Table des matières

Définition

Qu'est-ce qu'un système de gestion de version ou VCS (Version Control System) ?

- Un outil pour maintenir l'ensemble des versions d'un logiciel.
- Conserve les révisions successives du projet dans un référentiel (repository)
  possibilités de revenir en arrière, de voir les changements
- Facilite la collaboration entre les intervenants.
  chacun travaille avec son environnement.
  plusieurs personnes travaillent sur les mêmes fichiers simultanément.
- Chacun accède à une copie des fichiers, les originaux restent sur le référentiel.

Attention, un VCS n'est pas :

- Un système de construction de version.
- Un système de déploiement d'applications.
- Garant de la qualité d'un projet.
- Une alternative à la communication entre les membres d'une équipe projet.

Objectifs

Un gestionnaire de versions doit donc permettre :

- de gérer plusieurs versions d'un même document
- de garantir l'intégrité des versions
- de fusionner les versions

Les besoins

Dans le cadre d'un développement logiciel, on aura les besoins suivants :

- gérer les fichiers du projet (sources, fichiers de test et de configuration, documentation, ...)
- gérer les différentes « versions » des fichiers
- gérer le travail en collaboration locale et/ou distante (développeurs, testeurs, ...)

Les principaux utilitaires

Le précurseur

SCCS/GNU CSSC (Source Code Control System/Compatibly Stupid Source Control)

Les outils sous Licence GPL

GNU RCS (Revision Control System) : http://www.gnu.org/software/rcs/rcs.html
CVS (Concurrent Versions System) http://www.cvshome.org
Subversion : http://subversion.tigris.org/
GNU Arch : http://www.gnu.org/software/gnu-arch/

Les outils commerciaux

Il existe de nombreux systèmes commerciaux de gestion de sources, utilisant notamment une base de données, parmi lesquels Borland StarTeam® et LiveTeam®, Microsoft® Visual SourceSafe®, Rational® ClearCase, Perforce ...

CVS/Subversion

Présentation

CVS/Subversion gère un espace de stockage centralisé appelé le repository (dépôt ou référentiel) dans lequel il va stocker les fichiers, ainsi que l'historique des modifications, les journaux de chaque modification, la date, l'auteur d'une modification, etc.

CVS/Subversion va permettre :

- la gestion des sources d'un projet
- le travail en groupe
- une traçabilité totale du projet

Principe

Chaque développeur travaillera sur sa propre copie des fichiers, et pourra à tout moment demander à CVS/Subversion de synchroniser cette copie avec les fichiers communs stockés dans le repository.

CVS/Subversion fonctionne en client/serveur ce qui permet de travailler :

- sur des plates-formes de développement différentes
- sur des lieux différents.

On va distinguer ici plusieurs utilisations possibles de CVS/Subversion :

