Guide d'utilisation de Subversion

D. Renault

20 septembre 2011

Résumé

Le but de ce document est de donner une introduction sur Subversion, un système de gestion de version. Il définit les notions liées au contrôle de version, puis détaille l'utilisation de ce programme pour gérer un projet.

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

La réalisation d'un projet basé sur un ensemble de fichiers source, qu'il s'agisse du code d'un programme en langage `C` ou du source d'un rapport rédigé en LaTeX, passe inévitablement par de nombreuses versions successives. Une version du projet, appelée aussi *révision*, représente l'ensemble des fichiers appartenant au projet à un moment fixé de son développement, telle une photographie de son état instantané. Le projet évolue ainsi à partir de sa révision initiale vers une révision propre à sa mise en production (*release*).

Un système de gestion de version, en anglais *version control system* (VCS) ou *source code management* (SCM) est un ensemble d'outils permettant de stocker et de manipuler un code source ainsi que toutes les révisions qu'il a subies depuis sa création. Un tel système permet de gérer le code à la fois :

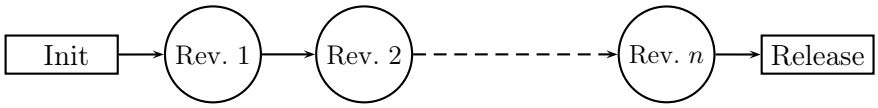
- à travers le temps (possibilité d'archiver des versions du code et d'inverser (*reversion*) des modifications) ;
- à travers l'espace (possibilité de faire partir le code dans plusieurs directions séparées (*branching*), et éventuellement de les rassembler (*merging*)).

Le fait de pouvoir accéder facilement à l'ensemble des versions d'un code permet ainsi de faciliter certaines phases de son développement. Cependant, la propriété fondamentale d'un tel système est le fait qu'il permette de *travailler à plusieurs* sur le code *de manière concurrente* dans de bonnes conditions : chaque programmeur peut travailler à partir d'une version du code, en développer une nouvelle version indépendante, et/ou décider de la rassembler avec la révision d'un autre programmeur, en *gérant les conflits*.

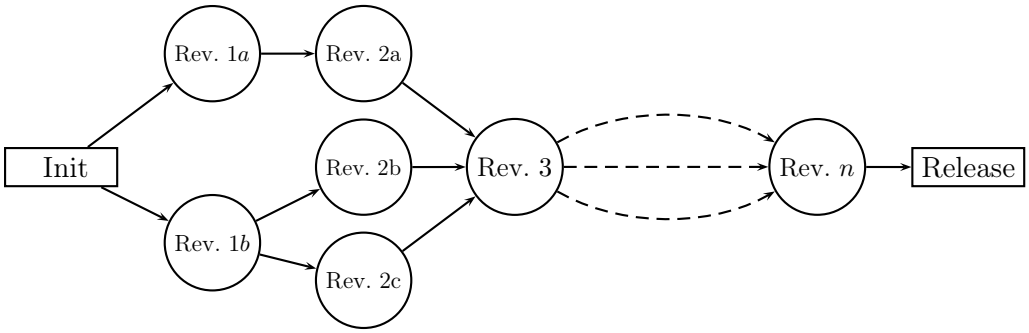
Ce document explique les concepts généraux et les modalités de fonctionnement d'un système de gestion de version particulier nommé Subversion (et parfois abrégé en `svn`), notamment utilisé à l'ENSEIRB pour la gestion des projets en 1ère année.

1 Processus de travail

Lorsque l'on travaille seul sur un projet sans utiliser de système de gestion de version, le processus de travail (*workflow*) consiste généralement à créer un répertoire de travail, puis à modifier graduellement les données dans ce répertoire en faisant des modifications successives, jusqu'à obtenir (dans le meilleur des cas) une version finale (*release*).



