



Bloc-note 'Subversion'

Le 28 octobre 2006 à 19:29.

Bloc-Notes sur Subversion

Organisation "typique" d'un dépôt

'organisation d'un dépôt est affaire de besoin et doit être bien pensée dès le début même si tout peut être changé, ou presque, par la suite.

Dépôt unique ou dépôts multiples

La première question à se poser est "un ou plusieurs dépôts". En effet, si nous avons deux projets à versionner, il n'est pas idiot de se dire que l'on va créer deux dépôts avec chacun leur URL. Mais il est aussi possible de créer deux dossiers dans le même dépôt et de gérer ces deux dossiers de manière indépendante. Ce qu'il faut prendre en compte dans son choix à ce stade, c'est :

Un compteur de révision par dépôt. Cela veut dire que si deux projets sont sur le même dépôt, chaque commit d'un côté fera augmenter le compteur de révision de l'autre côté. Pour moi la révision est comme un numéro de série, pas une version, donc je ne trouve pas cela gênant. Mais pour les anciens de CVS cela peut être perturbant de commiter il y a 6 mois une révision 10, et lors d'un nouveau commit, six mois plus tard, de se retrouver à la révision 4512 parce qu'entre temps, les autres projets ont vécu. Donc chacun ses goûts à ce stade.

Un dépôt = plus de fluidité. En effet, lorsqu'une sous-branche d'un projet devient bien dense et que l'on veut la transformer en projet indépendant, il suffit de faire un simple MOVE dans un nouveau dossier pour lui-donner son nouveau statut en conservant en prime tout l'historique.

Plusieurs dépôts = moins d'espace. Un argument pour le multi-dépôts est qu'il sépare les espaces disques. En effet, si le projet A meurt une fois pour toute, supprimer son dépôt libère de l'espace disque. Car il ne faut pas oublier que pour subversion, le mot supprimer n'existe pas.

En conclusion, au delà du numéro de révision, le critère de choix est "est-ce que mes projets vont se mélanger les uns les autres". Si je suis une SSII, c'est rarement le cas, je vais créer un dépôt par client/projet car il est rare que le sous-projet du client A se retrouve un jour projet du client B (et encore 😊). Si je suis une organisation type "Apache" ou "KDE", mes projets sont tous plus ou moins liés, j'ai donc tout intérêt à ne faire qu'un seul dépôt.

Organisation des dossiers par projet

Un dépôt peut contenir autant de sous-dossier que désiré jusqu'à arriver à celui d'un projet. Subversion permettant de gérer les droits de manière fine sur chacun des dossiers. Nous pouvons donc avoir une structure adaptée à nos besoins du type :

```
https://mondepot/  
  java  
    toolkits  
      network  
    ...  
  webapps  
    monApplication  
    ...
```

En revanche, arrivé au niveau du projet lui-même il convient de normaliser le premier niveau de dossiers comme suit :

```
monApplication  
  trunk  
    monApplication  
  branches  
    maBrancheExperimentale  
      monApplication  
  tags  
    Version 1.1  
      monApplication
```

Les sous-dossiers trunk, branches et tags vont permettre de s'y retrouver quant à l'étiquetage des versions. Le premier niveau mon application n'est donc en réalité jamais checkouté. Seuls les dossiers finaux monApplication le seront. Le trunk correspond à la version courant de développement (head en CVS), branches contient des sous-dossiers correspondant aux branches (expérimentales ou pas) et tags aux versions release. Dans les deux derniers cas le niveau en dessous est un dossier nommé pour s'y retrouver (ex. Version 1.1) et en dessous nous retrouvons le dossier de l'application elle-même.

Commandes de base

Import initial d'un dossier (et de son contenu) dans SubVersion

```
# On importe le projet dans le dépôt
svn import monProjet http://monSite.org/subversion/monProjet/trunk/monProjet

# Suppression du projet maintenant qu'il est dans le dépôt
rm [1] -rf monProjet

# pour le récupérer à nouveau, mais cette fois, géré par subversion
svn co [2] http://monSite.org/subversion/monProjet/trunk/monProjet
```

Ensuite tout marche comme un système de fichier classique

```
# Si l'on désire déplacer un fichier/dossier, grande nouveauté par rapport à CVS
svn mv [3] mauvaisEmplacement sousDossier/bonEmplacement

# Supprimer
svn rm [1] fichierADetruire

# Commiter ses modifications dans le dépôt
svn commit -m "Un commentaire est toujours le bienvenu !"
```

Etat de la copie locale

Pour s'assurer que tout est en ordre, ou pour avoir la liste des fichiers ajoutés et modifiés, la commande suivante est bien utile :

```
svn status
```

Liste des commentaires de révision

```
svn log
```

Modification des méta-données

Une des nouveautés de subversion est la possibilité de stocker des méta données au niveau des répertoires. Cela peut servir à renseigner la liste des dossiers/fichiers invisibles, à définir la liste des références externes, etc...

