Subversion,
un système de gestion de version

Lionel Meister
Codiciel, UPS CNRS 856
lionel.meister@mn.esm2.imt-mrs.fr

Table des matières

1 Introduction **2**

2 Présentation générale **2**

3 Utilisation de base de Subversion à travers un exemple **2**

3.1 Création du dépôt – Version 0 3
3.2 Import du projet dans la base Subversion – Version 1 4
3.3 Modifications de fichiers et ajouts de fichiers – Version 2 5
3.4 Création de répertoires et effacement de fichiers – Version 3 7
3.5 Déplacement de fichiers – Version 4 8
3.6 Résumé d'un cycle de travail sous Subversion 9
3.7 Les logs 9

4 Utilisation de Subversion avec différents développeurs **10**

4.1 Récupération de la base 11
4.2 Conflit géré par Subversion 11
4.3 Conflit à gérer manuellement 13

5 Comment utiliser Subversion pour suivre l'historique **15**

5.1 `svn log` 15
5.2 `svn diff` 18
5.3 `svn update` 20
5.4 `svn cat` 20
5.5 `svn revert` 20
5.6 `svn list` 21

6 Conclusion **21**

1 Introduction

Subversion est un système de gestion de versions[1]. L'outil le plus utilisé dans ce domaine est CVS, dont le développement est plus ancien et qui fait office de référence. Les concepteurs de Subversion l'ont en partie construit avec l'idée de simplifier un grand nombre de tâches qui sont fastidieuses et peu intuitives avec CVS. Ce document a pour but de présenter l'utilisation de base de Subversion.

2 Présentation générale

Dans Subversion, on crée tout d'abord un dépôt (*repository* en anglais) qui va servir à entreposer l'ensemble d'un projet. Une fois ce dépôt créé et rempli de l'ensemble des fichiers constituant le projet, chaque personne y ayant accès peut charger le contenu sur un disque local (c'est à dire autre que sur le disque où se trouve le dépôt). Une fois sur son disque local, cette personne peut effectuer un certain nombre d'opérations diverses (modifier des fichiers, en effacer, compiler le code source, etc.), et ceci sans affecter le contenu du dépôt.

Si elle y est autorisée, cette personne peut soumettre ses modifications au dépôt. Au lieu d'effacer le contenu du dépôt, Subversion va stocker les différences par rapport aux fichiers originaux, et ainsi établir une nouvelle version qui sera distribuée aux autres utilisateurs qui la demanderont. Comme tout gestionnaire de versions, Subversion permet bien entendu de consulter l'ensemble des modifications faites au cours du temps par les différents utilisateurs, de reprendre une ancienne version, etc. L'intérêt est de ne perdre aucune modification faite dans le développement d'un projet, tout en n'utilisant qu'un seul endroit pour le dépôt de fichiers. L'archivage est possible grâce à une numérotation des versions des fichiers.

Dans Subversion, le numéro de version est le même pour l'ensemble des fichiers et dossiers du projet, contrairement à CVS où chaque fichier possède son propre numéro de version. Ainsi, si l'on modifie uniquement un fichier dans tout le projet, celui-ci voit son numéro de version incrémenté de 1 : il n'est pas rare de voir dans un projet des numéros de version très élevés. Cela peut paraître étrange au départ, mais est en fait très pratique : on numérote un instantané du projet à chaque modification et il est ainsi plus facile de revenir à une version antérieure du projet complet. D'autre part, il est possible de donner un numéro de version, appelé tag, correspondant à un état du projet que l'on veut figer. Exemple du projet Subversion : la version 1.1.3 de Subversion correspond a la révision 12740.

On verra dans un premier temps comment créer un dépôt et y rajouter du contenu. Ensuite, nous verrons comment récupérer des fichiers et soumettre des modifications au dépôt. Différentes personnes peuvent avoir accès simultanément au dépôt, que ce soit pour en lire le contenu ou y écrire des modifications (figure 1). On verra également comment se gère le cas où différents développeurs font des modifications sur le même fichier.

3 Utilisation de base de Subversion à travers un exemple

Pour la présentation de l'utilisation basique de Subversion, nous allons suivre la création étape par étape d'un projet composé de différents dossiers et fichiers. Dans cet exemple, on se trouve dans le cas d'utilisation le plus simple :

[1] pour une définition complète, consulter http://en.wikipedia.org/wiki/Version_control_system

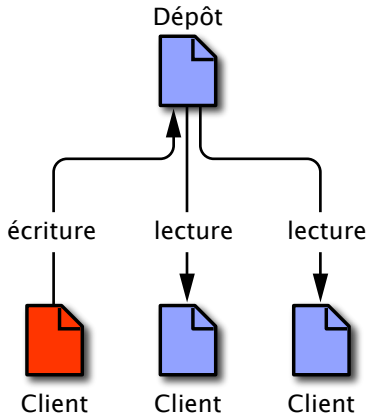

FIG. 1 – Subversion fonctionne sur un mode client/serveur. On peut voir ici que deux clients lisent un fichier du dépôt pendant qu'un autre soumet les modifications qu'il a faites à ce fichier.

- ▷ le dépôt Subversion est situé sur le disque local ; on verra par la suite comment se servir de Subversion lorsque le dépôt se trouve sur un serveur distant (rsh, ssh ou web) ;
- ▷ le développeur est seul à modifier les fichiers du projet ; on verra plus tard comment gérer les modifications opérées par plusieurs utilisateurs.

Sur la figure 2, nous avons représenté la chronologie du projet que nous allons gérer. Sous chaque flèche est représenté l'état de l'arborescence du projet correspondant à une version numérotée donnée dans la base Subversion. Les dossiers du projet sont nommés par une lettre majuscule (`A` et `B`), et les fichiers par une lettre minuscule (`a`, `b`, `c`, etc.). Les fichiers sont représentés avec des couleurs différentes : en bleu il s'agit du fichier original, c'est à dire celui qui apparaît dans le projet la première fois. A chaque fois que le contenu du fichier est modifié, sa couleur est changée : en rouge (fichier bleu modifié) puis en vert (fichier rouge modifié).

