Outils du développeur

TD n°6

Gestion de version avec svn

SVN (ou subversion) est un mécanisme de gestion de versions de code centralisé. Il repose sur un serveur dans lequel les différentes versions d'un code sont enregistrées. Le ou les développeurs peuvent ensuite se connecter sur le serveur pour y déposer la version initiale du code, récupérer la version courante (ou une version précédente), déposer les dernières mises à jour, …
SVN date un peu et aujourd'hui on lui préfère souvent GIT. Il est néanmoins plus facile à appréhender que GIT et constitue une bonne introduction aux systèmes de versionnage.

Nous utiliserons ici le client svn en mode console sous linux (ligne de commande). A noter qu'il existe des clients avec interface graphique. On peut faire du svn dans eclipse, dans intelliJ, … Il y a aussi, sous Windows, tortoisesvn.

Première partie : débuter avec svn

Le point d'entrée, sur un serveur svn, s'appelle un **dépôt**.
Vous disposez d'un dépôt svn sur un serveur du pôle informatique.
https://pdicost.univ-orleans.fr/svn/nom_du_depot/ où nom_du_depot est o minuscule suivi du numero etudiant
exemple du dépot SVN pour l'étudiant 2122248 : https://pdicost.univ-orleans.fr/svn/o2122248/
le login est votre n° d'étudiant (contrairement à ce qui est écrit dans la fenêtre de connexion)
le mot de passe est le code NNE que vous pouvez voir dans l'ENT rubrique scolarite , dossier etudiant , etat civil,champ NNE .

Connectez vous avec un navigateur web sur votre dépôt. Qu'y a-t-il dedans ? Pourquoi ?

Pour démarrer le versionnage, deux possibilités sont offertes : soit on code d'abord et ensuite on se connecte à svn. Soit l'inverse. Nous allons utiliser la première option.

1. Dans un shell bash, créer un répertoire « tpsvn »
2. Se placer dans « tpsvn ».
3. On va créer un répertoire « projet1 » qui sera une copie locale du dépôt (taper tout sur une seule ligne) :
   *svn checkout https://pdicost.univ-orleans.fr/svn/**nom_du_depot/** projet1 --username=**votre_login***
   Attention : il y a un espace avant et après « projet1 » !!!
   Attention : remplacez nom_du_depot par votre n° étudiant !!!
4. descendre dans projet1. Pour l'instant il ne contient rien (ou presque). C'est normal, le dépôt est vide.
5. créer un fichier « fichier1.txt ». Y placer un court texte de quelques lignes. Par exemple :
   Longtemps je me suis couché de bonne heure.
   Je m'appelle Ismaël. Mettons.
   Water, water everywhere. And not even a drop to drink.
6. Taper la commande *svn add fichier1.txt*
7. Faites votre première sauvegarde : *svn commit -m 'initial commit'*

- « -m 'commit initial' » : indique que cette opération est accompagnée du message « commit initial »

Chaque opération de sauvegarde sur le serveur est obligatoirement accompagnée d'un message. Cela permet de se souvenir du contenu de chaque sauvegarde, c'est à dire des changements qu'elle apporte au code sauvegardé.

Rafraîchissez la page web. Qu'observez vous ?
Dans la console bash, tapez « ls -la ». Que notez vous ?

Deuxième partie : ajout et observation de versions

1. Modifier le fichier « fichier1.txt » en ajoutant une quatrième ligne. Par exemple « Ce siècle avait deux ans ».
2. sauvez cette nouvelle version : *svn commit -m 'deuxième version'*
   La commande commit permet de sauver une nouvelle version…
3. observez la page web (après rafraîchissement)
4. créez un deuxième fichier, « fichier2.txt » Donnez lui un contenu, par exemple :« Façon puzzle »
5. tentez un deuxième commit : *svn commit -m 'troisième version'*
   que se passe-t-il ?
6. Tapez la commande : *svn add fichier2.txt*
   refaites ensuite la dernière commande de commit.
   Que se passe-t-il ?

Vous savez maintenant ajouter des fichiers au système de versionnage et faire des commits.

Le temps passe, les amours (programmatiques) s'en vont au loin et on oublie les versions…
Comment, alors, se souvenir du passé ? Grâce aux commande log et info

1. tapez la commande « svn log », puis essayez les variantes « svn log -v » et « svn log -r 2 ». Observez, comparez.
2. tapez la commande « svn info ». Observez

Sans aller aussi loin dans le passé, on peut tout simplement faire le point sur les modifications apportées au code récemment.

1. Tapez « svn status »
2. Tapez « svn diff »
3. Modifiez une des lignes de « fichier2.txt ».
4. Tapez à nouveau « svn status ». Observez.
5. Tapez à nouveau « svn diff». Observez.
6. Créez un nouveau fichier « fichier3.txt » avec un contenu quelquconque.
7. Tapez à nouveau « svn status ». Observez les différents statuts

Par défaut, svn diff compare au dernier commit connu localement.
On peut l'utiliser pour comparer à une version plus ancienne, par exemple « svn diff -r 1 » donnera les différences par rapport au premier commit. Essayez et observez.

