

CVS : Systèmes de gestion de versions

1 Qu'est-ce que cvs ?

CVS (*Concurrent Versions System*) aide à gérer le développement de projets réalisés par des équipes de une ou plusieurs personnes. Avec son successeur, le système Subversion, ce sont aujourd'hui les systèmes de gestion de versions de fichiers les plus répandus dans le monde Unix.

Il n'est en fait pas nécessaire d'avoir participé à la réalisation de tels projets pour comprendre l'intérêt de cet outil. Votre expérience de la programmation en binôme vous a déjà certainement permis de réaliser que travailler à deux en même temps sur un projet implique inévitablement des conflits de gestion de versions. Lequel d'entre vous ne s'est pas déjà posé au moins une fois les questions suivantes :

- Qui de nous deux a la dernière version du projet ?
- Si je fais une bêtise sur le version courante, comment revenir en arrière ?
- Comment faire pour partager la version courante du projet avec mon binôme ?

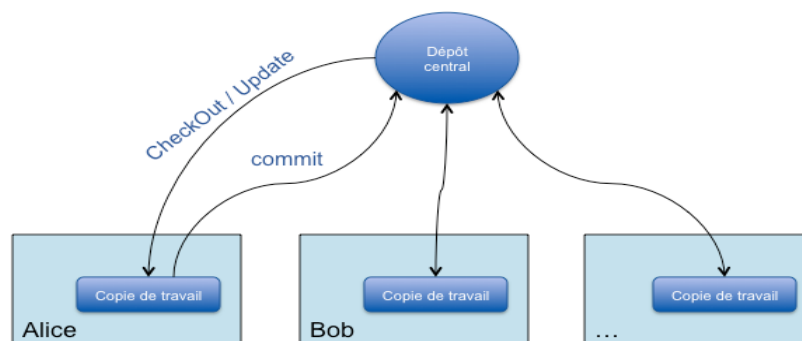
Les réponses apportées à ces questions sont généralement :

- Regarder les dates des fichiers.
- Faire des fichiers de sauvegarde.
- Stocker uniquement le projet chez moi (ou chez lui) et s'échanger nos mots de passe.
- S'envoyer les fichiers du projet par email, disquette, clé USB.

Ce sont ces problèmes que les outils de gestion de versions comme CVS s'attachent à résoudre. CVS permet de garder l'historique des modifications des fichiers du projet. Il garde en mémoire qui a fait quoi, pourquoi et comment. Cela permet bien sûr d'avoir toujours accès aux versions antérieures de son travail, et aussi de pouvoir travailler à plusieurs sur le même projet, c'est-à-dire d'avoir la possibilité de modifier (en même temps) le même fichier.

2 Architecture générale de cvs

CVS fonctionne selon une architecture **client/serveur** décrite par la figure ci-dessous.



Le serveur gère un *entrepôt centralisé* unique pour chaque projet où il stocke la version de référence de chaque fichier, ainsi que tout l'historique de ses modifications. Les clients travaillent alors sur une copie des fichiers et envoient à l'entrepôt, grâce à CVS, leurs modifications lorsqu'ils sont satisfaits de la version modifiée qu'ils ont chez eux.

3 Syntaxe de cvs

La syntaxe générale de la commande `cvs` est la suivante :

```
cvs [options générales] commande [options commande]
```

Les options générales affectent le comportement global de `cvs` tandis que les options de la commande sont spécifiques à celle-ci. Les principales commandes sont : `init`, `import`, `commit`, `checkout`, `update`, `add` et `remove`. Attention, certaines options (par exemple `-d` qui fixe l'emplacement de l'entrepôt) peuvent exister à la fois comme option générale et comme option de commande.

4 Premiers pas

Les exercices de cette section vont vous permettre de créer un entrepôt, d'y déposer votre premier projet et d'apprendre les manipulations de base (ajout, suppression, renommage de fichiers et répertoires).

