Équipe GXPlor
Université de Marne-la-Vallée
E-Mail : neilb@free.fr

CVS

Manuel d'utilisation

Table des matières

Auteur : Stéphane Moreau
Date : 16/10/03

1 Introduction

La philosophie de CVS[1] est de garder l'historique des modifications de fichiers ASCII ou binaires. Il garde en mémoire **qui** a fait **quoi**, **pourquoi** et **comment** (si l'auteur a bien voulu le dire).

Cela permet bien sûr d'avoir toujours accès aux versions antérieures de son travail, et aussi de travailler à plusieurs sur le même projet, c'est-à-dire de modifier « en même temps » le même fichier. Ce « en même temps » existe réellement car chaque utilisateur travaille sur une copie de l'original.

En fait, personne ne travaille sur les originaux (même l'administrateur du projet) ; toutes les modifications des originaux passent par CVS. Les fichiers originaux sont modifiés quand un utilisateur « renvoie » ses nouvelles versions des fichiers. Les anciens fichiers ne sont jamais détruits (ce qui évite l'écrasement d'un projet par un utilisateur peu scrupuleux) ; seules les différences sont enregistrées, il est alors possible de revenir en arrière, et ce indépendamment pour chaque fichier.

Chaque fichier possède donc son propre historique, et il est possible de garder un « souvenir », à l'aide d'une marque (*tag*), d'une version d'un projet (*release*) à un moment donné et de pouvoir le rappeler en une seule fois quand cela est nécessaire.

Le changement de version d'un fichier est appelé révision *revision*. Notez la différence entre une révision, c'est-à-dire l'enregistrement de l'évolution d'un fichier par CVS (lors de la soumission des modifications), et une version, c'est-à-dire une étape importante de l'évolution d'un fichier ou ensemble de fichiers (projet), enregistrée par l'administrateur à l'aide d'un *tag*.

Chaque révision de fichier « doit » être accompagnée d'un commentaire de l'utilisateur qui fournit sa modification ; ce commentaire est très utile pour la relecture future des modifications passées, par lui-même comme par les autres. Il ne faut donc pas hésiter à fournir quelques détails qui apporteront plus d'informations que les modifications elles-mêmes.

[1] Concurrent Version System

2 Configuration du système

2.1 Configuration de l'accès SSH[2] pour Savannah

Pour pouvoir se connecter au CVS de Savannah, il faut tout d'abord configurer l'accès SSH.

Premièrement, « Protocol 1 » doit être écrit dans le fichier `config` de SSH :

```
echo "Protocol 1" > ~/.ssh/config
```

Après cela, il faut créer la clef ssh. Pour ce faire, il faut taper :

```
ssh-keygen -t rsa1
```

Cette clef publique sera placé dans le fichier `~/.ssh/identity.pub`.

Ensuite, cette clef doit être enregistrée sur le compte de Savannah. Après s'être identifié sur ce serveur, il faut aller sur la page :

http://savannah.gnu.org/account/editsshkeys.php,

copier le contenu du fichier `~/.ssh/identity.pub` dans la zone de texte et appuyer sur le bouton "`Mise-à-jour`".

Il faut finalement attendre environs 6 heures pour que Savannah prenne en compte cette clef.

2.2 Initialisation des variables CVS

Tout d'abord, il faut initialiser la variable d'environnement `CVSRSH` :

```
export CVSRSH=ssh
```

Ensuite, la variable d'environnement `CVSROOT` :

```
export CVSROOT=<user>[3]@subversions.gnu.org:/cvsroot/<projet>[4]
```

Et enfin, la variable `CVSEDITOR` :

```
export CVSEDITOR=<editeur>[5]
```

qui permet à CVS de savoir quel éditeur de texte ouvrir lorsqu'il demandera d'entrer un commentaire pour une modification. Si cette variable n'est pas initialisée, CVS cherchera ensuite dans la variable `EDITOR` puis, si cette dernière n'est pas initialisée non plus, il retournera un message d'erreur comme celui-ci :

```
cvs [commit aborted]: no editor defined, must use -e or -m
```

L'option `cvs -e <editeur>` permet de spécifier un éditeur de texte lors de l'utilisation de la commande ; cette option prévaut sur les variables d'environnement.

L'option `cvs -m "<commentaire>"` permet d'éviter l'appel à un éditeur de texte en donnant le commentaire directement sur la ligne de commande.

Par la suite, il est bon de placer ces trois commandes `export` dans le fichier `.bashrc` pour éviter de les taper à chaque fois.

---

2 Secure SHell
3 Remplacer `<user>` par le login de l'utilisateur sur Savannah
4 Remplacer `<projet>` par le nom du projet, en l'occurrence `gxplor`
5 Remplacer `<editeur>` par l'éditeur de texte préféré comme `vi`, `emacs`, `nedit` ...

E-Mail : neilb@free.fr

3 Commandes utiles du CVS

3.1 La commande *checkout*

Après avoir configuré sa machine, la première chose à faire afin de travailler sur le projet est de le récupérer par la commande `checkout` :

```
cvs checkout <projet>
```

Cette commande sans options permet de récupérer un projet dans sa version la plus récente.

Il est aussi possible de travailler sur une version plus ancienne du projet. Cela veut dire qu'à un moment donné de ce projet, les fichiers étaient dans un état satisfaisant par l'administrateur du projet et que celui-ci a mis une marque (*tag*) sur tous les fichiers (indépendamment de leur numéro de révision).