3.1 Création du dépôt – Version 0

Il ne s'agit pas vraiment de la version 0 ici, mais d'une étape préliminaire qui sert à initialiser le dépôt. Nous faisons le choix de stocker le dépôt dans le compte de l'utilisateur : ici il s'agit de `/Users/meister/depot`, mais il est bien sûr possible de le mettre n'importe où.

La commande de création de dépôt est `svnadmin create` :

```
$ svnadmin create /Users/meister/depot
$ ls -l /Users/meister/depot/
total 16
-rw-r--r--   1 meister  meister  379 22 Feb 10:50 README.txt
drwxr-xr-x   3 meister  meister  102 22 Feb 10:50 conf
drwxr-xr-x   2 meister  meister   68 22 Feb 10:50 dav
drwxr-sr-x  18 meister  meister  612 22 Feb 10:50 db
-r--r--r--   1 meister  meister    2 22 Feb 10:50 format
drwxr-xr-x   7 meister  meister  238 22 Feb 10:50 hooks
drwxr-xr-x   4 meister  meister  136 22 Feb 10:50 locks
```

On peut voir que le dépôt n'est pas vide : il est composé de différents dossiers et fichiers qui

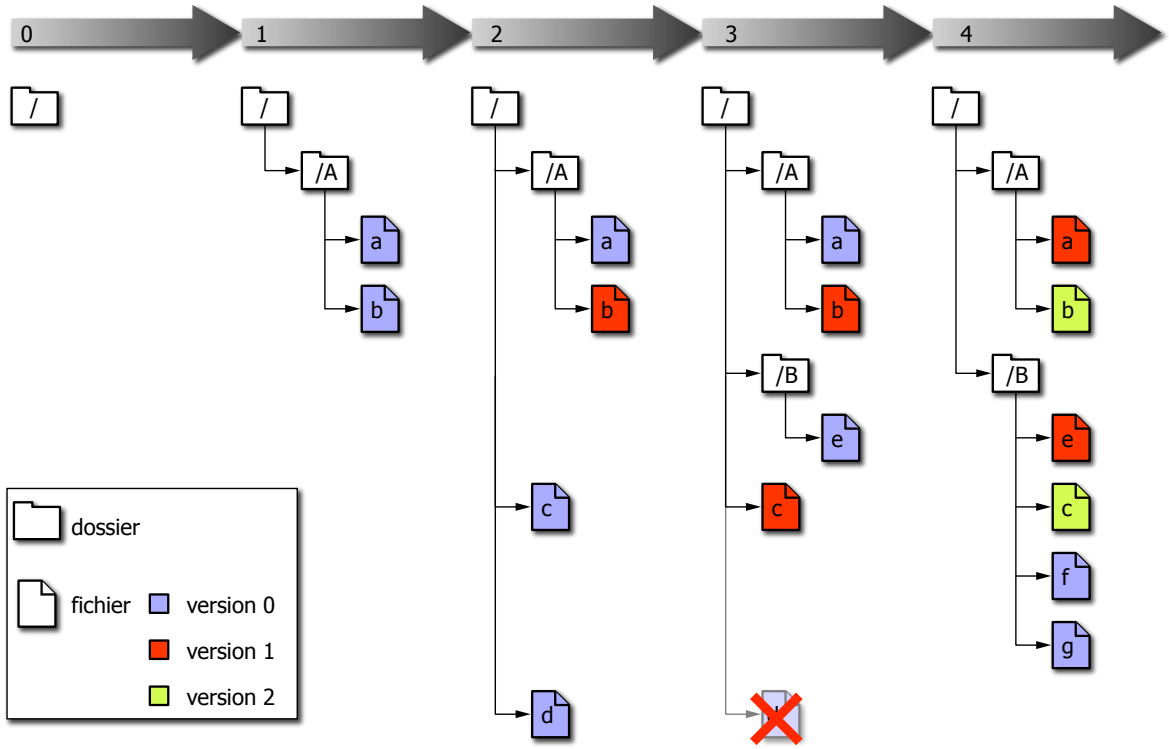

FIG. 2 – Chronologie du projet exemple étudié.

servent à Subversion pour la gestion du dépôt. En aucun cas l'utilisateur ne doit modifier le contenu du dépôt : ceci est précisé dans le fichier `README.txt` contenu dans la base :

```
$ more /Users/meister/depot/README.txt
This is a Subversion repository; use the 'svnadmin' tool to
   examine
it.  Do not add, delete, or modify files here unless you know how
to avoid corrupting the repository.

If the directory "db" contains a Berkeley DB environment,
you may need to tweak the values in "db/DB_CONFIG" to match the
requirements of your site.

Visit http://subversion.tigris.org/ for more information.
```

3.2 Import du projet dans la base Subversion – Version 1

Maintenant que le dépôt est créé nous allons y rajouter notre projet qui pour l'instant n'est constitué que d'un dossier (`A`) contenant deux fichiers (`a` et `b`), qui se trouvent dans le repertoire `/Users/meister/work/projet` :

```
$ cd /Users/meister/work/projet/
$ ls
A
```

```
$ ls A
a       b
```

Pour importer le dossier `projet` dans le dépôt il faut utiliser la commande `svn import` suivi du répertoire à importer et de l'adresse du dépôt :

```
$ svn import /Users/meister/work/projet/ file:///Users/meister/
   depot -m "import initial"
Adding          /Users/meister/work/projet/A
Adding          /Users/meister/work/projet/A/a
Adding          /Users/meister/work/projet/A/b

Committed revision 1.
```

A la fin de l'ajout il est marqué le numéro de version auquel se trouve le projet : `Committed revision 1.`

L'option `-m` permet de spécifier un message qui sert à identifier le numéro de révision. Si rien n'est précisé, Subversion lancera un éditeur de texte (souvent `vi` par défaut, mais cela est paramétrable) afin d'entrer le message. Enfin il est possible d'utiliser l'option `--file` pour spécifier le chemin d'un fichier qui servira de message : ceci peut être intéressant lorsque le message est long.

