Table des matières

1 Préambule

Subversion permet de gérer un projet (de programmation généralement) et de garder en mémoire l'historique de toutes les versions d'un ensemble de fichiers. Il permet de gérer un projet à plusieurs, de programmer afin de pouvoir revenir en arrière, comparer avec d'anciennes versions et cie.

Le principe est d'avoir un serveur subversion (un seul possible) qui va garder en mémoire l'historique de toutes les versions et un client subversion (plusieurs possibles) qui vont se connecter au serveur pour mettre à jour la version des fichiers ou en récupérer les dernières versions.

Remarque : Il est possible que le serveur soit lui aussi client, dans le cas où il n'y aurait qu'un seul développeur et qu'on ne souhaite pas passer par internet.

2 Commandes universelles subversion

Ici, je note les commandes qui sont valables à la fois pour svn installé sur un serveur internet, ou sur une machine locale pour un usage personnel

2.1 Mettre à jour sa copie locale : update

```
svn update
```

Pour celà, il faut que le dossier dans lequel on se trouve ait déjà été défini comme un dossier svn via un *checkout* (voir [§ 3.1, p4])

2.2 Ajouter des fichiers ou dossiers au projet : add

Pour ajouter des fichiers il faut faire :

```
svn add latex/ vim/
```

où **latex**/ et **vim**/ sont deux dossiers existant dans le dossier local de référence

Remarque : Au cas où ça serait pas clair. J'ai créé un dossier **/home/autiwa/Formulaires** grâce à [§ 3.1, p4]. Dans ce dossier, j'ai créé et rempli à la main les sous-dossiers **latex**/ et **vim**/. Maintenant, grâce à la commande ci-dessus, je définis ces sous-dossiers comme étant rattachés au projet. En faisant ainsi le contenu est rajouté récursivement.


Cette commande n'agit que sur le répertoire local (la *working copy*). Il faut ensuite faire un **commit** (voir [§ 3.2, p4]) pour valider les changements sur le serveur.

2.3 Supprimer des fichiers ou dossiers au projet : delete

Pour supprimer des fichiers il faut faire :

```
svn delete latex/ vim/
```

où **latex**/ et **vim**/ sont deux dossiers existant dans le dossier local de référence.


Cette commande n'agit que sur le répertoire local (la *working copy*). Il faut ensuite faire un `commit` (voir [§ 3.2, p4]) pour valider les changements sur le serveur.

Remarque : Supprimer manuellement dans le dossier courant des fichiers n'aura aucune incidence sur le dépot subversion (je crois). Il faut faire avec `svn delete` si on veut modifier quelque chose.

2.4 Lister les fichiers du dépot : list

```
svn list
```

va lister les fichiers présent dans le dépôt, mais par défaut, il fait les fichiers présent lors de la première révision.

Pour spécifier un numéro de révision (pour peu que les fichiers aient changés entre temps) :

```
svn list -r 7
```

pour afficher la liste des fichiers de la révision 7.

2.5 Revenir (tout ou en partie) à la révision courante : revert

Pour annuler les modifications locales et revenir à la dernière révision :

```
svn revert *.f90
```

va remettre en l'état tous les fichiers du svn qui sont présents dans le dossier courant (on peut aussi ne préciser qu'un seul fichier).

Si on souhaite forcer la mise à jour vers une révision plus ancienne (en cas de bug) il suffit de préciser le numéro de la révision

```
svn update --force -r39
```

pour revenir à la révision 39 ou

```
svn update --force -r39 fichier.f90
```

Pour ne changer qu'un seul fichier en particulier.

Remarque : Si la révision 39 est la dernière et qu'elle a été envoyée depuis le répertoire local, il est fort probable que la commande ci-dessus ne change rien. Il faut alors utiliser la commande `revert -force` permet de forcer les fichiers de la révision et d'écraser toute éventuelle modification locale. Dans le doute, tester sans cette option, quitte à la rajouter ensuite.

2.6 Annuler les modifications induite par une ou plusieurs révisions : merge

Pour annuler les modifications d'une ou plusieurs révisions, il faut utiliser la commande suivante :

```
svn merge -r131:129 .
```

 Pour supprimer les révisions 129 à 131, il faut entrer la commande :

```
svn merge -r131:128 .
```

Afin d'appliquer les changements, il faut faire un `commit` pour créer une nouvelle révision qui contiendra les modifications d'annulations.

Remarque : Pour supprimer les modifications de la révision 130, il faut faire :

```
svn merge -r130:129 .
```

On peut aussi supprimer les modifications de certaines révisions, mais à un seul fichier :

```
svn merge -r2349:2345 fichier.f90
```

Cette commande, qui doit être suivie d'un `commit`, va créer une nouvelle révision qui annule les commit entre 2345 et 2349 pour créer une révision 2350.

2.7 Historique des modifications : log

La commande `log` permet de voir un historique des commentaires de révision, soit de l'ensemble des fichiers, soit d'un fichier en particulier.

```
svn log
```

affiche l'historique de toutes les révisions (il est possible de spécifier un nombre maximal de lignes que l'on souhaite (les 10 dernières révisions par exemple via l'option `-l 10`).

```
svn log main.f90
```

va afficher l'historique des modifications du fichier **main.f90**, c'est à dire que n'apparaitront que les commentaires des révisions où le fichier **main.f90** a été modifié.

3 Subversion sur internet

Je vais prendre l'exemple de google code, qui est celui que j'ai choisi et que je suis en train d'apprendre.

