Table des matières

Cet exemple est une copie extraite d'un document sur Subversion réalisé par **Michael Jégat** (Corexpert).

Exemple pratique

Nous allons considérer que le projet n'est composé que d'un seul fichier **projet.txt**. Ce fichier contient 5 lignes, une pour chaque fonctionnalité à développer. Voici son contenu :

```
fonc 1 : vide
fonc 2 : vide
fonc 3 : vide
fonc 4 : vide
fonc 5 : vide
```

Lorsqu'une fonctionnalité sera réalisée, le texte "vide" sera remplacé par "ok". Si un bug est détecté sur une fonctionnalité et qu'il est corrigé, le texte sera "ok, correction 1".

Initialisation du dépôt

On suppose que l'import initial a été réalisé et que les trois répertoires trunk, tags et branches sont créés.

Ensuite, Clara va récupérer l'arborescence créée à l'aide de la commande suivante :

```
[clara] $ svn checkout URL .
```

Remarque : remplacer URL par l'adresse de votre dépôt

Maintenant, Clara va créer dans le répertoire /trunk/ le fichier projet.txt. Après, elle va l'ajouter au dépôt à l'aide de la commande suivante :

```
[clara] $ svn add projet.txt

[clara] $ svn status

[clara] $ svn commit -m "ajout du fichier projet"
```


Travail sur la version 1.0

Maintenant que le dépôt est initialisé, Clara et Morgane travaillent sur le projet, Clara sur la première fonctionnalité et Morgane sur la seconde. Chacune travaille sur une copie en local sur leur poste. Au bout d'un certain temps, la première fonctionnalité est terminée. Le fichier projet.txt devient donc :

```
fonc 1 : ok
fonc 2 : vide
fonc 3 : vide
fonc 4 : vide
fonc 5 : vide
```

Clara propage ses modifications en exécutant la commande suivante :

```
[clara] $ svn status
A     projet.txt

[clara] $ svn commit -m "réalisation de la fonc 1"
```

Le statut de chaque objet est représenté par un caractère : A ajouté, D supprimé, U modifié, C en conflit, G fusionné.

Morgane vient maintenant de terminer et comme elle n'a pas mis à jour sa copie en local, le fichier projet.txt possède le contenu suivant :

```
fonc 1 : vide
fonc 2 : ok
fonc 3 : vide
fonc 4 : vide
fonc 5 : vide
```

Elle effectue maintenant sa sauvegarde :

```
[morgane] $ svn commit -m "realisation func 2"
Envoi trunk/projet.txt
svn: Échec de la propagation (commit), détails :
svn: Out of date: '/trunk/projet.txt' in transaction '3-1'

[morgane] $ svn update
G     projet.txt
Actualisé à la révision 3.

[morgane] $ svn commit -m "realisation func 2"
Envoi trunk/projet.txt
Transmission des données .
Révision 4 propagée.
```

Vous remarquerez qu'un conflit a eu lieu sur le fichier projet.txt. En effet, Morgane n'a pas effectué de mise à jour et à modifier le même fichier que Clara. Cependant, comme elles n'ont pas travaillé sur les mêmes lignes, la fusion a été faite automatiquement par Subversion.

Tag de la version 1.0

Ca y est, les deux premières fonctionnalités sont réalisées, il faut maintenant *tagger* la version. Pour cela, Clara va tout d'abord effectuer une mise à jour de son dépôt.

```
[clara] $ svn update
U    projet.txt
Actualisé à la révision 4.
```

Ensuite, elle va *tagger* la version 1.0. En fait, dans Subversion, "*tagger*" consiste tout simplement à copier. Ainsi, Clara va exécuter la commande suivante :

```
[clara] $ svn copy URL/trunk/ URL/tags/1.0 -m "création du tag 1.0"
Révision 5 propagée.
```

Enfin, elle va mettre à jour sa copie en local et elle va s'apercevoir que le répertoire 1.0 a été créé.

```
[clara] $ svn update
A    tags/1.0
A    tags/1.0/projet.txt
Actualisé à la révision 5.

[clara] $ ls tag
1.0
```

Cette version du projet est donc sauvegardée et elle va pouvoir être livrée au client (voir la partie « obtenir une version livrable »).

Réalisation de la version 2.0 et correction des bugs

Deux tâches vont maintenant être réalisées en parallèle :

- la réalisation des fonctionnalités 3, 4 et 5 pour la version 2.0. Les fonctionnalités 3 et 4 vont être réalisées par Clara et la 5 par Morgane .
- la correction des bugs remontés par le client. Morgane est responsable de la correction des bugs.