Lorsque l'on travaille à plusieurs sur le même projet, ce workflow atteint ses limites. En effet, à partir d'une révision donnée, plusieurs personnes peuvent travailler et créer des révisions différentes. L'ensemble des révisions prend alors une forme plus complexe :



Plusieurs questions viennent à se poser :

- Problème n°1 : Quelle est la révision courante (de référence) des sources du projet ? Cette question se pose naturellement pour un développeur extérieur au projet qui désire contribuer, ou pour décider de la révision devant être communiquée au client lors de la release. Une façon de répondre à ce problème consiste à désigner une suite de révisions particulière pour être la version de référence. Concrètement, il peut s'agir d'un répertoire particulier désigné au départ comme le répertoire devant contenir toutes les révisions de référence. Toute révision extérieure à ce répertoire devra alors être fusionnée (*merging*) avec les données de référence. Mais alors :
- Problème n°2 : Comment gérer la fusion de deux révisions distinctes ?
  Ce problème est plus épineux, dans la mesure où deux versions des données peuvent avoir des différences non triviales menant à des *conflits*. Il devient alors nécessaire aux responsables du projet de gérer ces conflits à la main, ce qui consiste à construire une révision des données à partir de deux révisions distinctes et qui prennent au mieux en compte les modifications apparaissant dans ces révisions.

L'utilisation d'un système de gestion de version comme Subversion va permettre de fournir une réponse à ces problèmes.

2 Fonctionnement de Subversion

Subversion (`http://subversion.tigris.org`) est un système de gestion de version open-source, développé à partir de 2000 pour remplacer le vieillissant CVS (Concurrent Version System, `http://www.nongnu.org/cvs`, débuté à la fin des années 80), qui héritait lui-même de RCS (Revision Control System, `http://www.gnu.org/software/rcs`).

2.1 Dépôts

Subversion est un système *centralisé* : les données sont rangées dans un espace de stockage particulier appelé *dépôt* (*repository*). Le dépôt prend la forme d'un répertoire ayant la structure suivante :

README.txt	
conf/	Fichiers de configuration
dav/	Répertoire dédié au serveur web (Apache)
db/	Emplacement des données
format	Numéro de révision courant
hooks/	Scripts personnalisés
locks/	Informations concernant les fichiers vérouillés

L'existence du dépôt est une réponse à l'un des problèmes évoqués au paragraphe précédent : le dépôt sert de référence pour les données, ce qui assure que cette référence soit unique. L'accès aux données se fait uniquement à travers le dépôt, à la manière d'une architecture client/serveur : toute personne (le client) participant au projet peut récupérer une révision des données à partir du dépôt (le serveur), les modifier localement, pour enfin transmettre une nouvelle révision au dépôt, afin que celui-ci puisse les redistribuer.

Les données sont stockées dans le dépôt à l'intérieur d'un système de fichiers particulier appelé FSFS, et stocké dans le répertoire `db/`. Ce système de fichiers diffère des systèmes de fichiers usuels (NTFS, Ext3 ...) dans la mesure où il stocke à la fois les données et les différentes révisions qu'elles ont subies depuis la création du dépôt.

En pratique, pour chaque révision des données, Subversion crée un fichier *delta*, c'est-à-dire un fichier contenant toutes les modifications faites aux données à l'intérieur du dépôt par rapport à la révision précédente. Ce fichier delta est ensuite stocké dans un des sous-répertoires du répertoire `db/`, à l'intérieur du dépôt.

Ainsi, Subversion permet un stockage efficace des données : le poids d'une révision ne dépend que des modifications faites lors de cette révision. Par exemple, les déplacements, renommages ou suppressions de fichiers ont un poids négligeable.

La création d'un dépôt se fait avec la commande suivante : `svnadmin create ⟨nom_depot⟩` où ⟨nom_depot⟩ représente le nom du répertoire dans lequel sera stocké le dépôt. En général, les dépôts sont stockés sur des serveurs accessibles par Internet, comme `sourceforge.net`, mais il est aussi possible de créer des dépôts sur un serveur de fichier local.

