Subversion,

un logiciel de gestion de versions pour gérer un projet.

version 0.594 - le 08/10/2007 à 08:52:50 - PBo.

Table des matières

1-Concepts fondamentaux

Les systèmes de gestion de versions ou Version Control System[1] (VCS) en anglais, sont des outils facilitant le développement et la gestion d'un projet. Ils ne sont pas réservés à l'informatique et sont même issus de l'industrie aéronautique, mais nous ne verrons leur utilisation que dans le cadre informatique.

1-Qu'est-ce qu'un système de gestion de version ?

C'est un outil qui permet de maintenir l'ensemble des versions d'un logiciel. Conçu à la base pour faciliter le travail de développement seul ou en équipe, il est surtout concerné par le code source, mais peut s'appliquer à tout type de document informatique.

Par exemple, on trouve un mécanisme de gestion de version pour chaque article dans Wikipedia : il permet de retrouver chronologiquement toutes les modifications apportés à un article.

Tout projet peut tirer profit de l'utilisation d'un VCS, que ce soit pour la gestion d'un serveur Web, la gestion des fichiers de configuration d'un système, l'écriture d'un livre, d'un mémoire ou d'un article et bien sur pour la réalisation d'un programme et de sa documentation.

Depuis les tout-débuts jusqu'à aujourd'hui les logiciels de gestion de version VCS, (Version. Control System) ont beaucoup évolué.

Les premiers outils furent d'abord très simple, sans gestion des arborescences, ni gestion réseau, avec RCS (pour Revision Control System), dont la première version de Walter F. Tichy remonte à 1982. Puis vint dès 1986, mais véritablement à partir de 1989, l'aire de CVS qui est devenu au fil des années un des outils incontournables du monde de l'informatique, en particulier libre, à tel point que Brian Fitzpatrick[2] dira: *«Je ne peux pas imaginer programmer sans... Ce serait comme faire du parachutisme sans parachute».*

Aujourd'hui Subversion remplace avantageusement CVS et est utilisé par un grand nombre de projets open-sources (apache, kde ou gnome, par exemple) mais aussi commerciaux et est proposé en standard aussi bien par sourceforge que par google code.

Subversion est devenu un leader de la gestion de version centralisée. Pourtant d'autres modèles de VCS sont apparus et les dernières tendances sont aux systèmes décentralisés avec dépôts distribués et indépendants, sécurisés par des signatures numériques...

Quel que soit le mode de gestion de leur repository, les VCS fonctionnent en mode client serveur (voir 1: Architecture client serveur de subversion), ce qui permet à chacun de travailler sur des machines différentes avec son environnement. Ils permettent également aux différents acteurs d'un même projet, de travailler depuis des machines sous différentes plate-formes (Unix et non-Unix par exemple) pour un même projet. Que la personne travaille localement, sur la même machine que le serveur subversion ou à distance (en réseau), chacun n'accède jamais qu'à une copie des fichiers du projet, tandis que les originaux demeurent sur le « référentiel[3] » et ne sont modifiés qu'à travers les mécanismes sécurisés mis en place par le

1 On trouve aussi souvent SCM pour Source Code Management.
2 Membre de la célèbre Apache Software Foundation, il est aussi un développeur impliqué dans de nombreux projets opensources: http://www.red-bean.com/fitz/software.shtml.
3 Le référentiel est la traduction du terme anglo-saxon: repository, on trouve parfois également les termes dépôt ou entrepôt.

VCS.

Exemple d'architecture client/serveur:

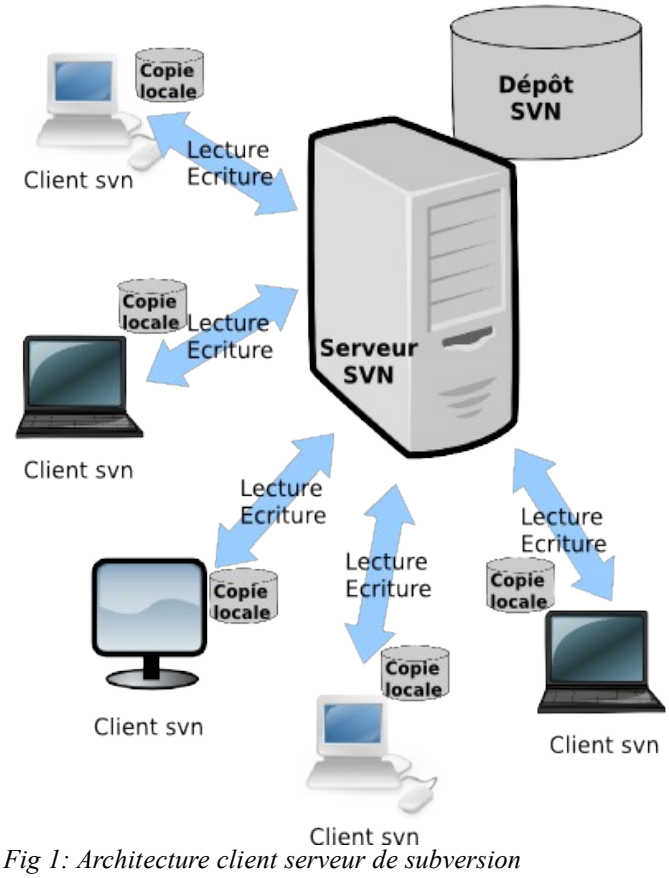


Fig 1: Architecture client serveur de subversion

1-Notion de dépôt et de copie de travail

Le dépôt, encore appelé repository en anglais est l'emplacement où sont gardées (entreposées) toutes les informations concernant les projets gérés par le VCS. Il contient l'historique des versions des fichiers stockés, les journaux de chaque modification, ainsi que toutes les informations comme la date, l'auteur d'une modification etc...

Ce dépôt est souvent sur un serveur, mais peut être sur une machine locale. Dans le cas de subversion il est unique, mais pour d'autres systèmes, il peut y en avoir plusieurs. L'accès au dépôt, se fait à travers différents protocoles en utilisant une URL locale ou distante voir le tableau « différentes URLs d'accès au dépôt » page 33.

Finalement le dépôt est un système de fichiers avec une dimension supplémentaire: le temps. Il est composé de répertoires et de fichiers parmi lesquels on peut naviguer dans le temps en plus de pouvoir lire et écrire en fonction des permissions.

Au sein d'un dépôt peuvent se trouver un ou plusieurs projets. Chaque projet correspond en général un répertoire situé à la racine du dépôt. Il contient lui-même les fichiers et dossiers du projet proprement dit.

Ce qu'il faut bien comprendre, c'est que l'utilisateur ne travaille jamais sur le dépôt, mais toujours sur une copie de travail.

La copie de travail est un répertoire local sur le poste de l'utilisateur qui contient une copie d'une version particulière des fichiers du repository. Les modifications seront effectuées en local et l'utilisateur pourra ensuite choisir de publier (« commiter ») ou pas les modifications sur le dépôt. C'est seulement quand les modifications seront publiées sur le dépôt qu'elles seront rendues « publiques » et connues des autres utilisateurs.

2-Notion de révisions

Centralisés ou décentralisés, les VCS utilisent pour la plupart la même unité de base: le patch[4] (ou changeset), c'est-à-dire un ensemble de changements pouvant concerner plusieurs fichiers. Mais dans les systèmes centralisés comme subversion, il y a un dépôt de référence et les utilisateurs lisent et écrivent uniquement dans ce dépôt alors que dans les systèmes décentralisés, il n'y a plus un unique dépôt de référence, mais plusieurs dépôts qui coexistent et on peut lire et écrire dans celui de son choix.

Le numéro de révision est un numéro identifiant de façon précise un état du projet sur le dépôt, une version du projet. Dans subversion, il commence à 1 et est incrémenté de 1 en 1 à chaque modification publiée. Sa valeur en elle-même n'a pas vraiment d'importance, mais c'est un repère qui permet d'identifier les différents états du projet ou une version particulière d'un ou plusieurs fichiers.

