SUBVERSION

svn.cnrm-game-meteo.fr

1 Introduction

Le serveur subversion du CNRM, **svn.cnrm-game-meteo.fr**, est accessible aussi bien depuis Météofrance que depuis l'extérieur du domaine aux personnes disposant d'un compte leur donnant les droits sur un ou plusieurs projets.
Il offre également la possibilté d'analyser l'état du projet placé sous subversion au travers de l'url suivante :

http://svn.cnrm-game-meteo.fr/viewsvn.

Ce qui suit n'a pas d'autre prétention que de permettre une prise en main rapide par la ligne de commande du client subversion sous linux. Plusieurs documentations sont accessibles sur le Webdav du CNRM à l'url :

(**http://webdav.cnrm.meteo.fr/public/cti/Documentation/Developpement**)

qui offrent une vision plus détaillée du serveur et de l'utilisation approfondie de subversion.

2 Définitions

2.1 Notions générales

2.1.1 dépôt (repository)

Le dépôt Subversion est l'emplacement central (**svn.cnrm-game-meteo.fr**) où sont stockées toutes les données relatives aux projets gérés.

Le dépôt contient l'historique des versions des fichiers stockés, les logs enregistrés lors des modifications, les dates et auteurs de ces modifications, etc. ..

Un dépôt apparaît de l'extérieur comme un système de fichiers composé de répertoires au sein desquels on peut naviguer, lire et écrire selon les permissions accordées.

2.1.2 projets

svn.cnrm-game-meteo.fr gèrent plusieurs projets. A chaque projet correspond un répertoire situé à la racine du dépôt et qui contient lui-même les fichiers et dossiers du projet proprement dit.

Exemple d'arborescence :

```
(dépôt)---+--/projet1-----+--/trunk
          |               |
          |               +--/branches
```

```
|               |
|               +--/tags
|
+--/projet2-----+--/trunk
|               |
|               +--/branches
|               |
|               +--/tags
|
+--/projet3-----+--/rep1
                |
                +--/rep2
```

2.1.3 copie de travail (working copy)

La copie de travail est un répertoire situé en local sur le poste de l'utilisateur et qui contient une copie d'une révision donnée des fichiers du dépôt. C'est cette copie qui sert de base de travail et qui est modifiée en local avant d'être importée (sauvegardée) vers le dépôt.

2.1.4 révisions

Chaque modification faite au dépôt constitue une révision. Le numéro de révision commence à 1 et augmente de 1 à chaque opération. Sa valeur n'a aucune importance, mais c'est un indicateur qui permet de revenir à une version donnée d'un ou plusieurs fichiers.

2.2 Opérations

2.2.1 checkout

Le checkout est l'opération qui consiste à récupérer pour la première fois les fichiers déjà existant au sein d'un projet du dépôt. Cette opération ne se fait en général qu'une fois par projet. Le résultat est une copie de travail.

2.2.2 import

L'import est l'opération inverse du checkout. Elle consiste à placer dans le dépôt des fichiers locaux déjà existants pour y créer un nouveau projet. Cette opération ne se fait en général qu'une fois par projet.

2.2.3 update

L'update consiste à synchroniser la copie de travail locale avec le dépôt en récupérant la dernière version des fichiers du dépôt. C'est à cette occasion que des conflits de version peuvent apparaître.

2.2.4 commit

Un commit est l'opération inverse d'un update. Elle consiste à mettre à jour le dépôt à partir de la copie de travail locale. Une nouvelle révision est alors créée. Un log (simple message texte contenant une description des modifications effectuées) doit être saisi à cette occasion.

A noter que pour qu'un commit soit possible, il faut que la copie de travail corresponde à la dernière version du dépôt (modifications locales exceptées). Si ce n'est pas le cas, il est nécessaire d'effectuer d'abord un update et de résoudre les conflits éventuels avant de réessayer le commit.

3 Utilisation

L'utilisation de Subversion suit en général un cycle assez répétitif.

Vous avez obtenu un compte **UTILISATEUR** (sour la forme **prenom_NOM**) et un mot de passe associé. Les droits sur le projet **PROJET** ont été définis sur le serveur par le responsable et vous permettent au minimum de lire l'ensemble des données.

