I) Manipulation locale de fichier (co/ci)

L'utilitaire rcs est une version simplifiée du développement sous cvs. Il permet d'enregistrer et maintenir des versions différentes d'un fichier.

I.1) Création initiale du fichier

La commande ci (check-In) permet de fabriquer une version rcs d'un fichier.

- Faire un répertoire tprcs dans lequel vous fabriquez un fichier vide Test.java
- Générer le rcs du fichier (ci Test.java)

  ==> Que ce passe t'il ?

I.2) Récupérez la version courante du fichier

La commande co(check-out) permet de récupérer la version courante du fichier

- Lancer le check-out sur le fichier.

  ==> Que ce passe t-il ?

  ==> Quels sont les droits sur le fichier ? Pourquoi ?

I.3) Récupérez la version courante en mode verrou(lock)

La commande co -l permet de placer un verrou sur le fichier.

- Lancer la commande

  ==> Quels sont les droits sur le fichier ?

I.4) Modifiez le fichier verrouillé

- Ajoutez quelques lignes dans le fichier
- Validez vos modification par un check-In du fichier

  ==> Que se passe t-il, que contient le fichier de version ?
- Contrôler les différentes versions du fichier avec la commande rlog

I.5) Insertion de la version dans le fichier

Il est possible que le contenu du fichier suive les révisions...

- Mettre un commentaire dans le fichier du type :

```
//$Id:$
 est mon commentaire du code
```

- Valider / récupérez votre fichier

  ==> Que contient le fichier source ?

Interlude, changement de clavier, résumé de la situation, etc....

On passe à cvs

II) Manipulation de cvs

Cvs est une version globale de rcs, elle permet de gérer un répertoire contenant les version des fichiers déposés :

II.1) Positionnement des variables d'environnement

L'accès à la base cvs est fait localement, sur les machines du département. Il faut se connecter sur un poste de travail, puis il faut indiquer la localisation du repository cvs (répertoire qui contiendra les fichiers d'administration de la base cvs).

- Positionnez la variable CVSROOT à $HOME/tpcvs/repository
- Créez ce répertoire
- Initialisez le repository cvs :

```
cvs init
```

==> Quels sont les fichiers créés

! Attention, sauf être sur de ce qu'on fait, on ne modifie jamais un fichier directement dans le repository

II.2) Créer un projet

Le plus simple est de partir d'un tp que vous aimez bien (hanoi.c, OhTomate.c, ftp etc). Créer un répertoire tpcvs/tptest/ dans lequel vous copiez vos fichiers (cela représente la source initiale du projet).

- **Déplacez vous dans ce répertoire une fois les fichiers copiés.**
- Importez votre projet avec la commande initiale :

```
cvs import <tp1> <sfrenot> <debut>
```

==> Regardez le contenu de la base cvs, que s'est-il passé ?

!! **Attention à ce point le code local (~/tpcvs/test) n'est pas en mode cvs. Il faut initialement faire un checkout du projet.**

- Placez-vous dans le répertoire ~/tpcvs/
- Renommer le répertoire tptest en tptest.ori (mv tptest tptest.ori)
- Lancer la commande :

```
cvs co <tp1>
```

==> Le projet est alors récupéré dans la copie locale de travail de cvs. Les modifications apportées dans cette copie ne seront propagées que si on fait un **commit**. On peut toujours revenir en arrière par un **update**.

- Vous pouvez maintenant éditer un fichier pour y ajouter du code (//$Id:$) par exemple.
- Pour valider une modification faire `cvs commit`

II.3) Quelques commandes "classiques"

- Pour ajouter un nouveau fichier `cvs add toto` (il faut faire un commit pour valider l'ajout).
- Pour supprimer un fichier `cvs remove toto` (il faut que le fichier n'existe plus), `cvs remove -f toto` (pour supprimer au vol le fichier). Toutes ces commandes sont validées par un commit.
- Pour indiquer directement le message de log, utilisez l'option `-m 'Bonjour'`
- Pour mettre à jour une modification faite par un autre utilisateur `cvs update`
- Pour récupérer une version précédente `cvs update -r 1.1 <toto.java>` (Attention cette commande est "Sticky" elle colle !)
- Pour lister l'état **cvs status**
- Pour déposer un fichier binaire ajouter le fichier en mode binaire **cvs add -kb <a.out>**

==>Pourquoi cette commande est elle nécessaire ?

==>Que se passe t-il si deux utilisateurs modifient le même fichier (testez l'action).