CVS
Concurrent Version System

D. Monniaux

23 juin 1998

Ce petit texte d'utilisation ne se substitue pas à la documentation officielle (`http://www.eleves.ens.fr:8080/tuteurs/cvs/reference`).

1 Qu'est-ce que le contrôle de version ?

Dès que l'on doit gérer un document ou un logiciel complexe, qui doit évoluer avec le temps, ou en collaboration entre plusieurs personnes, se pose le problème du contrôle de versions, à savoir des procédures à mettre en œuvre pour pouvoir garder des copies de sauvegardes des versions antérieures, pour pouvoir utiliser les modifications des autres etc... Les méthodes habituelles (copie de fichiers) aboutissent le plus souvent à une anarchie peu productive (« Mais tu n'as pas pris la dernière version ! — Ben elle était où ? — Dans `répertoire_obscur_au_nom_grotesque`, évidemment ! »).

CVS est un système où il y a un espace de stockage centralisé (*repository*). Cet espace contient toutes les versions déjà réalisées du logiciel ; afin d'économiser de la place, on ne stocke que les différences entre versions des fichiers. Le système tient un journal des modifications, avec le nom de leur auteur, leur date... Il n'y a pas de contraintes empêchant plusieurs personnes d'éditer le même fichier en même temps ; les modifications seront combinées ; au besoin on demandera à l'utilisateur de résoudre les conflits.

2 Utilisation pratique de CVS

2.1 Les commentaires

À plusieurs reprises, CVS demande un commentaire et ouvre un éditeur de texte (spécifiable par la variable d'environnement `EDITOR`). Les lignes commençant par `CVS :` seront effacées par CVS. Il est recommandé d'y indiquer ce que l'on a modifié, en gros.

2.2 *Repository*

2.2.1 Configuration

La variable d'environnement `CVSROOT` pointe sur un le *repository*. On peut aussi spécifier le chemin d'accès à ce dernier par l'option `-d`.

La première fois, on initialise le repository par la commande `cvs init`.

Une liste de modules est définie dans `CVSROOT/modules`. Pour l'éditer, faire

```
cvs checkout CVSROOT
cd CVSROOT
$EDITOR modules
cvs commit
```

Cette liste permet de définir des chemins d'accès standard. Par exemple, celle des tuteurs est :

```
# Three different line formats are valid:
# key -a    aliases...
# key [options] directory
# key [options] directory files...
#
# Where "options" are composed of:
# -i prog Run "prog" on "cvs commit" from top-level of module.
# -o prog Run "prog" on "cvs checkout" of module.
# -e prog Run "prog" on "cvs export" of module.
# -t prog Run "prog" on "cvs rtag" of module.
# -u prog Run "prog" on "cvs update" of module.
# -d dir Place module in directory "dir" instead of module name.
# -l Top-level directory only -- do not recurse.
#
# NOTE:  If you change any of the "Run" options above, you'll have to
# release and re-checkout any working directories of these modules.
#
# And "directory" is a path to a directory relative to $CVSROOT.
#
# The "-a" option specifies an alias.  An alias is interpreted as if
# everything on the right of the "-a" had been typed on the com-
mand line.
#
# You can encode a module within a module by using the special '&'
# character to interpose another module into the current module.  This
# can be useful for creating a module that consists of many directories
# spread out over the entire source repository.
CVSROOT CVSROOT
modules CVSROOT modules
images documentation/images
tex-tuteurs utilitaires/tex
doc-cvs documentation/cvs
```

2.2.2 Créer un module

`cvs import` *`répertoire_repos vendortag releasetag`* va recopier les fichiers « intéressants » du répertoire courant vers le sous-répertoire indiqué ; mettez votre nom dans *vendortag* et une indication de version dans *releasetag*.

2.3 Une session type

☞ *Ceci n'est qu'une présentation rapide. On se reportera à la documentation de CVS pour plus de renseignements.*

2.3.1 Obtenir une copie personnelle des sources

On utilise `cvs checkout` *`module`*.

```
bireme-~ $ cvstuteurs checkout images
cvs checkout: Updating images
U images/Makefile
U images/antialias.gif
U images/antialias.originale.gif
U images/bitmap.gif
U images/gimp-menu-legende.fig
U images/gimp-menu.gif
U images/hls.fig
U images/images.tex
U images/touche-meta-sun.fig
U images/touche-meta.gif
U images/xv-accueil.gif
U images/xv-menu-legende.fig
U images/xv-save-jpeg.gif
U images/xv-schnauzer.gif
```

2.3.2 Ajouter un fichier

Les commandes à utiliser sont `cvs add` *`fichier`* pour ajouter un fichier texte, `cvs add -kb` *`fichier binaire`* pour ajouter un fichier binaire[1].

☞ *Cette opération n'ajoutera le fichier dans le* repository *que lorsque les changements auront été confirmés (voir 2.3.5).*

2.3.3 Effacer un fichier

La commande à utiliser est `cvs delete` *`fichier`*. Les versions précédentes seront conservées dans un répertoire spécial. Le fichier doit être effacé dans le répertoire de travail pour que cette commande fonctionne.

2.3.4 Éditer des données...

Par les moyens habituels !

2.3.5 Confirmer ses changements

`cvs commit` met les changements dans le *repository*, en créant bien evidemment une nouvelle version pour tous les fichiers modifiés.

[1] La différence entre le traitement des fichiers textes et binaires est que dans le second cas, CVS n'essaiera pas de faire des mise à jour automatique des numéros de versions dans les fichiers et ne les comparera pas ligne à ligne.

3 Conseils d'utilisation

3.1 Comment collaborer

Il est conseillé de ne soumettre de modifications que lorsque le code sinon fonctionne, du moins compile correctement, afin de ne pas trop perturber les autres développeurs.

3.2 Pour les tuteurs

La commande à utiliser pour le CVS interne aux tuteurs est `~tuteurs/bin/cvstuteurs` au lieu de `cvs` ; il s'agit d'un script qui se charge des droits et du chemin d'accès corrects.