Diff, Patch, Subversion

1 Introduction

La gestion de projet nécessite des outils évolués permettant de faciliter cette gestion. En effet, la gestion des différentes personnes collaborant au projet et des différentes versions des composantes de ce projet ne peuvent pas se faire de façon rigoureuse sans un outil dédié. Les outils permettant de faire de la gestion de version répondent à deux besoins principaux :

— l'*édition collaborative*, c'est-à-dire le fait que plusieurs personnes travaillent sur un même projet, en prenant en compte les apports, les modifications et suppressions faits par toute personne du projet,

— l'*archivage* des différentes versions des composantes d'un système. Dans le cas d'un logiciel en développement, ces composantes correspondent alors aux fichiers sources, aux makefiles, aux différentes ressources...

La gestion de version permet la manipulation de plusieurs versions d'une même unité d'information, généralement un fichier. Les modifications faites sur cette unité sont identifiées par un numéro (la *version*) qui s'incrémente au fil des révisions. Les outils de gestion de projet sont typiquement utilisés pour le développement de logiciels, en gérant les différents fichiers sources (texte) de ces logiciels. Mais il peuvent être utilisés dans d'autres contextes (édition collaborative, projet de toute nature) et avec d'autres ressources (binaires par exemple).

2 Présentation de quelques outils

2.1 RCS

RCS (`http://www.gnu.org/software/rcs/rcs.html`, *Revision Control System*) est un logiciel qui permet la gestion de différentes versions d'une unité d'information, typiquement un fichier. Il automatise le stockage, l'identification et la fusion des différentes versions relatives à une unité. Il est essentiellement utilisé pour manipuler des fichiers textes, mais peut également travailler sur des fichiers binaires. Son rayon d'action est limité : il ne travaille que sur un seul fichier et n'a donc pas de « vision globale » d'un projet. Il est à la base de logiciels tels que CVS.

2.2 CVS

CVS (`http://www.nongnu.org/cvs/`, *Concurrent Version System*) est certainement le logiciel de gestion de version le plus connu. Il permet de garder la trace de toutes les versions successives d'un ensemble de fichiers (typiquement les sources d'un logiciel), tout en offrant des accès concurrents aux différentes personnes travaillant sur ces fichiers. CVS fonctionne selon un modèle client-serveur et est basé sur la notion de *dépôt* (*repository*). Le serveur stocke alors tous les sources d'un projet, ainsi que leur historique, et assure l'accès aux clients. Les clients se connectent au serveur afin de récupérer des copies complètes d'un projet, à partir desquelles ils travaillent, et répercutent les modifications effectuées sur le serveur.

Plusieurs clients peuvent éditer les différentes copies d'un même projet de façon simultanée. Lorsqu'ils répercutent par la suite leurs modifications sur le serveur, celui-ci essaie de

les fusionner. Si cette fusion échoue, ce qui est le cas par exemple lorsque plusieurs clients ont modifié la même ligne d'un même fichier, la fusion n'est pas effectuée et le serveur avertit alors les clients de la présence d'un conflit, qui doit alors être résolu « à la main » par les clients impliqués. Lorsque les modifications sont acceptées par le serveur, éventuellement après la résolution de conflits, le numéro de version de chaque fichier impliqué est automatiquement incrémenté et une description des modifications, éditée par les clients à l'origine des modifications, est ajoutée.

Les clients ont la possibilité de comparer différentes versions d'un fichier, de voir la liste complète des modifications effectuées, ou de récupérer l'état d'un projet à une date ou un numéro de version donné. CVS est également capable de maintenir plusieurs *branches* d'un projet, un branche représentant par exemple la version stable d'un projet, tandis qu'une ou plusieurs autres sont dédiées aux versions de développement du même projet. Les différences existant entre plusieurs versions d'un même fichier sont stockées sous forme de différences, calculées typiquement grâce à l'utilitaire UNIX `diff`.