2.2 Répertoire de travail

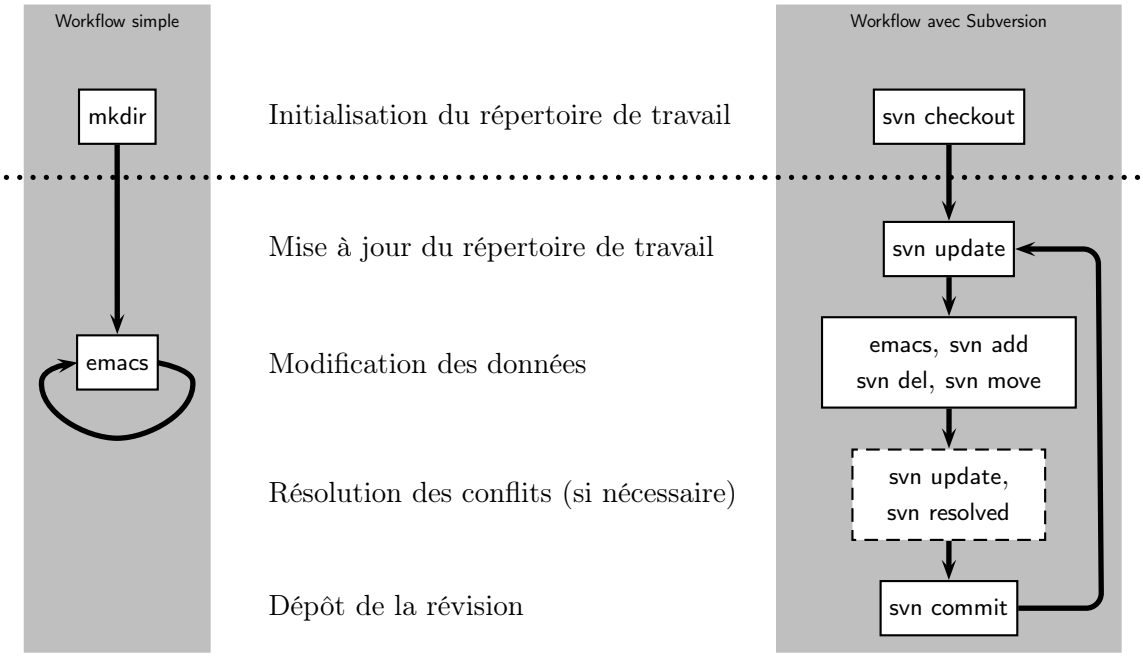
Les personnes utilisant Subversion ont rarement besoin de naviguer à l'intérieur du dépôt pour en examiner les fichiers, pour la simple raison que les données ne sont pas directement lisibles à l'intérieur du dépôt. Pour travailler sur les données, on utilise un répertoire de travail (*workspace* ou *working copy*) dans lequel les données sont accessibles sous la forme d'une arborescence de fichiers.

La création d'un répertoire de travail se fait généralement en effectuant un retrait (*checkout*) à partir d'un dépôt existant. Par exemple, si le dépôt est accessible dans le système de fichiers courant, il faut utiliser la commande `svn checkout file ://⟨chemin⟩ ⟨nom_rep⟩` où ⟨chemin⟩ est le chemin d'accès complet au dépôt et ⟨nom_rep⟩ représente le nom du répertoire de travail (cf. paragraphe 3 pour plus d'informations).

Le répertoire obtenu à partir d'un dépôt nouvellement créé est initialement vide, à l'exception d'un répertoire .svn/ contenant des données "administratives" sur le répertoire, comme le nom du dépôt lié à ce répertoire de travail ou les numéros de révision des fichiers dans le répertoire. Par la suite, chaque sous-répertoire du répertoire de travail contiendra un répertoire .svn/, ce qui en fait un moyen sûr de reconnaître un répertoire de travail Subversion.

2.3 Travailler en utilisant Subversion

Comment travaille-t'on concrètement avec Subversion ? Le *workflow* est présenté dans le diagramme suivant et mis face à face au même workflow sans système de gestion de version.



Globalement, le processus de travail, selon que l'on travaille avec ou sans gestionnaire de version, est le même : il faut d'abord initialiser un répertoire de travail, puis créer des révisions successives par modification des données. Néanmoins, il existe plusieurs subtilités. Premièrement, avec Subversion, il faut communiquer avec le dépôt à chaque fois que l'on désire obtenir ou transmettre des révisions des données. Ensuite, au moment de transmettre des données, il est possible qu'une personne ait déjà modifié la même partie du code que celle que l'on désire communiquer. Il faut alors fusionner ses modifications avec la révision courante existant dans le dépôt.

D'où deux règles de bonne conduite pour l'utilisation de Subversion :

- Règle n°1 : Il faut systématiquement mettre à jour le répertoire de travail, en particulier avant toute communication avec le dépôt.

  Puisque le dépôt contient la version de référence des sources, il est nécessaire de se mettre à jour *avant* de modifier les sources, mais aussi *avant* de les transmettre afin de s'assurer que l'on n'a pas fait de modification en même temps qu'une autre personne. Cela permet de minimiser les risques de conflits

- Règle n°2 : Deux personnes doivent (autant que possible) travailler sur des fichiers différents.

