TD 5: Développement collaboratif centralisé*

ED6 — Licence 3 — Université Paris Diderot

Séances du 3 et 10 avril 2012

L'objectif de ces travaux dirigés est de manipuler deux gestionnaires de versions : Subversion et Git, des outils qui s'appuient sur les principes de `diff` et `patch`, pour permettre la sauvegarde de l'historique d'un projet et le travail collaboratif.

À l'issu de ce TD, vous devrez savoir :
- mettre en place un dépôt personnel et y sauvegarder l'historique de votre travail ;
- utiliser un gestionnaire de versions centralisé pour travailler en équipe sur une même branche de développement ;
- travailler sur une branche parallèle au développement principal et fusionner deux branches ;
- effectuer les mêmes opérations dans un modèle de collaboration distribuée ;
- publier des *patch*s.

Sauvegarder son histoire Subversion est un logiciel de contrôle de versions développé par Karl Fogel. Il s'inscrit dans un modèle de développement **centralisé**, c'est-à-dire qu'il maintient une version de référence, appelée *révision*, d'un arbre de développement ainsi que les différentes évolutions de cet arbre. Ces données sont entreposées dans un *dépôt*[1] (*repository* en anglais). Les utilisateurs de ce dépôt collaborent en publiant (action *commit*) des modifications dans ce dépôt ou en important (action *update*) les dernières modifications depuis celui-ci. Ces modifications, à importer ou à publier, sont calculées en différenciant la *version de travail* (*working copy* en anglais) et le dépôt.

Subversion intègre de nombreuses interfaces de communication entre les utilisateurs d'un dépôt et le gestionnaire de ce dépôt. L'architecture du système est résumé dans la figure 1.

Subversion est subdivisé en plusieurs outils Unix en ligne de commandes (il existe aussi des interfaces graphiques ou des interfaces de programmation) ; en voici les principaux :

svn est l'outil principal permettant d'effectuer les opérations mettant en jeu la copie de travail et le dépôt.

svnadmin permet d'administrer le dépôt.

svnlook fournit des primitives d'observation d'un dépôt (sans avoir nécessairement de copie de travail).

svnserve est le serveur qui répond à des requêtes d'interaction avec le dépôt. (On n'utilise pas ce programme directement en général.)

Toutes les commandes suivent le même schéma :

```
(svn | svnadmin | svnlook | ...) sous-commande arguments...
```

Une aide en ligne, extrêmement utile, est disponible en utilisant la syntaxe :

```
(svn | svnadmin | svnlook | ...) help sous-commande
```

Dans cette section, nous allons découvrir l'utilisation la plus basique de Subversion : l'archivage de l'historique de son travail personnel.

1 Son histoire personnelle

Question 1.1. *À l'aide de la commande `svnadmin`, créez un dépôt sur votre compte à l'emplacement `$HOME/svnrepo`. Par la suite, on notera `$SVNREPO` cet emplacement. Du point de vue de Subversion, ce dépôt est accessible à l'aide d'une URL locale de la forme `file://$SVNREPO`.*

Question 1.2. *Placez-vous dans le répertoire contenant votre répertoire de scripts et fichiers sources créés lors des dernières séances. À l'aide de la sous-commande `import` de `svn`, publiez une version initiale de votre répertoire dans le dépôt.*

*Ce sujet de TD est inspiré d'un sujet proposé par Yann Régis-Gianas

1. parfois aussi appelé *repositoire*

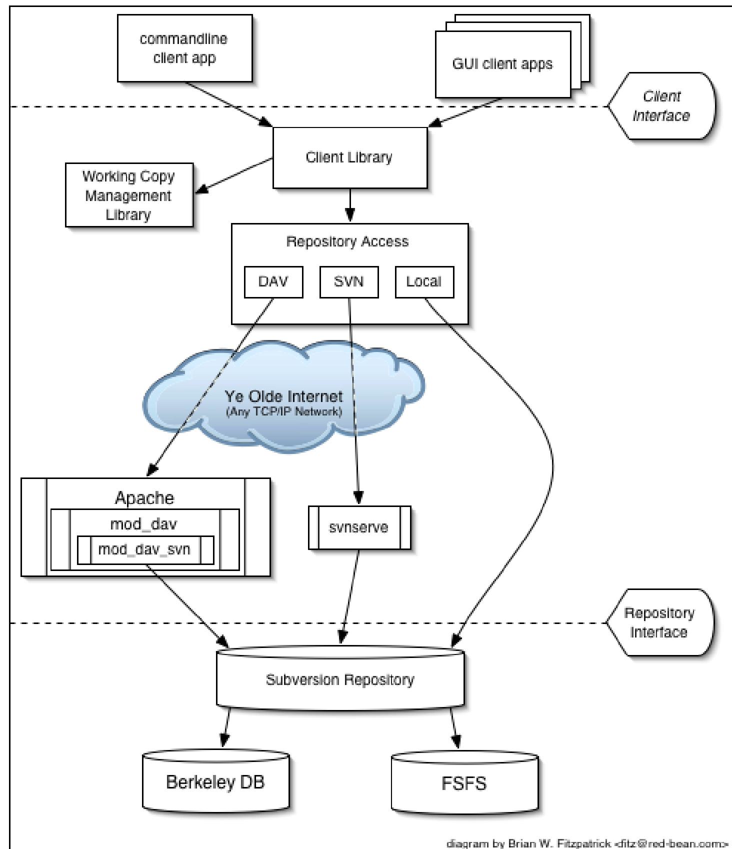

FIGURE 1 – Architecture de Subversion [1]