CVS est historiquement la première solution complète permettant de faire de la gestion de projet de façon satisfaisante. Il est encore largement utilisé dans beaucoup de projets, mais souffre de quelques défauts qui ont favorisé l'émergence de solutions alternatives.

2.3 Subversion

Subversion (https://subversion.apache.org/), parfois également appelé `svn`, est actuellement le successeur de CVS le plus utilisé. Il conserve une interface utilisateur proche de CVS, ce qui facilite les migrations, mais corrige la plupart de ses défauts :

— Les commits sont atomiques : un ensemble de modifications concernant différents fichiers est géré comme un ensemble unique de modifications, et non comme des modifications indépendantes.
— Le renommage et le déplacement de fichiers est possible, tout en conservant l'historique.
— Il est possible d'attacher des méta-données (des propriétés, des permissions) aux fichiers. Ces méta-données sont versionnées.

Les pages de manuel de Subversion sont peu pratiques à lire. Il est conseillé d'utiliser le livre *Version Control with Subversion* en ligne à l'adresse http://svnbook.red-bean.com/.

2.3.1 Commandes de base de Subversion

Consultez *Version Control with Subversion* pour plus de détails.

— `svnadmin create` *nom* : crée un dépôt Subversion.
— `svn checkout` : récupère une copie d'un dépôt.
— `svn update` : met à jour la copie locale à partir des modifications reportées sur le serveur.
— `svn status` : affiche l'état de la copie locale (fichiers modifiés, etc)
— `svn add` : ajoute un fichier au dépôt
— `svn commit` : met à jour des modifications locales sur le serveur.
— `svn log` : permet de consulter l'historique des modifications
— `svn diff` : permet de visualiser une modification
— `svn rm` : supprime un fichier du dépôt

2.4 Git

Git (`http://git-scm.com/`) est également une solution de remplacement pour CVS ou SVN. Tout comme Subversion, il étend la notion de modifications à un ensemble de fichiers, plutôt qu'à un fichier indépendant. Son modèle de fonctionnement, distribué, le différencie de solutions comme CVS ou subversion en apportant plus de flexibilité sur certaines opérations.

Git a initialement été développé par Linus Torvalds pour gérer les sources du noyau Linux. Mais il est maintenant utilisé par de nombreux autres projets, et tend à remplacer les autres solutions distribuées (Arch, Bazaar, Mercurial, Monotone, ...).

Une étude d'une petite quinzaine d'outils permettant de la gestion de projet peut être consultée sur `http://better-scm.shlomifish.org/`.

3 Questions

Le contexte de cette série d'exercices est la gestion d'un site Web, que vous pouvez récupérer d'un TP précédent. Le but de mettre en place une gestion collaborative et avec révision des différentes pages de votre site. Afin de vous concentrer sur l'essentiel, utilisez un site avec un nombre raisonnable de pages (5 maximum).

3.1 `diff` et `patch`

Q1. *Exercices avec `diff`.* Faites une copie de votre site dans un autre répertoire et effectuez une série de modifications dans un des fichiers. Utilisez ensuite la commande `diff` pour visualiser les différences entre le fichier d'origine et le fichier modifié. Étudiez la syntaxe utilisée par `diff` pour générer la liste de différences.

Q2. *Exercices avec `patch`.* La commande `patch` permet de répercuter une liste de différences sur un fichier pour en obtenir une version modifiée. Pour cela, il est nécessaire de construire la liste de différences avec `diff -u`.

(a) Testez cette nouvelle façon de construire des différences entre deux fichiers et étudiez les différences de syntaxe dans la liste établie par `diff`.

(b) Répercutez des différences établies par `diff -u` au moyen de la commande `patch`.

(c) La nouvelle version de votre fichier est-elle bien conforme à ce que vous attendez ?

(d) Comment le vérifier ?

Q3. *Exercices sur un ensemble de fichiers.*

(a) À quoi correspondent les différents flags de la commande `diff -rupN` ?

(b) Dupliquez le répertoire contenant votre site, modifiez certains de ses fichiers dans le répertoire de copie et lancez cette commande.

