CVS, le gestionnaire de sources

Article pour l'Echo de Linux (**Septembre 1996**)
Jean−Max REYMOND

Résumé :

CVS, proposé par la FSF foundation est un outil permettant la gestion de sources ainsi que de versions. Il permet entre autres de gérer la modification simultanée de mêmes fichiers par plusieurs utilisateurs.

---

1 Introduction

CVS, (Concurrent Version System) est un outil permettant de gérer l'évolution dans le temps d'un ensemble de fichiers (au sens UNIX de suite d'octets, mais plus fréquemment des sources). CVS, s'appuyant sur RCS permet donc au minimum d'extraire des sources, de les modifier, de soumettre ses modifications, de garder l'historique des modifications et de restaurer n'importe quelle version précédente. Mais l'atout majeur de CVS est de permettre à plusieurs développeurs de travailler sur le même groupe de fichiers: chacun travaille de manière indépendante dans son environnement personnel et CVS gère l'ensemble.

2 Principes généraux de CVS

Voici une explication sommaire des grands principes qui gouvernent CVS. Que le lecteur veuille bien m'excuser pour les anglicismes, mais quelquefois les traductions françaises sont laborieuses.

2.1 La repository

CVS maintient dans un répertoire pointé par la variable d'environnement CVSROOT, une arborescence correspondant à l'arborescence de développement. Cette arborescence abrite des fichiers RCS, chacun contenant le source d'origine ainsi que tous les différentiels à appliquer pour passer à une version plus récente. De même, on retrouve les commentaires à saisir lors d'une demande de modification.

Cette repository n’est pas forcément sur la machine de développement et peut se trouver sur un autre serveur afin de totalement isoler les sources des developpeurs.

2.2 Le mode client−serveur

Pour les cas où plusieurs machines doivent partager les mêmes sources ou dans le but d’isoler les sources des développeurs, un mode client serveur permet de s’affranchir de NFS et de ne passer sur le réseau que les informations utiles sous forme de patch. Ce mode client/serveur est de 4 à 10 fois plus rapide que par NFS et permet donc la gestion des sources entre deux sites très éloignés via des modems. Les accès se font par des commandes "remote−shells" qui doivent satisfaire aux conditions imposées par ce protocole (.rhost, host.equiv, ...).

2.3 La gestion des développements en parallèle

Classiquement, dès que plusieurs developpeurs modifient les mêmes fichiers, les problèmes commencent. Une méthode classique est de verrouiller le fichier afin d’empêcher les autres de faire des modifications. Malheureusement, cette méthode, si elle fonctionne, présente de nombreux inconvénients:

− Le bloquage intégral du fichier même pour modifier une ligne,

− le risque que l’emprunteur verrouille " advitam eternam " le fichier par exemple par une destruction du répertoire contenant le fichier.

Avec CVS, par contre, l’utilisateur possède une copie du fichier, le modifie comme il le veut et c’est au moment de soumettre ses modifications("commit") que CVS va faire une fusion (un "merge") de la version courante avec la version précédemment commitée. S’il y a un conflit dû à une modification simultanée et non identique de la même ligne, alors CVS le signale en entourant les lignes litigieuses.

Cette approche originale a entraînée des débats passionnés mais force est de reconnaître que très généralement, cela marche finement. A l’usage, il faut simplement faire très attention lorsque CVS refuse la livraison du fichier et demande une fusion préalable: il faut bien regarder ce que la fusion entraîne.

2.4 Les tags

De façon interne, CVS maintient des numéros de version pour chaque fichier. Ainsi, par exemple le fichier parserv.c pourra être numéroté 5.3, alors que le fichier scan.c sera à l'indice 1.12. CVS permet d'associer un nom symbolique à un ensemble de fichiers ayant des indices différents. Ainsi, le nom symbolique "release_V1" pourra être appliqué à parserv.c [1.3] et scan.c [1.12]. Plus tard, on pourra au besoin extraire la version release_V1 dans son intégralité, même si la version est beaucoup plus avancée.

2.5 Les branches de développement

Elles correspondent, en particulier, à un soucis très fréquent en informatique: la maintenance. Supposons que quelque mois après la sortie d'une version taggée release_V1, une correction s'impose. Hélas, depuis la release V1, les sources ont été profondément modifiés et ne sont pas livrables dans l'état. Dans ce cas là, on peut à partir de la version taggée release_V1 faire une branche de développement qui va permettre en // de travailler sur la correction, de la tester et de la livrer. Plus tard, le patch reviendra sur la branche principale, CVS assurant la fusion de la version corrigée et de la version en développement.