Troisième partie : travail collaboratif

On peut tout à fait utiliser svn pour travailler seul, mais souvent les projets sont réalisés en groupes. Cela ajoute potentiellement un peu de complexité.

Tout d'abord, vous allez simuler un deuxième utilisateur.

1. Ouvrez une deuxième console bash
2. placez vous dans « tpsvn »
3. il faut maintenant récupérer ce qui se trouve sur le serveur svn. On va réutiliser checkout, mais avec un nouveau nom de répertoire...
   *svn checkout https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/ copie_de_projet1 --username=**votre_login***
   notons que, dans l'absolu, le second paramètre (ici : copie_de_projet1) est optionnel. Il nous permet de dire explicitement comment s'appelle le répertoire où l'on va placer la copie.
4. observez...

Vous disposez maintenant de deux versions du projet.

1. Dans la console n°1 (répertoire « tpsvn/projet1 »), modifiez le fichier fichier2.txt (ajoutez une ligne, par exemple « Rome remplaçait Sparte »). Cela change-t-il quelque chose à la copie de fichier2.txt qui se trouve dans « tpsvn/copie_de_projet1 » ? Pourquoi ?
2. Maintenant, dans la console n°1, faites un commit. Cela change-t-il quelque chose à la copie de fichier2.txt qui se trouve dans « tpsvn/copie_de_projet1 » ? Pourquoi ?
3. On souhaite maintenant synchroniser « tpsvn/copie_de_projet1 » avec la dernière version comitée du projet. On va pour cela demander une mise à jour au serveur :
   svn update
   Observez...
4. Répéter l'opération en sens inverse : dans la copie (console n°2), modifiez fichier2.txt, faites un comit. Puis dans la console n°1, faites un update. Vérifiez que les modifications ont bien été reportées.

Nous allons un peu compliquer les choses. Jusque là les deux copies travaillent de manière séquentielle. On modifie dans l'une, puis on comite, puis on update de l'autre côté. Que se passe-t-il quand deux copies sont modifiées simultanément ?

Cas simple : on ne modifie pas les mêmes fichiers

1. dans la console n°1, modifiez fichier1.txt (ajoutez par exemple « Three nice mice ») et commitez.
2. dans la console n°2, modifiez fichier2.txt (ajoutez par exemple « O Combien de marins combien de capitaines ») et commitez. Que se passe-t-il ?
3. Terminez en faisant un update de chaque côté.

Cas plus compliqué : on modifie le même fichier.

1. dans la console n°1, modifiez fichier1.txt (ajoutez par exemple « Where is my mind ») et commitez.
2. dans la console n°2, modifiez fichier1.txt (ajoutez par exemple « On the sunny side of the street ») et commitez. Que se passe-t-il ?
3. Dans la console n°2, lancez svn update, observez les principales solutions proposées.
4. Choisissez « tc », et observez le résultat. Que serait-il arrivé si vous aviez choisi « mc » ?
5. Faites un commit. Que se passe-t-il ? Pourquoi ?
6. Recommencez plusieurs fois les étapes 1 et 2 pour créer un nouveau conflit ; à chaque fois essayez les options suivantes jusqu'à résoudre les conflits. Si vous n'en sortez pas demandez à l'enseignant.

1. Postpone (va modifier le fichier en conflit : cumule ce qui vient des deux versions en indiquant d'où viennent les différences. A vous d'ouvrir le fichier et de faire le ménage avant de lancer un resolve qui indiquera que vous avez résolu le problème ; on devra ensuite faire un commit).

- Svn resolve : Si on a choisi d'opérer une résolution « à la main » des conflits, cette commande permet d'indiquer, comment on l'a résolu :

  svn resolve --accept *solution* nomdufichier

  où *solution* vaut :

  - working si on a apporter les modifs à la main
  - theirs-full si on utilise la version du dépôt (similaire à tc, mais pour un fichier à la fois)
  - mine-full si on garde notre propre version (similaire à mc, mais pour un fichier à la fois)
  - base si on repart de la dernière version avant conflit

  cela se fait fichier par fichier...

2. show diff (indique les différences et redemande quelle solution on souhaite apporter)
3. edit file (comme postpone, mais ouvre directement les ou les fichiers en conflits ; on pourra ensuite directement faire un « mark resolved » sans passer par la commande « svn resolve »)
4. fusion (on laisse svn mixer les deux versions avec les apports de chacune, dans l'ordre et suivant les indications qu'on lui donnera – svn propose un dialogue).

Conclusion : si on travaille à plusieurs, commit et update sont très complémentaires et il faut penser à utiliser les deux…

A noter que pour « réinitialiser » la version locale d'un fichier, il y a toujours *svn revert*...

