Gestion de configuration avec CVS et Subversion

Maxence Guesdon et Guillaume Rousse

31 mars 2011

Table des matières

1 Introduction

1.1 Gestion de version

Les systèmes de gestion de version répondent à deux problématiques.

D'abord, le développement d'un projet informatique implique la création et la modification d'un ou plusieurs fichiers. Au fil du temps, ces fichiers sont modifiés (enrichis, supprimés, ...). Or, il arrive souvent que l'on veuille revenir à l'état précédent d'un fichier, afin par exemple de voir quelle modification a entraîné l'apparition d'un bug, ou de remettre un passage que l'on avait supprimé. Un système de gestion de version garde l'intégralité des états successifs de chaque fichier, permettant à tout instant de revenir en arrière.

Ensuite, le développement d'un projet informatique implique généralement plusieurs personnes simultanément. Dès lors que ces personnes sont amenées à intervenir sur les mêmes parties du projet, il convient de gérer correctement les modifications concurrentes des fichiers. Un système de gestion de version joue ce rôle de médiation.

1.1.1 Approches pessimiste/optimiste

La gestion de la concurrence peut s'effectuer selon deux modalités.

- La première empêche toute modification d'un fichier par les autres utilisateurs dès lors que l'un d'eux a commencé à l'éditer, et ce jusqu'à ce qu'il valide ses changements. Les conflits sont donc impossibles, mais cette méthode manque de souplesse puisqu'elle empêche la concurrence : un utilisateur ayant verrouillé un fichier empêche les autres de progresser. Elle est dite “pessimiste", dans le sens où elle considère que les conflits sont la règle, et qu'il faut tout faire pour les empêcher.
- La deuxième consiste à laisser chaque utilisateur modifier sa propre version du fichier, et à régler les conflits lors d'une synchronisation. Cette méthode est beaucoup plus souple, mais laisse la résolution des conflits à la charge des utilisateurs. Elle est dite “optimiste", dans le sens où elle considère que les conflits sont l'exception, et qu'il est préférable de les résoudre au cas par cas.

1.1.2 Approches centralisée/décentralisée

Il existe également deux modalités pour gérer l'archive de référence (le dépôt).

- La première utilise un dépôt central unique : on parle d'approche centralisée. Toute modification impose un accès en écriture sur ce dépôt, il est donc impossible de maintenir une branche parallèle d'un projet dont on n'est pas contributeur.
- La seconde utilise plusieurs dépôts : on parle d'approche décentralisée. Il devient possible de créer son propre dépôt à partir d'un autre, d'effectuer ses modifications localement, puis de récupérer les changement ultérieurs depuis la source, voire de propager ses propres modifications si l'on dispose d'un accès en écriture.

1.2 Historique

Voici un rapide historique des logiciels libres de gestion de version.

Le premier outil est RCS (Revision Control System). C'est un outil pessimiste, sans support réseau, donc forcément centralisé, bien qu'il n'y ait pas de dépôt à proprement parler, l'archive

versionnée du fichier étant stockée à côté de celui-ci. Un concurrent de RCS fut SCCS (Source Code Control System) développé par Sun.

Ensuite, il y eut CVS, implémenté d'abord comme un ensemble de scripts shell au dessus de RCS, de façon à apporter un support réseau. Il reste centralisé, mais devient optimiste. Il est ensuite ré-écrit en code natif, mais garde le format RCS pour stocker les différentes révisions d'un fichier.

Pendant longtemps, CVS reste la référence, malgré ses limitations. Le développement d'alternatives crédibles étant relativement long. Aujourd'hui, c'est un peu l'explosion.

Il y a d'abord Subversion (SVN), le descendant direct de CVS, dans le sens où il garde l'orientation initiale (optimiste et centralisée), et les mêmes sous-commandes, mais en corrige les défauts.

Dans le même temps, les systèmes décentralisés arrivent également à maturité, principalement poussées par des besoins très spécifiques, comme ceux du kernel linux de maintenir des branches en parallèle de façon autonome pendant longtemps, ou la pratique répétée de développer sa propre version d'un logiciel quand les contributions sont rejetées par l'équipe de développement. Parmi ceux-ci, on peut citer (liste non exhaustive) :

- SVK,
- darcs,
- tla,
- git,
- monotone,
- bazar.

Parmi ces derniers, SVK mérite une mention spéciale dans la mesure où il apporte les avantages d'un modèle décentralisé au-dessus d'un dépôt Subversion (en lecture/écriture) ou CVS (en lecture seulement) existant.

1.3 Plan de formation

Cette formation présente deux systèmes de gestions de versions libres, CVS et Subversion. Ce choix est basé sur deux critères. D'abord, ce sont ceux les plus utilisés à l'heure actuelle, CVS pour des raisons historiques, et Subversion parce ce qu'il est son successeur attitré. Ensuite, ce sont également ceux proposés sur la forge INRIA.

La formation comprend deux parties, l'une concernant l'utilisation, l'autre concernant l'administration d'un tel système. Elles sont prévues pour être suivies isolément l'une de l'autre.

Pour l'instant, seule la partie utilisation est prête. La première partie concerne CVS, la seconde Subversion.

2 Utilisation

2.1 Introduction

Cette première partie de la formation concerne l'utilisation d'un système de gestion de versions, et présente successivement CVS puis Subversion.

Dans les deux cas, la situation est la même. Il s'agit de développer en commun un site web à partir de pages HTML statiques présentant les participants. Le dépôt existe déjà, et contient uniquement la première page du site. Les participants vont dans un premier temps créer de nouveaux fichiers et travailler sur ceux-ci, de façon à se familiariser avec les commandes de base. Puis ils travailleront sur

les fichiers communs, afin d'expérimenter la concurrence et la gestion des conflits. Enfin, ils verront les concepts plus avancés, tels que la gestion de branches de développement parallèles.

2.2 Principes

2.2.1 Dépôt

CVS et Subversion fonctionnent avec un *dépôt* (*repository* en anglais), dans lequel se trouvent les différentes versions des fichiers.

Ce dépôt est subdivisé en sous-parties nommées *modules*, correspondant généralement à des projets distincts.

Dans un cas comme dans l'autre, l'accès direct au contenu du dépôt est réservé à des tâches administratives. De plus, le format interne de stockage de Subversion rend obligatoire l'utilisation de commandes dédiées à cet effet.

2.2.2 Mode de travail

Pour travailler, chaque utilisateur récupère une *copie locale* du module, stockée dans un *répertoire de travail*, par opposition au dépôt. Cette opération se nomme récupération (*check out* en anglais).

Ensuite, chacun travaille dans sa copie locale et, de temps en temps, se synchronise avec le dépôt :

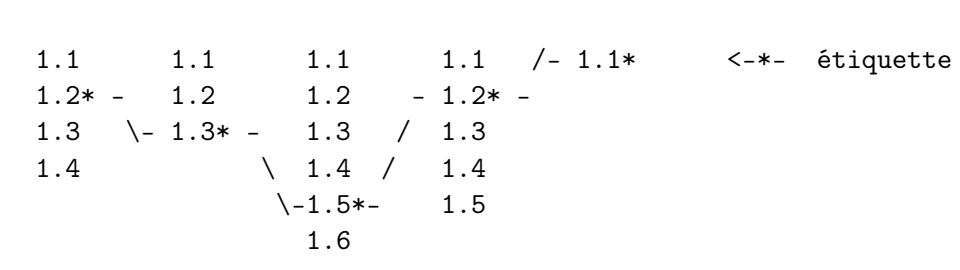
- soit en répercutant dans sa copie locale les modifications faites dans le dépôt par d'autres personnes : c'est la mise à jour (*update* en anglais),
- soit en répercutant ses propres modifications dans le dépôt : c'est la validation (*commit* en anglais).

La validation ne peut avoir lieu si des modifications sont intervenues dans le dépôt depuis la dernière mise à jour locale. Il faut alors mettre à jour sa copie locale et corriger manuellement les éventuels conflits qui peuvent apparaître, avant de pouvoir valider.