Chaque modification publiée sur le dépôt, après un « commit », constitue une révision. Pour la grande majorité des VCS, les commits, ou publications des modifications sont atomiques, ce qui signifie que l'ensemble des modifications qui sont publiées par les utilisateurs, sont vues par le système comme une unique commande. Que les modifications concernent un ou plusieurs fichiers, elles ne correspondent qu'à une révision particulière.

Les modifications sont appliquées sous forme de patchs, avec un résultat binaire: soit le patch est appliqué dans son ensemble et le numéro de révision est incrémenté, soit il est rejeté et le numéro de révision ne change pas car aucun fichier du repository ne sera modifié.

Remarque:

Dans les anciens VCS et en particulier dans CVS, les modifications n'étaient pas atomique, mais fonctionnaient par fichier, ce qui pouvait conduire à un dépôt inconsistant en cas d'echec d'une publication de modifications...

2-A quoi ça sert ?

Quand on a jamais travaillé avec aucun système de gestion de versions, on peut être un peu dérouté par le fait qu'ils sont utilisés pour deux raisons principales en apparence sans rapport:

- garder un historique du projet,
- faciliter la collaboration entre les différents intervenants du projet.

Il apparaît pourtant que ces tâches sont intimement liées.

L'historique devient nécessaire, quand les gens ont besoin de comparer l'état actuel du programme avec son état à un moment donné du développement. La récupération d'une version passée (un état ponctuel) d'un programme géré par un VCS ne pose aucun problème. Il suffit juste de demander au VCS: « donne moi le programme comme il était il y a 3 semaines »

4 Un patch est une une section de code que l'on ajoute à un logiciel, pour y apporter des modifications mineures. Il se présente comme une séquence de modifications à apporter au code source du logiciel concerné (d'après wikipédia).

ou encore « donne moi le programme comme il était lors de la dernière version publiée ».

Outre le fait que les originaux sur le référentiel ne sont jamais directement affectés par les utilisateurs, parce qu'ils éditent toujours des copies dont la mise à jour est assurée par le VCS, l'idée fondamentale derrière les VCS, est de ne conserver que l'historique des modifications des fichiers d'un projet. Plutôt que de stocker de multiples copies successives de ces fichiers au fil des modifications, on n'enregistre que la différence avec l'état précédent. D'où un gain de place important, pour parvenir au résultat souhaité: être en mesure de retrouver une version particulière du projet, ce qui permet par exemple, de localiser la première manifestation d'un bogue.

Pour la collaboration, plutôt que d'imposer aux développeurs de se concerter pour savoir qui peut travailler sur quel fichier et à quel moment, la majorité des VCS autorise tous les intervenants à travailler simultanément sur tous les fichiers (bien qu'il soit possible de mettre des verrous sur des fichiers), et gère ensuite le problème de l'intégration de tous les changements « auto-magiquement » chaque fois que c'est possible ou dans le cas contraire, prévient de l'apparition d'un conflit et demande de le résoudre manuellement.

En résumé, un VCS facilite la détection et la correction des bogues, permet d'économiser de la place et du temps, d'empêcher les erreurs de modifications/suppressions de code involontaires. Il permet à tout moment de revenir à un stade donné du développement.

3-A quoi ça ne sert pas ?

Malgré ses nombreuses qualités, un VCS n'est pas la « panacée universelle ». Voici en particulier une liste sommaire des choses pour lesquels il n'est pas d'un grand secours:

- Ce n'est pas un système de construction de version (comme le sont make[5] ou ant[6]), il n'aide pas à la construction d'un programme, mais il peut parfois être « adossé » à des outils de construction automatique comme buildbot[7].
- Ce n'est pas un système de distribution ou de déploiement d'application.
- Ce n'est pas un garant de la qualité: il n'y a aucun système de test ou de contrôle des éléments gérés par le VCS, c'est au développeur de s'assurer de la qualité de son travail.
- Ce n'est pas un outil de gestion de projet (au sens management/planification).
- Il ne doit pas se substituer à une bonne communication entre les membres d'une équipe projet, en particulier il ne peut en rien aider à la résolution de conflits relatifs à des logiques de programmation ou des différences de conceptions.

Il ne fera rien de magique, mais pourra apporter beaucoup de souplesse et de sécurité à une équipe de développeurs qui l'utilise correctement.

4-Relation entre CVS et subversion

Bien que CVS souffre de quelques défauts célèbres, il fut peut être le plus utilisé des VCS et même s'il est tombé en désuétude il reste encore employé aujourd'hui pour la gestion de

5 Pour plus d'information sur make, voir http://www.gnu.org/software/make/
6 Pour plus d'informations sur ant, voir http://ant.apache.org/
7 Voir http://buildbot.net

certains projets.

Subversion est l'héritier de CVS. Il a été créé avec l'idée que les principes mis en place avec CVS étaient les bons. Qu'il fallait combler les manques de CVS et corriger les bogues de conception.

De ce fait un utilisateur de CVS passe à subversion très facilement, d'autant plus facilement que les commandes sont bien souvent identiques.

Pourtant Subversion correspond à une grande avancée par rapport à CVS, en effet:

- Les commits, ou publications des modifications sont atomiques. L'unité de transaction est le patch et le serveur Subversion utilise de façon sous-jacente une base de données capable de gérer les transactions atomiques (FSF ou Berkeley DB[8]).
- Subversion permet de renommer et de déplacer des fichiers ou des répertoires facilement et sans en perdre l'historique.
- Les méta-données sont versionnées : on peut attacher des propriétés, comme les permissions, à un fichier, par exemple.

Du point de vue du simple utilisateur, les principaux changements lors du passage à Subversion, sont :

- Les numéros de révision qui sont désormais globaux à l'ensemble du dépôt et non plus par fichier : chaque patch a un numéro de révision unique, quels que soient les fichiers touchés. Il devient simple de se souvenir d'une version particulière d'un projet, en ne retenant qu'un seul numéro ;
- La commande « **`svn rename`** » (ou « **`svn move`** ») permet de renommer (ou déplacer) un fichier ou un répertoire facilement et sans pertes d'information;
- Les répertoires et méta-données qui sont versionnés.

Remarque:

Au niveau technique, Subversion[9] ne fait aucune distinction entre un label, une branche et un répertoire: ce sont tous des copies. Il devient très facile de comparer entre-eux labels, branches ou autres croisements. La distinction n'est qu'une convention de nommage pour les utilisateurs.

Pour conclure on peut dire que subversion est un CVS++, qui devrait ravir ceux qui utilisait CVS et convaincre les autres.

8 Les bases Berkeley DB sont des bases de données embarquées très utilisées et très réputées dans le monde de l'informatique. Son éditeur sleepycat a été racheté par oracle voir www.sleepycat.com.

9 Dans sa version actuelle 1.4, car la version 1.5 en préparation sera probablement différente sur ce point.

2-Utilisation de subversion

L'utilisation d'interfaces graphiques, est en général intuitive, et ne fournit qu'une sur-couche aux commandes standards, il est donc important de comprendre la mécanique de base et de savoir utiliser subversion en ligne de commande.

Souvent le référentiel est déjà créé, il suffit donc de savoir y accéder. Dans le cas contraire, on pourra le créer facilement à l'aide de la commande svnadmin (voir en annexe «Création d'un référentiel» page 33).

1-Utilisation basique de subversion

L'utilisation courante de subversion, est simple et intuitive, elles se résume à quelques commandes:

- Obtenir ou mettre à jour sa copie de travail: « svn checkout », « svn update ».
- Faire des changements: édition des fichiers, « svn add », « svn delete », « svn copy », « svn move ».
- Etudier les modifications: « svn status », « svn log », « svn diff », « svn revert ».
- Intégrer le travail des autres dans la copie locale: « svn update », « svn resolved ».
- Propager ses modifications: « svn commit ».

1-Comment obtenir de l'aide

La première des commandes à connaître, est celle qui permet d'apprendre les détails de toutes les autres:

```
svn help
```

En effet cette commande suivi d'une des sous-commandes svn affiche l'aide ainsi que le détail des informations concernant cette commande.

Par exemple la commande:

```
svn help help
help (?, h): Décrit l'usage de ce programme ou de ses sous-
commandes.
usage : help [SOUS_COMMANDE...]


Options valides:
  --config-dir ARG         : fichiers de configuration dans ce
répertoire
```

affiche l'aide de la sous-commande help.