Question 1.3. *Créez un nouveau répertoire du nom de votre choix. En utilisant la sous-commande `checkout` de `svn`, créez une version de travail de votre dépôt. Comparez le contenu de ce répertoire et du répertoire initial à partir duquel vous avez publié la version initiale de vos scripts et fichiers sources.*

Question 1.4. *Dans votre version de travail, modifiez un ou plusieurs fichiers. Observez le résultat de la commande `svn diff`.*

Question 1.5. *Publiez cette différence à l'aide de la commande `svn commit`. Que se passe-t-il ? Modifiez la variable d'environnement `$EDITOR` pour utiliser Emacs comme éditeur de message de modifications.*

Question 1.6. *Modifiez de nouveau le contenu d'un ou plusieurs fichiers. Que produit la commande `svn revert` ? Comment revenir à la version initiale de vos fichiers modifiés ?*

Question 1.7. *À quoi sert la commande `svn status` ?*

Question 1.8. *Créez un fichier `LISEZMOI` dans lequel vous devez lister vos fichiers et leur donner une documentation succincte. Rajoutez ce fichier dans le dépôt. Publiez cette version. Renommez le fichier `LISEZMOI` en `README`. Qu'indique `svn status` ?*

Question 1.9. *Supprimez le fichier `README` du dépôt. Publiez ce changement. Est-il possible de récupérer ce fichier ?*

Question 1.10. *À quoi sert la commande `svn log` ? La commande `svn annotate` ?*

2 Développement collaboratif

Comme l'a montré notre étude de l'outil `diff3`, le travail collaboratif pose le problème de la **modification concurrente** de fichier source.

Dans cette partie, vous allez jouer plusieurs scénarii de développement collaboratif en binôme (ou si vous êtes seuls, en jouant deux rôles à la fois). Pour cela, chaque groupe de travail doit mettre en place un dépôt Subversion partagé. Nous utiliserons un accès local car il est le plus simple à mettre en place : mais il faut que tous les utilisateurs du dépôt aient accès (en lecture/écriture) au dépôt.

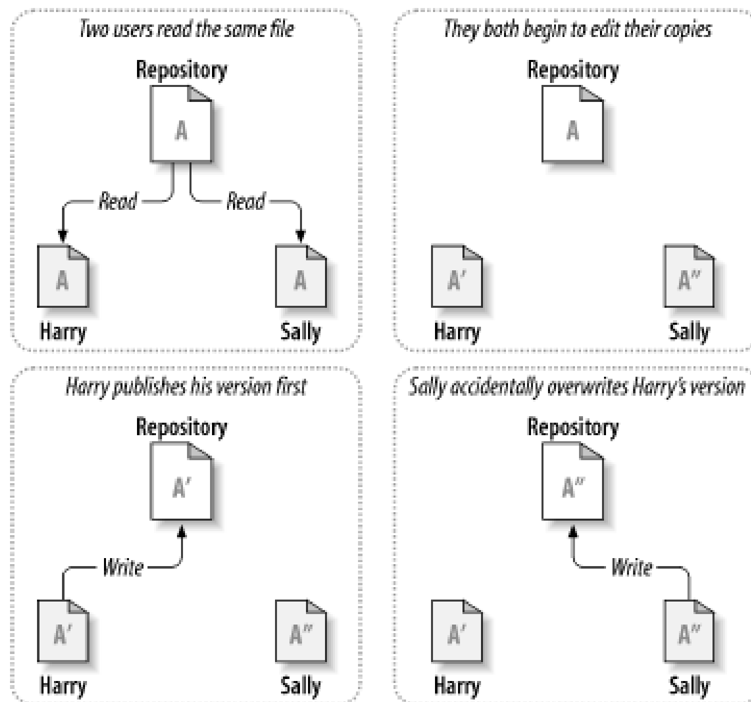

FIGURE 2 – Situation collaborative non-productive [1]

`/info/nouveaux` est un système de fichier accessible en écriture depuis tous les postes, ou se trouvent vos *répertoires personnels*, c'est ici que nous placerons les dépôts.