2.2.3 Versionnement

La façon de référencer les différents états du dépôt varie.

Dans CVS, on parle de *révision* pour qualifier l'état d'un fichier particulier, et chaque fichier possède donc son propre compteur numérique. L'état du dépôt complet, c'est-à-dire la combinaison d'une révision pour chaque fichier s'appelle une étiquette (*tag*), et utilise au contraire un nom symbolique.



Dans Subversion, il n'y a plus de notion d'état pour chaque fichier, le terme de révision désigne l'état de l'ensemble du dépôt à un instant donné. L'intérêt des étiquettes diminue mais elles restent utilisées pour qualifier ces états avec des noms symboliques plutôt que des indices numériques.

2.2.4 Branches

Lorsqu'il faut faire des modifications longues tout en continuant le développement "normal" du logiciel ou de l'article, plutôt que de développer dans sa copie locale sans faire de validation pendant très longtemps, il est préférable de créer une branche. Le long développement en question et les validations associées se font alors dans la nouvelle branche, par opposition à la branche principale, ou "tronc" (*trunk* en anglais), dans laquelle continue le développement "normal".

Lorsque le développement dans la branche est fait, on peut fusionner les modifications faites dans cette branche avec le tronc. Au cours du développement dans la branche, on peut répercuter les modifications du tronc dans la branche, pour éviter que le code ne diverge trop et que la fusion de la branche avec le tronc ne génère trop de conflits.

On peut utiliser les branches dans les cas suivants :

- Lorsqu'une version du logiciel est distribuée, on peut créer une branche dans laquelle seront faites les corrections de bugs sur cette version. Les corrections peuvent être répercutées dans le tronc si elles s'appliquent aussi aux développements en cours. Ceci permet de fournir une version corrective de la version distribuée.
- Lorsqu'un long développement implique de lourds changements dans l'organisation du code, qui lui doit continuer d'évoluer ; par exemple le changement d'un type de données. On crée alors une branche pour faire la modification du type de données ; on répercute régulièrement dans cette branche les modifications faites dans le tronc. Quand le développement dans la branche est terminé, on répercute dans le tronc les modifications faites dans la branche par rapport à la séparation d'avec le tronc. On peut ensuite détruire la branche. En répercutant régulièrement les modifications du tronc dans la branche, on évite d'avoir trop de conflits lors de la fusion finale.
- On peut aussi créer une branche pour un développement expérimental qui n'a pas pour but d'être intégré dans le tronc.

2.3 CVS

2.3.1 Syntaxe

Forme des commandes

Les commandes CVS sont de la forme suivante :

\# **cvs option_globales sous-commande options_spécifiques arguments**

Les `options_globales` sont des options générales, par exemple la spécification de l'endroit où trouver le dépôt. La `sous-commande` détermine l'opération à effectuer. Nous verrons les différentes sous-commandes par la suite. Les `options_spécifiques` sont les options modifiant le comportement de la sous-commande indiquée.

La commande suivante indique les options globales supportées :

\# **cvs --help-options**

tandis que celle-ci permet de connaître les options spécifiques de la sous-commande CVS indiquée :

\# **cvs --help *sous-commande***

La plupart des sous-commandes prennent en paramètre un ou plusieurs fichiers sur lesquels porte l'action. Si aucun fichier n'est précisé, le répertoire courant est utilisé. Toutes les sous-commandes sont récursives par défaut, mais le comportement inverse peut être obtenu par l'option `-l` (local).

Certaines sous-commandes possèdent une ou plusieurs formes abrégées, comme *co* pour *checkout* par exemple. Elles sont précisées dans le reste du document.

Localisation du dépôt

Le dépôt CVS peut être en local ou sur une machine distante. Lors des opérations CVS, le dépôt à utiliser est indiqué par l'option globale `-d`, suivie de l'emplacement du dépôt. Cet emplacement a l'une des formes suivantes :

`:local:`*répertoire*

`:ext:[`*utilisateur*`@]`*machine*`:`*répertoire*

La première forme indique que le dépôt est un répertoire sur la machine, tandis que la deuxième indique qu'il se trouve sur une autre machine dont on donne le nom et éventuellement un nom de login s'il est différent de celui de l'utilisateur actuel.

Des raccourcis existent pour ces deux formes. Ainsi, préciser seulement le *répertoire* est équivalent à la première forme et `[`*utilisateur*`@]`*machine*`:`*répertoire* est équivalent à la deuxième forme.

Lorsqu'on se trouve dans un répertoire de travail créé par un *checkout* (cf. 2.3.2), il n'est pas nécessaire de préciser l'emplacement du dépôt car celui-ci est stocké dans un fichier `CVS/Root`, présent dans chaque répertoire et sous-répertoire de travail.

Si l'option `-d` n'est pas précisée et qu'on ne se trouve pas dans un répertoire de travail, `cvs` cherche le dépôt à l'endroit indiqué dans la variable d'environnement `CVSROOT`. Au lieu de la commande

cvs -d Endroit_du_dépôt_CVS checkout ...

on peut donc faire :

export CVSROOT=Endroit_du_dépôt_CVS

cvs checkout ....

Nous verrons la sous-commande *checkout* plus loin.

Lorsque le dépôt se trouve sur une machine distante, la commande `cvs` utilise par défaut `rsh` pour se connecter à cette machine. La variable d'environnement `CVS_RSH` permet d'indiquer à `cvs` d'utiliser une autre commande que `rsh` pour se connecter à la machine distante, par exemple :

export CVS_RSH=ssh

afin d'utiliser `ssh` pour la connexion.

Pour notre exemple, le dépôt CVS à utiliser existe déjà. Tous les participants à cette formation accèdent au même, de façon à l'utiliser dans une situation classique de développement collaboratif en concurrence.

Configuration

Il est possible d'utiliser un fichier de configuration personnel `~/.cvsrc` pour indiquer des options globales ou spécifiques à ajouter automatiquement à chaque utilisation de la commande `cvs`. Chaque ligne de ce fichier commence par le nom canonique de la sous-commande concernée pour les options spécifiques, ou cvs pour les options globales.

Par exemple, le fichier suivant aura pour effet d'utiliser systématiquement la compression avec un niveau 6 pour toute communication réseau, et ajoutera les options -Nau a chaque invocation de

la sous-commande `diff` :

```
cvs -z6
diff -Nau
```

2.3.2 Utilisation mono-utilisateur

Récupération d'une copie locale (checkout)

La première chose à faire est de récupérer une copie locale, afin de travailler dessus. En effet, les modifications ne sont jamais faites directement dans le dépôt, mais dans la copie locale. Quand les modifications seront satisfaisantes (implémentation d'une fonctionnalité, rédaction d'un texte, ...), nous les répercuterons dans le dépôt à l'aide des commandes CVS.

Nous créons donc une copie locale :

cvs -d `/local/formation/cvs` **checkout formation**

Cette commande crée une copie locale du module `formation` du dépôt indiqué par l'option `-d`.

La commande indiquée nous a créé, dans le répertoire courant, un répertoire `formation`, car nous n'avons pas précisé un nom différent du nom de module. Pour crée une copie locale dans un répertoire avec un nom différent du module, il faut utiliser l'option `-d` de la sous-commande :

cvs -d `/local/formation/cvs` **co -d autre_nom formation**

La commande crée alors le répertoire `autre_nom` pour y mettre la copie locale.

Voici le contenu du répertoire de travail après extraction :

```
.
|-- CVS
|   |-- Entries
|   |-- Repository
|   `-- Root
`-- index.html
```

Le répertoire `CVS` contient des informations propres à CVS. Le répertoire de travail contient donc un seul fichier `index.html`.

Remarque : l'alias pour la sous-commande `checkout` est `co`.