Maintenant que le projet a été importé, il est possible de l'effacer : tout est stocké dans la base Subversion dont le chemin est `/Users/meister/depot`. Dans notre cas cela n'a aucune importance mais nous allons le faire afin de voir comment récupérer un projet dans une base Subversion.

```
$ rm -rf /Users/meister/work/projet/
```

Pour récupérer la base, on utilise la commande `svn checkout` suivie du chemin du dépôt et optionnellement du chemin du répertoire dans lequel on veut créer la copie. Dans notre cas nous allons supposer que nous allons copier le projet dans le répertoire `/Users/meister/subversion_exemple/projet` :

```
$ cd
$ svn checkout file:///Users/meister/depot/ /Users/meister/
   subversion_exemple/projet
A  /Users/meister/subversion_exemple/projet/A
A  /Users/meister/subversion_exemple/projet/A/a
A  /Users/meister/subversion_exemple/projet/A/b
Checked out revision 1.
```

A la fin de la copie, on peut noter le message `Checked out revision 1` qui renseigne sur la version récupérée.

3.3 Modifications de fichiers et ajouts de fichiers – Version 2

Nous allons voir dans cette section comment soumettre des modifications de fichiers dans la base ainsi qu'y rajouter des éléments. Si l'on suit le programme de la figure 2, à la version 2, le fichier `b` est modifié et deux fichiers sont rajoutés (`c` et `d`).

Plaçons nous tout d'abord dans notre répertoire de travail :

```
$ cd /Users/meister/subversion_exemple/projet
```

Il est possible d'obtenir des informations sur ce répertoire grâce à la commande `svn info` :

```
$ svn info
Path: .
URL: file:///Users/meister/depot
Repository UUID: 2b48c083-b0f0-0310-a719-c870f2c2c034
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: meister
Last Changed Rev: 1
Last Changed Date: 2005-02-22 15:32:55 +0100 (Tue, 22 Feb 2005)
```

on peut y apprendre l'adresse du dépôt gérant ce projet, le numéro de version, l'auteur de la dernière modification, etc.

Supposons que nous ayons fait des modifications sur le fichier `a` et créé deux fichiers `c` et `d` à la racine du projet :

```
$ ls /Users/meister/subversion_exemple/projet/*
/Users/meister/subversion_exemple/projet/c      /Users/meister/
   subversion_exemple/projet/d

/Users/meister/subversion_exemple/projet/A:
a       b
```

La commande `svn status` permet de renseigner sur les modifications faites par rapport au dépôt :

```
$ svn status /Users/meister/subversion_exemple/projet/
?      /Users/meister/subversion_exemple/projet/c
?      /Users/meister/subversion_exemple/projet/d
M      /Users/meister/subversion_exemple/projet/A/b
```

Dans la première colonne on peut lire que les fichiers `c` et `d` ont un statut inconnu signalé par le caractère " `?`" et que le fichier `b` a été modifié signalé par le caractère "`M`".

Il faut tout d'abord ajouter les fichiers `c` et `d` à la base grâce à la commande `svn add` :

```
$ svn add c d
A         c
A         d
$ svn status
M      A/b
A      c
A      d
```

et la commande `svn status` nous signale maintenant que les fichiers `c` et `d` ont été rajoutés à la base (par la lettre `A` dans la première colonne).

Il suffit maintenant de soumettre les modifications et ajout à la base en utilisant la commande `svn commit` à la racine de notre projet :

```
$ svn commit -m "modification b+ajout c et d"
Sending        A/b
Adding         c
Adding         d
Transmitting file data ...
Committed revision 2.
```

On peut alors utiliser la commande `svn update` pour synchroniser nos informations locales avec celles contenues dans la base :

```
$ svn update
At revision 2.
```

ce que l'on peut vérifier par :

```
$ svn info
Path: .
URL: file:///Users/meister/depot
Repository UUID: 2b48c083-b0f0-0310-a719-c870f2c2c034
Revision: 2
Node Kind: directory
Schedule: normal
Last Changed Author: meister
Last Changed Rev: 2
Last Changed Date: 2005-02-22 17:45:46 +0100 (Tue, 22 Feb 2005)
```

3.4 Création de répertoires et effacement de fichiers – Version 3

Dans cette partie, nous allons voir que pour Subversion, le traitement des dossiers et des fichiers n'est pas différent. Nous allons rajouter un dossier à notre projet. Nous verrons également comment effacer un fichier ou dossier. Il est important de rappeler que ce type de modification majeure (effacement) n'est qu'une vue d'esprit puisque tout est conservé dans la base Subversion : c'est tout l'intérêt d'un gestionnaire de version !

Supposons que nous sommes toujours à la racine de notre projet dans notre répertoire de travail et que nous avons rajouté un dossier `B` contenant un fichier `e` et modifié le fichier `c`. Pour l'effacement du fichier `d`, nous devons utiliser la commande `svn delete` afin que cela soit pris en compte par la base :

```
$ svn delete d
D          d
$ ls
A        B        c
$
```

et ajouter le dossier `B` dans la base (les éléments qu'il contient sont automatiquement rajoutés également) :

```
$ svn add B
A          B
A          B/e
```

Nous pouvons regarder ce qui a été fait :

```
$ svn status
A       B
A       B/e
M       c
D       d
```

`B` et `B/e` sont à ajouter, `c` a été modifié et la lettre `D` en première colonne indique que `d` est un fichier à effacer.