(c) Trouvez le moyen de lancer la commande `patch` pour que les différences générées puissent être appliquées au répertoire de départ.

3.2 Subversion

Attention, quand on travaille avec Subversion, il ne faut pas confondre le dépôt (*repository*), qui est la *base de données* stockant tout l'historique des modifications du projet, et la copie de travail (*working copy*) qui est l'endroit dans lequel le développeur va faire des modifications avant de les envoyer au dépôt à l'aide des commandes de Subversion. Les seules opérations faites directement dans le *repository* sont celles concernant l'administration du *repository* lui-même.

Q4. Créez un nouveau dépôt Subversion, à un endroit que vous avez préalablement fixé.

Q5. Placez-vous dans un nouveau répertoire `r1` et récupérez le contenu de votre site Web.
L'organisation suggérée est :

```
~   # repertoire personnel ($HOME)
|-- .svnrepos   # repertoire contenant les depots
|   `-- web       # un autre depot
`-- r1  # copies locales pour l'utilisateur 1
|   `-- web       # la copie locale de web
`-- r2  # copies locales pour l'utilisateur 2
    `-- web       # la copie locale de web
```

Q6. Importez votre site Web dans le dépôt précédemment créé.

Q7. Placez-vous dans un autre nouveau répertoire `r2` et récupérez à nouveau votre site.

Q8. Faites des modifications dans `r2` et répercutez-les au niveau du dépôt.

Q9. Placez-vous dans `r1` et répercutez les modifications soumises au dépôt.

Q10. Faites des modifications sur le même fichier dans `r1` et `r2`, puis répercutez vos modifications depuis `r1` et ensuite depuis `r2`.

(a) Que constatez-vous ?

(b) Comment contourner ce problème ?

3.3 Utilisation de Subversion en réseau

Subversion est le plus souvent utilisé pour permettre à des personnes travaillant à des endroits différents, ou simplement sur des postes de travail différents, de travailler ensemble.

La manière la plus courante d'utiliser Subversion à travers un réseau est de l'utiliser à travers des connexions SSH.

Pour cette partie du TP, nous allons utiliser l'environnement Vagrant du TP SSH (avec les deux machines *cli* et *srv*).

Cette partie du TP est à rendre. Dans un fichier texte, recopiez les questions, et indiquez toutes les commandes nécessaires pour réaliser les manipulations demandées (en précisant où elles doivent être exécutées - client ou serveur).

Q11. Vérifiez que vous arrivez à vous connecter en SSH sur le serveur.

Q12. Configurez la connexion SSH par clé sur le serveur, depuis la machine cliente.

Q13. Vérifiez que vous pouvez vous connecter sans entrer votre mot de passe depuis la machine cliente vers le serveur.

Nous allons maintenant configurer un dépôt Subversion utilisable par nos deux utilisateurs distants.

Q14. Créez un dépôt Subversion (sur le serveur).

Q15. Comme précédemment, récupérez une copie locale du dépôt (mais cette fois-ci en utilisant SSH pour le récupérer sur votre machine locale), faites y des modifications, et vérifiez que vos modifications sont bien prises en compte.

Q16. Côté client mais dans un autre répertoire, récupérez une autre copie locale, et faites des modifications.

Q17. Générez des modifications conflictuelles entre vos deux copies locales, et tentez de résoudre les conflits (il est nécessaire d'utiliser `svn resolved`).

3.4 Questions subsidiaires

Q18. Essayez les différentes interfaces graphiques disponibles pour Subversion (`subcommander`, `rapidsvn`, ...)

Q19. Pour permettre la navigation dans votre dépot avec une interface Web, installez et configurez ViewVC (http://www.viewvc.org/) et/ou WebSVN (https://websvnphp.github.io/)

Q20. Trouvez un projet libre utilisant SVN, et récupérez une copie locale de son code source. Vous pouvez utiliser le code source de Subversion lui-même (https://subversion.apache.org/source-code.html). Puis utilisez `statsvn` pour générer des statistiques sur ce projet.