Ajout de fichiers (add)

Chaque groupe va maintenant ajouter un fichier vide portant le nom de l'un de ses membres :

touch dupond.htm

Ensuite, il faut l'ajouter aux fichiers versionnés :

cvs add dupond.htm

Cette commande réserve le nom du fichier dans le dépôt, afin d'éviter les conflits. CVS indique cependant que l'ajout réel n'aura lieu qu'à la validation par une sous-commande `commit`.

Affichage du statut (status)

La sous-commande `status` permet d'afficher le statut des fichiers présents dans le répertoire de travail.

La commande :
cvs status
Devrait produire à peu près le résultat suivant :

```
cvs status: Examining .
===================================================================
File: dupond.htm        Status: Locally Added

   Working revision:    New file!
   Repository revision: No revision control file
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

===================================================================
File: index.html        Status: Up-to-date

   Working revision:    1.1.1.1 2006-05-11 11:48:17 +0200
   Repository revision: 1.1.1.1 /home/alceste/rousse/cvs/formation/index.html,v
   Commit Identifier:   LBD1nXuxirj6qAwr
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

Pour chaque fichier, elle indique le statut de celui-ci :
- Up-to-date
- Locally Modified
- Locally Added
- Locally Removed

Elle donne également d'autres renseignements, tels que la révision actuelle du dépôt, la révision de travail, un identifiant de la validation, et les éventuelles préférences de synchronisation de la copie locale. Elle est donc relativement verbeuse, ce qui lui fait souvent préférer une mise à jour simulée (cf. 2.3.3) pour juste connaître le statut exact des fichiers.

CVS signale également les fichiers inconnus présents dans le répertoire de travail, précédés d'un point d'interrogation. Typiquement, pour les utilisateurs de vim en train d'éditer le fichier `dupond.htm`, CVS signale un fichier `.dupond.htm.swp` (les utilisateurs d'emacs sont épargnés, le format de nom du fichier temporaire de celui-ci figurant parmi les exceptions connues par CVS). Lorsque le nombre de ces fichiers croit, ceci peut finir à la longue par être gênant pour la lisibilité du résultat des commandes. Il est cependant possible de demander à CVS d'ignorer ces fichiers, en fournissant une liste de patrons (utilisant la syntaxe shell) selon plusieurs moyens :
- un fichier global `~/.cvsignore`
- un fichier local `.cvsignore`
- la variable d'environnement CVSIGNORE

Les utilisateurs de vim peuvent donc tester le résultat de la commande :
CVSIGNORE=.*.swp cvs status

Annulation des modifications

Une modification prévue, mais non encore validée peut être annulée, mais il n'existe pas de commande précise pour le faire, tout dépend du type de modification.

La commande :

cvs remove dupond.htm

permet ainsi d'annuler l'ajout du fichier dupond.htm, comme le prouve une nouvelle exécution de `cvs status`. De même, `add` permet d'annuler un `remove`. Pour annuler une modification, par contre, il faudra forcer une mise à jour avec écrasement des modifications locales avec `cvs update -C`.

Une fois l'annulation vérifiée, le fichier peut de nouveau être ajouté pour continuer.

Validation des modifications (commit)

L'opération de validation consiste à propager les modifications locales vers le dépôt. Elle n'est possible que si la copie locale est à jour, c'est-à-dire s'il n'y a pas eu de modifications extérieures validées dans le dépôt depuis la dernière récupération. Dans le cas contraire, la validation est bloquée.

Lors de la validation, il faut préciser un message, que le système gardera associé au changement d'état qui s'ensuit. Ce message est important car il permet de baliser l'historique des modifications par des commentaires qui permettront par la suite de se repérer plus facilement dans l'historique du fichier. Il convient donc de le soigner, et de faire un résumé synthétique des modifications apportées.

Par défaut, CVS lance un éditeur interactif au moment de la validation pour saisir ce message. Cet éditeur est déterminé soit par une option de la ligne de commande (`-e`), soit par des variables d'environnement (`CVSEDITOR`, et `EDITOR`), soit enfin par une valeur par défaut (vi). Dès que l'on quitte l'éditeur, la validation se termine avec le message saisi, sauf si celui-ci est vide, auquel cas CVS propose d'interrompre celle-ci.

Ainsi, pour valider l'ajout du fichier `dupond.htm`, en utilisant emacs comme éditeur :

CVSEDITOR=emacs cvs commit dupond.htm

On peut court-circuiter ce mécanisme interactif en passant directement le message de validation par l'option `-m`, suivie du message lui-même. C'est ce qui sera fait dans les exemples suivants.

Remarque : l'alias pour la sous-commande `commit` est `ci`.

Modification de fichiers

Le fichier peut maintenant être modifié, avec un éditeur quelconque, ne serait-ce que pour en faire un véritable fichier HTML au lieu d'un fichier vide. Il faut ensuite valider ces changements :

cvs commit -m "ajout de la page sur Monsieur Dupond" dupond.htm

Suppression de fichiers (remove), renommage

L'extension du fichier créé est `.htm`, que l'on souhaite changer en `.html`. Il faut donc le renommer.

CVS ne permet pas le renommage direct des fichiers. Il faut supprimer l'ancien, et ajouter le nouveau :

mv dupond.htm dupond.html
cvs remove dupond.htm
cvs add dupond.html

Il faut noter que CVS se plaint si le fichier n'est pas effectivement supprimé de la copie locale avant la sous-commande `remove`.

Avec ces manipulations, l'historique du fichier avant le renommage est perdu. En effet, CVS considère que l'ancien fichier n'existe plus, et que le nouveau vient d'apparaître, sans aucun lien entre les deux. C'est l'une des faiblesses de CVS, qui le rend peu pratique dans les cas de changement de structure.

En fait, l'ancien fichier n'existe plus dans l'état actuel du projet, mais il reste présent dans le dépôt, puisque un système de gestion de version permet toujours de revenir en arrière. CVS utilise un répertoire particulier, le grenier (*Attic*), pour stocker ces fichiers, que l'on peut récupérer en spécifiant un numéro de révision inférieur à celui correspondant à leur élimination.

Le "renommage" ne devient effectif qu'après une validation :

cvs commit -m "renommage" dupond.htm dupond.html

Affichage de l'historique (log)

L'intérêt de la gestion de version est la conservation de l'historique des modifications. La sous-commande `log` permet d'afficher les modifications d'un ou plusieurs fichiers.

La commande suivante permet d'obtenir l'historique du fichier `dupond.html` :

cvs log dupond.html

Il n'y a qu'une seule révision pour l'instant.

Après une nouvelle modification, validée, la même commande permet cette fois-ci de constater que l'historique contient maintenant deux révisions.

Il est possible de restreindre l'affichage de l'historique par diverses options, indiquées par la commande suivante :

cvs --help log

Affichage des différences (diff)

Il est possible de demander la liste des modifications effectuées dans le répertoire de travail par rapport à la version du dépôt. C'est ce que fait la commande suivante, avec le format par défaut de diff :

cvs diff dupond.html

Il est également possible d'afficher les différences par rapport à une révision précise, voire entre deux révisions d'un fichier. Ainsi, la différence entre les révisions 1.1 et 1.2 du fichier `dupond.html` s'obtient par :

cvs diff -r 1.1 -r 1.2 dupond.html

Enfin, on peut ajouter des options pour changer le format d'affichage des différences, par exemple pour créer un patch :

cvs diff -Nau dupond.html

Note : Au contraire du programme `diff` standard, la sous-commande CVS du même nom est récursive. L'option `-l` peut être utilisée pour l'empêcher.

Mots-clés

Les mots-clés sont des chaînes de caractères qu'il est possible de mettre dans les fichiers gérés par CVS et qui sont remplacées par CVS lors de l'exécution de commandes. Le mot-clé le plus utilisé

est `Id` qui est remplacé par le nom du fichier, le numéro de version, la date de la version et le login de la dernière personne ayant modifié le fichier. Les mots-clés sont placés entre `$` dans les fichiers.

Ajouter le mot-clé `Id` dans `dupond.html`. En général cela est fait en tête du fichier. Comme nous ne voulons pas que cela apparaisse dans l'affichage de la page, nous le mettrons dans un commentaire HTML :