Il en est de même pour les différentes sous-commandes.

La liste des sous-commandes disponibles est affichée à la fin du svn help...

2-Importation (ajout) du projet dans le dépôt

La première étape dans la gestion d'un projet avec subversion consiste à « passer » le projet sous le contrôle de subversion. Cette opération que l'on ne réalise qu'une fois au début du projet, correspond à l'import des fichiers du projet dans le dépôt subversion.

On utilise pour ça, la commande « **svn *import*** »:

```
svn import [CHEMIN] URL –m "Un commentaire obligatoire"
```

qui charge récursivement une copie de CHEMIN vers URL, où le paramètre CHEMIN désigne le chemin vers le répertoire contenant le projet à importer, URL l'url du dépôt sur le serveur svn et « -m message » contient un texte descriptif utilisé pour les journaux...

De manière plus précise, les paramètres sont:

- CHEMIN: le répertoire du projet. Si CHEMIN est omis, le répertoire courant, « . » est utilisé. Si CHEMIN est un répertoire, son contenu est ajouté sur le serveur directement après l'URL dans le dépôt.
- URL: l'url vers le repository ou le répertoire dans le repository. Le dépôt peut être local ou distant comme expliqué dans l'annexe « Comment accéder à un dépôt subversion ? » page 33. L'url définie le protocole utilisé, le serveur et le chemin sur le serveur pour accéder au dépôt.
- l'option « -m » permet d'ajouter un message, ce message est obligatoire mais le contenu est évidement laissé à l'appréciation du développeur. Si on n'utilise pas cette option, « -m » suivi du descriptif de l'opération exécutée (entre double quotes), subversion appelle l'éditeur par défaut du système et demande à l'utilisateur de compléter un fichier contenant le message.

Remarques:

Attention, une fois importé dans subversion, il ne faut surtout pas continuer à travailler dans le répertoire contenant le projet. Ce répertoire peut même être supprimé, puisque sa copie est dans le dépôt. Si on travaille sur les fichiers du projet dans ce répertoire, le travail effectué ne pourra jamais être pris en compte par subversion.

Attention à préparer le projet avant de faire un « import » ! Tous les fichiers ne doivent pas obligatoirement être gérer avec subversion...

Pour travailler sur le projet, il faut obtenir une copie de travail à partir du serveur (voir « Travailler sur un projet géré avec subversion » page 13)

3-Organisation classique d'un dépôt svn

Un référentiel subversion peut contenir plusieurs projets. Chaque projet peut contenir un ou plusieurs répertoires. Un répertoire peut lui-même contenir des sous-répertoires etc... ce qui forme l'arborescence du repository.

- Par exemple:

```
projets2A
|------ projet alpha
|       |------ module m1
|       |       \------ sous-module m1-a
|       \------ module m2
\------ projet beta
```

Avec subversion, il est très facile de gérer et d'organiser « proprement » le dépôt. Il est possible d'ajouter, de supprimer ou de déplacer des répertoires dans le repository, à l'aide des commandes « ***mkdir url/répertoire*** », « ***delete (del, remove, rm) url/répertoire*** » et « ***move (mv, rename, ren) url/répertoire*** » comme on le ferait sur un système de fichiers distant ou par ftp.

Par exemple la commande:

```
svn mkdir file://localhost/svn/projets1A
```

ajoutera le répertoire « projets1A » au dépôt situé dans /svn de la machine locale.

Bien que ça ne soit en rien une obligation, il est recommandé d'organiser le projet de la façon suivante :

- un répertoire « trunk » pour le développement courant du projet,
- un répertoire « tags » pour y stocker des versions identifiées (voir le chapitre branches et tags),
- un répertoire « branches » pour le développement d'expérimentations ou la correction de bugs concernant une version antérieure du projet (voir le chapitre branches et tags).

On retrouve donc en principe dans chaque répertoire d'un projet sur le dépôt l'architecture suivante:

```
projets2A
       |------ projet alpha
       |       |------ trunk
       |       |       |------ README
       |       |       |------ source.c
       |       |------ tags
       |       |------ branches
       \------ projet beta
       |       |------ trunk
       |       |------ tags
       |       |------ branches
```

Remarques:

Cette organisation n'est pas obligatoire, mais elle est standard, il est donc recommander de s'y conformer.

On peut également trouver d'autres répertoires, en particulier un répertoire « vendortag » qui contient les dépendances externes au projet.

4-Travailler sur un projet géré avec subversion.

Pour pouvoir travailler sur un projet géré avec subversion, il faut commencer par récupérer une version depuis le dépôt et la copier dans un répertoire de travail.

En effet, on travaille toujours dans un répertoire de travail[10] qui servira à faire des expérimentations, que l'on reportera ensuite éventuellement dans le référentiel sur le serveur.

La commande nécessaire à la récupération d'une version du projet depuis le dépôt est « checkout », qu'il est aussi possible d'abréger en « co »:

```
svn checkout repositoryUrl (destination)
```

ou plus rapidement:

```
svn co repositoryUrl (destination)
```

où « **repositoryUrl** » est le chemin vers le dépôt du projet sur lequel on souhaite travailler (voir « Comment accéder à un dépôt subversion ? » page 33) et « **destination** » est le chemin où le projet doit être copié.

En l'absence de « destination », le dernier élément de l'URL (basename) est utilisé à la place de destination. Le répertoire contenant la copie de travail portera donc ce nom.

5-Modifier sa copie de travail

Maintenant que l'on a obtenu une copie locale, on peur travailler librement sur le projet (modifier, ajouter, supprimer etc...).

Le travail sur un projet géré par subversion, consiste à modifier la copie de travail, en éditant un fichier par exemple, puis à propager la modification sur le référentiel, pour la rendre publique.

1.Modifier un fichier de la copie locale

La modification des fichiers de la copie locale est exactement identique à l'édition d'un fichier ordinaire. On travaille sur le fichier, puis on l'enregistre pour valider les modifications.

Subversion détecte que la version locale est différente de la version du dépôt, mais tant que les modifications ne sont pas transmises au dépôt, la version modifiée reste locale et est inconnue des autres développeurs. Pour la rendre publique il faut la valider sur le serveur par un « commit », voir « Validation d'une modification » page 14.

2.Ajouter un fichier ou un répertoire

Lorsque l'on souhaite inclure un nouvel élément dans un projet, il faut l'indiquer par la commande « svn add » :

```
svn add CHEMIN
```

Avec CHEMIN le chemin vers le ou les éléments (fichiers ou répertoires) à ajouter.

Subversion informe alors que la prise en compte des nouveaux éléments ne sera effective qu'après la publication des modifications (le prochain « commit »).

Comme pour l'édition des fichiers où il est nécessaire de répercuter les modifications sur le serveur, il en va de même pour la création ou la suppression des fichiers.

10 On appelle souvent ce répertoire « bac à sable » ou sandbox en anglais.

Les modifications ne seront effectives sur le dépôt (le serveur) qu'après le « commit » !

Remarques:

Sauf indication explicite, les fichiers n'appartenant pas au projet de départ et qui n'ont pas fait l'objet d'un « **svn add** » ne seront pas transférés sur le repository du serveur. Subversion se contentera d'indiquer qu'il a remarqué la présence de fichiers qu'il va ignorer, en faisant précéder leur nom d'un point d'interrogation.

D'une manière générale, il est préférable d'éviter de placer sous le contrôle de subversion un fichier que l'on peut générer automatiquement à partir d'un autre qui se trouve déjà dans le dépôt. Il suffit dans ce cas de laisser sous subversion le fichier source et un fichier permettant de le re-générer.

6- Validation d'une modification

Tant que l'on a pas reporté les modifications sur le serveur, celles-ci restent locales (dans la sandbox). Pour les rendre publique, il est nécessaire de les publier sur le dépôt.

La commande « **svn commit** » (ou « **svn ci** » en abrégé) envoie les modifications de la copie locale vers le dépôt:

```
svn commit (CHEMIN) –m "descriptif des modifications"
```

Avec CHEMIN le chemin vers le ou les éléments que l'on souhaite publier sur le repository. Si CHEMIN n'est pas indiqué, le « commit » s'applique au répertoire courant, ce qui permet de mettre à jour sur le référentiel l'ensemble des fichiers connus de subversion et qui ont été modifiés en local (la commande est récursive par défaut).

