:44 CVS
-rw-rw-r--   1 stephane fubar         116 Feb  6 11:43 .wmlrc
drwxrwxr-x   4 stephane fubar        1024 Feb  6 11:42 WML
```

Pour contrôler la récupération par le Web (on `cvs checkout` directement dans le `DocumentRoot`), on protège :

```
<DirectoryMatch "/CVS/">
order allow,deny
deny from all
</DirectoryMatch>
<FilesMatch "^Makefile">
order allow,deny
deny from all
</FilesMatch>
```

Par contre, les sources WML sont récupérables, ce qui me semble un avantage. Le "*template*" WML peut fournir un petit lien en bas de chaque page ("*See the source*").

Je ne montre pas le `Makefile`, il est assez simple. Il y a juste une petite astuce pour les répertoires contenant de l'HTML fait par un autre moyen que WML (il ne faut pas les détruire par un `make clean`!). Il utilise toutes les spécificités de GNUmake en matière de régles à suffixe et de `ifneq`.

À propos de MacCVSPro `<http://www.maccvs.org>` : bien que son nom puisse faire penser à la version commerciale d'un friouère, c'est un logiciel libre (GPL), ce qui est exceptionnel dans le monde Mac. Très, très bien, très Machinetocien. Gros avantage : on peut raisonnablement demander à des purs Machinetociens de l'utiliser, et donc de participer à un développement géré par CVS. Le fait que les commandes WML soient toutes exécutées par make, lui même exécuté par loginfo, fait que le Machinetocien n'a à pas connaitre le shell.

Ses faiblesses : nécessite Open Transport, messages d'erreur imbitables en cas de mot de passe erroné ou de démon ne tournant pas sur le serveur (Erreur -3158), ne supporte pas SSH comme méthode d'accès.

Pour plus de renseignements sur CVS, voir le serveur de référence `<http://www.loria.fr/~molli/cvs-index.html>`, l'excellent serveur de Cyclic `<http://www.cyclic.com/cyclic-pages/cvsdev.html>`, ou bien mon article Versionnage : garder facilement trace des versions successives d'un document `<https://www.bortzmeyer.org/cvs-versionnage-document-numerique.html>`.