Après quelques temps de travail, Clara termine la fonctionnalité 4 et propage ses modifications. Le contenu du fichier est désormais :

```
fonc 1 : ok
fonc 2 : ok
fonc 3 : ok
fonc 4 : vide
fonc 5 : vide
```

Et voici la commande exécutée :

```
[clara] $ svn status
M    projet.txt

[clara] $ svn commit -m "realisation func 3"
Envoi trunk/projet.txt
```



```
Transmission des données .
Révision 6 propagée.
```

Pendant ce moment là, le client a utilisé la version livrée et vient de détecter un bug sur la fonctionnalité 2. Il le remonte donc à Morgane. Pour satisfaire le client, Morgane va stopper son travail sur la fonctionnalité 5 et va corriger le bug. Seulement, le bug ne va pas être corrigé directement sur la version située dans /trunk/. En effet, cette version est en cours de développement, certaines fonctionnalités ont été rajoutées mais tout n'est pas encore parfaitement stable. Ainsi, pour ne pas déranger le développement de cette version, une branche va être créée à partir de la version que possède le client. Sachant que cette version est taggée à l'emplacement /tags/1.0/, Morgane va tout simplement copier ce répertoire à l'emplacement /branches/1.1/. En fait, comme les tags, une branche est tout simplement une copie d'un répertoire.

Morgane va donc effectuer la commande suivante :

```
[morgane] $ svn copy URL/tags/1.0/ URL/branches/1.1/ -m "creation de la branche 1.1"
Révision 7 propagée.
```

Ensuite, elle va mettre à jour sa copie en local avec la commande suivante :

```
[morgane] $ svn update
A     branches/1.1
A     branches/1.1/projet.txt
Actualisé à la révision 7.
```

Voilà, la nouvelle branche est créée, Morgane va maintenant pouvoir corriger le bug sur la fonctionnalité 2. Elle modifie donc le fichier projet.txt situé dans le répertoire /branches/1.1/. Ce fichier possède maintenant le contenu suivant :

```
fonc 1 : ok
fonc 2 : ok,  correction 1
fonc 3 : vide
fonc 4 : vide
fonc 5 : vide
```

La correction est terminée, Morgane propage ses modifications sur la base :

```
[morgane] $ svn status
M     projet.txt

[morgane] $ svn commit -m "correction de la fonc 2"
Envoi 1.1/projet.txt
Transmission des données .
Révision 8 propagée.
```

Cette version est bonne et va être livrée au client. Un nouveau tag va donc être créé pour cette version. Ainsi, Morgane crée le tag 1.1 à l'aide de la commande suivante :

```
[morgane] $ svn copy URL/branches/1.1 URL/tags/1.1
-m "création du tag 1.1"
Révision 9 propagée.
```

Morgane termine par une mise à jour de sa copie en local :


```
[morgane] $ svn update
A     tags/1.1
A     tags/1.1/projet.txt
Actualisé à la révision 9.
```

Morgane va pouvoir maintenant retourner au développement de la fonctionnalité 5. Si un nouveau bug est détecté par le client, la même procédure sera effectuée, mais cette fois-ci en récupérant la version taggée 1.1.

Fusion de branches

Lorsque le bug sur la fonctionnalité 2 a été corrigé, Clara aurait pu fusionner cette correction avec son développement courant.

Cette fusion est effectuée grâce à la commande **svn merge**. En fait, la fusion va consister à prendre les changements qui ont été apportés sur la branche 1.1 et à les appliquer sur la branche principale.

On peut voir ces changements à l'aide de la commande **svn diff**. Clara obtient par exemple :

```
[clara] $ svn diff -r 7:8 URL/branches/1.1/Index: projet.txt
===================================================================
--- projet.txt (révision 7)
+++ projet.txt (révision 8)
@@ -1,5 +1,5 @@
fonc 1 : ok
-fonc 2 : ok
+fonc 2 : ok, correction 1
fonc 3 : vide
fonc 4 : vide
fonc 5 : vide
```

A noter que les numéros de révisions indiqués dans l'option –r ont été obtenus à l'aide de la commande **svn log**.

On va donc prendre ces modifications pour les apporter sur la branche principale. Pour cela, Clara va se déplacer dans le répertoire /trunk/, puis elle va excécuter la commande suivante :

```
[clara] $ svn merge -r 7:8 URL/branches/1.1/
U     projet.txt
```

Comme vous pouvez le voir, le fichier projet.txt a été mis à jour, et voici son nouveau contenu :

```
fonc 1 : ok
fonc 2 : ok,  correction 1
fonc 3 : ok
fonc 4 : vide
fonc 5 : vide
```

La fusion a été réalisée automatiquement, mais il se peut que des conflits apparaissent. Dans ce cas, il faut suivre la même procédure qu'énoncée précédemment.

La dernière étape de Clara va d'être de sauvegarder les changements sur la base. Pour cela, elle exécute la

commande suivante :

```
[clara] $ svn commit -m "fusion avec la branche 1.1"
Envoi trunk/projet.txt
Transmission des données .
Révision 10 propagée.
```

Enfin, les filles vont devoir travailler pour corriger toutes les erreurs qu'elles ont faites et pour terminer la deuxième version du logiciel.

Obtenir une version « livrable » du projet

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 suivante exporte dans le répertoire livrable-1.1 la dernière révision du répertoire « branches/1.1 » :

```
$ svn export URL/branches/1.1/ livrable-1.1
```

On l'utilise pour obtenir du code livrable « débarrassé » des informations inutiles à l'utilisateur final (le client).

L'argument REV permet de préciser une révision et 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.

Résolution de conflit

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 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 ...

