Introduction rapide à CVS

Jérôme Petazzoni, `skaya@enix.org`

Ceci document permet une prise en main ultra-rapide de CVS, pour les impatients qui veulent coder maintenant, tout de suite. Les exemples supposent que l'on dispose d'une station de travail (si possible sous Unix) connectée à Internet, avec CVS installé (peu importe la version).

Contents

1 Informations générales sur CVS

1.1 Qu'est-ce que ça veut dire?

Concurrent Version System.

1.2 Qu'est-ce que ça peut faire?

Le but principal de CVS est de maintenir dans un endroit centralisé l'intégralité des sources d'un projet, avec toutes les versions successives de chaque fichier. Ainsi, on peut facilement retracer l'évolution du source. De plus, CVS prévoit que plusieurs programmeurs travaillent simultanément sur les mêmes fichiers, et tente de gérer au mieux les mises à jour simultanées de ces fichiers.

Par exemple : Alice modifie `types.h` en rajoutant un type, Bob modifie ce même fichier en modifiant un autre type. Lors de la mise à jour, CVS regarde les différences entre le fichier original et celui édité par le programmeur et incorpore les modifications. Lorque elles concernent exactement la même section de code, il y a "conflit" et il faut qu'un des deux programmeurs s'en occupe manuellement.

1.3 Qu'est-ce que ça ne peut pas faire?

Le rôle de CVS n'est pas de gérer la compilation. Les Makefile, autoconf et autres sont là pour ça. De plus, CVS n'est pas spécialement adapté au suivi d'un standard ou d'un protocole, ou d'autres documents textuels. Il ne remplace pas non plus les réunions de travail et autres brainstormings.

2 Se préparer à utiliser CVS

De quoi a-t-on besoin? Il faut le programme CVS. On peut l'utiliser en "stand-alone", lorsque tous les développeurs se logent sur la même machine. Mais nous allons ici détailler l'utilisation d'un *pserver*, où les programmeurs se connectent par le biais d'un réseau TCP/IP (Internet par exemple, ou un réseau local).

L'administrateur du serveur CVS doit donner aux développeurs les informations suivantes : le nom du serveur, le chemin d'accès (`/var/cvs/superprojet`), ainsi qu'un login et un mot de passe. Il ne faut pas nécessairement un compte Unix sur le serveur pour utiliser le CVS.

Pour vérifier que CVS est bien installé sur votre machine, essayez de lancer `cvs`, tout simplement. S'il trouve, il doit répondre par une page d'aide. D'autre part, pour vérifier que le serveur CVS de démonstration est disponibles, essayez de faire `ping cvs.enix.org`.

2.1 Configurer l'environnement

Il n'y a presque rien a faire. Sauf : positionner une variable d'environnement, CVSROOT, afin d'indiquer le serveur CVS qu'on souhaite utiliser, ainsi que le nom d'utilisateur et la repository.

Par exemple :

```
export CVSROOT=:pserver:ecrivain@cvs.enix.org:/var/cvs/demo
```

NB : il est aussi possible d'utiliser l'option `-d` du programme `cvs`, par exemple `cvs -d :pserver:ecrivain@cvs.enix.org:/var/cvs/demo login`.

2.2 Organisation des *repositories*

Sur un même serveur CVS, on peut avoir plusieurs projets qui cohabitent, dans des *repositories*. Et dans chaque repository (en général, on peut envisager une par projet), on a des modules. En effet, lorsqu'on travaille sur un projet, celui-ci est compose de plusieurs modules (completement independants) qu'on rapatrier separement depuis le CVS. Par exemple, on peut avoir un module doc, un module src, et un module bench. Ici, le login est `ecrivain`, le serveur est `cvs.enix.org`, et la repository est `/var/cvs/demo`. On suppose qu'on est dans un nouveau repertoire (par exemple DemoCVS).

2.3 Se "loger" sur le serveur CVS

Cette operation n'est à effectuer que la première fois qu'on travaille sur la repository. Il suffit de taper :

```
cvs login
```

Le mot de passe va être demandé et verifié auprès du serveur. Pour cette demonstration, utiliser le mot de passe `ecrivain`. Si le programme revient à l'invite sans rien dire, c'est qu'il n'y a pas de problème. Sinon il doit donner un message d'erreur (type : mot de passe invalide, par exemple).

2.4 Créer un nouveau module

Créons un repertoire `trucmuche`. (Pour que la demo marche, mettez autre chose que `trucmuche`, il y a surement quelqu'un qui l'a déjà mis!) Dans ce répertoire, placez un fichier `main.c` et un `Makefile`, par exemple, avec quelques lignes chacun. Ensuite, il faut (en restant dans le répertoire `trucmuche`) taper la commande :

```
cvs import trucmuche HEAD DEVEL
```

Normalement, un éditeur va se lancer pour demander de taper un message, qui sera incorporé au *log* des fichiers.

- `import` signifie qu'on veut ajouter un module avec les fichiers existants dans le repertoire,
- `trucmuche` est le nom du module,
- `HEAD` et `DEVEL` sont des *tags* qui servent à marquer des *branches*. Sans entrer dans les détails, cela permet de maintenir de front plusieurs versions du programme, par exemple une version `ALPHA` et une version `STABLE`, et de les faire évoluer séparément, en intégrant au coup par coup les modifications de l'une dans l'autre.

Une fois que l'import est effectué, on peut (il est même recommandé) d'effacer le repertoire trucmuche (sauf bien entendu s'il y a eu une erreur!).

3 Utilisation concrète

3.1 Récupérer les sources

Maintenant, se replacer dans le répertoire DemoCVS. Pour commencer à travailler sur le projet, il faut utiliser la commande de `checkout` :

```
cvs checkout trucmuche
```

Cela va créer un répertoire `trucmuche` contenant les fichiers qu'on a importés tout à l'heure, ainsi qu'un sous répertoire CVS contenant des informations propres au programme (ne pas y toucher).

3.2 Faire des modifications

Une fois qu'on a fait un checkout, on peut éditer les sources avec ses outils favoris, comme si de rien n'était.

3.3 Mettre sa copie de travail à jour

Lorsqu'on souhaite récuperer les dernières mises à jour, par exemple juste avant de synchroniser sa copie de travail avec les sources du serveur, il faut utiliser `cvs update`.

3.4 Propager ses modifications

Une fois les modifications jugées satisfaisantes, pour les incorporer au source, utiliser `cvs commit`. Comme pour un import, un éditeur va etre lancé pour taper un commentaire qui sera ajoute au log.

3.5 Résolution des conflits

S'il y a des conflits, CVS l'indique et modifie les fichiers concernés en plaçant des repères aux endroits qui posent problème. Il suffit d'éditer les fichiers et d'effectuer la fusion à la main.

3.6 Ajouter / enlever des fichiers

Pour ajouter un fichier, faire `cvs add nomdufichier.c`, le fichier ne sera vraiment ajouté qu'au prochain `commit`.

Pour enlever un fichier, c'est `cvs remove`. Pour renommer un fichier, il faut l'ajouter sous son nouveau nom puis enlever l'original. Le problème est que cela fait perdre les informations de version (il faut examiner l'ancien fichier pour remonter les versions).