Question 2.1. *Formez des binômes. Choisissez le* répertoire personnel *d'une personne pour chaque binôme et créez un nouveau dépôts dans un sous-répertoire. Modifiez les permissions d'accès à ce dépôt pour que les deux binômes puissent y accéder à l'aide de la commande*[2] `chmod -R a+rwX /info/nouveaux/HOME/NOTREDEPOT`

Question 2.2. *Récupérez l'état actuel du dépôt.*

Question 2.3 (À faire par le binôme 1)**.** *Créez l'arborescence suivante :*

```
|– nom-prénom-binôme-1
|– nom-prénom-binôme-2
‘ – commun
    |– src
    ‘ – scripts
```

et publiez-la.

Question 2.4 (À faire par le binôme 2)**.** *Mettez à jour votre version de travail. Que se passe-t-il ?*

Question 2.5. *Partagez les scripts et fichiers sources que vous avez écrits lors des dernières séances à l'aide de ce dépôt.*

Deux méthodes de travail collaboratif

Pour collaborer correctement, il faut pouvoir travailler de manière concurrente sur un arbre de sources sans introduire de conflits. Dit autrement, il est contre-productif de perdre le travail d'un membre de l'équipe parce que le travail concurrent d'un autre membre l'a écrasé.

La figure 2 résume la situation à éviter. Harry et Sally font des modifications concurrentes sur un même fichier A. Harry publie ses changements. Sally a la possibilité d'écraser le travail de Harry sans l'avoir pris en compte.

Deux solutions à ce problème sont possibles :

2. La commande `chmod` permet d'éditer les permissions d'accès de fichiers et de répertoires :
– Le flag `-R` indique que les changement doivent être appliqués récursivement à toute l'arborescence
– `a` indique que l'on modifie les permissions de tous (l'utilisateur, les membres du groupe et les autres utilisateurs)
– `+rwx` indique que l'on ajoute (`+`) les permissions en lecture (`r`), écriture (`w`) et exécution (`X` – qui indique que l'on rajoute cette permission en exécution si et seulement si quelqu'un d'autre a une permission d'exécution sur le fichier ou répertoire)

Lock-Modify-Unlock Harry indique qu'il va travailler sur le fichier A en posant un verrou sur A. Sally peut lire le travail de Harry mais ne peut pas modifier le fichier tant que Harry n'a pas relâché le verrou.

Copy-Modify-Merge Harry et Sally font évoluer leur copie de travail. Harry publie ses modifications. Lorsque Sally essaie de publier ses modifications, le système rejette la publication de Sally : elle doit d'abord prendre en compte les modifications de Harry. Une fusion des modifications indépendantes est effectuée. Si un conflit existe, Sally doit d'abord le résoudre pour être autorisée à publier ses changements.

Question 2.6. *Quels sont les inconvénients et les avantages des deux méthodes ?*

Question 2.7. *Subversion propose des commandes pour ces deux solutions. Listez-les !*

Question 2.8. *Simulez en binôme une situation de travail collaboratif non conflictuelle à l'aide des deux méthodes précédentes.*

Question 2.9. *Simulez en binôme une situation de travail collaboratif conflictuelle et la résolution de ce conflit à l'aide des deux méthodes précédentes.*

Il est parfois nécessaire de ne pas se synchroniser pendant un moment avec le développement du reste de l'équipe. Un exemple d'une telle situation est la coexistence de deux versions de développement d'un logiciel : une version stable en voie d'être publiée et dans laquelle seules les modifications de correction d'erreurs mineures sont admises et une version instable dans laquelle est développée une fonctionnalité supplémentaire. Ces deux versions de développement sont appelées des **branches**.

Les interactions entre les branches sont de deux types :

Transfert de modifications particulières d'une branche à l'autre Tant que le développement de la nouvelle fonctionnalité n'est pas mature, les modifications qui la concernent sont publiées uniquement dans la branche instable. Durant ce développement, il se peut que des corrections d'erreurs soient effectuées dans la branche stable : il doit être possible d'importer ces modifications dans la branche instable.

Synchronisation totale entre deux branches Une fois que le développement de la nouvelle fonctionnalité est terminé, il est temps de l'importer dans la branche stable. Pour cela, il ne faut importer que les modifications qui ont trait à la nouvelle fonctionnalité et non pas les modifications de correction d'erreurs qui existent déjà dans la branche stable.

Les branches

Question 2.10. *Dans le répertoire personnel du binôme 1, créez une branche de développement expérimental du répertoire* `scripts`*.*

Question 2.11. *Dans cette branche expérimentale, créez un nouveau script. Ce script s'appelle* `svn-publish.sh`*. Il consiste à lancer la commande* `svn update` *puis la commande* `svn commit`*.*

Question 2.12. *Dans la branche stable, modifiez un script. Importez cette modification dans la branche instable à l'aide de la commande* `svn merge`*.*

Question 2.13. *Publiez une modification du script expérimentale : il ne faut lancer la seconde commande que si la première réussit.*

Question 2.14. *Observez et comparez l'historique des deux branches à l'aide de la commande* `svn log`*.*

Question 2.15. *Fusionnez les deux branches à l'aide de la commande* `svn merge --reintegrate`*. Note : une fois fusionné, la branche instable est inutilisable et doit être détruite.*

Références

[1] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version control with Subversion.* 2004.