Maintenance de Logiciels

Travaux Dirigés N° 02

Gestionnaire de version - Subversion

Préliminaires :

Subversion (**SVN**) est un gestionnaire de version moderne qui permet de stocker d'une manière centralisée un ensemble de documents (code sources...) tout en conservant l'intégralité de leur historique des modifications.

SVN est particulièrement utilisé pour gérer les projets logiciels (GCC, Apache...). Les fichiers sources du projets sont stockés dans un **dépôt**. Ensuite, chacun des développeurs peut télécharger une copie locale des sources du dépôt sur sa propre machine, effectuer des modifications puis les soumettre à nouveau sur le dépôt pour qu'elles soient accessibles par les autres développeurs du projet.

SVN est compatible avec CVS (un gestionnaire de version plus ancien et extrêmement populaire). De plus, SVN corrige plusieurs lacunes présentes dans CVS. Il est inclus dans la plupart des distributions des systèmes de type UNIX. Sous windows, un outils appelé **TortoiseSVN** permet d'intégrer l'utilisation de SVN dans l'explorateur Windows rendant son utilisation encore plus conviviale. SVN peut être téléchargé manuellement depuis le site http://subversion.tigris.org/.

La documentation de référence de SVN est disponible sur le site http://svnbook.red-bean.com/.
Une documentation pour chaque sous commande de SVN est aussi fournie en utilisant la sous commande `help` :

- `svn help <cmd>`
- `svnadmin help <cmd>`
- `svnlook help <cmd>`

L'objectif de ces travaux dirigés est de vous initier aux fonctionnalités les plus communes de l'outil **Subversion**. La méthode d'accès au dépôt utilisée sera la méthode **svnserve** (`svn://`).

Tout le long du TD, vous allez utiliser une archive de code source qui vous sera fournie pendant la séance. Il s'agit de l'archive des codes sources du TD précédent sur l'outil **GNU Make**. Décompresser l'archive `td_svn.tar.gz` :

```
~% tar xzvf td_svn.tar.gz
```

Exercice 01 : Créer, gérer et utiliser un dépôt SVN

Tout d'abord, nous allons créer un nouveau dépôt subversion dans le dossier personnel. La création du dépôt subversion se fait en utilisant la sous-commande `create` de la commande d'administration `svnadmin` :

```
~% cd
```



```
~% mkdir td_svn
~% cd td_svn
~% svnadmin create mon_depot
```

Question 1 :

Créer un nouveau dépôt SVN et explorer le contenu des sous-répertoires du répertoire `mon_depot/`.

La seule création du dépôt ne suffit pas pour le rendre accessible aux développeurs (sauf depuis la machine locale en utilisant le protocole `file://`). Il faut également démarrer un serveur subversion pour qu'il permette l'accès au dépôt à distance. Dans ce TD nous allons utiliser le serveur **`svnserve`** pour accéder au dépôt créé.

Question 2 :

Consulter la documentation de la commande `svnserve` (en saisissant `man svnserve` dans un terminal). Déduire comment démarrer un nouveau serveur SVN en mode ***daemon*** pour servir votre dépôt fraichement créé et démarrer-le **dans un nouveau terminal**.

Notre dépôt est encore vide, nous allons le peupler en utilisant la sous commande `import` de la commande `svn`.

Question 3 :

Créer la hiérarchie suivante de répertoires dans un endroit à l'extérieur du dépôt (exemple `td_svn/`) :

```
arithm/
 |_ trunk/
      |___ arithm.c
      |___ fonctions_basiques.h
      |___ fonctions_avancees.h
      |___ fonctions_basiques.c
      |___ fonctions_avancees.c
      |___ Makefile
 |_ branches/
 |_ tags/
```

Effectuer un premier import de ces répertoires dans le dépôt :

```
%~ svn import arithm svn://localhost
```

Une fenêtre de l'éditeur `vim` vous demandera un message de log pour cette première révisions :

- Appuyer sur la touche `'i'` pour passer en mode *insertion* sous `vim`
- Saisir le message de log
- Repasser en mode *commande* sous `vim` en appuyant sur la touche `ECHAP`
- Valider le message en tapant `':x'` suivi de `ENTREE`

Que constatez-vous?

Le premier import n'a pas réussi par ce que le dépôt n'a pas encore été configuré pour permettre aux utilisateurs d'y écrire. Pour configurer le dépôt, nous allons effectuer les tâches suivantes :

- Interdire les accès anonymes au dépôt
- Activer les accès authentifiés
- Créer un nouveau compte d'utilisateur avec mot de passe

Question 4 :

Éditer les deux fichiers `conf/svnserve.conf` et `conf/passwd`, lire la documentation qu'il contiennent et les modifier pour réaliser les trois points évoqués plus haut.

Question 5 :

Ré-exécuter la commande `svn import` et vérifier sont bon déroulement. On pourra supprimer le dossier `td_svn/tmp` qui a servi pour accueillir la hiérarchie de répertoire initiale car on n'en a plus besoin pour le reste du TD.

Maintenant on va supposer que l'on est un nouveau développeur et qu'on veut télécharger une copie locale des fichiers sources qu'il y a sur le dépôt.

Question 6 :

Créer un nouveau répertoire à l'extérieur du dépôt. Il servira pour accueillir la copie locale :

```
%~ mkdir td_svn/copie_locale
%~ cd copie_locale
%~ svn co svn://localhost
```

Saisir le nom d'utilisateur et le le mot de passe du compte créé. L'issue de cette commande produira un répertoire nommé `localhost/` contenant la hiérarchie de répertoires créée.