3.1 Récupération de la dernière version du projet

Une seule commande suffit pour effectuer un **checkout** et récupérer la dernière version des fichiers : il s'agit de la commande **svn checkout (ou svn co en abrégé).**

```
$ svn co svn://UTILISATEUR@svn.cnrm-game-meteo.fr/PROJET .
ou
$ svn co http://UTILISATEUR@svn.cnrm-game-meteo.fr/projets/PROJET .
```

Le système vous demande votre mot de passe et en cas de succès vous obtenez les données. La directory de travail est créée dans la directory courante. Noter que le chemin est différent d'une méthode à l'autre. La méthode **svn** est préférable à **http**, car plus rapide, quand elle est permise.
Remarque :
En cas de succès, le mot de passe et le compte sont mémorisés dans un fichier de configuration si vous avez pris la peine de spécifier dans le fichier **$HOME/.subversion/config** l'entrée suivante :

```
store-passwords = yes
```

Avant de travailler sur les fichiers du projet, il faut s'assurer que l'on est bien synchronisé avec le dépôt, c'est à dire que la copie de travail correspond bien à la dernière révision en cours. Pour cela, il faut effectuer un update à l'aide de la commande **svn update** depuis la directory du projet:

```
$ svn update
U index.htm
Updated to revision 37.
```

La commande indique qu'un fichier du répertoire, vraisemblablement modifié par quelqu'un d'autre depuis notre dernier update, a été mis à jour dans notre copie de travail.

3.2 Mise à jour des modifications dans le dépôt

Une fois qu'on a modifié des fichiers, il faut basculer ces modifications au sein du dépôt pour qu'elles soient accessibles aux autres utilisateurs. Cette opération s'effectue à l'aide de la commande **svn commit** :

```
$ svn commit -m "Ajout d'une personne dans les credits"
Sending            credits.htm
Transmitting file data .
Committed revision 38.
```

Toute opération de commit s'effectue en indiquant un message décrivant les modifications effectuées (ici directement dans la ligne de commande). Il est possible d'effectuer cette opération sur un répertoire entier, ou sur seulement un ou plusieurs fichiers. Si des modifications ont eu lieu par un autre utilisateur du dépôt depuis le dernier update, un message d'erreur le signale. Il faut alors effectuer un nouvel update et résoudre d'éventuels conflits avant de relancer le commit.

3.3 Récupération d'une version antérieure d'un fichier

3.3.1 Récupération de la dernière version

Lorsqu'on travaille sur un fichier, il peut arriver que les modifications effectuées ne soient pas bonnes et qu'on souhaite retourner au fichier tel qu'il était lors du dernier update. La commande **svn revert** est faite pour cela :

```
$ svn revert credits.htm
Reverted 'credits.htm'
```

Cette commande annule les modifications effectuées depuis le dernier update. À noter que tout est effectué en local, et qu'un accès au dépôt n'est pas nécessaire.

3.3.2 Récupération d'une version antérieure du dépôt

On peut aussi souhaiter revenir à une version antérieure d'un fichier situé dans le dépôt. Il faut alors utiliser **svn update** en précisant le numéro de la révision et le ou les fichiers :

```
$ svn update -r 36 credits.htm
U credits.htm
Updated to revision 36.
```

3.4 Gestion des fichiers du dépôt

Subversion propose un ensemble de commandes pour ajouter, supprimer ou renommer des fichiers du dépôt.

3.4.1 Ajout d'un fichier

Il faut utiliser **svn add**. A noter que l'ajout n'est effectif qu'au prochain commit :

```
$ svn add liens.htm
A           liens.htm
$ svn commit -m "Ajout du fichier de liens"
Adding             liens.htm
Transmitting file data .
Committed revision 39.
```

3.4.2 Suppression d'un fichier

Il faut utiliser **svn delete** :

```
$ svn delete liens.htm
D            liens.htm
$ svn commit -m "Suppression d'un fichier"
Deleting           liens.htm
Committed revision 40.
```

Là aussi, la suppression n'est effective qu'au commit suivant.