Par contre si CHEMIN est renseigné, c'est-à-dire que l'on fait suivre la commande d'un ou plusieurs fichiers, ou d'un répertoire, ils seront les seuls à être « commiter ».

Par exemple pour:

```
svn ci –m "descriptif des modifications" exemple.c module_alpha
```

seul *exemple.c* et *module_alpha* seront mis à jours dans le repository.

Un message descriptif de l'opération doit être fourni pour les journaux. S'il n'est pas donné , l'éditeur par défaut est lancé pour taper le message. Il est également possible d'utiliser un fichier contenant le message. Dans ce cas, on utilisera l'option -F fichierMessage à la place du -m « message ».

Remarques:

Si le projet sur le dépôt a été modifié par un autre développeur entre le « checkout » (le retrait de la copie locale) et le « commit » (la publication sur le serveur), le « commit » échouera expliquant que la copie de travail n'est pas à jour.

Il faut alors mettre à jour la copie de travail, avec la commande « **svn update** » avant de faire le « commit » (voir « Synchroniser la copie locale avec le dépôt du serveur »).

La plupart du temps, cet « update », suffira pour corriger le problème, mais il peut arriver qu'un conflit apparaisse. Dans ce cas, il faudra résoudre le conflit manuellement pour pouvoir valider les modifications sur le dépôt (voir « Résolution des conflits » page 16).

7-Synchroniser la copie locale avec le dépôt du serveur

La commande « **svn update** » (« **svn up** » en abrégé) permet de synchroniser la copie de travail avec le serveur. Cette opération n'aura pas d'influence sur les modifications locales qui seront intégralement conservées.

La syntaxe est simplement:

```
svn update (CHEMIN)
```

Avec CHEMIN le chemin vers la copie locale à mettre à jour. Si aucun chemin n'est donné, la commande mettra à jour le répertoire courant.

Si on veut traiter uniquement les fichiers du répertoire courant, sans récursion, il est nécessaire d'ajouter l'option « -N » à la commande, sinon par défaut svn étant récursif, la commande sera exécutée récursivement à partir du répertoire courant.

Subversion affiche alors les fichiers concernés et leur statut.

Le statut de chaque objet est représenté par un caractère:

A ajouté,

D supprimé,

U modifié,

C en conflit,

G fusionné.

Par défaut la copie de travail est actualisée par rapport à la révision HEAD, c'est à dire la révision la plus récente disponible sur le serveur, mais il est également possible de préciser une révision particulière en utilisant l'option « -r ARG » ou « --revision ARG ».

Dans ce cas l'argument ARG permettant de préciser une révision peut être:

- NUMÉRO un numéro de révision,
- '{' DATE '}' la date d'une révision,
- 'HEAD' la dernière révision du dépôt (la plus récente révision),
- 'BASE' la révision de base de la copie de travail. Si la copie de travail a été modifiée, BASE renvoi à la révision avant toute modification locale,
- 'COMMITTED' la dernière révision équivalente ou juste avant BASE,
- 'PREV' la révision juste avant COMMITTED, c'est-à-dire la révision immédiatement antérieure à la dernière révision qui a modifié un élément de la copie locale.

Par exemple, la commande:

```
svn update –r PREV foo.c
```

annule les derniers changements du fichier foo.c et fait descendre son numéro de révision.

Les commandes suivantes:

```
svn update -r {2006-02-17}
svn update -r {15:30}
svn update -r {"2006-02-17 15:30"}
```

mettent à jour l'ensemble de la copie locale avec une révision donnée par une date précise.

Si des fichiers ont été modifiés sur le serveur, les modifications qui ont été faites sont reportées dans les fichiers concernés localement.

Quand c'est possible subversion fusionne « auto-magiquement » les modifications du repository dans les fichiers locaux, sinon il génère un conflit qu'il faut résoudre manuellement (voir « Résolution des conflits »).

Remarques:

Attention pour subversion, la date {2006-02-17} est comprise comme {"2006-02-17 00:00:00"}...

En règle générale, il est recommandé de faire un « update » avant un « commit », mais pour éviter les conflits, mieux vaut effectuer des mises à jour fréquentes.

8-Résolution des conflits

La résolution d'un conflit correspond à l'intégration des changements effectués par les autres sur une même section d'un fichier que l'on a modifié localement.

Les conflits interviennent au moment d'un « **`svn update`** », lorsque des modifications ont été faites simultanément dans la copie de travail et dans le dépôt.

Par exemple, lorsqu'on édite en local un fichier pour rajouter ou éditer une ligne, et qu'un autre utilisateur a « commité » entre temps une modification de cette même ligne, le « commit » échoue avec l'erreur suivante:

```
svn commit
Sending        foo.c
svn: Commit failed (details follow):
svn: Your file or directory 'foo.c' 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).
```

Comme on vient de le voir, il faut effectuer un « update », qui va mettre à jour le ou les fichiers concernés automatiquement quand c'est possible.

Dans ce cas le message est le suivant :

```
svn update
G  foo.c
Updated to revision 12.
```

Le conflit est résolu, il ne reste plus qu'à faire le commit, mais il est préférable de vérifier manuellement le résultat de la résolution « auto-magique », avant de « commiter ».

Dans certains cas, les modifications ne peuvent pas être fusionnées automatiquement,

quand elles concernent les mêmes parties d'un fichier, comme dans l'exemple ci-dessus.

Le conflit est alors signalé au moment de l'update par la lettre C:

```
svn update
C  foo.c
Updated to revision 13.
```

Des nouveaux fichiers font leur apparition dans la copie de travail:

foo.c.mine, la version que l'on vient de modifier en local (celle que l'on voulait publier avant le conflit),

foo.c.r12 la version de base (celle commune à la copie locale et au dépôt),

foo.c.r13 la version actuelle du dépôt (celle modifiée par un autre utilisateur),

foo.c une version spéciale, dédiée à la résolution du conflit, qui présente les différences entre les différentes versions.

La résolution du conflit consiste à éditer le fichier foo.c et à faire le choix d'une version ou la synthèse des deux.

Pour guider le développeur, subversion signale les parties qui posent problème entre les sections:

<<<<<<< .mine

et

>>>>>>> .rX

(avec X correspondant au numéro de révision sur le dépôt ; r13 dans l'exemple).

```
<<<<<<< .mine
bool isReady() { if (_bReady) return true ; else return false; }
=======
bool isReady() { return _bReady ; }
>>>>>>> .r13
```

Dans cet exemple, la même fonction est écrite de deux façons différentes, il faut en choisir une, supprimer l'autre, puis de signaler que le conflit est résolu.

Cette opération est réalisée à l'aide de la commande « **`svn resolved`** »:

```
svn resolved foo.c
Resolved conflicted state of 'foo.c'
```

Comme précédemment, il faut encore propager la nouvelle version du fichier avec un « commit ».

Remarque:

Pour éviter les conflits, le mieux c'est encore de communiquer. Il faut garder à l'esprit que subversion ne peut pas se substituer à la communication entre les membres de l'équipe projet...

9- Obtenir des informations sur le projet

Un des intérêts de subversion est de pouvoir suivre l'évolution d'un projet et si besoin, de pouvoir revenir en arrière (à une version donnée).

Pour cela, il est nécessaire d'avoir des informations sur les fichiers du projet et il peut être utile de connaître l'historique des modifications.

Un certain nombre de sous-commandes svn sont dédiées à cette tâche.

1 - La sous-commande « info »

La commande « **svn info** » permet d'obtenir des informations sur la copie locale ou sur le serveur.

```
svn info (CIBLE(@REV)...)
```

Par défaut cette commande affiche des informations sur le répertoire courant ou pour chaque CIBLE si CIBLE est précisée.

CIBLE peut être un répertoire ou un fichier dans une copie de travail ou une URL.

REV permet de spécifier, une révision particulière qui sera d'abord examinée.