  Avec Subversion, Les conflits n'apparaissent que lorsque deux personnes modifient une même zone de code dans le même fichier au même moment. Toutes les modifications dans des endroits distincts du répertoire de travail (fichiers séparés / zones du code clairement éloignées dans un même fichier) sont gérées automatiquement par Subversion. Pour éviter les conflits, il suffit donc de ne pas travailler en même temps sur une même zone de code.

Dans les paragraphes suivants sont décrites les principales commandes offertes par Subversion, dans le cadre d'une utilisation raisonnable du système de gestion de version.

3 Échanges avec le dépôt

• Retrait du dépôt : `svn checkout` , `svn co`

Permet de créer un répertoire de travail à partir de l'adresse d'un dépôt donnée sous la forme d'une URL.

svn checkout ⟨protocole⟩ ://⟨chemin⟩ ⟨nom_rep⟩

```
% svn co https://thor.enseirb.fr/depot local_dir
```

⟨protocole⟩	Peut prendre la valeur file pour un dépôt local, http pour un dépôt accessible par WebDAV [1], svn ou svn+ssh pour un dépôt accessible à travers un serveur Subversion.
⟨chemin⟩	Le chemin complet d'accès au dépôt. Par chemin complet, on entend l'URL dans le cas d'un serveur distant et le chemin absolu pour des ressources locales.
⟨nom_rep⟩	Le nom du répertoire de travail.

• Mise à jour du répertoire de travail : `svn update` , `svn up`

Permet d'amener un répertoire de travail à la révision actuelle dans le dépôt. Si le répertoire de travail contient des modifications par rapport à la révision qu'il est supposé contenir, cette opération peut mener à des conflits.

svn update
svn update -r ⟨revision⟩

```
% svn update
U    code.c
Updated to revision 3.
```

⟨revision⟩	Le numéro de révision que l'on veut atteindre. Sans cet argument, Subversion mettra à jour le répertoire à la révision la plus récente.

1. Web Distributed Authoring and Versioning, un protocole utilisé pour s'authentifier sur un serveur distant.

• Communication d'une révision au dépôt : `svn commit` , `svn ci`

Permet de créer une nouvelle révision à l'intérieur du dépôt. De manière semi-automatique, Subversion encourage à associer un commentaire lors de tout ajout d'une révision dans le dépôt. Si l'on ne passe pas de commentaire en paramètre à la commande, Subversion va exécuter un éditeur de texte dans lequel on pourra éditer ledit commentaire.

svn commit
svn commit -m ⟨commentaire⟩

```
% svn commit
/* Ecriture du commentaire avec emacs */
Sending        code.c
Transmitting file data .
Committed revision 4.
```

⟨commentaire⟩	Le commentaire lié à cette nouvelle révision.

Usuellement, le commentaire décrit les modifications apportées par la révision. Il décrit les tests réalisés pour assurer un fonctionnement correct du code, et il donne des informations sur l'état du code après la révision (Le programme compile t'il ? Génère t'il des erreurs ?)

• Import de données dans le dépôt : `svn import`

Permet d'ajouter directement toute une arborescence à l'intérieur d'un dépôt. Cette commande prend son utilité lorsque l'on dispose d'un ensemble de sources que l'on désire intégrer à l'intérieur du dépôt Subversion.

svn import ⟨nom_rep⟩ ⟨protocole⟩ ://⟨chemin⟩ -m ⟨commentaire⟩

```
% svn import local_dir https://thor.enseirb.fr/depot/local
```

⟨protocole⟩ ⟨chemin⟩	} Se référer à la commande svn checkout
⟨nom_rep⟩	Le nom du répertoire local que l'on désire importer.
⟨commentaire⟩	Le commentaire associé à cet import de données.

Concrètement, cette commande correspond à un commit de l'ensemble des fichiers dans le répertoire ⟨nom_rep⟩, et il convient de lui faire correspondre un commentaire décrivant cet ajout massif de fichiers.