éditeur externe

Pour éditer les propriétés, nous allons avoir besoin d'un éditeur externe. Cela peut-être n'importe quelle application capable d'éditer du texte de la manière qui vous convient :

```
export VISUAL=vi
```

Les commandes de gestion des propriétés

```
# donne la liste des propriétés dans le dossier courant
svn proplist .

# Affiche le contenu d'une propriété du dossier courant
svn propget ma:propriete .

# Edition du contenu d'une propriété du dossier courant
svn propedit ma:propriete .

# Changer la valeur sans passer par l'éditeur (moins pratique)
svn propset ma:propriete "Sa valeur" .
```

Déclarer un dossier comme ignoré

La propriété responsable du fait d'ignorer un fichier, un dossier, un ensemble, etc, est `svn:ignore` . Il suffit donc d'éditer cette propriété pour ajouter une règle par ligne contenant le nom d'un fichier, d'un dossier, ou un "pattern" du type `*.bak` .

Charger des sous-projets externes

Une des possibilités très utile de SubVersion est de pouvoir "monter" dans l'arborescence d'un projet des branches (au sens système de fichier du terme) provenant d'autre dépôts. Cela se fait très simplement par la commande d'édition de propriétés. Par exemple, imaginons que dans un dossier `monProjet`, nous cherchions à grêffer le projet `moduleAmi`.

Pour ajouter une référence, il faut éditer la propriété `svn:externals` du dossier où l'on désire établir la greffe. Ensuite nous ajoutons la référence comme suit :

```
| moduleAmi https://siteAmi.org/subversion/moduleAmi/trunk/moduleAmi
```

Il peut y avoir autant de ligne de référence que désiré. Une fois la propriété sauvegardée, il suffit de faire une mise à jour du dossier `monProjet` (`svn co`) pour mettre à jour ou, si le dossier externe n'existe pas encore, checker le dossier greffé automatiquement.

De manière générale, une mise à jour (`svn up`) à la racine, entraîne une mise à jour des dossiers externes. En revanche, les commits (`svn co`), eux, ne franchissent pas les "barrières" des externals. Pour commiter une version d'un sous-dossier externe, il faut explicitement commiter celui-ci.

Astuces diverses

Revenir en arrière sur une action (avant commit)

```
| svn revert fichier/chemin
```

Commiter un dossier complet

```
| find [4] . -name "*" -exec rm [1] -rf {} \;  
svn add `svn status | grep [5] ? | cut [6] -d" " -f7`  
svn rm [1] `svn status | grep [5] ! | cut [6] -d" " -f7`  
svn commit -m "$1"
```

Obtenir l'url d'une copie de travail

```
| svn info | grep [5] URL
```

Changer l'url d'un espace de travail

Il arrive parfois que l'URL de base d'un dépôt doive être changé.

Par exemple imaginons que votre copie locale de `monProjet` ait été obtenue par :

```
| svn co [2] http://svn.masociete.com/depots/monProjet
```

Or, l'url de base a changé entre temps, et n'est aujourd'hui plus `http://svn.masociete.com/depots` mais `http://intranet.masociete.com/subversion`. Vous avez alors deux solutions, la plus "simple" consiste à détruire votre copie locale et à refaire un checkout. C'est jouable si vous n'avez aucune modification en cours sinon vous perdez tous vos ajouts...

La "vraie" solution consiste quant à elle à relocaliser votre copie locale vers le nouveau dépôt. Pour cela, allez dans le dossier racine de votre projet et tapez la commande suivante :

```
| svn switch --relocate http://svn.masociete.com/depots  
http://intranet.masociete.com/subversion
```

Notez bien que seule l'url de base est utilisée, par le nom du projet. Ceci fait, vous pouvez à nouveau commiter vos modifications.

Annuler le dernier commit

Grosse cata, vous avez commité des horreurs et pour votre malheur, la règle absolue de subversion est de ne rien effacer. Pas de panique, voici la marche à suivre.

Imaginons que vous ayez commité des erreurs dans la révision 666 et que vous vouliez revenir à la révision 665. L'astuce est de fusionner la révision 665 sur la 666 et de commiter l'ensemble en 667 qui sera donc un nouveau 666. Pour cela, placez-vous dans votre dossier à restaurer :

```
| cd mon_projet  
  
# au cas où. Personnellement je fait un même un rm -rf * dans ce dossier, puis un svn  
up  
# être certain d'avoir le dernière revision.  
svn up
```

```
# On merge avec les deux révisions
svn merge -r 666:667 .

# On commit la revision "patch"
svn commit -m "annulation des horreurs de la révision 666"
```

Changer les propriétés de révisions

mettre en place le hook pre-revprop-change

<http://artisan.karma-lab.net/node/1071>
(C) artisan numerique - CC BY-SA

Liens:

- [1] <http://pwet.fr/man/linux/commandes/rm>
- [2] <http://pwet.fr/man/linux/commandes/co>
- [3] <http://pwet.fr/man/linux/commandes/mv>
- [4] <http://pwet.fr/man/linux/commandes/find>
- [5] <http://pwet.fr/man/linux/commandes/grep>
- [6] <http://pwet.fr/man/linux/commandes/cut>