4.1 Création de l'entrepôt : `cvs init`

La première chose à faire pour utiliser `cvs` est de créer un entrepôt. La commande suivante initialise un entrepôt à la racine de votre compte :

```
cvs -d ~/entrepot_cvs init
```

Cette commande crée un répertoire `entrepot_cvs` (s'il n'existe pas déjà) à la racine de votre compte. Ce répertoire contiendra les versions de référence et les historiques des fichiers de vos projets. On peut se passer d'utiliser l'option `-d` dans cette commande. Pour cela, il suffit d'indiquer l'endroit souhaité pour l'entrepôt dans la variable d'environnement `CVSROOT` de la manière suivante :

```
export CVSROOT=~/entrepot_cvs (pour BASH)
```

ou

```
setenv CVSROOT ~/entrepot_cvs (pour TCSH)
```

En plus du répertoire `entrepot_cvs`, la commande d'initialisation a créé un sous-répertoire `CVSROOT` qui contient tous les fichiers administratifs relatifs à la gestion de l'entrepôt par CVS.

À vous de jouer. Taper cette commande et vérifier que le répertoire `entrepot_cvs` (et son sous-répertoire `CVSROOT`) a bien été créé à la racine de votre compte. Effacer ce répertoire, créer la variable d'environnement `CVSROOT` comme indiqué et taper la commande `cvs init`. Vérifier qu'elle a bien le même effet.

4.2 Déposer un projet dans l'entrepôt : cvs import

Afin d'illustrer le dépôt d'un projet dans l'entrepôt, récupérer le fichier `projet-tp5.tgz` à l'aide de la commande suivante :

```
wget www.lri.fr/~conchon/projet-cvs.tar
```

En tapant la commande suivante :

```
tar xvf projet-cvs.tar
```

un répertoire `monprojet` est créé. Supposons maintenant que vous souhaitez utiliser CVS pour vous aider à gérer les fichiers qu'il contient. Le dépôt de ce projet dans l'entrepôt se fait alors à l'aide des commandes suivantes :

```
cd tp5
cvs import -m "creation" monprojet <votre login> initial
```

À vous de jouer. Exécuter ces différentes commandes et regarder le contenu du répertoire `~/entrepot_cvs`. Consulter la page `man` de CVS, et trouver ce que signifie les différentes options de la ligne de commande ci-dessus, en particulier l'option `-m`.

4.3 Projets gérés par CVS : cvs checkout

Maintenant que le projet est enregistré dans votre entrepôt CVS, vous pouvez détruire le répertoire `monprojet` (qui ne peut de toute façon pas être utilisé comme copie locale). La commande `checkout` (dont l'abréviation est `co`) permet de faire une copie locale des fichiers composant un projet sous CVS.

La commande suivante permet de récupérer le projet `monprojet` enregistré sous CVS :

```
cvs checkout monprojet
```

À vous de jouer. Vérifier que la copie est correctement faite : pour cela se déplacer dans le répertoire `monprojet/`. Vous noterez la présence d'un nouveau répertoire `CVS`. Ce répertoire est utilisé par CVS pour sa gestion interne, et contient notamment l'adresse du serveur à partir duquel l'import a été effectué. Ainsi la variable `CVSROOT` ou l'option `-d` ne sont plus nécessaire pour spécifier l'adresse du serveur pour les commandes à venir. Taper la commande `cvs status` qui vous renseigne sur l'état de votre copie locale par rapport à la version située sur le serveur.

4.4 Enregistrer une nouvelle version d'un fichier : cvs commit

Une fois la copie locale (*working revision*) sur votre compte, vous pouvez modifier les fichiers qu'elle contient à votre guise, sans craindre de gêner qui que ce soit. En effet, personne ne peut voir vos modifications tant que vous ne les avez pas enregistrées dans l'entrepôt.

La commande permettant de remplacer un fichier de l'entrepôt par votre version de travail est la suivante :

```
cvs commit -m "commentaire" <fichier>
```

Si aucun fichier n'est précisé, cette commande enregistre les modifications de tous les fichiers du projet.

À vous de jouer. Modifiez le fichier `numerote.ml` contenu dans le répertoire `sources` du projet en rajoutant un commentaire. Quelles indications vous donne la commande `cvs status sources/numerote.ml` exécutée avant et après la modification d'un fichier.