• Extraction du dépôt sans les fichiers liés à Subversion : `svn export`

Permet d'obtenir un répertoire contenant la dernière révision des sources, mais ne contenant aucune information relative au système de gestion de version (en particulier, aucun répertoire .svn). Cette commande trouve son utilité lorsque l'on désire produire une *release*, une archive destinée à un public extérieur à l'équipe de développement.

svn export ⟨nom_source⟩ ⟨nom_dest⟩

```
% svn export local_dir release_dir
```

4 Informations sur le dépôt

• Etat courant d'un fichier dans le répertoire de travail : `svn status`, `svn st`

Permet d'obtenir des informations concernant les fichiers décrivant leur raison d'être sous contrôle de version. Cette information est codée sous la forme d'un caractère, suivi du nom du fichier.

svn status ⟨fichier⟩

```
% svn status
    A     code.c
    U     type.c
    U     Makefile
```

La liste des codes utilisés par Subversion est :

'␣'	La version du fichier correspond à la révision courante.
'?'	Le fichier n'est pas contrôlé par Subversion.
'A'	Le fichier va être ajouté lors du prochain dépôt.
'C'	Le fichier est en conflit avec la version de référence.
'D'	Le fichier va être supprimé lors du prochain dépôt.
'I'	Subversion est configuré pour ignorer ce fichier..
'M'	Le fichier a été modifié par rapport à la dernière mise à jour.
'R'	Le fichier va être remplacé lors du prochain dépôt.

Les caractères indiquant le statut d'un fichier peuvent aussi apparaître lors de l'utilisation d'autres commandes de Subversion. C'est en particulier le cas lors des mises à jour (*update*), pendant lesquelles peuvent apparaître les codes suivants :

'C'	Le fichier est en conflit avec la version de référence.
'G'	Le fichier a été mis à jour par rapport au dépôt, il contenait des modifications par rapport à la version de référence, mais ces modifications ont été intégrées automatiquement.
'U'	Le fichier a été mis à jour à la dernière version présente dans le dépôt sans avoir à gérer de modifications locales.

• Liste des messages de commentaires : `svn log`

Fournit l'historique complet des révisions associées à un fichier ou à un répertoire du dépôt, ainsi que les commentaires associés.

svn log ⟨fichier⟩
svn log -r ⟨revision⟩ ⟨fichier⟩
svn log -r ⟨r1⟩ :⟨r2⟩ ⟨fichier⟩

```
% svn log code.c
------------------------------------------------------------------------
r18 | renault | 2008-03-23 | 2 lines

Correction du bug de la mangouste.
------------------------------------------------------------------------
```

⟨fichier⟩	(Optionnel) Le fichier dans le dépôt que l'on souhaite analyser. Si aucune révision n'est donnée, affiche les différences avec la dernière version dans le dépôt.
⟨revision⟩	Indique la révision du dépôt à laquelle on veut comparer la version de travail.
⟨rev1⟩ :⟨rev2⟩	Indique les deux révisions du dépôt entre lesquelles on désire calculer les différences.

Par exemple, la commande `svn log -r 1:HEAD` permet d'obtenir l'ensemble des révisions associées au dépôt.

• Affichage des différences entre révisions : `svn diff`

Cette commande permet de visualiser les différences entre deux révisions dans le dépôt, voir entre une révision du dépôt et les fichiers actuellement dans le répertoire de travail (cf. paragraphe 6.1 pour plus d'informations).

svn diff ⟨fichier⟩
svn diff -r ⟨revision⟩ ⟨fichier⟩
svn diff -r ⟨rev1⟩ :⟨rev2⟩ ⟨fichier⟩

```
% svn diff code.c
--- code.depot.c
+++ code.local.c
@@ -1,10 +1,7 @@
 #include <stdio.h>
-#include <stdlib.h>
+#include <math.h>
```

⟨fichier⟩	(Optionnel) Le fichier dans le dépôt que l'on souhaite analyser. Si aucune révision n'est donnée, affiche les différences avec la dernière version dans le dépôt.
⟨revision⟩	Indique la révision du dépôt à laquelle on veut comparer la version de travail.
⟨rev1⟩ :⟨rev2⟩	Indique les deux révisions du dépôt entre lesquelles on désire calculer les différences.