3.1 Récupérer le contenu du projet

Une fois le projet créé (sur la page http://code.google.com/hosting/createProject), il faut faire :

```
svn checkout --username autiwa@gmail.com --password "passwd" \
https://autiwa-tutorials.googlecode.com/svn/trunk/ Formulaires
```

Cette commande permet de récupérer le contenu du projet et de le copier dans un dossier **Formulaires** qui sera créé dans le dossier courant.

Définition 1 (Checkout)
Opération d'extraction d'une version d'un projet du repository vers un répertoire de travail local.

3.2 Appliquer les changements qu'on vient de faire localement : commit

Pour mettre à jour les versions sur serveur à partir des modifications effectuées localement, il faut :

```
svn commit -m "initialisation" --username autiwa@gmail.com --password passwd
```

où **"initialisation"** est le commentaire qui décrit la mise à jour et les modifications effectuées, **passwd** étant le mot de passe associé à votre identifiant (ici **autiwa@gmail.com** pour moi).

Remarque : Je n'ai eu besoin d'entrer le mot de passe que la première fois.

4 Subversion localement

Le serveur ET le client seront alors sur la même machine.

4.1 Création du projet

J'ai créé un dossier **svn** dans mon **$HOME**, puis je fais, dans mon répertoire utilisateur :

```
svnadmin create svn/mercury
```

pour un projet que j'appelle **mercury**.

4.2 Récupérer le contenu du projet

Une fois le projet créé (sur la page http://code.google.com/hosting/createProject), il faut faire :

```
svn checkout file:///home/cossou/svn/mercury
```

où **file :///home/cossou/svn/mercury** est une URL qui désigne le dossier /**home**/**cossou**/**svn**/-**mercury**. La présente de **file :**// est à ma connaissance indispensable.

Cette commande permet de récupérer le contenu du projet et de le copier dans un dossier **Formulaires** qui sera créé dans le dossier courant.

Définition 2 (Checkout)
Opération d'extraction d'une version d'un projet du repository vers un répertoire de travail local.

4.3 Appliquer les changements qu'on vient de faire localement : commit

Pour mettre à jour les versions sur serveur à partir des modifications effectuées localement, il faut :

```
svn commit -m "initialisation"
```

où **"initialisation"** est le commentaire qui décrit la mise à jour et les modifications effectuées, **passwd** étant le mot de passe associé à votre identifiant (ici **autiwa@gmail.com** pour moi).

5 Importer un arbre subversion

À la création d'un dépot subversion (sur internet par exemple) on peut souhaiter importer un arbre subversion déjà existant (localement par exemple). Afin de l'importer, il suffit d'avoir le dépôt de destination entièrement vide (pas initialisé). Il peut donc être nécessaire de faire un *reset*. Puis il faut faire

```
svnsync init --username YOURUSERNAME \
https://YOURPROJECT.googlecode.com/svn file:///path/to/localrepos
  Copied properties for revision 0.
```

pour initialiser le dépôt. Vous aurez alors un retour de cette forme :

```
$ svnsync init --username YOURUSERNAME \
https://mercury-90.googlecode.com/svn file:///home/login/svn/mercury
  Copied properties for revision 0.
```

Puis il faut synchroniser toutes les révisions locales sur le nouveau dépôt :

```
svnsync sync --username YOURUSERNAME https://YOURPROJECT.googlecode.com/svn
  Committed revision 1.
  Copied properties for revision 1.
  Committed revision 2.
  Copied properties for revision 2.
  [...]
```

Remarque : J'ai eu droit à un

```
svnsync: Le serveur a envoyé une valeur inattendue (502 Bad Gateway)
 en réponse à la requête PROPFIND pour '/svn/!svn/bln/21'
```

J'ai re-exécuté la même commande de synchronisation quelques minutes après et j'ai pu synchroniser, je ne sais pas d'où le problème venait.

6 Gérer un conflit

Parfois il arrive qu'il y ait un conflit lors d'un **commit** en particulier quand les modifications locales ont été faites sur une version plus ancienne que la dernière version disponible sur le serveur svn. En gros, ça signifie qu'il y a des modifications dans les deux versions, la version locale et la version de la base de données Subversion.

Lors du commit, on aura alors quelque chose du genre :

```
Conflict discovered in 'recettes/chapitres/05_desserts.tex'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: s

  (e)  edit             - change merged file in an editor
  (df) diff-full        - show all changes made to merged file
  (r)  resolved         - accept merged version of file

  (dc) display-conflict - show all conflicts (ignoring merged version)
  (mc) mine-conflict    - accept my version for all conflicts (same)
  (tc) theirs-conflict  - accept their version for all conflicts (same)

  (mf) mine-full        - accept my version of entire file (even non-conflicts)
  (tf) theirs-full      - accept their version of entire file (same)

  (p)  postpone         - mark the conflict to be resolved later
  (l)  launch           - launch external tool to resolve conflict
  (s)  show all         - show this list
```

Je ne vais parler que de ce que je connais. Il existe deux familles d'actions principales, les commandes `*-conflict` et les commandes `*-full` :

- Les commandes `*-full` vont écraser les modifications par le fichier soir local (**mine-full**) soit sur le serveur (**theirs-full**) ;
- Les commandes `*-conflict` vont au contraire chercher à rajouter toutes les modifications en prenant pour base le fichier local (**mine-conflict**) ou le fichier distant (**theirs-conflict**)

On peut aussi laisser le coté la résolution du conflit avec **postpone**, en rentrant la commande 'p' dans le terminal dans le prompt pour la résolution du conflit. Le conflit sera alors marqué, on devra s'en occuper (ou pas) à la main, puis marquer le conflit comme résolu en faisant

```
svn resolved recettes/chapitres/05_desserts.tex
```

Index