L'exécution de la commande « **svn info** » donne par exemple:

```
svn info
Chemin : .
URL : file:///tmp/prep-svn/testsvn
Racine du dépôt : file:///tmp/prep-svn/testsvn
UUID du dépôt : f19eb3ad-fe6f-48c6-85f5-697b89699a04
Révision : 11
Type de noeud : répertoire
Tâche programmée : normale
Auteur de la dernière modification : pbo
Révision de la dernière modification : 11
Date de la dernière modification: 2007-08-31 23:54:20 +0200 (ven,
31 aoû 2007)
```

2 - La sous-commande « status »

La commande « **svn status** » permet d'obtenir des informations sur les objets de la copie de travail en allant éventuellement regarder dans le dépôt si on l'utilise avec l'option --show-updates. Elle affiche les informations de révision complète de chaque objet si on l'utilise avec en mode verbose (--verbose).

```
svn status –v README
M               11       11 pbo           README
```

L'affichage de cette commande se lit en ligne: chaque ligne affiche les informations correspondant à un fichier du projet.

Les six premières colonnes font un caractère:

1ère colonne: indique si l'objet est ajouté, supprimé ou modifié

- ' ' pas de modification
- 'A' ajouté
- 'C' en conflit
- 'D' supprimé
- 'I' ignoré
- 'M' modifié
- 'R' remplacé
- 'X' non versionné, mais utilisé par une référence externe
- '?' non versionné
- '!' manquant (supprimé par une autre commande) ou incomplet
- '~' objet versionné dissimulé par un objet différent
- 2ème colonne: indique modification d'une propriété
- ' ' pas de modification
- 'C' en conflit
- 'M' modifiée

3ème colonne: indique si le répertoire local est verrouillé

- ' ' non verrouillé
- 'L' verrouillé (locked)

4ème colonne: indique si la prochaine propagation (commit) comportera un ajout avec reprise de l'historique

- ' ' pas d'ajout avec reprise d'historique prévu
- '+' ajout avec reprise d'historique prévu
- 5ème colonne: indique si l'élément a une référence différente de son parent
- ' ' normal
- 'S' ré-aiguillé (switched)
- 6ème colonne: si un verrou est posé

(sans -u)

- ' ' normal
- 'K' verrouillé

(avec -u)

- ' ' pas de verrou dans le dépôt, pas de verrou local
- 'K' verrouillé dans le dépôt, verrou local (lock toKen) présent
- 'O' verrouillé dans le dépôt, verrou dans une autre copie (Other)

- 'T' verrouillé dans le dépôt, verrou local présent mais volé (sTolen)
- 'B' non verrouillé dans le dépôt, verrou local cassé (Broken)
- 8e colonne (avec -u) : indique l'obsolescence d'un objet
- '*' une nouvelle version existe sur le serveur
- ' ' la copie de travail est à jour

Les autres colonnes, sont de taille variable et délimitées par des espaces. Elles indiquent:

- La révision de travail (avec -u ou -v)
- La dernière révision propagée et son auteur (avec -v)

Le chemin de l'objet est toujours dans le dernier champs et peut inclure des espaces.

3 - La sous-commande « log »

Pour voir l'historique des modifications d'un fichier ou d'un répertoire, on utilisera la commande « **`svn log`** ». Elle affiche les entrées du journal pour un ensemble de révisions ou de fichiers:

```
log (CHEMIN)
```

Par défaut, elle affiche les entrées du journal pour le chemin local « . ». Les révisions prises en compte par défaut sont BASE:1.

Mais on peut aussi utiliser:

```
log URL(@REV) (CHEMIN...)
```

Dans ce cas, si CHEMIN est présent les entrées du journal pour les CHEMINs sous l'URL seront affichées.

Si REV est précisée, ce sera la révision d'abord prise en compte au lieu de HEAD:1 par défaut.

Exemple:

```
svn log index.html
------------------------------------------------------------------------
r37 | pbo | 2007-05-30 14:42:26 +0200 (lun, 30 mai 2007)| 2 lines
Modification du titre de la page
```

Si l'option -v est utilisée, elle affiche également les chemins concernés par le message.

Si l'option -q est utilisée, le corps du message n'est pas affiché (compatible avec -v).

Par défaut, le journal suit l'historique des copies, sauf si l'option --stop-on-copy est précisée.

Remarques:

Chaque entrée du journal n'est affichée qu'une seule fois, même si plusieurs chemins explicitement spécifiés sont concernés.

La sous-commande log est particulièrement utile pour connaître les révisions utilisées quand on veut fusionner des branches.

4 - La sous-commande « diff »

La commande « **svn diff** » permet de regarder les différences entre deux révisions données.

Elle permet de comparer les fichiers modifiés localement avec ceux de la base, mais aussi de comparer les fichiers locaux avec ceux d'une version particulière, indiquée en utilisant l'option −r suivie d'un identifiant de révision, ou de deux révisions séparées par le caractère « : ».

Par exemple, la commande « svn diff -r5:11 » donne le résultat suivant:

```
svn diff -r5:11
Index: README
===================================================================
=
--- README      (révision 5)
+++ README      (révision 11)
@@ -1,6 +1,7 @@
 Un petit exemple de projet pour trac et svn.
```

```
-Il faut utiliser:
+Après avoir installer le sccript post-commit-hook dans le
repertoire hook  de svn,
+il faut utiliser:
        - fixes #N
 ou
        - closes #N
@@ -16,4 +17,6 @@

 (avec #N representant le numéro de l'issue (du ticket) concerné.

+Remarque:
+Il faut que l'utilisateur est la permission (dans trac) de
cloturer un ticket...
```

Les éléments ajoutés par rapport à la version la plus ancienne sont précédés d'un « + », alors qu'à l'inverse les éléments supprimés par rapport à la version la plus récente sont précédés d'un « - ».

Remarque:

Il est facile de créer un patch avec la commande « **svn diff** » de subversion:

```
svn diff > patchfile
```

On obtient alors le patch patchfile qui contient les différences entre les deux révisions (sur le serveur et dans la copie locale).

5 - La sous-commande « blame »

La sous-commande « **svn blame** » (également appelée « praise », « annotate », « ann ») permet d'afficher le contenu d'un élément ligne par ligne, avec la révision et l'auteur concerné:

```
svn blame index.html
    36   pbo <html>
    36   pbo <head></head>
    36   pbo <body>
    37   pbo <h1>Introduction à subversion</h1>
    36   pbo </body>
    36   pbo </html>
```

Contrairement à la commande « **diff** », la commande « **annotate** » ne compare pas deux versions, elle affiche la chronologie des modifications.

10-Revenir à une version antérieure

Si les modifications effectuées dans la copie locale ne nous conviennent pas et qu'on souhaite retrouver le projet tel qu'il était lors du dernier « **update** », on peut utiliser la commande « **svn revert** »:

```
svn revert index.html
Reverted 'index.html'
```

Cette commande n'a pas besoin d'un accès au dépôt tout est effectué en local.

Si ce n'est pas suffisant, on peut récupérer une version antérieure à partir du dépôt. Dans ce cas, il faut utiliser « **svn update** » en précisant le numéro de la révision et éventuellement le ou les fichiers concernés:

```
svn update –r 16 index.html
U  index.html
Updated to revision 16.
```

Remarque:

La commande « revert » annule aussi toutes les opérations programmées, comme l'ajout ou la suppression de fichiers (quand elle n'ont pas été suivies d'un « commit »).

11-Résumé du cycle de travail basique:

Le cycle de travail typique peut se résumer comme suit:

- Obtenir ou mettre à jour sa copie de travail (svn checkout, svn update)
- Faire des changements (édition des fichiers, svn add, svn delete, svn copy, svn move)
- Etudier les modifications (svn status, svn diff, svn revert)
- Intégrer le travail des autres dans la copie locale (svn update, svn resolved)
- « Commiter » ses modifications (svn commit)

2-Quelques conseils:

L'usage de subversion n'est pas difficile, et laisse une grande liberté au développeur, mais il convient de respecter certaines règles notamment concernant les mises à jour.

a - Quand « mettre à jour » (commit) ?

Les mises à jour par commit devraient être d'autant plus fréquentes qu'un projet comporte peu de modules mais compte beaucoup de participants.