5 Manipulation des fichiers

• Ajout d'un fichier : svn add

• Suppression d'un fichier : svn delete , svn del , svn remove , svn rm

L'ajout et la suppression d'éléments dans le dépôt se fait naturellement avec les commandes suivantes. Ces commandes ne modifient que le répertoire de travail. Pour les transmettre au dépôt, il est impératif de les faire suivre d'un svn commit.

svn add ⟨fichier⟩
svn del ⟨fichier⟩

```
% svn add README.txt
   A     README.txt
```

• Déplacement d'un fichier : svn move , svn mv , svn rename , svn ren

• Copie d'un fichier : svn copy , svn cp

La modification des fichiers à l'intérieur de cet espace de travail se fait la plupart du temps avec les outils usuels, comme `emacs`. Néanmoins, il existe des cas particuliers qui sont la copie, le déplacement, ou le renommage des fichiers. Si l'on utilise les commandes usuelles `cp` ou `mv`, on va créer de nouveaux fichiers qui ne sont pas dans le dépôt. Et ces nouveaux fichiers n'auront pas l'historique des fichiers à partir desquels ils ont été créés. A la place, on utilise des commandes pratiquement homonymes propres à Subversion

svn mv ⟨source⟩ ⟨dest⟩
svn cp ⟨source⟩ ⟨dest⟩

```
% svn move barre.c choc.c
   A     choc.c
   D     barre.c
```

Noter que Subversion affiche à côté de chaque fichier un caractère décrivant l'état courant de ce fichier dans le dépôt (cf. la commande svn status).

6 Gestion d'un conflit

Lorsque plusieurs personnes travaillent de manière indépendante sur les mêmes fichiers source, il arrive parfois que leurs modifications soient conflictuelles : dans le dépôt coexistent deux versions différentes d'un même fichier. Dans les dernières versions de Subversion, un conflit est annoncé de la manière suivante lors de l'appel à la commande `svn update` :

```
Conflict discovered in 'code.c'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

Dans la mesure où il n'est pas trivial de rassembler ces modifications en une seule et unique nouvelle révision du code, il existe un certain nombre de possibilité pour résoudre les conflits. En fait, Subversion est capable de gérer automatiquement un certain nombre de conflits (cf. la commande `svn status`, codes U et G). Lorsque le menu précédent apparaît, il est possible de prendre plusieurs décisions :

- Repousser la décision (p) : l'utilisateur désire gérer le conflit manuellement. Il s'agit de la procédure la plus libre, et elle est détaillée dans les paragraphes suivants.
- Afficher les conflits et éditer à la volée (df, dc, e,r) : l'utilisateur peut vouloir gérer directement le conflit à la main à l'intérieur même du processus de mise à jour. Si l'on manque de pratique, ce choix est déconseillé, dans la mesure où les commandes fournies sont limitées.
- Choisir une version automatiquement (mc, tc) : l'utilisateur peut choisir de préférer une version directement sans résoudre le conflit, la sienne (mc) ou celle du dépôt (tc). Le conflit est alors automatiquement résolu

Dans les paragraphes suivants, nous montrons comment résoudre un conflit manuellement, en ayant choisi l'option de repousser la décision (p).

6.1 Différences entre deux fichiers

Supposons l'existence d'un fichier de code existant dans le dépôt Subversion. Actuellement, voici la version actuelle d'un fichier du dépôt :

• Révision initiale (`code.init.c`) :

```
1 #include <stdio.h>
2
3 int main(void) {
4   printf("Value of Pi : %.10f\n",
5          3.1415926);
6 }
```

Ce fichier a subi des modifications de la part d'un développeur dans son répertoire de travail, et en même temps de la part d'un autre développeur distant qui a communiqué ses modifications personnelles dans le dépôt. Au final, la situation est la suivante :

- Révision distante (`code.depot.c`) :

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  /* Print the value of pi */
  printf("Value of Pi : %.10f\n",
         3.1415926);

  return(EXIT_SUCCESS);
}
```

- Fichier local (`code.local.c`) :

```c
#include <stdio.h>
#include <math.h>

int main(void) {
  printf("Value of Pi : %.10f\n",
         4*atan(1));
}
```

Manifestement, ces deux fichiers sont dans des états différents. Une manière de représenter les différences entre ces deux versions consiste à utiliser la commande `diff` entre les deux fichiers avec la commande suivante : `diff -u code.depot.c code.local.c >diff.txt`

Le résultat obtenu est le fichier `diff.txt` suivant :

```
--- code.depot.c
+++ code.local.c
@@ -1,10 +1,7 @@
 #include <stdio.h>
-#include <stdlib.h>
+#include <math.h>

 int main(void) {
-  /* Print the value of pi */
   printf("Value of Pi : %.10f\n",
-         3.1415926);
-
-  return(EXIT_SUCCESS);
+         4*atan(1));
 }
```