```
<!-- $Id$ -->
```

Ensuite, valider la modification et regarder la ligne du fichier sur laquelle se trouve le mot-clé ajouté.

Fichiers binaires

Les fichiers binaires doivent être traités spécialement, car afficher les différences ligne à ligne n'a pas de sens pour ces fichiers, et la substitution des mots-clés peut même corrompre le fichier.

Pour indiquer qu'un fichier est binaire et doit être traité comme tel, il faut mettre l'option `-kb` lors de l'ajout du fichier.

Pour mettre en pratique cet aspect, chaque groupe ajoute un répertoire `dupond` dans le CVS :

mkdir dupond
cvs add dupond

Maintenant, chaque groupe ajoute une image qui sera utilisée dans les pages HTML que nous construisons :

cp une_image_existante dupond/dupond.jpg

Ensuite, chaque groupe référence l'image depuis sa page `dupond.html` en ajoutant par exemple :

```
<img src="dupond/dupond.jpg"/>
```

Ensuite, on ajoute l'image dans CVS et on valide les modifications. En effet, il est de bon ton de valider en même temps les différentes modifications de fichiers ayant trait à une fonctionnalité ou un aspect commun :

cvs add -kb dupond/dupond.jpg
cvs commit -m "ajout de mon image" dupond/dupond.jpg dupond.html

La commande

cvs log dupond/dupond.jpg

affiche le log du fichier et quelques informations. Sur la ligne "keyword substitution", le `b` indique que le fichier est binaire.

2.3.3 Utilisation multi-utilisateurs

Mise à jour de la copie locale (update)

Bien que chaque groupe ait ajouté ses fichiers, les autres groupes ne voient pas encore ces fichiers dans leur copie locale. Il est donc temps pour chaque groupe de mettre à jour sa copie locale :

cvs update -d

Par défaut, CVS ne met pas à jour les répertoires qui manquent dans la copie locale, d'où l'utilisation de l'option `-d` qui indique à CVS de créer et mettre à jour les répertoires qui n'existaient pas dans le répertoire de travail.

La commande indique les actions réalisées par une lettre suivie du fichier en question. La signification des lettres est la suivante :

Lettre	Signification
A	Le fichier a été ajouté dans la copie locale mais sans validation
R	Le fichier a été supprimé dans la copie locale mais sans validation
U	Le fichier vient d'être mis à jour depuis le dépôt
P	Le fichier vient d'être mis à jour depuis le dépôt par un simple patch
M	Le fichier vient d'être mis à jour depuis le dépôt, en fusionnant les modifications locales avec succès
C	Le fichier vient d'être mis à jour depuis le dépôt, mais la fusion avec les modifications locales a provoqué un conflit
?	Le fichier est inconnu de CVS

Lors de notre mise à jour, nous de devrions avoir que des `U` et éventuellement des `?` car la seule action réalisée par CVS a été de mettre à jour notre copie locale en ajoutant les fichiers et répertoires manquants ajoutés par les autres groupes.

Il peut être intéressant de récupérer cette information sans faire la mise à jour réellement. Cela est permis par l'option globale `-n`, qui indique à CVS de n'effectuer aucune modification dans la copie locale. Ici, cela donnerait :

cvs -n update -d

Enfin, l'option `-P` (pour *Prune*, élaguer en anglais) permet de supprimer les répertoires vides, par exemple ceux qui ne contiennent plus de fichiers gérés par CVS. En effet, il est impossible de retirer un répertoire dans CVS (contrairement à Subversion). Cette option est donc utile pour faire le ménage dans le répertoire de travail.

Remarque : l'alias pour la sous-commande `update` est `up`.

Fusion des modifications

La manipulation qui suit va maintenant consister à effectuer des modifications concurrentes sur un même fichier. Chaque groupe va donc éditer le fichier `index.html` et ajouter dans la partie "Liens" un lien vers son fichier, de la forme `<a href="dupond.html">Dupond</a><br/>`.

Ensuite, chaque groupe valide sa modification :

cvs commit -m "ajout du lien vers dupond.html" index.html

Cependant, ceci ne fonctionne pas. En effet, pour valider une modification, il faut que la copie locale soit à jour, et ici, des modifications ont été apportées dans le dépôt entre temps. Il faut donc d'abord effectuer une mise à jour :

cvs update

CVS signale alors par un `M` que des modifications faites dans `index.html` on été fusionnées (*merge*) avec les modifications du fichier dans la copie locale. Cette fusion est faite automatiquement car les modifications avaient eu lieu à des endroits différents du fichier dans le dépôt et dans la copie locale.

Maintenant que la copie locale est à jour par rapport au dépôt, modulo les modifications locales, il est possible de valider ses modifications :

cvs commit -m "ajout du lien vers dupond.html" index.html

Cependant, ceci ne fonctionne que pour un seul groupe, le premier à effectuer cette validation. La copie locale des autres groupes n'est effectivement plus à jour, et chacun d'entre eux doit fusionner les modifications du premier groupe avec ses propres modifications :

cvs update

Mais cette fois-ci, la mise à jour ne se passe pas correctement. CVS indique par un `C` un conflit dans le fichier `index.html`. Ce conflit est dû au fait que cette fois-ci, les modifications concurrentes concernent les mêmes parties du fichier, et qu'une résolution automatique n'est pas possible. Il faut alors corriger manuellement le problème.

Chaque conflit est signalé dans le fichier incriminé par des marques `<<<<<<<` et `>>>>>>>`, entre lesquelles on trouve d'abord la version locale, puis l'autre version, séparées par une marque `=======`. Il faut résoudre chaque conflit, en fusionnant les deux versions, et en enlevant les marques. Ensuite, il devient possible de valider à nouveau sa modification.

Ici, pour éviter les conflits à répétition, chaque groupe fusionnera puis validera sa modification successivement.

Affichage des auteurs (annotate)

Il est possible d'afficher l'auteur de la dernière modification pour chaque ligne d'un fichier. Ceci permet par exemple de savoir qui a introduit un bogue et de lui faire payer sa tournée de carambars. Ou plus généralement, de demander des explications sur une partie du code à la bonne personne. Ceci s'obtient par :

cvs annotate dupond.html

Sur ce fichier, il ne devrait y avoir pour l'instant, qu'un seul auteur d'indiqué. Par contre, la même commande sur `index.html` :

cvs annotate index.html

montre différents auteurs, selon les lignes.

De même que pour diff, il est possible de spécifier une révision particulière d'un fichier sur laquelle exécuter la sous-commande, plutôt que celle du répertoire de travail, avec l'option `-r`. Par exemple :

cvs annotate -r 1.1 dupond.html

2.3.4 Étiquettes et branches

La mise en place d'étiquettes et de branches de développement sort du cadre de l'utilisation quotidienne d'un système de gestion de version, et correspond à des problématiques de cycle de développement. Leur usage est donc en général réservé à une personne particulière, celle qui a la charge de gérer cette politique. Ce sont les formateurs qui joueront ce rôle ici, en effectuant les opérations concernant l'ensemble du dépôt.

Par ailleurs, étiquettes et branches utilisent des noms symboliques. Avec CVS, ces noms doivent commencer par une lettre, et ne peuvent comporter que des caractères alphanumériques, ainsi que les caractères '-' et '_'. Le point, souvent utilisé dans les numéros de versions, est notamment interdit. Ces noms sont partagés, et ne peuvent donc être utilisés pour désigner à la fois une étiquette et une branche. Enfin, les noms `BASE` et `HEAD` sont déjà réservés pour un usage interne.

Etiquettes (*tags*)

La création d'une étiquette peut se faire de deux manières :

- à partir d'une copie locale, avec la sous-commande `tag`,
- sans copie locale, avec la sous-commande `rtag`.

Dans les deux cas, cette sous-commande prend en argument le nom de l'étiquette :