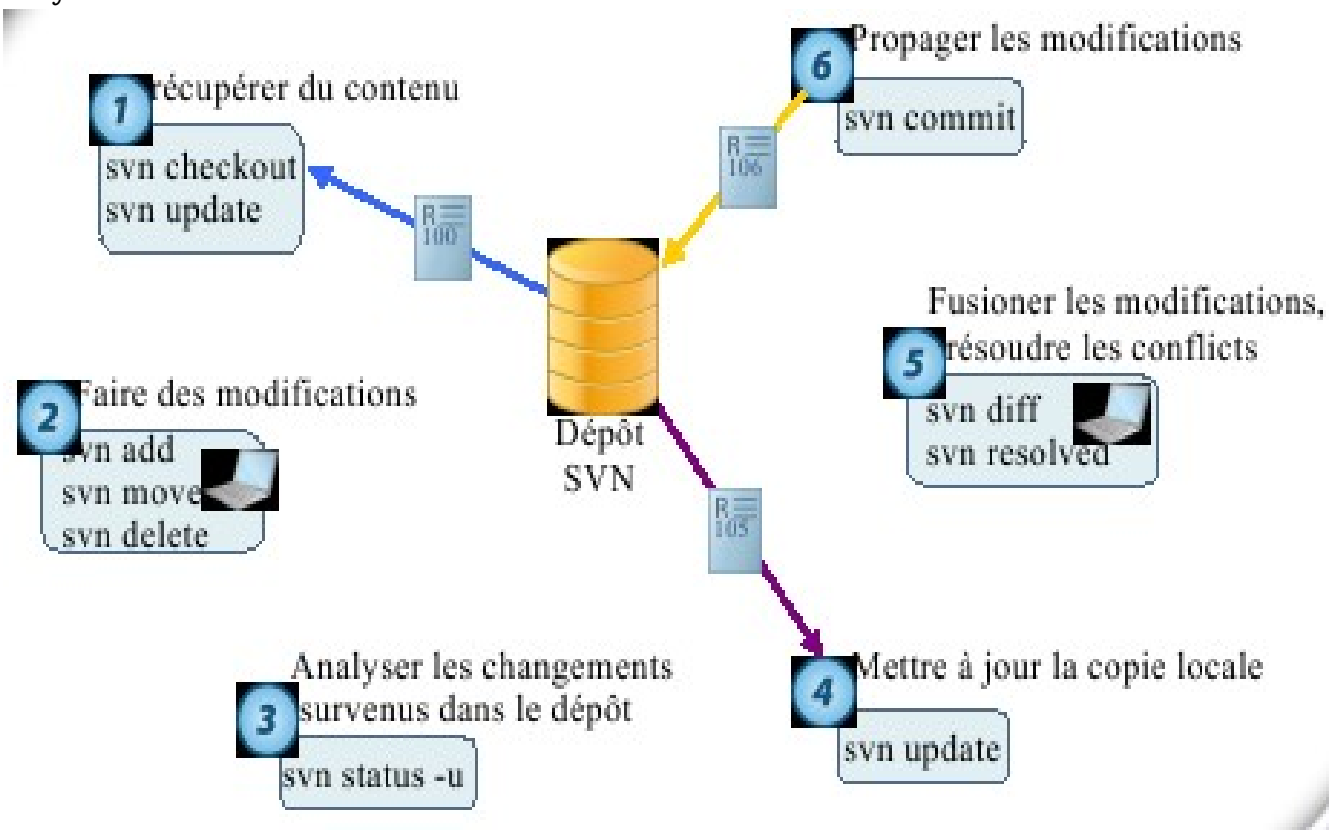
- en Local : vous êtes seul à travailler sur un projet de développement et vous voulez profiter des fonctionnalités de CVS/Subversion. Il vous faut alors une machine client/serveur sous Unix/Linux.
- en Intranet : vous êtes une équipe de développement sur un même lieu géographique (établissement scolaire, université, association ou entreprise). Il vous faut un réseau LAN, un serveur Unix/Linux et des machines clientes sous Linux/Windows/Mac/etc ...
- par Internet : vous êtes une équipe de développement dispersée sur plusieurs lieus géographiquement distincts. Il vous faut un réseau WAN ou internet et un serveur Unix/Linux chez un hébergeur spécialisé le plus souvent (http://sourceforge.net/, ...).

On distinguera deux types d'utilisateurs ou clients d'un projet de développement CVS/Subversion :

- les membres ou développeurs qui auront les droits de lecture/écriture sur le repository
- l'accès anonyme qui permet à quiconque d'avoir un accès en lecture

Fonctionnement

Un cycle de travail avec subversion :



Organisation standard du dépôt

il est recommandé d'organiser le projet de la façon suivante :

- un répertoire "trunk" (développement courant),
- un répertoire "tags" (stockage des versions identifiées),
- un répertoire "branches" (expérimentations et bugs)

| projet
| | trunk
| | tags
| | branches

Version vs Révision

Il faut distinguer ces deux termes :

- Version : indice attribué à une application (ou à un fichier) lors de sa release en tenant compte de la convention nom.majeur.mineur. Le numéro Majeur est souvent utilisé pour indiquer une évolution majeure et plus généralement lorsqu'elle devient incompatible avec la précédente. Le numéro Mineur est donc utilisé pour des évolutions mineures (et aussi correction progressive d'anomalies ou bugs) en n'affectant pas la compatibilité pour la version en cours. Une autre convention (pour les noyaux linux par exemple) précise des versions stables pour des numéros pairs et expérimentales pour des numéros impairs.
- Révision : indice attribué à un fichier lors d'une modification prise en compte par le gestionnaire de révisions. La numérotation utilisée représente des branches dans l'arbre de révision. De manière générale, un numéro identifiant un état du projet. Il commence à 1 et est incrémenté de 1 en 1 à chaque modification (subversion).