De la même façon que précédemment, il ne nous reste plus qu'à faire :

```
$ svn commit -m "repertoire B rajoute dans le projet"
Adding          B
Adding          B/e
Sending         c
Deleting        d
Transmitting file data ..
Committed revision 3.
```

3.5 Déplacement de fichiers – Version 4

Nous n'allons préciser dans cette étape que le déplacement du fichier `c` dans le répertoire `B`, car nous avons déja vu comment faire les autres opérations nécessaires pour arriver à la version 4 de la figure 2. Ce déplacement se fait à l'aide de la commande `svn move` :

```
$ svn move c B/
A          B/c
D          c
```

on peut noter que pour Subversion il s'agit d'ajouter un fichier dans `B` et d'effacer l'autre, ce qui apparaît avec `status` sous forme d'un `+` en troisième colonne :

```
$ svn status
M       A/a
M       A/b
A  +    B/c
M       B/e
A       B/f
A       B/g
D       c
```

et on soumet les modifications :

```
$ svn commit -m "dernieres modifications de l'exemple"
Sending         A/a
Sending         A/b
Adding          B/c
Sending         B/e
Adding          B/f
Adding          B/g
```

```
Deleting       c
Transmitting file data .....
Committed revision 4.
```

3.6 Résumé d'un cycle de travail sous Subversion

Nous allons résumer ici les différentes commandes de base pour travailler avec Subversion sous forme du cycle de commande habituel :

1. Création d'un dépôt : `svnadmin create`
2. Copie d'un projet pour gestion dans ce dépôt : `svn import`
3. Récupération d'une copie de projet depuis le dépôt : `svn checkout`
   i. Modification du contenu du projet :
      ▷ de façon classique (éditeur de texte)
      ▷ rajout de fichiers : `svn add`
      ▷ copie de fichiers : `svn copy`
      ▷ déplacement de fichiers : `svn move`
      ▷ effacement de fichiers : `svn delete`
   ii. Consultation des modifications : `svn status`
   iii. Soumission des modifications : `svn commit`

3.7 Les logs

A chaque fois que nous avons soumis des modifications dans la base, nous les avons accompagnées d'un message par le biais de l'option -m de `svn commit`. L'ensemble de ces messages peut être consulté par le biais de l'instruction `svn log`, qui nous fournit en outre la date de chaque modification ainsi que son auteur :

```
$ svn log
------------------------------------------------------------------------
r3 | meister | 2005-02-23 11:04:20 +0100 (Wed, 23 Feb 2005) | 1
   line

modification de la definition du tableau xcol
------------------------------------------------------------------------
r2 | meister | 2005-02-23 11:02:39 +0100 (Wed, 23 Feb 2005) | 1
   line

changement des en-tetes
------------------------------------------------------------------------
r1 | meister | 2005-02-23 10:59:26 +0100 (Wed, 23 Feb 2005) | 1
   line

Import initial
------------------------------------------------------------------------
```

4 Utilisation de Subversion avec différents développeurs

Dans cette partie, nous allons traiter le cas où différents développeurs ont accès à la base Subversion : c'est le cas de la plupart des projets en équipe. Dans l'exemple précédent, tout allait bien car un seul développeur pouvait modifier la base. Mais en équipe, il sera fréquent que deux développeurs ou plus fassent des modifications sur le même fichier de façon simultanée. Il ne faut évidemment surtout pas que chacun puisse écraser les modifications des autres, ni que tous soient paralysés en attendant qu'un utilisateur ait fini de modifier un fichier et pouvoir à leur tour y apporter des modifications.

Que se passe-t-il alors ? Sur la figure 3 est représentée de façon schématique la gestion des conflits de fichiers. Quand un fichier est modifié par deux personnes différentes, le premier développeur à enregistrer ses modifications dans la base, change la version du fichier concerné. Lorsque le second développeur désire soumettre son fichier, la base l'avertit que le fichier a déjà été modifié et ne l'autorise pas à soumettre. Ce développeur peut alors récupérer le fichier de la base et le comparer au sien localement. Il réalise alors lui-même un fichier hybride qui mêle les développements des deux utilisateurs. Il soumet ce nouveau fichier à la base, qui incrémente le numéro de version de la base.

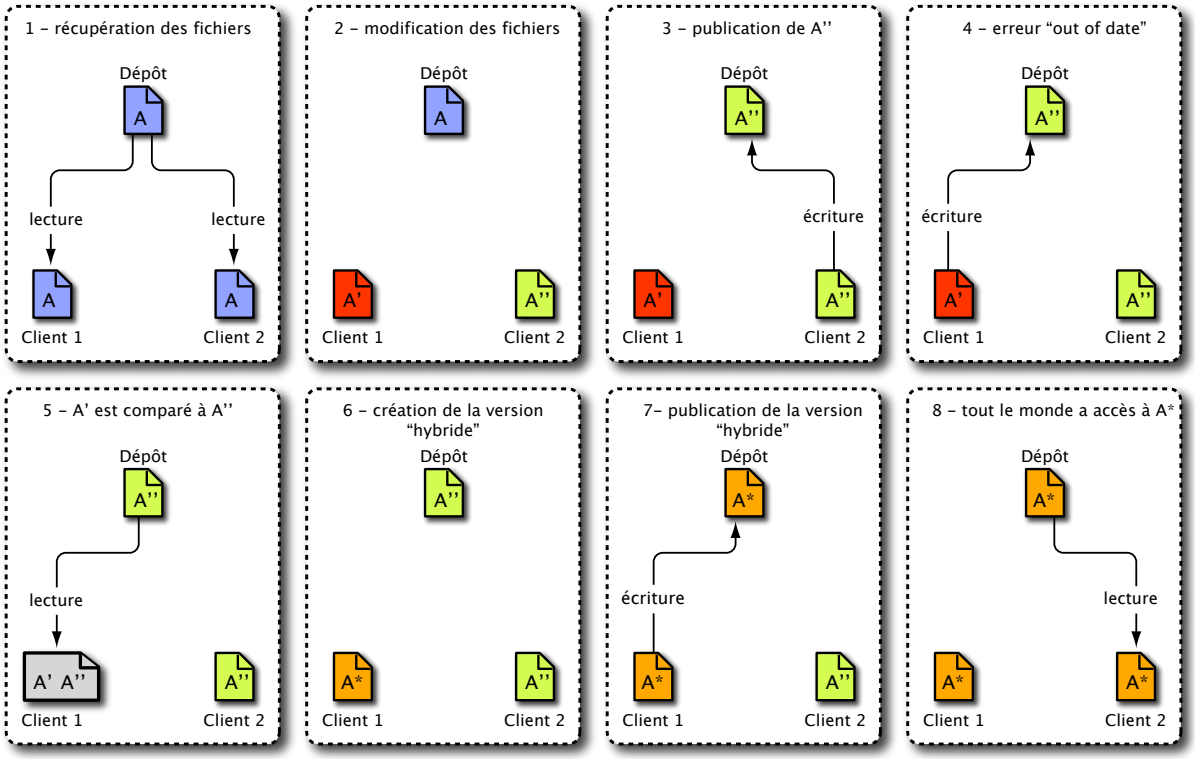

Fig. 3 – Schéma de gestion des conflits.