cvs tag <étiquette>

Dans le cas de l'utilisation d'une copie locale, il vaut mieux s'assurer au préalable que celle-ci est à jour par rapport au dépôt, puisque l'étiquette correspond à l'état de celui-ci. L'option `-c` force cette vérification, et empêche l'étiquetage si des modifications locales sont détectées.

Les formateurs s'occupent de marquer l'ensemble du dépôt avec l'étiquette `v1` :
cvs tag -c v1
Cet étiquetage est immédiat et impacte directement le dépôt. Il ne requiert pas de validation.

Par la suite, chaque fois que l'on fait référence à une révision particulière d'un fichier, on peut ensuite substituer un nom symbolique pour désigner la révision de ce fichier correspondant à ce nom. Par exemple pour demander les différences entre la révision 1.1 et la révision correspondant à l'étiquette `v1` du fichier `index.html` :
cvs diff -r 1.1 -r v1 index.html

Création de branches

La création d'une branche utilise les mêmes sous-commandes `tag` ou `rtag`, suivie du nom de la branche, que la création d'étiquette, mais avec l'option supplémentaire `-b` :
cvs tag -c -b <branche>

Les mêmes remarques que précédemment concernant l'option `-c` par précaution dans le cas de l'utilisation d'une copie locale s'appliquent.

Les formateurs s'occupent de créer une branche de développement consacrée à la mise en place de feuilles de style :
cvs tag -c -b style

La moitié des groupes va continuer à travailler dans la branche principale, l'autre va travailler dans la nouvelle branche. Pour cela, il faut récupérer celle-ci, en passant l'option `-r style`, soit à la sous-commande `checkout` pour extraire une nouvelle copie de travail du dépôt, soit à la sous-commande `update` pour convertir le répertoire de travail actuel. Dans un cas comme dans l'autre, la sous-commande `status` permet de vérifier que l'option `sticky tag` est positionnée sur la branche.

Fusion de branches

La fusion des modifications de la branche `style` avec la branche principale correspond à la répercussion, dans la branche principale, des modifications ayant eu lieu dans la branche `style` depuis sa création.

Nous nous plaçons dans le répertoire de travail correspondant à la branche principale et lançons la commande :
cvs update -j style

Celle-ci demande la fusion des modifications ayant eu lieu dans la branche `style` depuis son ancêtre commun avec la branche correspondant au répertoire de travail. Il est possible que des conflits surviennent lors de cette fusion, il faut alors les résoudre. Il est ensuite recommandé de valider cette fusion en indiquant dans le commentaire de validation de quelle fusion il s'agit.

Il est également recommandé de mettre une étiquette sur la branche, après la fusion, de façon à pouvoir identifier facilement les modifications faites dans la branche depuis cette fusion, et répercuter ces dernières dans le tronc. En effet, sans étiquette pour cet 'endroit' de la branche, il faut utiliser les dates pour indiquer les 'endroits' de la branche dont il faut prendre les différences pour les appliquer au tronc, ce qui est plus délicat et beaucoup plus difficile à mémoriser.

Ainsi, après avoir reporté les modifications de la branche style dans le tronc, nous mettons un tag dans la branche. Pour cela, nous nous plaçons dans un répertoire de travail de la branche et nous faisons :

cvs tag "style-fusion1"

Nous pouvons continuer le développement dans cette branche, sachant que nous avons un nom pour préciser l'endroit de la branche à partir duquel il y aura de nouvelles modifications à reporter dans le tronc.

Ensuite, quand nous voudrons reporter les modifications de la branche dans le tronc, nous ferons :

cvs update -j style-fusion1 -j style

Cela indique qu'il faut reporter les modifications effectuées entre "style-fusion1" et l'état courant de la branche "style".

Ensuite, on met une étiquette sur la branche, pour indiquer qu'une nouvelle fusion a été faite avec le tronc, comme pour la première fusion, avec la commande suivante dans le répertoire de travail de la branche :

cvs tag -m "style-fusion2"

Évidemment, il faut ensuite gérer dans le répertoire de travail du tronc où nous avons intégré les modifications de la branche les conflits éventuels puis, lors de la validation des modifications, indiquer dans le commentaire de quelles modifications il s'agit, par exemple :

cvs commit -m "intégration des modifications de syle-fusion1 a style-fusion2"

Une politique de nommage pour les étiquettes devrait être précisé dans le projet, de façon à garder une cohérence dans les noms et ainsi les retrouver et les comprendre plus facilement. Par exemple, si on fait une release 1.0, on pourra mettre une étiquette rel-1-0 dans le tronc, et créé aussitôt une branche rel-1-0-bugfix pour faire les corrections sur cette release pendant que le développement continue dans le tronc. On pourra de temps en temps reporter les corrections de bugs dans le tronc avec les manipulations indiquées plus haut.

2.4 Subversion

2.4.1 Syntaxe

Forme des commandes

Les commandes Subversion utilisent une syntaxe très proche de CVS :

svn sous-commande options paramètres

La `sous-commande` détermine l'opération à effectuer (comme la récupération ou la validation). Les `options` sont facultatives (le degré de verbosité par exemple), les paramètres sont obligatoires (comme l'url du dépôt).

La commande suivante indique les sous-commandes disponibles :

svn --help

tandis que celle-ci permet de connaître l'utilisation d'une sous-commande précise :

svn --help *sous-commande*

La plupart des sous-commandes prennent en paramètre un ou plusieurs fichiers sur lesquels porte

l'action. Si aucun fichier n'est précisé, le répertoire courant est utilisé. Toutes les sous-commandes sont récursives par défaut, mais le comportement inverse peut être obtenu par l'option `-N` (non-recursive).

Certaines sous-commandes possèdent une ou plusieurs formes abrégées, comme *co* pour *checkout* par exemple. Elles sont précisées dans le reste du document.

Localisation du dépôt

Contrairement à CVS, où il faut indiquer séparément la racine du dépôt et le chemin du module désiré à l'intérieur de celui-ci, Subversion utilise une URL pour désigner ces deux informations à la fois.

Un module dans un dépôt local sera donc accessible par la commande suivante :
svn co file://*répertoire*
et un module dans un dépôt distant par :
svn co svn+ssh://[*utilisateur*@]*machine*/*répertoire*

De même que pour CVS, une fois à l'intérieur du répertoire de travail, il n'est plus nécessaire de préciser l'URL du dépôt.

Configuration

Le comportement de la commande `svn` peut être également configurée globalement, dans le fichier `/etc/subversion/config`, ou pour chaque utilisateur, dans son fichier `~/.subversion/config`. Il s'agit de fichiers texte au format INI classique, dont les options sont abondamment commentées.

2.4.2 Utilisation mono-utilisateur

Récupération d'une copie locale (checkout)

Comme pour CVS, il faut commencer par créer une copie locale.

Cette commande crée un répertoire de travail portant le même nom que le dernier élément de l'URL :
svn checkout `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`

Cette commande crée un répertoire de travail portant le nom précisé en deuxième paramètre :
svn checkout `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/` **formation**

Voici le contenu du répertoire de travail après extraction :

```
.
|-- branches
|-- tags
`-- trunk
```

Ce sont les répertoires conventionnels d'un dépôt Subversion. Les deux premiers servent à accueillir des copies de la branche de développement principale, elle-même contenu dans le troisième, `trunk`. C'est dans ce dernier qu'il convient de travailler pour le moment :
cd formation/trunk

Voici le contenu détaillé de ce répertoire :