Exemple pour un projet BTS IRIS

L'administrateur a créé un dépôt par projet sur le serveur de la section :

```
# svnadmin create /var/lib/repositories/projets/2009/pesage/
# svnadmin create /var/lib/repositories/projets/2009/marine/
# svnadmin create /var/lib/repositories/projets/2009/theatre/
# svnadmin create /var/lib/repositories/projets/2009/meteo/
```

Chaque dépôt a été initialisé avec l'arborescence suivante (remplacé xxx par le nom du projet) :

```
| xxx
|     |  trunk
|     |     | README
|     |     | bin
|     |     | etc
|     |     | doc
|     |     | include
|     |     | lib
|     |     | src
|     |        | Makefile
|     | tags
|     | branches
```

Pour cela, l'administrateur a déjà réalisé un *import* pour chaque projet :

```
# svn import ./pesage/ svn://localhost/projets/2009/pesage/ -m "initial import" --username tv --password xxx
```

```
# svn import ./marine/ svn://localhost/projets/2009/marine/ -m "initial import" --username tv --password xxx
```

```
# svn import ./theatre/ svn://localhost/projets/2009/theatre/ -m "initial import" --username
tv --password xxx

# svn import ./meteo/ svn://localhost/projets/2009/meteo/ -m "initial import" --username tv
--password xxx

Adding       trunk
Adding       trunk/include
Adding       trunk/doc
Adding       trunk/lib
Adding       trunk/src
Adding       trunk/src/Makefile
Adding       trunk/bin
Adding       trunk/etc
Adding       branches
Adding       tags

Committed revision 1.
```

Première utilisation : il vous faudra récupérer une copie de travail locale (*checkout*). Vous pouvez soit utiliser le client svn en ligne de commande soit un client avec une IHM graphique sous Linux ou sous Windows (voir les clients pour Subversion en Annexe).

En ligne de commande, cela donne par exemple pour le projet pesage :

```
$ mkdir $HOME/projets/2009/pesage/
$ cd $HOME/projets/2009/pesage/
$ svn co svn://localhost/projets/2009/pesage/
```

Vous obtenez alors une copie de travail dans le répertoire courant. Vous pouvez maintenant travailler sur cette copie locale puis enchaîner les *update/commit* comme indiqué dans le cycle de travail vu précédemment.

Conseil : consultez les documents disponibles sur le serveur de la section ...

Mots clés de substitution

RCS/CVS/Subversion permettent d'insérer automatiquement des informations directement dans les fichiers sources en utilisant les mots clés de substitution.

Les mots-clés de substitution doivent souvent être placés dans des zones du fichier où ils ne perturberont pas les compilateurs ou les interpréteurs : entre commentaires, dans des chaînes de caractères, ...

Le mot clé le plus couramment utilisé est Id qui identifie l'état d'un document en une ligne).

Chacune de ces substitutions est disponible individuellement ($Author$, $Date$, $Revision$ ...) et peut être insérée à n'importe quel endroit du fichier, bien qu'on les place généralement en tête, pour des raisons évidentes de lisibilité.

Exemple pour Subversion :

Dans le fichier test.cpp :

```
/*
 * $Rev$
 * $URL$
 * $Date$
 * $Author$
 * $Id$
 */
```

Puis, on exécute les commandes suivantes :

```
$ svn propset svn:keywords "Rev URL Date Author Id" src/test.cpp

$ svn ci src/test.cpp -m "une modification"
```

Une fois le commit réalisé, on obtient :

```
/*
 * $Rev: 11 $
 * $URL: svn://localhost/projets/2009/pesage/src/test.cpp $
 * $Date: 2009-01-29 18:09:49 +0100 (jeu, 29 jan 2009) $
 * $Author: tv $
 * $Id: test.cpp 11 2009-01-29 17:09:49Z tv $
 */
```

Stratégies

- d'organisation du projet
  - définir la bonne organisation des répertoires
  - structurer en modules indépendants si nécessaire

- d'utilisation
  - ne posséder qu'une seule copie locale, et donc une seule copie par utilisateur
  - enregistrer régulièrement les modifications (commit ou check-in)
  - n'enregistrer que du code ayant été compilé et testé semble la meilleure solution à adopter

- de développement
  - utiliser les branches pour créer des release stable du projet
  - utiliser des noms (tags et labels) pour identifier des révisions compilables
  - importer les sources externes dans branches séparées

Limites

- d'espace disque
  - L'utilisation de RCS/CVS/Subversion provoque un accroissement du volume des sources utiles.
  - Dans tous les cas, moins que s'il fallait conserver l'intégralité des versions successives !

- de gestion de projet
  - CVS/Subversion n'est pas un outil de gestion de projet mais seulement un outil de gestion de révision.
  - Il ne peut donc pas se substituer aux autres outils ou à l'indispensable communication entre les développeurs.

- des fichiers binaires
  - CVS/Subversion offre toutes ses fonctionnalités pour les fichiers de type texte (ASCII).
  - Les autres type de fichiers peuvent être conservés dans le repository comme de fichiers binaires, mais ils ne pourront pas profiter des possibilités offertes par CVS/Subversion.