Dans ce type de conflit, rien ne remplace la communication entre les développeurs qui doivent s'assurer de la cohérence de leurs développements avec ceux des autres. Sur des projets avec de nombreux développeurs, les conflits sont gérés par des administrateurs qui sont chargés de prendre en compte les modifications de chacun. Cependant lorsque les équipes de travail ne sont pas trop grandes, un contact par téléphone ou par mail est suffisant pour résoudre la plupart des conflits rencontrés.

Nous allons supposer le cas de deux développeurs (`deleutre` et `meister`) travaillant sur une même base Subversion hébergée sur le serveur `codiciel4`. L'authentification sur ce serveur se fera avec `ssh` mais il existe d'autres possibiilités d'accès à un serveur Subversion. En particulier, on peut noter que nativement, Subversion peut fonctionner avec un serveur web Apache. En fonction du mode serveur avec lequel Subversion fonctionne, on utilisera pour le début du chemin du dépôt :

▷ `file://` : pour un dépôt sur un disque local ;

▷ `svn://` : pour un dépôt distant avec un serveur Subversion dont l'authentification est gérée par Subversion ;

▷ `svn+ssh://` : pour un dépôt distant avec un serveur Subversion dont l'authentification est gérée par ssh ;

▷ `http://` : pour un dépôt distant avec un serveur web ;

Dans le dépôt sur lequel `deleutre` et `meister` vont travailler il y a deux fichiers : `conflit_a_gerer.txt` et `conflit_automatise.txt`. Le contenu de ces fichiers est le suivant :

▷ `conflit_automatise.txt`

```
Ligne a modifier par deleutre :
Ligne a modifier par meister :
```

▷ `conflit_a_gerer.txt` :

```
cette ligne est modifiee par : <Nom developpeur>
Je suis deleutre (oui/non) :
Je suis meister (oui/non) :
```

4.1 Récupération de la base

Sur leurs comptes respectifs, `deleutre` et `meister` récupèrent le contenu de la base :

```
meister $ svn checkout svn+ssh://codiciel4/depot_exemple/projet
   work
meister@codiciel4's password:
A  work/conflit_a_gerer.txt
A  work/conflit_automatise.txt
Checked out revision 1.
```

et

```
deleutre $ svn checkout svn+ssh://codiciel4/depot_exemple/projet
   travail_subversion
deleutre@codiciel4's password:
A  travail_subversion/conflit_a_gerer.txt
A  travail_subversion/conflit_automatise.txt
Checked out revision 1.
```

4.2 Conflit géré par Subversion

En fait, si les modifications faites par les développeurs ne concernent pas les mêmes lignes, Subversion gère lui-même la fusion des différences. Le fichier `conflit_automatise.txt` modifié par :

▷ `deleutre` est maintenant :

```
Ligne commune
Ligne a modifier par deleutre : bonjour meister
Ligne a modifier par meister :
une autre ligne commune
```

▷ `meister` est maintenant :

```
Ligne commune
Ligne a modifier par deleutre :
Ligne a modifier par meister : salut deleutre
une autre ligne commune
```

On peut voir que dans ce cas aucune modification de l'un n'empiète sur celle de l'autre.

Supposons que `deleutre` soit le premier à soumettre ses modifications :

```
deleutre $ svn status
M       conflit_automatise.txt
deleutre $ svn commit -m "message pour meister"
deleutre@codiciel4's password:
Sending        conflit_automatise.txt
Transmitting file data .
Committed revision 2.
```

C'est maintenant le tour de `meister` :

```
meister $ svn status
M       conflit_automatise.txt
meister $ svn commit -m "message pour deleutre"
meister@codiciel4's password:
Sending        conflit_automatise.txt
Transmitting file data .svn: Commit failed (details follow):
svn: Out of date: 'conflit_automatise.txt' in transaction '3'
```

Le message `svn : Out of date : 'conflit_automatise.txt' in transaction '3'` indique à `meister` qu'il doit mettre son répertoire local à jour. Pour cela, il faut utiliser la commande `svn update` :

```
meister $ svn update
meister@codiciel4's password:
G  conflit_automatise.txt
Updated to revision 2.
```

La lettre `G` dans la première colonne indique que les modifications contenus dans la base n'empiètent pas sur celles du répertoire local. Il suffit de recommencer la soumission :

```
meister $ svn commit -m "message pour deleutre"
meister@codiciel4's password:
Sending        conflit_automatise.txt
Transmitting file data .
Committed revision 3.
```

Maintenant, chacun des développeurs peut remettre son répertoire local à jour et regarder le contenu du fichier `conflit_automatise.txt` :

```
meister $ svn update
meister@codiciel4's password:
At revision 3.
meister $ more conflit_automatise.txt
Ligne commune
Ligne a modifier par deleutre : bonjour meister
Ligne a modifier par meister : salut deleutre
une autre ligne commune
```

Toutes les modifications sont dans la base.