Ce fichier représente les différences entre les deux codes. Les deux premières lignes indiquent quels sont les fichiers dont on inspecte les différences, et quel symbole leur est associé (ici `+` pour le fichier `code.local.c` et `-` pour le fichier `code.depot.c`). Les différences entre les deux fichiers sont alors listées sous la forme suivante :

- Ligne de localisation (ligne 3) : `@@ -1,10 +1,7 @@`
  Cette ligne permet de localiser l'endroit où se situe la différence. Ici, elle se situe entre les lignes 1 et 9 du fichier `-`, et entre les lignes 1 et 6 du ficher `+`.
- Lignes précédées d'un `+` ou d'un `-` (lignes 5, 6, 9, 11, 12, 13, 14) :
  Ces lignes sont les lignes présentes dans un seul des deux fichiers.
- Lignes non précédées d'un symbole (lignes 4, 7, 10) :
  Ces lignes sont les lignes communes aux deux fichiers.

Si l'on possède le fichier `diff.txt`, il est alors possible de passer d'une version à l'autre du fichier `code.c` en utilisant la commande `patch` (l'option `-b` permet de faire un fichier de sauvegarde) :

- pour faire passer le fichier de dépôt à la version locale : `patch code.depot.c diff.txt`
- pour transformer la version locale en celle du dépôt : `patch −R code.local.c diff.txt`

C'est le principe à la base du système de fichiers utilisé par Subversion : ne stocker que les différences entre les révisions du code. C'est aussi le mécanisme à la base du processus de gestion des conflits de Subversion.

Remarque : la commande `svn diff` permet d'afficher les différences entre les fichiers actuellement dans le répertoire de travail et une révision donnée à l'intérieu du dépôt, voire entre deux révisions du dépôt.

6.2 Apparition d'un conflit

Le scénario d'apparition d'un conflit le plus courant est le suivant : après avoir travaillé sur les fichiers dans le répertoire local, l'opération de `commit` renvoie un message d'erreur annonçant que les fichiers locaux sont périmés (`out of date`)[2]. Cela signifie que la version locale est différente de la version distante. La commande `update` affiche alors le statut suivant pour un fichier :

```
% svn update
   C    code.c
   Updated to revision 2.
```

La lettre `C` indique qu'il existe un conflit pour le fichier `code.c`. De plus, Subversion a créé un certain nombre de fichiers dans le répertoire de travail, à savoir :

- `code.c.mine` : la version locale avant mise à jour ;
- `code.c.rOLD` : la révision du fichier avant les modifications locales ;
- `code.c.rNEW` : la révision du fichier la plus à jour.

En plus de cela, Subversion a ajouté un certain nombre de modifications au fichier initial :

2. Subversion privilégie systématiquement la version du dépôt par rapport à la version locale.

- Fichier conflictuel : (code.c)

```
1  #include <stdio.h>
2  <<<<<<< .mine
3  #include <math.h>
4  =======
5  #include <stdlib.h>
6  >>>>>>> .r2
7
8  int main(void) {
9    /* Print the value of pi */
10   printf("Value of Pi : %.10f\n",
11 <<<<<<< .mine
12           4*atan(1));
13 =======
14           3.1415926);
15
16   return(EXIT_SUCCESS);
17 >>>>>>> .r2
18 }
```

Ces modifications peuvent se regrouper en blocs (*chunks*) délimités par :

- Marqueurs de début (lignes 2,11) : `<<<<<<<.mine`
  Puis viennent les lignes de code modifiées localement, suivies par :
- Lignes de séparation (lignes 4, 13) : `=======`
  Ensuite se trouvent les lignes de code modifiées dans le dépôt, terminées par :
- Marqueurs de fin (lignes 6, 17) : `>>>>>>>.r2`

6.3 Résolution du conflit

Tant que le conflit n'est pas résolu, Subversion ne permettra pas d'envoyer le fichier dans le dépôt. Afin d'éliminer le conflit, il est nécessaire de replacer les données à l'intérieur du répertoire de travail dans un état propice à un svn commit. Pour ce faire, il existe plusieurs solutions :

1. Recopier l'un des fichiers `code.c.rOLD` ou `code.c.rNEW` sur le fichier `code.c`. Naturellement, ceci supprimera toutes les modifications locales.
2. Gérer les conflits à la main : ceci consiste à modifier le fichier `code.c` de manière à obtenir une version finale qui rassemble l'ensemble des modifications du code. Ce travail est facilité par le fait que les différences entre les deux fichiers sont encadrées par les caractères `<<<<<<<` et `>>>>>>>`, et les deux versions sont séparées par `=======`.