On remarquera la présence de répertoires cachés nommés `.svn` dans tous les répertoires gérés par **svn** dans la copie locale (en utilisant la commande `ls -a`). Ce sont les fichiers contenus dans ces répertoires là qui vont permettre de savoir quels éléments de la copie locale ont été modifiés par rapport au dépôt.

Question 7 :

Créer un nouveau fichier appelé `VERSION` dans le dossier `trunk/` et y ajouter une ligne contenant

un numéro de version pour le programme **arithm** (exemple **1.0**).
Ajouter ce fichier au gestionnaire de version avec la commande `svn add`. Avant de soumettre les changements avec `svn commit`, tester l'état du dépôt avec les commandes `svn status` et `svn diff`.

Nous allons maintenant illustrer les autorisations que peut mettre en place l'administrateur d'un dépôt SVN ainsi que le mécanisme des des branches.

Question 8 :

Créer deux comptes d'utilisateurs supplémentaires : `branch_maker` et `release_maker`
Modifier les fichiers `svnserve.conf` et `authz` pour attribuer aux trois utilisateurs les permissions suivantes :

- Tout le monde peut accéder en lecture seule au répertoire racine du dépôt
- 1er utilisateur créé : accès en lecture/écriture à `trunk` et en lecture seule à `branches` et `tags`
- `branch_maker` : accès en lecture/écriture à `trunk` et `branches` et en lecture seule à `tags`
- `release_maker` : accès en lecture seule à `trunk` et en lecture/écriture à `branches` et `tags`

Tester le bon déroulement des permissions en essayant par exemple de modifier un des fichiers sources avec le compte `release_maker` (il va falloir créer un nouvelle copie locale et utiliser l'option `--username` de `svn co`).

Question 9 :

Avec le compte `branch_maker` :

- Créer une nouvelle branche **"1"** du programme `arithm` :
  - **`svn copy`** `trunk branches/arithm_1`
- Passer le numéro de version à **"2.0"** dans `trunk`
- Soumettre les changements

Avec le compte `release_maker` :

- Créer une nouvelle release **"1.0"** du programme `arithm` :
  - **`svn copy`** `branches/arithm_1 tags/arithm_1.0`
- Passer le numéro de version à **"1.1"** dans `arithm_1`
- Soumettre les changements

Avec le premier compte créé :

- Effectuer quelques modifications sur les fichiers sources (en ajoutant une fonction `ppcm` dans `fonctions_basique.[h|c]`).

Avec les comptes `branch_maker` et `release_maker` :

- Créer la branche **"2"**
- Créer la version **"2.0"**

Question 10 :

Vérifier avec la commande `svn log` les différents messages de log des différents changements.

Exercice 02 : Les Hooks

Nous allons maintenants explorer le mécanisme de **hooks** fournis par le gestionnaire Subversion. Un **hook** est un un exécutable (script ou autre) ayant un nom spécifique et qui, s'il existe, est exécuté à un moment spécifique du fonctionnement du serveur subversion. Les deux hooks que l'on va utiliser sont :

- **`pre-commit`** : exécuté avant l'application finale d'un commit
- **`post-commit`** : exécuté après le succès d'un commit

Visualiser le contenu du répertoire `hooks/` et localiser les deux fichiers `pre-commit.tmpl` et `post-commit.tmpl`.

Question 1 :

Renommer le fichier `pre-commit.tmpl` en `pre-commit` et lui ajouter le droit d'exécution :

```
%~ mv pre-commit.tmpl pre-commit
%~ chmod a+x pre-commit
```

Éditer le script `pre-commit` et garder uniquement la partie qui vérifie que l'utilisateur a donné un message de log non vide. Ensuite, ajouter un second test qui vérifie que le message de log ne comporte pas des lignes trop longues :

```
MAX_LINE_LENGTH=`svnlook log -t "${TXN}" "${REPOS}" | wc -L`
if test ${MAX_LINE_LENGTH} -gt 80; then
    echo "Le messag de log contient des lignes trop longues" >&2
    echo "Vous avez probablement utilisé svn commit -m" >&2
    echo "Ceci est une mauvaise habitude :-)" >&2;
    # C'est le code d'erreur ≠ 0 qui va causer l'échec du commit
    exit 1
fi
```

Tester le bon fonctionnement de ces deux tests.

Question 2 :

Renommer le fichier `post-commit.tmpl` en `post-commit` et lui ajouter le droit d'exécution :

```
%~ mv post-commit.tmpl post-commit
%~ chmod a+x post-commit
```

Remplacer, dans `post-commit`, le mécanisme qui envoie un e-mail par un mécanisme qui, à chaque commit :

- Crée un répertoire `/tmp/mon_depot` (avec `mkdir -p`)
- Ajoute un fichier portant le numéro de la nouvelle révision dans ce répertoire
- Écrit dans ce fichier les éléments suivants :
  - Les informations générales de la modification :
    ```
    svnlook info --revision ${REV} ${REPOS}
    ```
  - Les répertoires affectés : `svnlook dirs-changed --revision ${REV} ${REPOS}`
  - Les fichiers exacts : `svnlook changed --revision ${REV} ${REPOS}`
  - Les modifications effectuées : `svnlook diff --revision ${REV} ${REPOS}`