4.5 Mise à jour du projet : `cvs update`

La commande suivante met à jour les fichiers se trouvant dans votre copie locale en accord avec ceux contenus dans l'entrepôt.

```
cvs update
```

Cette commande est utilisée pour mettre à jour ses fichiers lorsqu'une autre personne a modifié la version courante du projet stockée dans l'entrepôt. Elle conserve les changements locaux effectués. Aussi, cette mise à jour peut donc provoquer des *conflicts* entre les fichiers de votre copie locale et ceux de l'entrepôt. Nous reviendrons sur ces conflits un peu plus tard.

À vous de jouer. Effacer le fichier `LISEZMOI.txt` de votre copie locale (à l'aide de la commande `rm`) puis utiliser la commande `update`. Que se passe-t-il ?

4.6 Ajout et suppression de fichiers/répertoires

L'ajout d'un fichier dans un projet se fait à l'aide de la commande :

```
cvs add <fichier>
```

La suppression d'un fichier nécessite d'abord de détruire votre copie locale du fichier (à l'aide de la commande `rm`) puis de l'indiquer au serveur avec la commande :

```
cvs remove <fichier>
```

Néanmoins, ces modifications ne sont pas immédiates. Pour confirmer un ajout ou une suppression, il est nécessaire de faire ensuite un `cvs commit` sur les fichiers concernés. Sinon les changements ne seront jamais pris en compte !

À vous de jouer. Créez un fichier `AUTEURS.txt` qui contient votre nom et prénom. Ajoutez-le à l'entrepôt. Vérifiez le statut du répertoire `monprojet/` vis à vis de l'archive CVS. Détruisez le répertoire `monprojet/`. Puis refaites un `checkout` du module `monprojet`. Supprimez localement le fichier `LISEZMOI.txt`. Vérifiez le statut du répertoire `monprojet/` vis à vis de l'archive. Supprimez le fichier de l'archive CVS.

CVS ne sait pas déplacer de fichier (c'est une de ses limitations). Pour pouvoir simuler un déplacement, il faut donc retirer les fichiers, puis les réinsérer au bon endroit. Sachant cela, débrouillez-vous pour renommer votre répertoire `sources/` en `src/`, à la fois localement et sur l'archive CVS. Que remarquez-vous lors de la prochaine mise à jour ? Trouver l'option de la commande `update` permettant de ne pas faire de copie locale des répertoires vides.

5 Gestion des conflits

Lorsque vous serez plusieurs à travailler sur le même projet, il apparaîtra certainement des conflits. Le cas se présente si vous tentez de modifier une partie de code sur laquelle un autre développeur a déjà enregistré des modifications. Vous allez simuler vous même ces situations en travaillant sur deux copies locales du projet.

Effacer votre copie locale `monprojet/`. Trouver l'option de la commande `checkout` permettant de copier un projet dans un répertoire portant un nom différent de l'archive et créer deux copies locales du projet `monprojet` dans les répertoires `monprojet-version1/` et `monprojet-version2/`.

Modifier, dans la version 1, le fichier `numerote.ml` (en ajoutant un commentaire par exemple) et valider le changement. Quel est le statut de la version 2 ? Pour la mettre à jour par rapport à la dernière version de l'archive CVS, utiliser la commande `update` et vérifier une nouvelle fois le statut de cette version.

Modifier `AUTEURS.txt` dans la version 1 et enregistrer le changement. Modifier sensiblement dans la version 2 le même fichier sans faire de mise à jour préalable, et enregistrer la modification. Que se passe-t-il alors ?

Pour résoudre le conflit il faut mettre à jour `AUTEURS.txt` par rapport à l'archive CVS (en utilisant `update`). La mise à jour modifie le fichier en signalant les zones pour lesquelles des conflits existent. Ces zones sont mises en évidence dans le texte par des séquences de la forme :

```
<<<<<<<<<
texte dans la version 1
=====
texte dans la version 2
>>>>>>>>
```

Effacer dans les zones de conflits le texte correspondant à la version se trouvant dans l'entrepôt et valider ces changements. Vérifier le statut de la version 2 et celui de la version 1. Mettre à jour la version 1 (toujours avec `update`). Quel est le statut de cette version après cette commande ? Comment expliquez-vous cela ?