2.6 Un outil graphique tkcvs

CVS est caractérisé par un jeu de commandes commençant toutes par cvs et avec une multitude d'options. Un remarquable outil graphique tkcvs, écrit en Tcl/Tk, permet avec des icones et des écrans de bien visualiser l'arbre de développement (avec ses branches) ainsi que les différences entre deux versions d'un même fichier.

3 Quelques commandes CVS

Nous décrivons ici quelques commandes CVS parmi les plus usitées

3.1 cvs update

Cette commande permet de mettre à jour son arborescence CVS par rapport à la repository. Si des modifications ont été commitées depuis le précédent cvs update, elles seront automatiquement reportées chez soi. Si un conflit arrive, il faut le résoudre chez soi à l'aide d'un éditeur.

3.2 cvs commit

Cette commande permet de soumettre une modification. Lors d'un cvs commit, CVS vérifie d'abord qu'il n'y a pas de conflit avec la version stockée dans la repository, puis il passe le fichier à un shell−script user qui pourra vérifier si des règles de programmation sont bien respectées. Puis, CVS appellera l'éditeur favori pour faire saisir un texte explicitant la modification. Enfin, après incrément du numéro interne de version, CVS le stocke dans la repository. A partir de ce moment, toute personne faisant un cvs update aura son fichier mis à jour.

3.3 cvs add

Cette commande est nécessaire pour ajouter un fichier dans la repository. En effet, par défaut un fichier n'est pas connu de CVS. Ceci permet d'avoir dans l'arborescence des fichiers hors CVS, par exemple des fichiers générés automatiquement (un .c issu de yacc).

3.4 cvs history

Cette commande permet de trouver l'historique des modifications pour tout un module (ensemble de fichiers). Par exemple, on peut demander l'ensemble des fichiers modifiés depuis 4 jours:

```
osirisv3 cvs history −a −xM '−D 4 days ago'

M 08/20 10:00 +0000 gfv22 1.5.2.1 listener.c
Coriolis/Src/Noyau/Unix/Main ==
~/Coriolis/Src/Noyau/Unix/Main

M 08/20 10:00 +0000 gfv22 1.2.2.2 testcorio.c
Coriolis/Src/Noyau/Unix/Main ==
~/Coriolis/Src/Noyau/Unix/Main

M 08/20 14:45 +0000 gfv22 1.3.2.1 SYSTFONC.c
Coriolis/Src/Noyau/Unix/Commun == ~/Coriolis/Src/Noyau
```

3.5 cvs log

Cette commande permet de voir l'ensemble des modifications apportées à un ou plusieurs fichiers. Ainsi cvs log parser.c donnera pour le fichier parser.c chaque modification apportée ainsi que le commentaire associé:

```
osirisv3 cvs log parser.c

RCS file:
/coriolis/gcode/Gcode/Coriolis/Tools/Doc/Essai/parser.c,v

Working file: parser.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: 1996/08/23 06:38:12; author: gfv3;
state: Exp; lines: +1 -1

adaptation aux nouvelles API's

----------------------------

revision 1.1 date: 1996/08/23 06:37:47; author: gfv3;
state: Exp;

Création
=============================================================
```

4 LINUX en serveur de sources

Dans notre projet, nous avons eu à mettre en place CVS afin d'assurer la cohérence des sources sur lesquels travaillaient les développeurs. En effet, notre projet reposant sur un concept client/serveur, les personnes développants la partie cliente en Visual Basic 3.0 ont rapidement eu des problèmes de gestion des sources (modifications régulièrement égarées). La solution a été de leur faire voir le disque d'un serveur Linux grâce à Samba et de mettre leurs arborescences de développement sur ce disque. Alors, avec quelques commandes CVS, ils pouvaient voir l'historique des fichiers, les modifications et soumettre leurs propres modifications (ceci nécessite la sauvegarde des sources VB au format texte). De même, pour les sources serveurs, nous disposions de deux petits serveurs AIX, géographiquement très éloignés. Les disques coûtant fort cher pour ce type de plate−forme, tous la repository a été stockée sur le serveur Linux, les deux serveurs AIX accédants le serveur Linux via le mode client/serveur de CVS, très efficace même sur une ligne TRANSFIX à 64 kbits/s.

5 Conclusion

Linux et CVS donnent pleine satisfaction pour la gestion de ce gros projet comportant 439 écrans et 4000 fichiers sources serveurs. Depuis la mise en place du duo, nous n'avons pas eu à déplorer de pertes de sources ou de régression. Avec des outils du type tkcvs, on obtient un remarquable outil de gestion de source et de version qui n'a rien à envier à des solutions dites professionnelles et qui sont proposées à des coûts prohibitifs.