Il ne faut pas perdre de vue que subversion n'archive que l'historique des modifications et pas les fichiers eux-mêmes. Ainsi la différence entre de nombreuses mises à jour par commit (modifications mineures successives des fichiers) et une politique de mises à jour plus espacées (modifications plus substantielles) n'a en pratique que peu d'impact en termes d'occupation du disque.

Il est tout de même recommandé de ne « commiter » que du code ayant été compilé (voire testé) avec succès. Le bon sens conduit en effet à considérer qu'il n'est peut n'être pas souhaitable que des éléments incomplets soient envoyés sur le dépôt si d'autres personnes s'attendent à ne trouver que du code fonctionnel. À l'inverse, il ne saurait être question de garantir que tout est en ordre avant de lancer le commit... Bref, c'est là évidemment une décision « politique », qui ne relève pas directement de l'usage de subversion.

Mais il faut noter que l'utilisation des « branches » permet de conserver parallèlement une version stable accessible au public en même temps qu'une version de développement (instable) sur le même référentiel. Dans le même esprit, l'usage des « labels » se révèle très pratique pour identifier, par exemple, la dernière révision livrée au client.

b - Comment se préserver des collisions ?

Malgré la modeste contrainte qu'imposent les « commits » fréquents, une politique de mises à jour rapprochées peut faire gagner beaucoup de temps en évitant la survenue de collisions. Si subversion sait en effet identifier (voire régler tout seul) les conflits de révision, cela ne dispense guère les utilisateurs d'agir avec prudence et bon sens.

En règle générale, il est souhaitable de vérifier qu'on dispose bien d'une révision « à jour » avant de se lancer dans une intervention. Si la commande status répond Up to date, alors tout va bien. Mais si subversion indique qu'une mise à jour est nécessaire (Needs Checkout), inutile d'hésiter.

Enfin, on veillera à respecter les standards mis en place pour ne pas intervenir sur la mise en forme des lignes ou paragraphes déjà édités.

Par ailleurs, bien qu'il soit possible de « verrouiller » un fichier pour en empêcher l'édition simultanée, ce mécanisme n'est pas recommander, parce qu'il ajoute des contraintes mais ne résout pas les problèmes. Une bonne communication entre les intervenants s'avère de toute façon incontournable...

3-Utilisation avancée de subversion

1-Obtenir une version « livrable » du projet avec export

La commande « **svn export** » crée une copie non versionnée d'une arborescence subversion. Elle est similaire à « **svn checkout** » et permet d'obtenir une copie locale du projet (ou d'un sous répertoire du projet), mais sans les informations de gestion de versions (répertoires internes de subversion « .svn »).

Cette commande peut s'utiliser aussi bien pour récupérer le projet à partir d'un serveur ou bien à partir d'une copie locale.

On peut préciser la révision avec l'option -r REV où REV correspond à un identifiant de révision.

Par exemple la commande:

```
svn export file:///tmp/depot-svn/trunk livrable_0.1
A    livrable_0.1
Exporté à la révision 11.
```

exporte dans le répertoire livrable_0.1 la dernière révision du répertoire « trunk ».

On l'utilise pour obtenir du code livrable « débarrassé » des informations inutiles à l'utilisateur finale.

2-Propriétés et substitution de mot-clés

Avec subversion, il est possible d'ajouter dans les fichiers des variables qui seront mises à jour automatiquement au fil des révisions.

La syntaxe est commune pour tous les mots clés: « **$keyword$** ».

Les mots clés sont:

- Revision (ou Rev en abrégé): la dernière révision.
- HeadURL (ou URL en abrégé): l'url du fichier (sa localisation dans le dépôt).
- LastDateChange (ou Date en abrégé): la date de la dernière révision.
- Author: l'auteur de la dernière révision.
- Id: un identifiant rassemblant les informations principales des autres mots clés.

Ces mots clés sont très utiles pour pouvoir identifier facilement un fichier.

Pour qu'ils soient remplacés par leur valeur, il faut modifier les propriétés des fichiers pour activer la substitution des mots clés, sinon rien ne se passera...

Par exemple pour activer la substitution de tous les mots clés, on exécute:

```
svn propset svn:keywords "Rev URL Date Author Id" README
Propriété 'svn:keywords' définie sur 'README'
svn ci -m "Mise en place de la substitution des mots clés"
```

L'affichage du fichier avant la validation des propriétés était le suivant:

```
svn cat -r12 README
           Fichier explicatif
================================================
INFORMATIONS:
$Rev$
$URL$
$Date$
$Author$
$Id$
================================================
Pour compiler le projet
Taper dans le répertoire des sources du projet:
    ./configure
    make
    make install
```

Avec l'expansion des mots clés, le résultat est le suivant:

```
svn cat README
           Fichier explicatif
================================================
INFORMATIONS:
$Rev: 13 $
$URL: file:///tmp/depot-svn/trunk/README $
$Date: 2007-10-06 11:06:18 +0200 (sam, 06 oct 2007) $
$Author: pbo $
$Id: README 13 2007-10-06 09:06:18Z pbo $
================================================
Pour compiler le projet
Taper dans le répertoire des sources du projet:
    ./configure
    make
    make install
```

Remarque:

L'activation manuelle des propriétés pour chaque fichier peut vite devenir fastidieuse si le projet contient beaucoup de fichiers. C'est pourquoi il est possible d'activer certaines propriétés automatiquement en éditant le fichier de configuration de subversion « **`~/.subversion/config`** » sur son poste client.

```
### Set enable-auto-props to 'yes' to enable automatic properties
### for 'svn add' and 'svn import', it defaults to 'no'.
### Automatic properties are defined in the section 'auto-props'.
enable-auto-props = yes
### Section for configuring automatic properties.
[auto-props]
*.c = svn:eol-style=native;svn:keywords=Rev Id Date Author URL
*.cpp = svn:eol-style=native;svn:keywords=Rev Id Date Author URL
*.h = svn:eol-style=native;svn:keywords=Rev Id Date Author URL
*.php = svn:eol-style=native;svn:keywords=Rev Id Date Author URL
*.php3 = svn:eol-style=native;svn:keywords=Rev Id Date Author URL
*.html = svn:eol-style=native;svn:keywords=Rev Id Date Author URL
*.css = svn:eol-style=native;svn:keywords=Rev Id Date Author URL
*.dsp = svn:eol-style=CRLF
*.dsw = svn:eol-style=CRLF
*.sh = svn:eol-style=native;svn:keywords=Rev Id Date Author
URL;svn:executable
Makefile = svn:eol-style=native;svn:keywords=Rev Id Date Author
URL
*.txt = svn:eol-style=nativer
*.png = svn:mime-type=image/png
*.jpg = svn:mime-type=image/jpeg
```

Cet extrait constitue un exemple classique de configuration des propriétés. Les fichiers sont identifiés par leur nom. Il est possible d'utiliser des jokers (« wildcards ») pour diférentier les fichiers par leur extension. Les différentes propriétés sont séparées par un « ; ».

3-Tags et Branches de développement

Jusqu'à maintenant, toutes les opérations ont été toujours effectuées dans le répertoire principa du projet.

Mais bien souvent le développement d'un logiciel n'est pas linéaire et il est nécessaire de pouvoir travailler sur une version particulière tandis que le développement principal continue par ailleurs.

C'est possible grâce aux tags et aux branches.

1-Etudions un exemple pour mieux comprendre

Supposons que l'on développe un logiciel « Lambda » dont la gestion de version est confiée à subversion.

Le projet « Lambda » est très ambitieux et pour être efficace, les développeurs qui sont expérimentés, ont prévu un plan de développement en plusieurs étapes.

L'équipe se met au travail et publie ses modifications dans le répertoire « trunk », le « tronc » du projet. Après un certain temps le projet est arrivé à l'étape 1. Les développeurs décident donc de publier la première version du projet: la release 0.1.

Ils annoncent au monde entier que leur projet est en bonne voie et que l'on peut dores et déjà utiliser la « release 0.1 » du projet.