3.4.3 Renommer un fichier

Il faut utiliser **svn move** :

```
$ svn move credits.htm merci.htm
A          merci.htm
D          credits.htm
$ svn commit -m "Renommage d'un fichier"
Deleting           credits.htm
Adding             merci.htm
Committed revision 41.
```

3.5 Résolution des conflits

Des conflits peuvent intervenir au moment d'un update, lorsque des modifications ont été faites à la fois dans la copie de travail et dans le dépôt. Par exemple, si vous éditez en local un fichier

pour lui rajouter une ligne, et qu'un autre utilisateur du dépôt a « commité » entre temps une modification différente sur le même fichier, votre commit va générer l'erreur suivante :

```
$ svn commit
Sending            merci.htm
svn: Commit failed (details follow):
svn: Your file or directory 'merci.htm' is probably out-of-date
svn:
The version resource does not correspond to the resource within the transaction.
Either the requested version resource is out of date (needs to be updated), or
the requested version resource is newer than the transaction root (restart the
commit).
```

Il vous faut alors effectuer un update, ce qui va mettre en concurrence les deux versions du ou des fichiers concernés. Deux cas de figure peuvent alors se présenter.

Dans le premier cas, le conflit peut être résolu automatiquement par Subversion car les modifications ne concernent pas les mêmes parties du fichier. Dans ce cas vous obtiendrez le message suivant :

```
$ svn update
G merci.htm
Updated to revision 42.
```

Il est quand même conseillé de vérifier manuellement le résultat de cette résolution « automatique ». Dans le deuxième cas, les modifications ne peuvent être fusionnées automatiquement car elles concernent les mêmes parties d'un fichier. Dans ce cas un conflit est signalé lors de l'update :

```
$ svn update
C merci.htm
Updated to revision 43.
```

Dans ce cas, deux nouveaux fichiers font leur apparition dans votre copie de travail. Dans l'exemple précédent, on se retrouve avec :

- merci.htm.mine : copie du fichier tel qu'il se trouvait dans votre copie de travail, en local, avant de faire l'update. C'est la version que vous souhaitiez « commiter » avant de détecter un conflit ;
- merci.htm.r42 : version du fichier pour la révision 42, c'est à dire lors de votre dernier update. C'est la version qui a servi de base pour les deux utilisateurs du dépôt qui ont travaillé en parallèle ;
- merci.htm.r43 : version du fichier pour la revision 43, c'est à dire la version actuellement dans le dépôt. Il s'agit de la version modifiée par un autre utilisateur, « commitée » avant votre update, et dont le contenu est à l'origine du conflit.
- merci.htm : il s'agit d'une version qui, en quelque sorte « résume » les trois autres en faisant apparaître les différences entre versions au sein d'un seul fichier.

Dès lors, le travail consiste à éditer le fichier merci.htm jusqu'à ce que le conflit soit résolu. Une fois ce travail terminé, on signale que le conflit est résolu à l'aide de la commande **svn resolved** :

```
$ svn resolved merci.htm
Resolved conflicted state of 'merci.htm'
```

On peut alors effectuer le commit final.

4 Trunk, branches, tags...

Les notions de **tronc**, de **branches** et d'**étiquettes** sont assez spécifiques aux logiciels de contrôle de versions. C'est ce qui explique que les arborescences des répertoires de projet contiennent souvent comme premier niveau de sous-répertoires les dossiers **trunk**, **branches** et **tags**.

En général, on définit par « tronc » la version centrale du programme, le développement principal « officiel ». Une « branche » est en général créée lorsqu'un développement « secondaire » est mis en route, que ce soit pour ajouter une nouvelle fonctionnalité ou parce que certains développeurs souhaitent essayer de prendre une autre direction pour certains aspects du développement. Une branche peut, au bout d'un certain temps, soit être à nouveau fusionnée dans le « tronc », soit disparaître, soit donner lieu à un nouveau programme.

La notion de tags correspond en partie à celle de release, c'est à dire de marquage d'une certaine révision du projet comme composant une version du projet. Une fois que le développement a atteint une certaine stabilité, on pourra par exemple créer un tag pour marquer la sortie de la version 1.0. Ceci permettra de revenir facilement à cette version, indépendamment du numéro de révision sous-jacent correspondant.