4.3 Conflit à gérer manuellement

Le fichier `conflit_a_gerer.txt` modifié par :

▷ `deleutre` est maintenant :

```
ligne commune
cette ligne est modifiee par : deleutre
Je suis deleutre (oui/non) : oui
Je suis meister (oui/non) : non
```

▷ `meister` est maintenant :

```
ligne commune
cette ligne est modifiee par : meister
Je suis deleutre (oui/non) : non
Je suis meister (oui/non) : oui
```

Toutes les modifications sont cette fois-ci incompatibles. `deleutre` soumet ses modifications :

```
deleutre $ svn commit -m "remplissage du formulaire"
deleutre@codiciel4's password:
Sending        conflit_a_gerer.txt
Transmitting file data .
Committed revision 4.
```

puis c'est au tour de `meister` :

```
meister $ svn commit -m "formulaire rempli"
meister@codiciel4's password:
Sending        conflit_a_gerer.txt
Transmitting file data .svn: Commit failed (details follow):
svn: Out of date: 'conflit_a_gerer.txt' in transaction '6'
```

Comme précédemment, on utilise `svn update` pour mettre à jour le contenu du répertoire de travail local :

```
meister $ svn update
meister@codiciel4's password:
```

```
C  conflit_a_gerer.txt
Updated to revision 4.
```

La lettre `C` dans la première colonne indique cette fois-ci un conflit à gérer manuellement. Lorsque ceci arrive sur un fichier `file`, Subversion modifie le répertoire local du développeur en créant différents fichiers supplémentaires :

1. `file` : contient maintenant l'ensemble des modifications :
2. `file.mine` : contient le fichier `file` modifié par l'utilisateur, avant que celui-ci ne mette à jour son répertoire ;
3. `file.r#old` (où `#old` est le numéro de l'ancienne révision) : c'est le fichier de la base avant que l'utilisateur ne fasse ses propres modifications ;
4. `file.r#new` (où `#new` est le numéro de la nouvelle révision) : c'est le fichier de la base le plus récent.

Dans notre cas :

```
meister $ ls -l
conflit_a_gerer.txt
conflit_a_gerer.txt.mine
conflit_a_gerer.txt.r3
conflit_a_gerer.txt.r4
conflit_automatise.txt

meister $ more conflit_a_gerer.txt
ligne commune
<<<<<<< .mine
cette ligne est modifiee par : meister
Je suis deleutre (oui/non) : non
Je suis meister (oui/non) : oui
=======
cette ligne est modifiee par : deleutre
Je suis deleutre (oui/non) : oui
Je suis meister (oui/non) : non
>>>>>>> .r4

meister $ more conflit_a_gerer.txt.mine
ligne commune
cette ligne est modifiee par : meister
Je suis deleutre (oui/non) : non
Je suis meister (oui/non) : oui

meister $ more conflit_a_gerer.txt.r3
ligne commune
cette ligne est modifiee par : <Nom developpeur>
Je suis deleutre (oui/non) :
Je suis meister (oui/non) :

meister $ more conflit_a_gerer.txt.r4
ligne commune
cette ligne est modifiee par : deleutre
```

```
Je suis deleutre (oui/non) : oui
Je suis meister (oui/non) : non
```

Le développeur a alors trois choix possibles :

1. gérer le conflit manuellement ;
2. écraser le fichier du répertoire par l'une de ces variantes temporaires ;
3. taper la commande `svn revert` suivi du nom du fichier pour éliminer toute modification locale.

Nous allons ici gérer le conflit manuellement. `meister` change le contenu du fichier en :

```
meister $ more conflit_a_gerer.txt
ligne commune
cette ligne est modifiee par : deleutre puis meister
Je suis deleutre (oui/non) : non
Je suis meister (oui/non) : oui
```

Il faut ensuite indiquer à Subversion que le conflit a été résolu avec la commande `svn resolved` suivie du nom du fichier concerné. Cette commande efface également les fichiers temporaires. Ensuite on peut soumettre les modifications classiquement.

```
meister $ svn resolved conflit_a_gerer.txt
Resolved conflicted state of 'conflit_a_gerer.txt'
meister $ ls -l
conflit_a_gerer.txt
conflit_automatise.txt
meister $ svn commit -m "meister a le dernier mot"
meister@codiciel4's password:
Sending        conflit_a_gerer.txt
Transmitting file data .
Committed revision 5.
```

5 Comment utiliser Subversion pour suivre l'historique

Nous avons jusqu'ici vu comment il était possible d'utiliser Subversion pour archiver l'ensemble des modifications faites à un projet. Comme tout gestionnaire de versions, il est aussi possible de revoir des versions plus anciennes, de reprendre des fichiers de versions différentes et d'en faire une nouvelle version de projet, de comparer des versions successives de fichiers entre elles, etc. Nous allons voir ces différents aspects dans cette partie.

5.1 `svn log`

La commande `svn log` permet de suivre les modifications faites à chaque nouvelle version. Exemple pour l'exemple précédent :

```
[508 codiciel4:meister]$ svn log
meister@codiciel4's password:
------------------------------------------------------------
```

```
r5 | meister | 2005-02-23 17:53:04 +0100 (Wed, 23 Feb 2005) | 1
   line

meister a le dernier mot
------------------------------------------------------------
r4 | deleutre | 2005-02-23 17:27:34 +0100 (Wed, 23 Feb 2005) | 1
   line

remplissage du formulaire
------------------------------------------------------------
r3 | meister | 2005-02-23 17:10:01 +0100 (Wed, 23 Feb 2005) | 1
   line

message pour deleutre
------------------------------------------------------------
r2 | deleutre | 2005-02-23 17:01:34 +0100 (Wed, 23 Feb 2005) | 1
   line

message pour meister
------------------------------------------------------------
r1 | meister | 2005-02-23 16:52:48 +0100 (Wed, 23 Feb 2005) | 1
   line

import initial des fichiers exemples
------------------------------------------------------------
```

L'option -v détaille plus précisément les changements de la base pour chacune des versions :

```
[509 codiciel4:meister]$ svn log -v
meister@codiciel4's password:
------------------------------------------------------------
r5 | meister | 2005-02-23 17:53:04 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   M /conflit_a_gerer.txt

meister a le dernier mot
------------------------------------------------------------
r4 | deleutre | 2005-02-23 17:27:34 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   M /conflit_a_gerer.txt

remplissage du formulaire
------------------------------------------------------------
r3 | meister | 2005-02-23 17:10:01 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   M /conflit_automatise.txt