Et si on voulait vraiment travailler à plusieurs ?
Il n'y aurait presque rien à changer. Il faudrait juste que l'administrateur qui gère le serveur svn autorise plusieurs personnes à accéder au même dépôt…

Notons l'existence de *svn annotate* pour savoir qui a fait quoi… (affiche le code avec l'auteur et le numéro de version de chaque ligne...)

Quatrième partie : accrobranche

On entre ici dans le niveau au dessus. Lorsque l'on est bien aguerri à la gestion de version et que l'on commence vraiment à travailler sur du projet collaboratif, on va sans doute se lancer dans des modifs de fond, sur du long terme.
Il est alors souhaitable de bricoler dans son coin. Si on commence de modifier l'appli dans tous les sens d'une façon peu compréhensible par les autres programmeurs, cela va finir en Tour de Babel…

La solution consiste à se faire un petit coin à soi : sa propre copie de l'appli, versionnée, mais identifiée comme une version « parallèle ». Cela s'appelle une *branche*. Elle peut être partagée par plusieurs utilisateurs, mais son rôle est de se distinguer de la version « officielle » du code.

La branche n'est pas une notion interne à svn, c'est une surcouche utilisateur, une convention.

En général, avant de faire quoi que ce soit dans un dépôt, on y créer trois répertoires :

- trunk : c'est là où on va faire les versionnages du code « officiel »

- branches : c'est là où on va versionner les copies « personnelles »
- tags : c'est l'endroit où on stockera les « release », c'est à dire les versions importantes du code officiel.

Mais tout cela reste une convention…

Mettre au propre le dépôt
- placez vous à la racine de votre répertoire versionné (normalement, vous y êtes déjà…)
- créez un répertoire « trunk » : cela peut se faire de deux manières :
  - **mkdir trunk**, suivi de *svn add trunk* puis *svn commit*
  - sinon, directement *svn --parents mkdir https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/trunk*
  - le --parents est optionnel, il est juste utile si dans le chemin donné on crée plusieurs niveaux de répertoires d'un seul coup. Dans notre cas on est juste sous la racine du dépôt, donc il ne sert à rien…
- répétez pour branches et tags

Remarque : si on voulait gérer plusieurs projets dans le même dépôt, on aurait commencé par créer à la racine du dépôt un répertoire « mon_premier_projet », dans lequel on aurait mis trunk, branches et tags...

On peut maintenant déplacer nos fichiers dans le trunk
- svn move fichier1 trunk
- svn move fichier2 trunk
- svn commit -m 'déplacmeent dans le trunk'

Et maintenant, on pourra faire soit un checkout global du dépôt (pas forcément conseillé), soit faire des checkout partiels :
*svn checkout https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/trunk copiedutrunk –username=**votre_login***

Reste la question des branches…

Pour se créer une branche de travail :
*svn copy https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/trunk*
*https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/branches/mabrancheamoi -m 'creation d une branche perso'*

(le saut à la ligne c'est uniquement dans ce fichier, pas quand vous tapez la commande…)

et ensuite vous faites votre checkout :

*svn checkout https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/branches/mabrancheamoi macopieamoi –username=**votre_login***

Cela crée le répertoire macopieamoi, qui est versionné, et en lien avec le répertoire branches/mabrancheamoi de votre dépôt.

Vous pouvez maintenant y modifier fichier1 et fichier2 (faites le), commiter, et vérifier que le commit s'est bien fait dans cette branche, pas dans le trunk…

Enfin, les branches ont souvent une fin : quand on considère que nos modifs sont mûres pour être intégrées dans la version officielle, on refusionne les deux : on fait un merge.

Comment ?

Le merge sert à deux choses : rester compatible avec le trunk (c'est à dire faire un update mais entre le trunk et sa propre branche) et pousser ses propres modifs dans le trunk.

On va donc procéder en 6 étapes :

- en étant dans la copie locale de notre branche :
  - svn merge *https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/trunk*
    qui permet de faire l'update du trunk vers notre branche
  - svn commit -m 'version finale branche'
    qui permet de faire un commit propre de la branche (avec nos modifs+la dernière version du trunk)
- en se plaçant dans une copie locale du trunk :
  - svn update pour s'assure que l'on a bien le dernier trunk
  - svn merge --reintegrate *https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/branches/maversionamoi*
    qui intègre les modifs de notre branche dans le trunk
  - svn commit -m 'integration'
    qui sauve le trunk avec intégration de nos apports…

et enfin supprimer notre branche :
svn delete *https://pdicost.univ-orleans.fr/svn/**nom_du_depot**/branches/maversionamoi* -m 'réintégrée'

Est-ce vraiment grave d'avoir supprimé cette branche ?

Si vous êtes arrivés jusque là, félicitations vous êtes en possession du kit de survie du versionnage svn, et le passage a git ne sera qu'une formalité (ou du moins une marche moins haute).

Et si vous voulez en savoir plus sur svn :
http://svnbook.red-bean.com/fr/1.5/