```
.
|-- .svn
|   |-- README.txt
|   |-- empty-file
|   |-- entries
|   |-- format
|   |-- prop-base
|   |   `-- index.html.svn-base
|   |-- props
|   |   `-- index.html.svn-work
|   |-- text-base
|   |   `-- index.html.svn-base
|   |-- tmp
|   |   |-- prop-base
|   |   |-- props
|   |   |-- text-base
|   |   `-- wcprops
|   `-- wcprops
`-- index.html
```

Le répertoire `.svn` contient des informations propres à Subversion. À l'inverse de CVS, il s'agit non seulement de la configuration du répertoire de travail, mais également d'une copie du fichier tel qu'il existait dans le dépôt au moment de l'extraction. Ceci permet d'effectuer la plupart des opérations de comparaison localement, sans avoir à contacter le dépôt systématiquement, et de ne transmettre que le différentiel lors de la validation. Il en résulte un usage du réseau largement diminué, au détriment de la taille du répertoire de travail par contre.

Remarque : l'alias pour la sous-commande `checkout` est `co`.

Ajout de fichiers (add)

Chaque groupe va maintenant ajouter un fichier vide portant le nom de l'un de ses membres :
touch dupond.htm

Ensuite, il faut l'ajouter aux fichiers versionnés :
svn add dupond.htm

Cette commande réserve le nom du fichier dans le dépôt, afin d'éviter les conflits. Subversion indique cependant que l'ajout réel n'aura lieu qu'à la validation par une sous-commande `commit`.

Affichage du statut (status)

La sous-commande `status` permet d'afficher le statut des fichiers présents dans le répertoire de travail. La commande
svn status
devrait produire le résultat suivant :