Subversion : mémo

Types de connexion

Les dépôts peuvent être accédés de plusieurs façons (local, http, ssh...). Dans chaque commande, le mode d'accès est indiqué au début du chemin d'accès.
Voici le code à mettre :
file:/// : accès direct à un dépôt sur un disque en local
http:// : accès via le protocole WebDAV à un serveur Apache avec le plugin Subversion
https:// : comme pour http :// mais avec le cryptage SSL
svn:// : accès via le protocole spécifique à un serveur Subversion de type svnserve
svn+ssh:// : comme pour svn :// mais à travers un tunnel SSH.

Pour lancer simplement un serveur svnserve, utilisez la commande suivante :
svnserve -d .

Un démon sera lancé et les utilisateurs pourront se connecter par l'option svn://.

Commandes courantes

obtenir l'aide :
    # svn help <commande>
création d'un dépôt :
    # svnadmin create <cheminlocal>
importation d'un projet :
    # svn import <cheminlocal> <depot> -m "commentaire"
récupération d'un projet :
    # svn checkout <depot> <cheminlocal>
obtenir des infos sur la copie en local :
    # svn info
visualiser les états des fichiers :
    # svn status <cheminlocal>
ajout d'un fichier :
    # svn add <nomfichier>
suppression d'un fichier :
    # svn delete <nomfichier>
déplacement d'un fichier :
    # svn move <source> <destination>
copie d'un fichier :
    # svn copy <source> <destination>
validation des modifications :
    # svn commit -m "commentaire"
résolution d'un conflit :
    # svn resolved <nomfichier>
mise à jour à la dernière version :
    # svn update

Gestion des modifications

suivre les modifications sur le projet :

```
# svn log
```

suivre les modifications sur le projet (affichage détaillé) :

```
# svn log -v
```

suivre les modifications sur un intervalle de révisions :

```
# svn log -r <numéro> :<numéro>
```

visualiser les différences entre les fichiers modifiés localement et ceux de la dernière révision de la base :

```
# svn diff
```

visualiser les différences entre les fichiers modifiés localement et ceux d'une révision donnée de la base :

```
# svn diff -r <numéro>
```

visualiser les différences entre un fichier particulier modifié localement et celui d'une révision donnée de la base :

```
# svn diff -r <numéro> <nomfichier>
```

visualiser les différences entre deux révisions d'un fichier :

```
# svn diff -r <numéro> :<numéro> <nomfichier>
```

mise à jour à une révision précise :

```
# svn update -r <numéro>
```

affichage du contenu d'un fichier pour un révision donnée :

```
# svn cat -r <numéro> <nomfichier>
```

annulation des modifications effectuées localement sur un fichier :

```
# svn revert <nomfichier>
```

lister les fichiers dfiun dépôt (affichage détaillé) :

```
# svn list -v
```

appliquer les modifications apportées sur un répertoire sur un autre répertoire :

```
# svn merge -r <numéro> :<numéro> <source> <destination>
```

Interfaces graphiques pour subversion

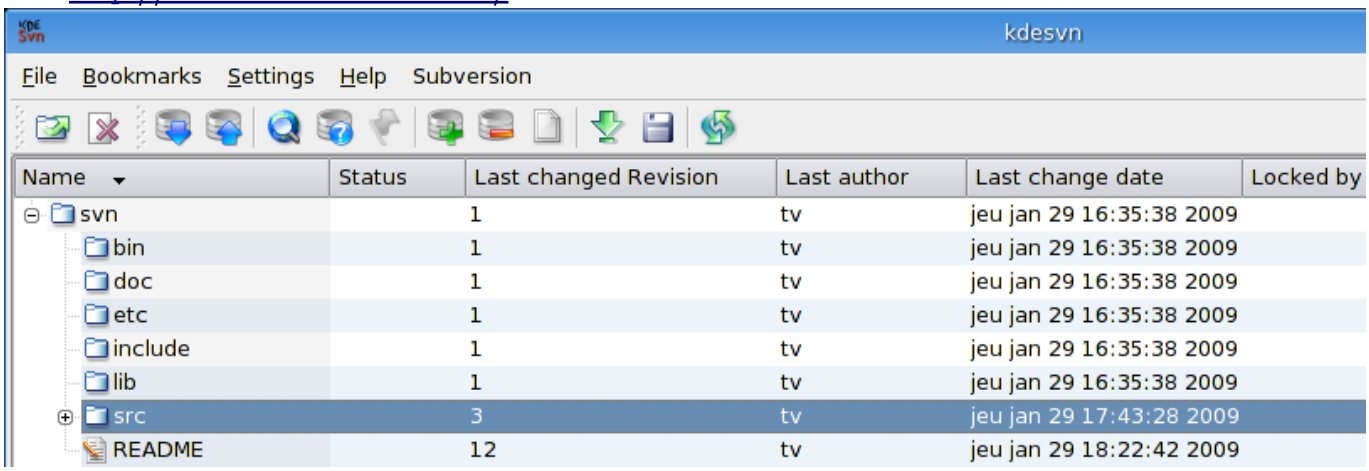
Il existe de nombreuses interfaces graphiques pour subversion disponibles pour un grand nombre de plate-forme: unix/linux, windows, mac etc...