```

```
message pour deleutre
------------------------------------------------------------------------
r2 | deleutre | 2005-02-23 17:01:34 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   M /conflit_automatise.txt

message pour meister
------------------------------------------------------------------------
r1 | meister | 2005-02-23 16:52:48 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   A /conflit_a_gerer.txt
   A /conflit_automatise.txt

import initial des fichiers exemples
------------------------------------------------------------------------
```

Enfin, il est possible de préciser avec l'option `-r` un seul numéro de révision, ou une plage de révisions si l'on donne deux numéros de révision séparés par le caractère " :" :

```
[504 codiciel4:meister]$ svn log -r 3:5 -v
meister@codiciel4's password:
------------------------------------------------------------------------
r3 | meister | 2005-02-23 17:10:01 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   M /conflit_automatise.txt

message pour deleutre
------------------------------------------------------------------------
r4 | deleutre | 2005-02-23 17:27:34 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   M /conflit_a_gerer.txt

remplissage du formulaire
------------------------------------------------------------------------
r5 | meister | 2005-02-23 17:53:04 +0100 (Wed, 23 Feb 2005) | 1
   line
Changed paths:
   M /conflit_a_gerer.txt

meister a le dernier mot
------------------------------------------------------------------------
```

5.2 `svn diff`

La commande `svn diff` permet de regarder les différences (comme le donne `diff`) entre deux révisions données.

Avant de soumettre les modifications à la base Subversion, on peut utiliser `svn diff` pour comparer les fichiers modifiés localement à ceux de la base. Exemple, si `meister` a rajouté une modification à un fichier de l'exemple précédent :

```
[518 codiciel4:meister]$ svn status
M       conflit_a_gerer.txt
[519 codiciel4:meister]$ svn diff
Index: conflit_a_gerer.txt
===================================================================
--- conflit_a_gerer.txt (revision 5)
+++ conflit_a_gerer.txt (working copy)
@@ -2,3 +2,4 @@
 cette ligne est modifiee par : deleutre puis meister
 Je suis deleutre (oui/non) : non
 Je suis meister (oui/non) : oui
+rajoutee pour exemple avec diff
```

Ceci ne fonctionne que si les fichiers n'ont pas encore été soumis.

Il est également possible de comparer les fichiers locaux à ceux d'une version donnée en utilisant l'option `-r` :

```
[527 codiciel4:meister]$ svn diff -r 1
meister@codiciel4's password:
meister@codiciel4's password:
Index: conflit_a_gerer.txt
===================================================================
--- conflit_a_gerer.txt (revision 1)
+++ conflit_a_gerer.txt (working copy)
@@ -1,4 +1,4 @@
 ligne commune
-cette ligne est modifiee par : <Nom developpeur>
-Je suis deleutre (oui/non) :
-Je suis meister (oui/non) :
\ No newline at end of file
+cette ligne est modifiee par : deleutre puis meister
+Je suis deleutre (oui/non) : non
+Je suis meister (oui/non) : oui
Index: conflit_automatise.txt
===================================================================
--- conflit_automatise.txt      (revision 1)
+++ conflit_automatise.txt      (working copy)
@@ -1,4 +1,4 @@
 Ligne commune
-Ligne a modifier par deleutre :
-Ligne a modifier par meister :
+Ligne a modifier par deleutre : bonjour meister
```

```
+Ligne a modifier par meister : salut deleutre
 une autre ligne commune
```

Et on peut préciser un fichier en particulier :

```
[528 codiciel4:meister]$ svn diff -r 3 conflit_a_gerer.txt
meister@codiciel4's password:
meister@codiciel4's password:
Index: conflit_a_gerer.txt
===================================================================
--- conflit_a_gerer.txt (revision 3)
+++ conflit_a_gerer.txt (working copy)
@@ -1,4 +1,4 @@
 ligne commune
-cette ligne est modifiee par : <Nom developpeur>
-Je suis deleutre (oui/non) :
-Je suis meister (oui/non) :
\ No newline at end of file
+cette ligne est modifiee par : deleutre puis meister
+Je suis deleutre (oui/non) : non
+Je suis meister (oui/non) : oui
```

Enfin il est possible de comparer deux versions en séparant leurs numéros par le caractère ": " :

```
[529 codiciel4:meister]$ svn diff -r 2:5
meister@codiciel4's password:
Index: conflit_a_gerer.txt
===================================================================
--- conflit_a_gerer.txt (revision 2)
+++ conflit_a_gerer.txt (revision 5)
@@ -1,4 +1,4 @@
 ligne commune
-cette ligne est modifiee par : <Nom developpeur>
-Je suis deleutre (oui/non) :
-Je suis meister (oui/non) :
\ No newline at end of file
+cette ligne est modifiee par : deleutre puis meister
+Je suis deleutre (oui/non) : non
+Je suis meister (oui/non) : oui
Index: conflit_automatise.txt
===================================================================
--- conflit_automatise.txt      (revision 2)
+++ conflit_automatise.txt      (revision 5)
@@ -1,4 +1,4 @@
 Ligne commune
 Ligne a modifier par deleutre : bonjour meister
-Ligne a modifier par meister :
+Ligne a modifier par meister : salut deleutre
 une autre ligne commune
```

5.3 svn update

La commande `svn update` permet de récupérer les fichiers d'une révision donnée en utilisant l'option `-r` :

```
[533 codiciel4:meister]$ svn update -r 3
meister@codiciel4's password:
At revision 3.
[~/work]
[534 codiciel4:meister]$ svn info
Path: .
URL: svn+ssh://codiciel4/depot_exemple/projet
Repository UUID: ce7af9ab-c9f0-0310-bae6-926fc795d926
Revision: 3
Node Kind: directory
Schedule: normal
Last Changed Author: meister
Last Changed Rev: 3
Last Changed Date: 2005-02-23 17:10:01 +0100 (Wed, 23 Feb 2005)
```

Si l'on ne précise rien, `svn update` récupère la version la plus récente de la base.