```
A       dupond.htm
```

À la différence de CVS, la sortie de cette commande est largement moins verbeuse par défaut, et n'indique que les fichiers pour lesquels il y a effectivement une différence par rapport au dépôt. Ici il n'y a qu'un fichier sur le point d'être ajouté. Les autres valeurs possibles sont les mêmes que pour la mise à jour (cf. 2.4.3).

Annulation des modifications (revert)

Une modification prévue, mais non encore validée, peut être annulée par la sous-commande `revert`.

La commande :

svn revert dupond.htm

permet ainsi d'annuler l'ajout du fichier dupond.htm, comme le prouve une nouvelle exécution de `svn status`.

Une fois l'annulation vérifiée, le fichier peut de nouveau être ajouté pour continuer.

Validation des modifications (commit)

L'opération de validation consiste à propager les modifications locales vers le dépôt. Elle n'est possible que si la copie locale est à jour, c'est-à-dire s'il n'y a pas eu de modifications extérieures validées dans le dépôt depuis la dernière récupération. Dans le cas contraire, la validation est bloquée.

Lors de la validation, il faut préciser un message, que le système gardera associé au changement d'état qui s'ensuit. Ce message est important car il permet de baliser l'historique des modifications par des commentaires qui permettront par la suite de se repérer plus facilement dans l'historique du fichier. Il convient donc de le soigner, et de faire un résumé synthétique des modifications apportées.

Par défaut, svn lance un éditeur interactif au moment de la validation pour saisir ce message. Cet éditeur est déterminé soit par une option de la ligne de commande (`--editor-cmd`), soit par une option de configuration (`editor-cmd`), soit par des variables d'environnement (`SVN_EDITOR`, `VISUAL` et `EDITOR`), soit enfin par une valeur par défaut (généralement vi). Dès que l'on quitte l'éditeur, la validation se termine avec le message saisi, sauf si celui-ci est vide, auquel cas Subversion propose d'interrompre celle-ci.

Ainsi, pour valider l'ajout du fichier `dupond.htm`, en utilisant emacs comme éditeur :

SVN_EDITOR=emacs svn commit dupond.htm

On peut court-circuiter ce mécanisme interactif en passant directement le message de validation par l'option `-m`, suivie du message lui-même, ou alors par l'option `-f`, suivie du nom d'un fichier contenant ce message. L'option `-m` sera utilisée dans les exemples suivants.

Remarque : l'alias pour la sous-commande `commit` est `ci`.

Modification de fichiers

Le fichier peut maintenant être modifié, avec un éditeur quelconque, ne serait-ce que pour en faire un véritable fichier HTML au lieu d'un fichier vide. Il faut ensuite valider ces changements :

svn commit -m "ajout de trucs et machins" dupond.htm

Renommage de fichiers (move)

L'extension du fichier créé est `.htm`, que l'on souhaite changer en `.html`. Il faut donc le renommer.

A la différence de CVS, Subversion gère la chose parfaitement :

svn move dupond.htm dupond.html

Le renommage ne devient effectif qu'après une validation :

svn commit -m "renommage" dupond.htm dupond.html

Remarque : les alias pour la sous-commande `move` sont `mv`, `rename` et `ren`.

Affichage de l'historique (log)

L'intérêt de la gestion de version est la conservation de l'historique des modifications. La sous-commande `log` permet d'afficher les modifications d'un ou plusieurs fichiers.

La commande suivante permet d'obtenir l'historique du fichier `dupond.html` :

svn log dupond.html

Il n'y a que deux révisions pour l'instant.

Après une nouvelle modification validée, la même commande permet cette fois-ci de constater que l'historique contient maintenant trois révisions.

Il est possible de restreindre l'affichage de l'historique par diverses options, indiquées par la commande suivante :

svn help log

Affichage des différences (diff)

Il est possible de demander la liste des modifications effectuées dans le répertoire de travail par rapport à la version du dépôt. C'est ce que fait la commande suivante, avec le format unifié de diff :

svn diff dupond.html

Il est également possible d'afficher les différences par rapport à une révision précise, voire entre deux révisions. Ainsi, la différence du fichier `dupond.html` entre les révisions X et Y s'obtient par :

svn diff -r X:Y dupond.html

Par exemple, prendre un X et un Y d'après le log du fichier obtenu par la commande du paragraphe précédent.

Remarque : l'alias pour la sous-commande `diff` est `di`.

Alias des révisions

Chaque dépôt subversion possède un numéro unique de révision, automatiquement incrémenté à chaque modification. Ces numéros de révisions peuvent être utilisés tels quels dans un certain nombre de commandes, pour préciser l'objet de ces commandes, comme montré précédemment.

Mais il est également possible d'utiliser certains alias pour les cas les plus courants :

- HEAD correspond à la révision la plus récente du dépôt
- BASE correspond à la révision locale avant modification, telle qu'elle a été extraite du dépôt
- COMMITTED correspond à la dernière révision pour laquelle une modification a été validée
- PREV correspond à la révision située juste avant COMMITTED

La commande suivante permet ainsi d'examiner les modifications survenues lors de la dernière validation :

svn diff -r PREV:COMMITTED dupond.html

Celle-ci permet d'examiner les modifications intervenues dans le dépôt depuis la dernière mise à jour :

svn diff -r BASE:HEAD dupond.html

Propriétés

Les propriétés sont des meta-données, versionnées elles aussi, que l'on peut ajouter aux fichiers. Un certain nombre de ces propriétés sont déjà définies par Subversion, et sont utilisées par le client pour effectuer des traitements particuliers sur certains fichiers. Il est également possible de définir des propriétés arbitraires supplémentaires, et de s'en servir pour ses besoins spécifiques.

Toutes ces propriétés se manipulent par les commandes suivantes :

svn propset <clé> <valeur>
svn propdel <clé>
svn propget <clé>
svn proplist
svn propedit <clé> <valeur>

Les quatre premières permettent respectivement de positionner une propriété, d'effacer une propriété, d'afficher la valeur d'une propriété et de lister l'ensemble des propriétés existantes. La dernière permet d'éditer la valeur d'une propriété avec un éditeur interactif (selon les mêmes modalités que le message de validation).

Les propriétés définies par Subversion ont toutes un nom préfixé par `svn:`. Voici la liste des principales :

- `svn:keywords`,
- `svn:eol-style`,
- `svn:mime-type`,
- `svn:executable`,
- `svn:ignore`.

La propriété `svn:keywords` autorise l'expansion de la listes des mots-clés correspondant à sa valeur. Celle-ci n'est en effet plus effectuée par défaut, pour protéger l'intégrité des fichiers binaires.

La propriété `svn:eol-style` autorise la conversion à la volée des caractères délimitant les fins de ligne, afin de les normaliser entre les différentes plateformes. Ceci n'est pas fait par défaut, encore une fois pour protéger l'intégrité des fichiers binaires.

La propriété `svn:mime-type` permet à Subversion de ne pas utiliser un diff contextuel sur les fichiers binaires (ceux dont le type MIME n'est pas `text/`), ainsi que de positionner correctement le type MIME dans les communications http.

La propriété `svn:executable` autorise l'ajout à la volée du bit exécutable sur les systèmes où il est nécessaire.

La propriété `svn:ignore`, définie sur un répertoire, permet d'ignorer les fichiers correspondant à la liste de patrons définie par sa valeur, de la même façon que les fichiers `.cvsignore` de CVS.

L'insertion du mot-clé `$Id$` dans le fichier `dupond.html` suivie d'une validation est sans effet pour le contenu du fichier (le mot-clé n'est pas remplacé) tant que la propriété `svn:keywords` n'est pas positionnée pour effectuer cette substitution : La commande

svn proplist dupond.html

n'indique aucune propriété pour ce fichier.

Mettre les propriétés `svn:keywords`, `svn:eol-style` et `svn:mime-type` sur `dupond.html` :

svn propset svn:keywords "Id" dupond.html
svn propset svn:eol-style "LF" dupond.html
svn propset svn:mime-type "text/html" dupond.html

Afficher la liste des propriétés du fichier `dupond.html` :

svn proplist dupond.html

Valider les modifications :
svn commit -m"propriétés" dupond.html
et regarder le contenu du fichier : le mot-clé `$Id$` a été remplacé.

Un exemple d'utilisation de propriétés non utilisées par Subversion est le suivant. Imaginons que l'on gère un site web avec Subversion. On définit une propriété "copyright" à laquelle on associe une chaîne de caractères indiquant un détenteur de droits. On met cette propriété sur les fichiers d'images, par exemple celles apparaissant dans une galerie de photos du site. Lors de la mise en ligne, on peut se servir de cette propriété pour modifier au vol l'image afin d'ajouter l'indication de *copyright* dans l'image elle-même. Ainsi, les images dans le dépôt sont les images originales, tandis que celles en ligne comportent la mention de *copyright*.

Fichiers binaires

Il ne faut pas mettre les propriétés `svn:keywords` ou `svn:eol-style` sur les fichiers binaires car ces derniers pourraient en être altérés.

Pour mettre en pratique cet aspect, chacun crée un répertoire `dupond` dans sa copie locale :
mkdir dupond
svn add dupond
Ces deux opérations peuvent être effectuées par
svn mkdir dupond

Maintenant, chaque groupe ajoute une image qui sera utilisée dans les pages HTML que nous construisons :
cp une_image_existante dupond/dupond.jpg
Ensuite, chaque groupe référence l'image depuis sa page `dupond.html` en ajoutant par exemple :
`<img src="dupond/dupond.jpg"/>`
Ensuite, on ajoute l'image dans Subversion et on valide les modifications. En effet, il est de bon ton de valider en même temps les différentes modifications de fichiers ayant trait à une fonctionnalité ou un aspect commun :
svn add dupond/dupond.jpg
svn commit -m "ajout de mon image" dupond/ dupond.html

La commande
svn proplist dupond/dupond.jpg
indique que Subversion a trouvé tout seul le type MIME du fichier et positionné la valeur de la propriété `svn:mime-type` en conséquence.

2.4.3 Utilisation multi-utilisateurs

Mise à jour de la copie locale (update)

Bien que chaque groupe ait ajouté ses fichiers, les autres groupes ne voient pas encore ces fichiers dans leur copie locale. Il est donc temps pour chaque groupe de mettre à jour sa copie locale :
svn update

La commande indique les actions réalisées par une lettre suivie du fichier en question. La signification des lettres est la suivante :

Lettre	Signification
A	Le fichier a été ajouté
D	Le fichier a été supprimé
U	Le fichier vient d'être mis à jour depuis le dépôt
G	Le fichier vient d'être mis à jour depuis le dépôt, en fusionnant les modifications locales avec succès
C	Le fichier vient d'être mis à jour depuis le dépôt, mais la fusion avec les modifications locales a provoqué un conflit

Ici, la mise à jour ne devrait afficher que des ajouts (`A`), puisque les seules modifications dans le dépôt concernent des nouveaux fichiers et répertoires.

Remarque : l'alias pour la sous-commande `update` est `up`.

Remarque (bis) : la sous-commande `update` met à jours les nouveaux répertoires sans qu'il soit nécessaire de le spécifier, comme c'est le cas avec l'option `-d` de CVS.

Fusion des modifications

La manipulation qui suit va maintenant consister à effectuer des modifications concurrentes sur un même fichier. Chaque groupe va donc éditer le fichier `index.html` et ajouter dans la partie "Liens" un lien vers son fichier, de la forme `<a href="dupond.html">Dupond</a><br/>`.

