Introduction à Subversion (2)

La première partie du TP d'initiation à SVN consistait à comprendre les principes généraux d'utilisation en travaillant sur la même machine. La seconde partie du TP consiste à vous initier aux aspects multi-utilisateurs de SVN que vous serez amenés à utiliser lors de la conception de votre projet. Dans cette partie, vous travaillerez sur des machines différentes et vous utiliserez le dépôt du premier TP sur SVN.

1 Références et aide en ligne

Le site de Subversion :
`http://subversion.tigris.org`
Les documents de Philippe Durif, dont le TP s'inspire :
`http://www2.lifl.fr/~durif/svn/index.php`
Le livre en ligne sur Subversion :
`http://svnbook.red-bean.com`
L'aide fournie par le client svn :

- La commande **svn help** pour avoir une aide générale à propos de svn;
- La commande **svn help commit** pour avoir de l'aide à propos de la sous-commande commit. Bien entendu, cela fonctionne avec toutes les sous-commandes de svn.

2 Préparation de cette partie du TP

Chaque membre du binôme va dans un premier temps créer une copie de travail du dépôt. Pour cela, vous allez utiliser une option particulière de la commande **checkout**, qui permet de spécifier le nom de l'utilisateur qui accède au dépôt :
`svn checkout --username loginDuMembre`
`https://svn.fil.univ-lille1.fr/svn/login_PL_TP_SVN`
Par exemple, si le binôme est constitué de deux membres dont les logins sont *"dupontm"* et *"jacksonj"*, alors la création de la copie de travail se fera :

- Pour *"dupontm"* avec la commande :
  `svn checkout --username dupontm`
  `https://svn.fil.univ-lille1.fr/svn/dupontm_PL_TP_SVN`
- Pour *"jacksonj"* avec la commande:
  `svn checkout --username jacksonj`
  `https://svn.fil.univ-lille1.fr/svn/dupontm_PL_TP_SVN`

3 Subversion en multi-utilisateurs

LES DEUX MEMBRES DU BINÔME DOIVENT TRAVAILLER SUR DES MACHINES DIFFÉRENTES POUR CETTE PARTIE DU TP.
A ce point du TP, les 2 versions locales sont identiques. Nous allons travailler sur le fichier `src/ant/example/HelloWorld.java`.

3.1 Résolution automatique de conflits

dupontm ajoute un attribut de classe 'x' de type entier qu'il initialisera à 2 dans le constructeur.

```
public class HelloWorld {
    public int x;

    public HelloWorld() {
      this.x = 2;
    }

    public static void main(String args[]) {
            System.out.println("Hello world !");
    }
}
```

```
dupontm@lxa13:~/PL_TP_SVN\$ svn commit -m "Ajout et initialisation de x"
Sending        src/ant/example/HelloWorld.java
Transmitting file data .
Committed revision 2.
```

De son coté *jacksonj* ajoute un méthode d'affichage du message "It works".

```
public class HelloWorld {
    public String printMessage() {
      System.out.println("It works");
    }

    public static void main(String args[]) {
            System.out.println("Hello world !");
    }
}
```

```
jacksonj@lxa14:~/PL_TP_SVN\$svn commit -m "Ajout d'une méthode
                                        d'affichage du message 'It works'"
```

Le `commit` de la modification de *jacksonj* ne se déroule pas normalement. Subversion répond le message suivant :

```
Sending        src/ant/example/HelloWorld.java
Transmitting file data .svn: Commit failed (details follow):
svn: File '/src/ant/example/HelloWorld.java' is out of date
```

En effet, ici les modifications ont été faites sur le même fichier. Lorsque *jacksonj* effectue son `commit`, Subversion lui signale qu'il n'est pas à jour et lui refuse l'opération. *jacksonj* doit alors effectuer un `update` :

```
jacksonj@lxa14:~/PL_TP_SVN\$svn update
G    src/ant/example/HelloWorld.java
Updated to revision 2.
```