Pour simplifier la récupération de cette release, ils créent un tag, c'est-à-dire une copie du projet au stade « release 0.1 », avec la commande:

```
svn copy URL/trunk URL/tags/release-0.1.
```

Ainsi tous ceux qui veulent essayer le projet peuvent facilement récupérer les sources à partir du dépôt subversion avec la commande:

```
svn co URL/tags/release-0.1 ou svn export URL/tags/release-0.1
```

Le premier avantage, c'est que pendant ce temps l'équipe projet peut continuer le développement du projet dans le « trunk » pour passer à l'étape 2. Même si la version en cours de développement n'est pas très stable, les utilisateurs utiliserons la version 0.1, qui est plus fiable.

Le second avantage concerne la gestion du dépôt: il est possible d'autoriser un accès public en lecture seul pour le répertoire « tags », alors que l'accès au répertoire « trunk » sera réservé aux développeurs par exemple...

Supposons maintenant que les utilisateurs qui emploient la version 0.1 détectent un problème sérieux dans cette version. Il faut que l'équipe corrige ce bogue.

Pour résoudre le problème sans perturber l'avancement du projet vers l'étape 2, l'équipe créé une branche à partir de la release 0.1, dans laquelle elle va pouvoir travailler pour corriger le bogue. Ce sera encore une copy:

```
svn copy URL/tags/release-0.1 URL/branches/bugfix-0.1
```

Les développeurs corrigent le problème et publient une nouvelle version 0.1 corrigée. Comme précédemment ils « étiquettent » cette version en copiant la branche dans un tag:

```
svn copy URL/branches/bugfix-0.1  URL/tags/release-0.1.1
```

Parallèlement le développement du tronc continue vers l'étape 2, mais il serait dommage que la nouvelle version ne profite pas de la correction, alors l'équipe décide de fusionner la branche « bugfix-0.1 » avec le « tronc ». Pour ça le développeur en charge de la fusion fait un « merge » des deux versions dans sa copie de travail.

```
svn merge URL/branches/bugfix-0.1 trunk@head
```

Quand l'étape 2 est achevée, l'équipe publie la version 2 avec un tag « release 0.2 ».

Et ainsi de suite jusqu'à la fin du projet.

Remarque:

L'utilisation des tags et des branches entraîne nécessairement une multiplication des copies du projet dans le dépôt, sans que la place occupée par le projet sur serveur n'augmente énormément car les données ne sont pas dupliquées. Subversion fait des copies « intelligentes » qui sont similaires à des alias.

2-Création des « tags » ou des branches

D'un point de vue technique, la création d'une branche ou d'un tag avec subversion est exactement identique. Mais d'un point de vue logique, les labels sont toujours en lecture seul. Par principe on ne travaille jamais dans un label, mais dans une branche. Ce n'est qu'une convention, mais il est vraiment recommander de la rescpecter.

La commande utilisée pour créer une branche ou un tag est « **svn copy** ». Elle recopie quelque chose dans une copie de travail ou un dépôt, en conservant l'historique des objets.

```
svn copy SRC DST
```

avec SRC et DST qui sont un chemin dans la copie de travail (CT) ou une URL.

Les cas suivants sont possibles:

- CT -> CT : copie et mise en attente pour ajout (avec historique)
- CT -> URL : propage immédiatement une copie de CT vers URL
- URL -> CT : extrait une URL dans CT, mise en attente pour ajout
- URL -> URL : copie côté serveur ; utilisée pour les branches et les tags

Chaque fois que le résultat final est dans la copie de travail, il faut valider l'opération par un « commit » comme pour l'ajout d'un fichier.

3-Fusion des « branches »

La fusion des branches est réalisée avec la commande « **svn merge** ». Elle applique les différences entre deux sources à une copie de travail.

Son usage nécessite souvent l'utilisation des numéros de révisions de chaque branche fusionnées. La commande « **svn log** » permettra d'obtenir facilement ces numéro.

Il existe plusieurs façons d'utiliser la commande « **svn merge** »:

```
merge URL1[@N] URL2[@M] [CHEMIN]
```

Les URLs des deux références sont précisées. Les révisions peuvent aussi être précisées, si elles ne le sont pas, la révision utilisée est HEAD.

```
merge CHEMIN1@N CHEMIN2@M [CHEMIN]
```

Les URLs sont déduites des chemins dans la copie de travail. Dans ce cas, les révisions doivent être précisées.

```
merge [-c M|-r N:M] SOURCE[@REV] [CHEMIN]
```

La SOURCE est une URL ou un chemin dans une copie de travail. L'URL vue à la révision REV est comparée entre les révisions N et M. Si REV n'est pas précisée, HEAD est utilisée.

L'option '-c M' est équivalente à '-r N:M' avec N = M-1.

Utiliser '-c -M' fait l'inverse : '-r M:N' avec N = M-1.

Dans tous les cas, CHEMIN est l'élément à modifier dans la copie de travail. S'il est omis, le répertoire courant « . » est utilisé, sauf si les noms des sources ont un dernier composant identique à un fichier dans le répertoire courant « . », auquel cas c'est ce fichier qui sera modifié.

Remarque:

La fusion des branches est une opération délicate, qu'il ne faut pas réaliser dans la précipitation. Une fois réalisée il est important de bien la commenter dans le message du commit.

4-La commande « switch »

Elle permet de basculer entre deux branches dans une copie de travail. Elle actualise la copie de travail à partir d'une URL.

Par exemple la commande :

```
svn switch URL (CHEMIN)
```

actualise la copie de travail vis-à-vis d'une autre URL dans le dépôt. Le comportement est similaire à 'svn update', et constitue la bonne manière d'aiguiller une copie de travail vers une branche dans le même dépôt.

La commande:

```
switch --relocate DE VERS (CHEMIN...)
```

quand à elle, réécrit les URL internes de la copie pour refléter un simple changement syntaxique, par exemple si l'URL racine du dépôt a changé comme dans le cas d'un renommage de la machine où la copie de travail reflète toujours le même répertoire du même dépôt.

4-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

1-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/

2-Kdesvn

Kdesvn est un « clone » de TortoiseSVN pour KDE.

voir http://kdesvn.alwins-world.de/

3-SubEclipse

SubEclipse est un environnement intégré à Eclipse.

voir http://subeclipse.tigris.org/

4-Quelques autres interfaces:

eSVN voir http://esvn.umputun.com/

JSVN voir http://jsvn.alternatecomputing.com/

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

AnkhSVN voir http://ankhsvn.tigris.org/

Emacs VC un add-on pour Emacs (add-to-list 'vc-handled-backends 'SVN)

etc...

5-Les concurrents de subversion

Il existe un grand nombre de logiciels dédiés à la gestion de version.

Il est difficile de les connaître tous, mais on trouve parmi les plus connus: CVS, AccuRev, Aegis, Arch, BitKeeper, ClearCase, CMSynergy, Co-Op, Darcs, Mercurial, Monotone, OpenCM, Perforce, PureCM, Superversion, svk, Vesta, Visual, SourceSafe, Git...

Il existe plusieurs comparatifs disponible en ligne.

Un comparatif aussi complet que possible pourra être consulté à la page suivante:

http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

Un autre comparatif dédié au seuls logiciels libre:

http://better-scm.berlios.de/comparison/comparison.html

Un comparatif plus synthétique est disponible à l'adresse suivante:

http://zooko.com/revision_control_quick_ref.html

Pour comparer subversion avec la concurrence, on peut retenir quelques points essentiels:

- subversion est stable.
- subversion est centralisé.
- subversion est multi-plateforme.
- subversion est un logiciel libre.
- subversion permet différents modes d'accès au dépot, dont un acèss direct, à travers SSH, SVN ou WebDAV via Apache.

On peut dire que subversion est sans doute le logiciel libre pour la gestion de version centralisée le plus abouti et qu'il est aussi un des plus utilisés et les mieux documentés ce qui en fait un excellent choix.