Modificat
concurren
par
formateur

Ensuite, chaque groupe valide sa modification :

svn commit -m "ajout du lien vers dupond.html" index.html

Cependant, ceci ne fonctionne pas. En effet, pour valider une modification, il faut que la copie locale soit à jour, alors qu'ici des modifications ont été apportées dans le dépôt entre temps. Il faut donc d'abord effectuer une mise à jour :

svn update

Cette mise à jour se passe sans encombre, mais à la différence de la précédente, Subversion effectue une mise à jour avec fusion des modifications, signalée par un `G` devant le nom du fichier. Ceci est possible, car les modifications locales et les modifications en provenance du dépôt concernaient des endroits différents du fichier.

Chaque groupe peut à nouveau valider sa modification :

svn commit -m "ajout du lien vers dupond.html" index.html

Cependant, ceci ne fonctionne que pour un seul groupe, le premier à effectuer cette validation. La copie locale des autres groupes n'est effectivement plus à jour, et chacun d'entre eux doit fusionner les modifications du premier groupe avec ses propres modifications :

svn update

Mais cette fois-ci, la mise à jour ne se passe pas correctement. Subversion indique par un `C` un conflit dans le fichier `index.html`. Ce conflit est dû au fait que cette fois-ci, les modifications concurrentes concernent les mêmes parties du fichier, et qu'une résolution automatique n'est pas possible. Il faut alors corriger manuellement le problème.

Chaque conflit est signalé par des marques `<<<<<<<` et `>>>>>>>`, entre lesquelles on trouve d'abord la version locale, puis l'autre version, séparées par une marque `=======`. Il faut résoudre chaque conflit, en fusionnant les deux versions et en enlevant les marques. Enfin, contrairement à CVS, il faut explicitement indiquer à Subversion que le conflit a été résolu, par la commande suivante :

svn resolved index.html
Ensuite, il devient possible de valider à nouveau sa modification. Cette procédure permet d'avoir des <<<<<< et >>>>>> dans le fichier sans que Subversion en déduise qu'il s'agit d'un conflit non résolu.

Ici, pour éviter les conflits à répétition, chaque groupe fusionnera puis validera sa modification successivement.

Affichage des auteurs (blame)

Il est possible d'afficher l'auteur de la dernière modification pour chaque ligne d'un fichier. Ceci permet par exemple de savoir qui a introduit un bogue et de lui faire payer sa tournée de carambars. Ou, plus généralement, de demander des explications sur une partie du code à la bonne personne. Ceci s'obtient par :
svn blame dupond.html

Pour l'instant, il n'y a qu'un seul auteur, puisque chaque équipe a travaillé sur son propre fichier. Ceci changera dans les manipulations suivantes.

De même que pour diff, il est possible de spécifier une révision particulière du dépôt sur laquelle exécuter la commande, plutôt que celle du répertoire de travail, avec l'option `-r`. Par exemple :
svn blame -r X dupond.html
avec X un numéro de version de `dupond.html` (utiliser `svn log dupond.html` pour voir les numéros de version).

Remarque : les alias pour la sous-commande `blame` sont `annotate`, `praise` et `ann`.

2.4.4 Étiquettes et branches

Les mêmes remarques que pour CVS concernant l'utilisation des étiquettes et des branches s'appliquent. Ici encore, les manipulations concernant l'ensemble du dépôt seront effectuées par les formateurs.

Par contre, il n'y a plus de contraintes concernant la syntaxe des noms à utiliser pour les étiquettes ou pour les branches, ce sont des noms de répertoires quelconques pour Subversion.

Étiquettes (*tags*)

L'intérêt des étiquettes sous Subversion est réduit par rapport à CVS, puisque chaque révision correspond directement à un état global du dépôt. Néanmoins, il reste plus facile d'utiliser des noms symboliques plutôt que des numéros pour se référer à un état précis, comme par exemple 'release-1.1', plutôt que '488'.

L'étiquetage d'un dépôt se résume à une simple copie de la branche de développement courante sous le nom désiré, par convention dans la section `tags` du dépôt :
cd ..
svn copy trunk tags/release-1.1

Cet étiquetage peut même se faire sans copie locale, en utilisant directement des URLs comme arguments de la sous-commande `copy` :
svn copy `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**trunk** `svn+ssh://formation1-`
1.1

Dans ce dernier cas, cette commande implique une modification immédiate dans le dépôt. Elle requiert donc aussi un message de validation, selon les mêmes modalités que la sous-commande `commit`.

Création d'une branche

La création d'une branche dans Subversion se fait également par copie de la branche de développement courante sous le nom désiré, par convention dans la section `branches` du dépôt :
svn copy trunk branches/style

De même que précédemment, ceci peut se faire sans copie locale :
svn copy -m "nouvelle branche" `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**trunk**
`svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**branches/style**

Si les commandes sont les mêmes, c'est qu'en fait Subversion n'attache aucune sémantique particulière à la notion de branche ou d'étiquette : toute copie de répertoire constitue de facto une nouvelle branche potentielle. Les noms de répertoires de plus haut niveau classiquement utilisés dans un dépôt Subversion (`trunk`, `branches`, `tags`) sont purement conventionnels, et correspondent à cet usage.

Fusion des branches

La sous-commande `merge` permet de fusionner des modifications provenant des deux branches de développement. Comme chaque branche est en fait une "simple copie" du répertoire contenant le projet, cette sous-commande revient en fait à extraire les différences entre deux hiérarchies différentes du dépôt (comme diff), et à appliquer le résultat à une copie de travail (comme patch), à la différence près que ces différences concernent également les répertoires et les renommages.

Pour fusionner deux branches, l'idée instinctive consiste à se placer dans la branche destination et à utiliser `merge` en lui indiquant l'autre branche comme origine. Or c'est une erreur... Comme cette sous-commande commence par extraire les différences entre les deux hiérarchies, l'utiliser de cette façon revient à éliminer de la branche principale toutes les modifications qui y ont eu lieu depuis la création de l'autre branche.

La bonne solution consiste en fait à appliquer à la branche cible la différence entre la révision du répertoire au moment de la création de la branche et l'état actuel de la branche dont on veut importer les modifications.

Pour trouver facilement cette révision initiale, le plus simple est d'utiliser la sous-commande `log` avec l'option `--stop-on-copy` :
svn log --verbose --stop-on-copy `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**bran**

Au lieu de remonter dans l'historique jusqu'au début du dépôt, la commande s'arrête dès qu'elle détecte une copie commune, ce qui correspond en fait à la création de la branche. On note XXX le numéro de cette révision.

Ensuite, on peut connaître la révision correspondant à l'état courant de la branche `style`, on se place dans le répertoire de la branche style, et on lance la commande
svn update
qui indique la dernière révision de la branche, sous la forme
`At revision YYY.`

Il ne reste plus qu'à effectuer la fusion proprement dite :
svn merge -r avant:après `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**branches/sty**
avec avant et après étant ici respectivement XXX et YYY. En nous plaçant dans le répertoire de travail du tronc, la commande suivante répercute donc les modifications de la branche style dans le notre répertoire de travail :
svn merge -r XXX:YYY `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**branches/sty**

Ensuite, il faut résoudre les conflits éventuels, puis valider les modifications en indiquant dans le commentaire de quel fusion il s'agit. Il est préférable de ne pas indiquer "style" comme numéro de révision mais le vrai numéro, car "style" évoluera en même temps qu'on modifie la branche style. Nous faisons donc :
svn commit -m"fusion de la branche style XXX -> YYY dans le tronc."
avec `XXX` le numéro de révision actuel de la branche style. Il n'est pas nécessaire de mettre ensuite une étiquette à la branche comme dans CVS, car Subversion a déjà affecté un numéro unique de révision que nous utiliserons pour répercuter les modifications ultérieures de la branche dans le tronc.

Ainsi, supposons que des modifications successives de la branche style aboutissent à une révision ZZZ, nous pouvons répercuter les modifications entre YYY et ZZZ dans le tronc par la commande suivante :
svn merge -r YYY:ZZZ `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**branches/styl**
Les révisions entre YYY et ZZZ qui ne concernent pas la branche style ne sont bien sûr pas prises en compte.

De la même manière, si le développement dans une branche est long, on peut souhaiter synchroniser la branche avec le tronc, de façon à minimiser les conflits lorsqu'on reportera les modifications de la branche dans le tronc. On fait cela de la même manière, par exemple en supposant que nous voulons reporter dans la branche les modifications du tronc entre les révisions YYY et ZZZ : nous nous plaçons dans le répertoire de travail de la branche et nous exécutons la commande
svn merge -r YYY:ZZZ `svn+ssh://formation1-rocq.inria.fr/local/formation/svn/`**trunk**
Ensuite, on valide les modifications (après résolution de conflits éventuels) en indiquant dans le commentaire de quelle fusion il s'agit :
svn commit -m"modifications du tronc de YYY à ZZZ".

Subversion ne conserve pas trace du fait que les différences entre telles révisions de telle branche ont été reportées dans telle autre branche. Il est donc important de le noter, de façon à ne pas reporter deux fois les mêmes modifications, ce qui peut avoir des effets néfastes. Le minimum est donc d'indiquer ces fusions dans le commentaire, et si possible également dans un fichier ChangeLog.

3 Conclusion

3.1 Bilan

Nous avons vu les différentes utilisations de CVS et Subversion en ce qui concerne la modification des fichiers et répertoires gérés. D'autres aspects, davantage orientés "administration" seront exposés dans une prochaine formation (création du dépôt, gestion des droits sur le dépôt, export, contrôle avant validation, ...).

Même si CVS permet de faire de la gestion de version, y compris les branches, Subversion apporte tout de même un plus grand confort, et surtout des possibilités qui manquent à CVS, comme le

renommage des fichiers et la gestion des répertoires.

En tout cas, vous n'avez plus d'excuse pour ne pas utiliser de gestionnaire de version !

Pour aller plus loin, voici quelques liens vers des outils et documentations.

3.2 Ressources pour CVS

Il existe d'autres clients CVS que la commande `cvs`. On peut notamment citer :
- VC (mode Emacs)
- cvscommand (plugin Vim)
- Cervisia
- Eclipse
- TortoiseCVS

Des documentations sont disponibles en ligne :
- Documentations de référence, didacticiels, FAQs, ...
- Didacticiel en français
- Introduction à CVS, en français
- Utiliser CVS sous AFS à Rocquencourt

3.3 Ressources pour Subversion

Il existe d'autres clients Subversion que la commande `svn`. On peut notamment citer :
- psvnl (mode Emacs)
- svncommand (plugin Vim)
- subclipse (plugin eclipse)
- TortoiseSVN
- RapidSVN
- eSvn

Des documentations sont disponibles en ligne :
- Le livre de référence sur Subversion
- La page de Subversion chez Tigris