- Fichier final : (code.c)

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main(void) {
  /* Print the value of pi */
  printf("Value of Pi : %.10f\n",
         4*atan(1));

  return(EXIT_SUCCESS);
}
```

Lorsque l'on a obtenu une version finale correcte, la résolution du conflit se fait à l'aide de la commande : `svn resolved code.c`. Cette commande supprime les fichiers précédemment rajoutés[3], et prépare le répertoire de travail pour permettre ensuite de transmettre les modifications. Si aucune autre modification conflictuelle n'a été apportée au dépôt entre temps, la commande `svn commit` peut alors s'exécuter sans encombres.

7 Compléments

7.1 Ignorer des fichiers

Dans un répertoire de travail, il existe des fichiers qui sont générés naturellement mais qui n'ont pas vocation à être mis sous contrôle de version. Un exemple naturel est celui des exécutables lorsque l'on gère le code source d'un programme. Néanmoins, il n'est pas pratique de les supprimer avant chaque commit du code.

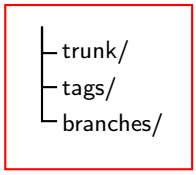
La bonne solution consiste à demander à Subversionde les ignorer. Pour cela, chaque répertoire est configurable à l'aide de la commande `svn propedit svn:ignore .`. Cette commande ouvre un fichier dans un éditeur de texte, fichier dans lequel il est possible d'insérer une liste de *patterns* de fichiers à ignorer. Pour un programme compilé en `C`, il est par exemple intéressant d'ajouter la ligne `"*.o"` dans la liste des fichiers à ignorer.

7.2 Créer des tags et des branches

Au cours d'un développement logiciel, certaines situations demandent à mettre l'ensemble du code dans un état particulier. L'exemple concret est celui d'un délivrable (*release*), pour lequel le code doit vérifier des contraintes d'utilisabilité, de stabilité, de documentation . . .Il est possible de fixer certaines révisions comme étant dédiées à ces releases, mais alors il n'est plus très pratique d'y accéder.

3. Attention : lors de l'application de cette commande, Subversion ne vérifie pas si le fichier contient encore des modifications à faire ou pas.

La bonne pratique pour un dépôt Subversion consiste à le découper à la racine en trois sous-répertoires différents :



Le répertoire `trunk` contient tout le code actuellement en développement. Si jamais l'on désire faire une sauvegarde de l'état courant du code, il suffit de créer une étiquette (*tag*) dans le répertoire `tags` en utilisant la commande `svn copy` . Par exemple, depuis le répertoire racine :

```
% svn copy trunk tags/release-1.0
```

Le fait d'utiliser la commande `svn copy` permet à Subversion de ne pas recopier entièrement les fichiers, mais simplement un identifiant les reliant à leur révision dans le répertoire `trunk`. Par convention, on ne fait pas d'autre commit dans un tag après l'avoir créé.

Le répertoire `branches` fonctionne quant à lui de manière similaire, mais permet de créer une nouvelle branche de développement du code différente de `trunk`. Cette branche peut vivre une vie indépendante du répertoire principal, quitte à être ultérieurement fusionnée (*merged*) avec le code initial en utilisant la commande `svn merge` :

```
% svn merge branches/branch1 trunk -r 3:HEAD
```

Remarquer que les fusions de code amènent souvent à résoudre des conflits.

Bibliographie

- La documentation de Subversion, accessible à l'URL suivante :
  `http://svnbook.red-bean.com/en/1.5/index.html`.
- Le livre correspondant, *Version Control with Subversion*
  par C Pilato, Ben Collins-Sussman, et Brian Fitzpatrick, aux éditions *O'Reilly*.

A voir aussi ...

D'autres systèmes de gestion de versions :

CVS :	`http://www.nongnu.org/cvs/`
Git :	`http://git-scm.com/`
Bazaar :	`http://bazaar.canonical.com/en/`