5.4 svn cat

La commande `svn cat` sert à examiner le contenu d'un fichier à une révision que l'on précise avec l'option `-r` :

```
[539 codiciel4:meister]$ svn cat -r 2 conflit_automatise.txt
meister@codiciel4's password:
meister@codiciel4's password:
Ligne commune
Ligne a modifier par deleutre : bonjour meister
Ligne a modifier par meister :
une autre ligne commune
```

Ceci peut permettre par exemple de récupérer un fichier à une révision donnée dans un fichier local afin d'utiliser un outil de comparaison graphique plus pratique que l'austère `svn diff` :

```
[540 codiciel4:meister]$ svn cat -r 2 conflit_automatise.txt >
   conflit_automatise.txt.r2
meister@codiciel4's password:
[541 codiciel4:meister]$ ls -l
conflit_a_gerer.txt
conflit_automatise.txt
conflit_automatise.txt.r2
```

5.5 svn revert

La commande `svn revert` permet d'annuler les modifications faites dans un fichier localement et de récupérer sa version la plus récente dans la base :

```
[544 codiciel4:meister]$ svn status
M       conflit_automatise.txt
[545 codiciel4:meister]$ svn revert conflit_automatise.txt
Reverted 'conflit_automatise.txt'
[546 codiciel4:meister]$ svn status
```

5.6 svn list

Enfin, la commande `svn list` permet de lister les fichiers d'un dépôt. En utilisant l'option `-v`, on obtient un résultat plus détaillé :

```
[550 codiciel4:meister]$ svn list -v
meister@codiciel4's password:
      5 meister          132 Feb 23 17:53 conflit_a_gerer.txt
      3 meister          133 Feb 23 17:10 conflit_automatise.txt
```

La première colonne donne le numéro de révision à laquelle a eu lieu la dernière modification, ensuite on peut lire le nom du développeur qui a fait cette modification et ensuite la taille du fichier et la date du dépôt dans la base.

6 Conclusion

Nous n'avons vu que les aspects de base de Subversion, mais son utilisation peut se limiter aux commandes que nous avons présentées. En revanche pour tout ce qui concerne l'administration d'un dépôt, pour la gestion d'un serveur dédié à Subversion ou pour l'utilisateur désient tirer parti de toute la puissance de Subversion, on consultera la documentation (très abordable) disponible sur le site http://subversion.tigris.org/.

La liste des commandes disponibles est donnée par `svn help` :

```
[553 codiciel4:meister]$ svn help
usage: svn <subcommand> [options] [args]
Type "svn help <subcommand>" for help on a specific subcommand.

Most subcommands take file and/or directory arguments, recursing
on the directories.  If no arguments are supplied to such a
command, it will recurse on the current directory (inclusive) by
default.

Available subcommands:
   add
   blame (praise, annotate, ann)
   cat
   checkout (co)
   cleanup
   commit (ci)
   copy (cp)
   delete (del, remove, rm)
   diff (di)
```

```
   export
   help (?, h)
   import
   info
   list (ls)
   log
   merge
   mkdir
   move (mv, rename, ren)
   propdel (pdel, pd)
   propedit (pedit, pe)
   propget (pget, pg)
   proplist (plist, pl)
   propset (pset, ps)
   resolved
   revert
   status (stat, st)
   switch (sw)
   update (up)

Subversion is a tool for version control.
For additional information, see http://subversion.tigris.org/
```

et de l'aide peut être obtenue sur une option en tapant `svn help <commande>` :

```
[554 codiciel4:meister]$ svn help list
list (ls): List directory entries in the repository.
usage: list [TARGET...]

  List each TARGET file and the contents of each TARGET directory as
  they exist in the repository.  If TARGET is a working copy path, the
  corresponding repository URL will be used.

  The default TARGET is '.', meaning the repository URL of the current
  working directory.

  With --verbose, the following fields show the status of the item:

    Revision number of the last commit
    Author of the last commit
    Size (in bytes)
    Date and time of the last commit

Valid options:
  -r [--revision] arg      : ARG (some commands also take ARG1:
     ARG2 range)
```

```
                                A revision argument can be one of:
                                   NUMBER       revision number
                                   "{" DATE "}" revision at start of
                                       the date
                                   "HEAD"       latest in repository
                                   "BASE"       base rev of item's
                                      working copy
                                   "COMMITTED"  last commit at or
                                      before BASE
                                   "PREV"       revision just before
                                       COMMITTED
  -v [--verbose]             : print extra information
  -R [--recursive]           : descend recursively
  --username arg             : specify a username ARG
  --password arg             : specify a password ARG
  --no-auth-cache            : do not cache authentication tokens
  --non-interactive          : do no interactive prompting
  --config-dir arg           : read user configuration files from
     directory ARG
```

Enfin, nous donnons ici un tableau récapitulatif des différentes commandes les plus courantes :

`svnadmin create`	créer une base
`svn checkout`	récupérer le contenu de la base
`svn commit`	soumettre des modifications à la base
`svn import`	importer un projet dans la base
`svn resolved`	indiquer que les conflits ont été résolus
`svn revert`	annuler toute modification locale
`svn update`	mettre à jour le répertoire local
`svn cat`	lire le contenu d'un fichier de la base
`svn diff`	regarder les différences entre des versions de la base
`svn info`	obtenir des infos sur le réperoire local
`svn list`	lister le contenu de la base
`svn log`	voir les messages accompagnant chaque révision de la base
`svn status`	afficher l'état des fichiers/dossiers du répertoire local
`svn add`	ajouter un fichier/dossier dans l'arborescence de la base
`svn copy`	copier des fichiers/dossiers dans l'arborescence de la base
`svn delete`	supprimer des fichiers/dossiers de l'arborescence de la base
`svn mkdir`	créer un dossier dans l'arborescence de la base
`svn move`	déplacer des fichiers/dossiers dans l'arborescence de la base

Tab. 1 – Liste des commandes usuelles de Subversion classées par *genre*.