Une liste complète de clients est maintenue à l'adresse :
http://subversion.tigris.org/project_links.html

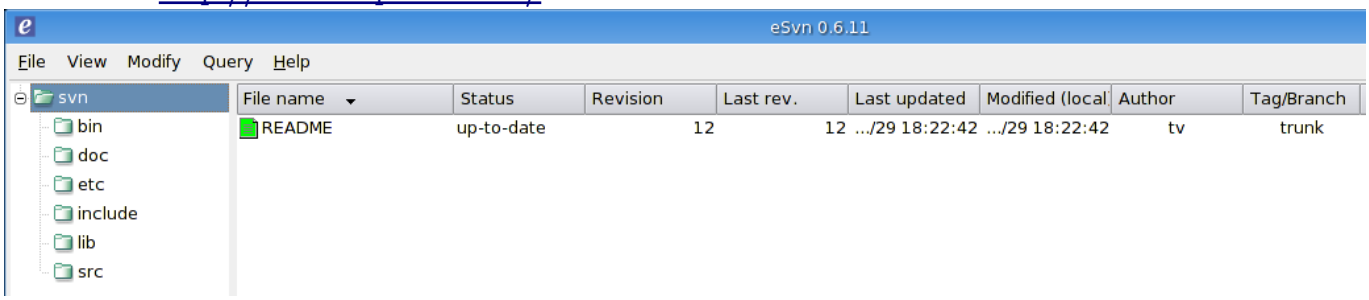
Une comparaison des clients est disponible sur
http://fr.wikipedia.org/wiki/Comparaison_des_clients_pour_Subversion

TortoiseSVN :
C'est sans doute la plus classique des interfaces graphiques pour subversion. A la fois sobre, mais complète et totalement intégrée à l'explorateur Windows elle est intuitive et très efficace.
voir http://tortoiseSVN.tigris.org/

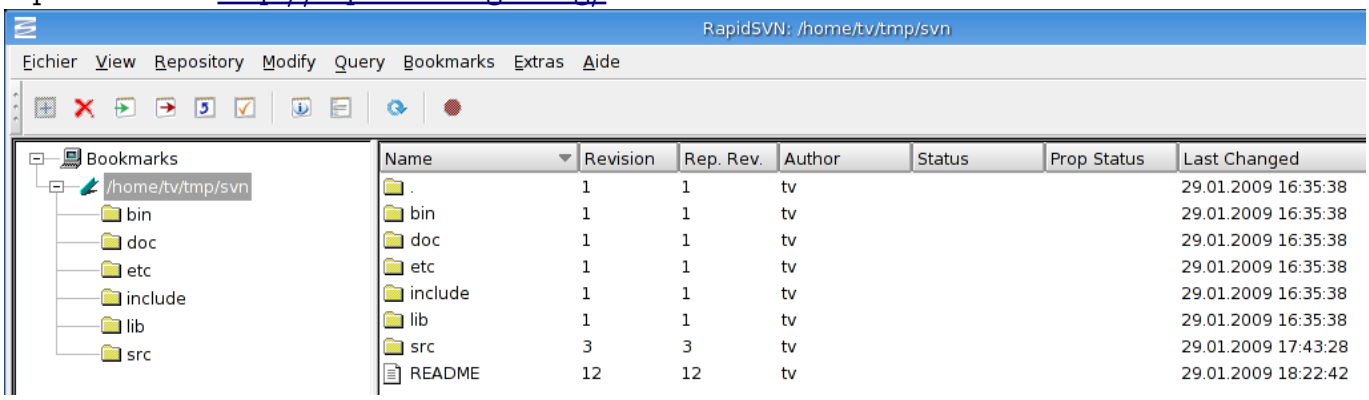
Kdesvn :
Kdesvn est un « clone » de TortoiseSVN pour KDE.
voir http://kdesvn.alwins-world.de/



Quelques autres interfaces :
eSVN voir http://esvn.umputun.com/



rapidSVN voir http://rapidSVN.tigris.org/



etc . . .