Pour récupérer une version plus ancienne :

```
cvs checkout -r <version>[6] <projet>
```

3.2 La commande *update*

Comme la plupart des commandes CVS, `update` s'utilise en précisant ou non un nom de fichier après ; cette commande permet de se mettre à jour par rapport au dépôt, pour un fichier donné ou l'ensemble du projet :

```
cvs update <fichier>[7]   pour mettre à jour un fichier
ou cvs update             pour mettre à jour un projet
```

Lors d'une mise à jour, CVS rassemble (*merge*) alors les modifications, avec celles des dernières versions existantes. En général cela se passe sans mal, mais si les modifications sont importantes (d'où l'intérêt de ne pas soumettre ses modifications trop peu fréquemment) ou si celles qui ont été faites sur ce fichier par quelqu'un d'autre sont trop importantes, un message de ce genre peut s'afficher :

```
cvs update : Updating .
RCS file: <chemin du projet>toto.java,v[8]
retrieving revision x.x
retrieving revision x.y[9]
Merging differences between x.x and x.y into toto.java[10]
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in toto.java
C toto.java
```

C'est le message d'un conflit, c'est-à-dire que CVS n'a pu rassembler les différences tout seul ; il faut donc le faire à la main.

---

6 Remplacer `<version>` par le nom de la version.
7 Remplacer `<fichier>` par le nom du fichier.
8 `toto.java,v` est le nom sous lequel CVS sauvegarde les fichiers dans un dépôt.
9 `x.x` et `x.y` sont les numéros de révision de `toto.java` concernés.
10 `toto.java` est le fichier problématique.

En général cela n'arrive pas souvent, et quand cela arrive, les corrections sont triviales à effectuer pour un humain. CVS indique à l'intérieur du fichier `toto.java` les conflits comme ceci :

```
<<<<<<< toto.java
bla bla bla …
=======
bli bli bli …
>>>>>>> x.y
```

Il faut alors choisir (ou faire un « mélange ») entre « bla bla bla » et « bli bli bli ». Lorsque cela est fait, un `cvs commit toto.java` devrait marcher, sauf bien sûr, si quelqu'un d'autre a encore modifié le fichier pendant cette modification, auquel cas … il faut recommencer (cela n'arrivera quasiment jamais).

3.3 La commande *commit*

Pour soumettre les modifications d'un fichier sur le CVS, il faut taper :

```
cvs commit <fichier>
```

Lors de la soumission, un éditeur va s'ouvrir permettant d'écrire un commentaire sur les modifications du fichier. Il est fortement conseillé d'écrire un commentaire ce qui facilitera la relecture future des modifications passées.

Pour soumettre toutes les modifications de plusieurs fichiers d'un seul coup, et de n'écrire qu'un seul commentaire, il faut taper :

```
cvs commit
```

Si jamais un message du genre :

```
cvs commit: Examining .
cvs commit: Up-to-date check failed for 'toto.java'
cvs [commit aborted]: correct above errors first!
```

apparaît à votre écran, c'est que le fichier `toto.java` a été modifié depuis la dernière récupération, il faut alors faire une mise à jour (*update*) avant de refaire un `commit`.

3.4 La commande *add*

Lors d'un `commit` d'un projet, CVS n'ajoute et ne retire de fichiers **que** si ça lui a été demandé explicitement.

Pour ajouter un fichier :

```
cvs add <fichier>
```

est la commande appropriée, et CVS va congratuler par un :

```
cvs add: adding <fichier>
cvs add: use 'cvs commit' to add this file permanently
```

comme indiqué, il faut poursuivre par un :

```
cvs commit <fichier>
```

3.5 La commande *remove*

Pour retirer un fichier, il faut utiliser la commande :

```
cvs remove <fichier>
```

de la même manière que la commande `add`. Cependant, il faut que ce fichier soit réellement absent du répertoire de travail, sinon CVS va envoyer un message de complainte.

Pour CVS, effacer un fichier n'est qu'une opération « virtuelle », il reste toujours un souvenir (susceptible d'être rappelé) d'un fichier qui a été effacé, ainsi que toute son historique de révisions.

3.6 La commande *release*

Cette commande permet de supprimer un répertoire du CVS :

```
cvs release -d <répertoire>
```

Elle indique que le répertoire n'est plus utilisé tandis que l'option `-d` le supprime effectivement du CVS. Toutefois le répertoire demeurera sur les copies locales des autres utilisateurs, tant que la commande `update -P` n'aura pas été effectuée dans leurs répertoires.

3.7 La commande *log*

Pour visualiser les révisions d'un fichier, c'est-à-dire : les numéros de révisions, qui les a faites, quand, les commentaires, etc., taper :

```
cvs log <fichier>
```

Cette commande ne sera pas plus détaillée ici, pour en savoir plus, consulter le manuel de `cvs` :

```
man cvs
```

4 Préfixes des messages CVS

Lors des opérations sur les fichiers (*add*, *commit*, *update*…), CVS affiche des messages où le nom des fichiers concernés sont précédés d'une lettre en majuscule. Voici les cas les plus courants :

`U <fichier>` = Updated (récupération de la dernière version du fichier) ;
`M <fichier>` = Modified (modification du fichier) ;
`C <fichier>` = Conflict (le fichier contient un conflit) ;
`T <fichier>` = Tagged (marquage du fichier) ;
`? <fichier>` = Unknown (le fichier ne fait pas partie du projet).