Comme les modifications ont été faites dans des portions distinctes du fichier, Subversion arrive à fusionner les changements de *dupontm* avec ceux de *jacksonj*. Le conflit a été résolu automatiquement par Subversion. Le G signifie que Subversion a réussi à "fusionner" les modifications.
jacksonj peut alors effectuer son `commit`.

```
jacksonj@lxa14:~/PL_TP_SVN\$svn commit -m "Ajout d'une méthode
                                        d'affichage du message 'It works'"
Sending        src/ant/example/HelloWorld.java
Transmitting file data .
Committed revision 3.
```

3.2 Résolution manuelle de conflits

dupontm veut maintenant que, dans le constructeur, l'attribut 'x' soit initialisé à la valeur 3.

```
public class HelloWorld {
    public int x;

    public HelloWorld() {
      this.x = 3;
    }

    public String printMessage() {
      System.out.println("It works");
    }

    public static void main(String args[]) {
            System.out.println("Hello world !");
    }
}
```

```
dupontm@lxa13:~/PL_TP_SVN\$svn commit -m "x initialisé à 3"
Sending        src/ant/example/HelloWorld.java
Transmitting file data .
Committed revision 4.
```

De son côté, *jacksonj* veut effectuer la même modification sauf qu'il souhaite que l'attribut 'x' soit initialisé à la valeur 4.

```
jacksonj@lxa14:~/PL_TP_SVN\$svn commit -m "x initialisé à 4"
Sending        src/ant/example/HelloWorld.java
Transmitting file data .svn: Commit failed (details follow):
svn: File '/src/ant/example/HelloWorld.java' is out of date
```

Comme précédemment les modifications ont été faites sur le même fichier donc il y a un conflit et Subversion veut que la version locale soit mise à jour.

```
jacksonj@lxa14:~/PL_TP_SVN\$svn update
Conflict discovered in 'src/ant/example/HelloWorld.java'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h) help for more options:
```

Ici, les modifications ont été apportées sur les mêmes portions du fichier et donc Subversion ne peut pas résoudre le conflit. Le fichier a été modifié par Subversion et l'utilisateur doit donc l'éditer (choix (e) proposé) manuellement pour résoudre le conflit. N.B. Le choix de cette commande (e) nécessite que SVN soit lié à un éditeur (nano, vi, emacs...). Si aucun éditeur n'est associé par défaut, vous devrez ouvrir le fichier avec l'éditeur de votre choix pour le modifier.
Vous pouvez choisir de ne pas modifier le fichier en mode interactif : commande `postpone` (choix (p) proposé) et de modifier le fichier concerné ultérieurement. Dans ce cas, le processus de mise à jour est interrompu.
Ci-après le fichier modifié par Subversion mentionnant les lignes posant problème.

```
public class HelloWorld {
    public int x;

    public HelloWorld() {
<<<<<<< .mine
      this.x = 4;
=======
```

```
      this.x = 3;
>>>>>>> .r4
    }

    public String printMessage() {
      System.out.println("It works");
    }

    public static void main(String args[]) {
            System.out.println("Hello world !");
    }
}
```

Les conflits sont délimités par les lignes :

```
<<<<<<< .mine
      this.x = 4;
=======
      this.x = 3;
>>>>>>> .r4
```

`.mine` est la modification faite par *jacksonj* sur sa version locale et `.r4` est la modification en provenance du dépot correspondant à la version 4 du fichier commité par *dupontm*
La résolution manuelle du conflit se fait donc en 3 étapes :

1. *jacksonj* doit modifier le fichier HelloWorld.java ”à la main” et supprimer les lignes ajoutées par SVN (`<<<<<< .mine`, `======`, `>>>>>> .r4`) et celles posant problème pour avoir finalement :

   ```
   public HelloWorld() {
     this.x = 4;
   }
   ```

2. *jacksonj* signale localement que le conflit est résolu soit en choisissant (r) s’il a procédé en mode interactif, soit en tapant la commande : `svn resolved src/ant/example/HelloWorld.java`

3. Enfin, il peut rendre sa modification effective à l’aide de la commande : `svn commit -m "x initialisé à 4"`