6-Bibliographie et sites internet:

1-Articles et publications:

[1] Laurent Bloch, Systèmes de gestion de version et d'archivage, mai 2006, http://www.laurent-bloch.org/spip.php?article93

[2] Stéphane Bortzmeyer, Les nouveaux Systèmes de Gestion de Version, décembre 2005, http://2005.jres.org/paper/2.pdf

[3] St?phane VANPOPERYNGHE, Installation et utilisation de base de Subversion, mars 2004, http://toutprogrammer.com/article_19.html

[4] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato, Version Control with Subversion, O'Reilly, 2004, http://svnbook.red-bean.com/

[5] Garrett Rooney, Practical Subversion, Apress

[6] Mike Mason; Pragmatic Version Control Using Subversion; Pragmatic Bookshelf

[7] William Nagel; Subversion Version Control: Using the Subversion Version Control System in Development Projects; Prentice Hall, http://www.phptr.com/content/images/0131855182/downloads/Nagel_book.pdf

2-Sélection de sites internet:

[8] Le site officiel de Subversion: http://subversion.tigris.org/

[9] La FAQ de Subversion: http://subversion.tigris.org/faq.html

[10] La documentation de TortoiseSVN (en anglais): http://tortoisesvn.net/support

[11] Wikipedia, http://fr.wikipedia.org/wiki/Subversion_%28logiciel%29

[12] Tutorial très complet: http://eiffelsoftware.origo.ethz.ch/index.php/Subversion_Tutorial

[13] Toutes les références de subversion sur une page recto-verso, http://www.cs.put.poznan.pl/csobaniec/Papers/svn-refcard.pdf

Annexes

5.2 - Création d'un référentiel

Pour créer un nouveau dépôt subversion, il suffit d'utiliser la commande « svnadmin » qui est destiné à l'administration d'un dépôt.

Comme pour la commande svn, on pourra accéder à l'aide en ligne en tapant « svnadmin help », par exemple pour obtenir l'aide de la sous-commande create:

```
svnadmin help create
```

Pour créer un nouveau dépôt la commande est:

```
svnadmin create CHEMIN_DÉPÔT
```

avec CHEMIN_DÉPÔT correspondant au répertoire qui contiendra le dépôt.

Il est possible d'utiliser des options pour paramétrer le repository, mais les options par défaut sont en principe recommandées.

L'option « --fs-type ARG » avec ARG égale à 'fsfs' (par défaut) ou 'bdb' permet de choisir le mécanisme de stockage du dépôt: soit le système de fichier développé par l'équipe de subversion, soit une base de donnée Berkeley DB, mais le système FSFS est hautement recommandé.

Remarque:

Le dépôt créé par svnadmin create est vide, il faut ensuite importer un projet à l'intérieur du repository.

5.3 - Comment accéder à un dépôt subversion ?

On peut accéder à un dépôt svn de plusieurs façons différentes. Subversion identifie le protocole utilisé en fonction de l'URL. Le schéma général d'une URL permettant d'accéder à un dépôt est le suivant:

```
Protocole Hôte Chemin_vers_le_dépôt
```

où Protocole détermine le protocole utilisé, Hôte le nom ou l'adresse de la machine contenant le dépôt et Chemin_vers_le_dépôt le chemin vers le dépôt ou un sous-répertoire du dépôt.

Tableau des différentes URLs d'accès au dépôt

Protocoles	Méthode d'accès
`file:///`	accès direct au dépôt (disque local ou disque réseau).
`http://`	accès via le protocole WebDAV a un serveur Apache + Subversion.
`https://`	comme `http://`, mais avec cryptage SSL.
`svn://`	accès à travers le protocole svn à un serveur `svnserve` .
`svn+ssh://`	comme `svn://`, mais à travers un tunnel SSH.

Le tableau ci-dessus résume les différents protocoles possibles et leurs urls respectives.

1- Protocole « file »:

La première et la plus simple des possibilités c'est d'accéder à un dépôt local. Dans ce cas le protocole est *file*.

Il suffit d'indiquer le chemin vers le dépôt ou le répertoire à l'intérieur du dépôt. L'url sera de la forme:

```
file:///chemin/du/depot
```

Dans ce cas l'hôte « localhost » est « sous-entendu », l'url pourrait aussi bien s'écrire

```
file://localhost/chemin/du/depot.
```

Cette méthode est rapide (puisqu'il n'y a pas d'accès réseau) et peut être très utile dans le cas d'un utilisateur seul, ou pour administrer un dépôt.

Remarque pour les utilisateurs de windows:

L'accès à un dépôt sur un répertoire partagé par un serveur de fichier Windows bien qu'utilisant le protocole file nécessite une syntaxe particulière:

```
C:\> svn command file:///X:/chemin/du/depot

ou

C:\> svn command "file:///X|/chemin/du/depot"
```

Les deux commandes sont équivalentes, mais attention à ne pas oublier les double quote « " » encadrant l'url dans la seconde écriture, sinon la barre « | » sera interprété comme un pipe.

On peut également noter qu'il faut utiliser des slash « / » et pas des anti-slash « \ » pour la construction de l'url.

2-Protocole « svn »:

Le second protocole est le protocole SVN, défini par l'équipe du projet Subversion. Il fait appel au programme serveur svnserve.

Les options concernant l'authentification sont définies dans le fichier de configuration du serveur svnserve.conf (présent dans le dépôt, à l'emplacement conf/svnserve.conf). L'url est de la forme:

```
svn://login@machine/chemin/du/depot
```

En cas de besoin, le client svn affiche un prompt pour interroger l'utilisateur sur son mot de passe.

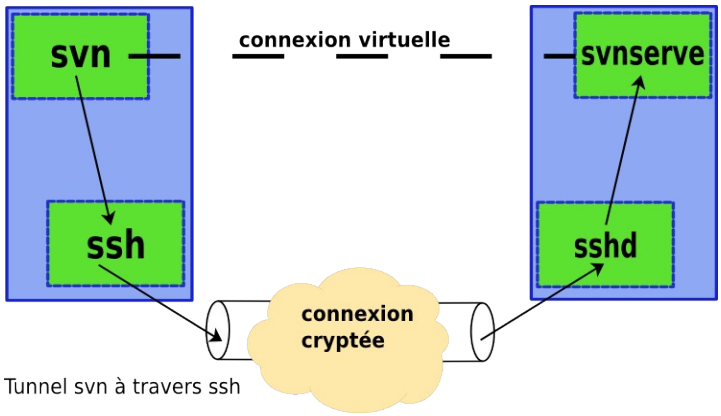
Les niveaux d'accès sont définis dans le fichier svnserve.conf.

3-Protocole « Svn+ssh »:

Le protocole svn peut être sécurisé au travers d'un tunnel SSH, dans ce cas l'url sera du type:

```
svn+ssh://login@machine/chemin/du/depot
```

Le principe est le même que pour le protocole SVN, mais le transport est assuré et sécurisé par SSH.



Le schéma ci-dessus résume le circuit logique des données par la connexion virtuelle et le circuit physique à travers la connexion réelle.

4-Protocole « WebDav » (http & https) :

Le dernier protocole est le protocole WebDav. WebDav[11] est une extension du protocole HTTP, qui permet de simplifier la gestion de fichiers avec des serveurs distants.

Ce protocole utilise le module « mod_dav_svn » pour apache 2. Il permet à la fois de donner un accès au dépôt en lecture seul, permettant de parcourir les répertoires et les fichiers du projet, mais aussi un accès en écriture permettant l'enregistrement des modifications.

Les options concernant l'authentification sont alors définies dans le fichier de configuration du serveur apache, dans la directive « Directory » ou « Location » concernant le dépôt SVN.

L'url est alors de la forme:

```
http://login@machine/chemin/du/depot
```

ou (si on ne donne pas de login)

```
http://machine/chemin/du/depot
```

ou encore

```
https://login@machine/chemin/du/depot
```

si l'on a mis en place SSL sur le serveur apache.

Si l'on ne fournit pas le login, une fenêtre d'authentification apparaît permettant de préciser le login et le mot de passe.

La configuration minimale d'un accès au dépôt est assez simple. Par exemple:

```
<Location /depot>
  DAV svn
  SVNPath /chemin/absolu/jusqu'au/depot
</Location>
```

La réalisation d'une configuration plus complète est aussi plus complexe, mais on trouvera beaucoup d'informations sur le site http://svnbook.red-bean.com/.

11 Le protocole WebDav “Distributed Authoring and Versioning” est décrit dans la RFC 2518.