Introduction de subversion dans un groupe de travail

Karl Heinz Marbaise

Le contrôle de version et le suivi des modifications d'un logiciel sont de plus en plus exigés, et cela vaut non seulement pour les grands projets en équipe mais concerne tout aussi bien les petites équipes. Le contrôle de version constitue la pierre angulaire de l'assurance de qualité dans le cadre du développement logiciel, et le choix de l'outil est la question qui se pose d'emblée.

D'autres questions se posent, comme le coût et l'ampleur du travail pour maîtriser un tel outil. Sur ces questions, le logiciel open-source se présente bien et Subversion vient immédiatement à l'esprit, car bien que jeune il intègre des concepts solides et résout de nombreux problèmes de son prédécesseur CVS (Concurrent Versions Systems).

1. Le concept de Subversion

Le concept de base de Subversion réside dans une structure centralisée, au sein de laquelle les modifications apportées au projet sont codifiées à l'intérieur de ce que l'on appelle un Repository (ou répertoire de synchronisation). Cela signifie que toutes les modifications apportées aux fichiers/répertoires sont décrites à l'aide de Subversion et seront disponibles à l'avenir. De plus, il est possible de rétablir chacune de ces modifications et, aussi de spécifier la date et l'auteur d'une modification.

L'installation

Il y a principalement trois manières de mettre en œuvre Subversion : l'installation orientée fichier, l'installation en tant que démon resp. Service (windows) ou en tant que module du serveur web Apache (http resp. https). L'expérience m'a appris à l'installer directement en tant que module Apache, car après quelque temps, apparaissent des exigences que seuls cette méthode permet de satisfaire. La méthode d'installation au moyen du serveur web Apache sera présentée ici, car c'est la plus délicate mais aussi la plus flexible.

Installation sous Linux

Vous pouvez utiliser des fichiers RPM déjà prêts, auquel cas il vous faudra prendre connaissance de leur contenu exact. L'autre possibilité consiste à installer Subversion à partir des fichiers source; dans ce dernier cas, il vous faudra avoir une expérience des installations et des compilations à partir de paquetages source.

Sur l'auteur :

Karl Heinz Marbaise, Dipl.Ing.(FH), Travailleur indépendant dans le domaine de la gestion des configurations et du développement logiciel. Auteur spécialisé et formateur. dans le domaine de la gestion des configurations et des languages de programmation.

Pour contacter l'auteur : *sdj@soebes.de*.

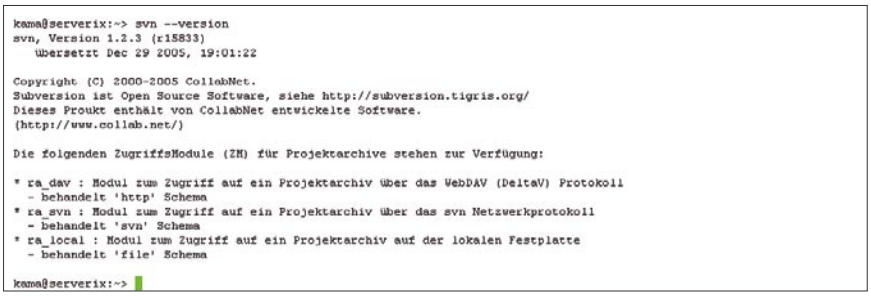

Figure 1. *svn appel après l'installation*

Si vous n'avez pas cette expérience, je ne peux que vous conseiller d'éviter cette méthode, et de la confier éventuellement à quelqu'un qui s'y connaît ;-). Les autres prés-requis pour une installation sous Linux sont un serveur web Apache (Release 2.0), une DB Berkeley (au moins la 4.2, optionnelle) et quelques autres paquetages. Il faut lire exactement les instructions d'installation et installer éventuellement les paquetages manquants. Il vous faudra simplement télécharger le paquet source sur [1] et l'installer par le jeu de commandes (configure; make; make install), le tout bien sûr en tant qu'utilisateur root. Subversion sera alors installé dans /usr/local, (préfixe par défaut qui peut être modifié en argument de configure).

Mise en place d'un Repository

L'étape importante qui suit consiste à trouver un emplacement adéquat pour un Repository, et à le créer par Subversion, au moyen de la commande svnadmin create Repository chemin, ceci devant se passer sans message d'erreur. La mise en place d'un Repository ne doit être effectué qu'une fois.

Installation dans le serveur web Apache

Pour rendre possible le travail collaboratif entre le serveur web et Subversion, il vous faut copier à partir du répertoire source (sous unix) les modules mod_dav_svn et mod_authz_svn dans le répertoire des modules du serveur web. Sous Windows, il vous faudra les copier à partir du répertoire bin. Il faudra s'assurer bien sûr que les modules sont chargés, c'est à dire que les les entrées LoadModul correspondantes sont renseignées comme dans la table 1. En outre, si le serveur a déjà été lancé, il doit être redémarré. Pour faciliter l'accès au serveur Subversion, celui-devrait se faire par une entrée dans l'intranet, il peut aussi se faire par une adresse IP, avec l'inconvénient que cela n'est pas aussi simple à mémoriser que svn.domain.tld par exemple. Cela suppose qu'un serveur local de noms soit présent; dans le cas le plus simple on peut aussi ajouter une entrée dans le fichier hosts. La mise en oeuvre d'un DNS et de la configuration correspondante simplifie quelque peu la procédure.

Il nous faut donner un nom pour la société imaginaire Virtual Software Corporation : nous souhaitons gérer tout cela dans le réseau local et non dans l'internet. Nous utiliserons donc svn.visoco.local.

La configuration d'un VHOST dans un serveur web Apache se passe comme indiqué dans le tableau : Configuration SVN Apache. Sous Windows il faut compléter la directive SVNPath“ par un nom de lecteur et bien sûr par un chemin vers les fichiers de log. Il faudra redémarrer le serveur Apache, ce qui devrait se produire sans message d'erreur indiquant que le configuration, ou des paramètres de configuration ne sont pas corrects. Si l'installation est correcte, l'url *svn list http://svn.visoco.local* est disponible. Sous linux, Il est très important que les droits user/group des répertoires, dans lesquels se trouvent les Repository soient identiques à ceux indiqués dans le fichier httpd.conf du serveur web, afin que celui-ci ait aussi le droit d'accès aux répertoires et aux fichiers. Si ce n'est pas le cas, vous obtiendrez avec la requête précédente les messages correspondants (tableau 3 : Droits erronés pour le Repository).

Le premier projet

Après une installation réussie, nous pouvons envisager d'intégrer notre premier projet dans Subversion. Comment cela se passe-t-il? On importe une ébauche de projet dans le Repository. Supposons que nous ayons établi une ébauche de projet dans le répertoire projekt1 sur une machine et un disque donnés. Nous pouvons alimenter cette ébauche par la commande *svn import . http://svn.visoco.local/projekt1/trunk -m“- Start“*. Les options et les paramètres seront expliqués plus tard. Selon les circonstances, l'importation peut prendre un peu de temps. Cela dépend entre autres choses de l'importance du projet. Les écrans correspondants ressemblent à ceci :

Après la réussite de l'importation, nous donnons un nom au répertoire que nous avons importé, par ex. Projekt1-imp. Pour pouvoir poursuivre le travail sur le projet,

Listing 1. *Apache Konfiguration*

```
LoadModule dav_module
                      modules/mod_
                      dav.so
LoadModule dav_fs_module
                      modules/mod_
                      dav_fs.so
LoadModule dav_svn_module
                      modules/mod_
                      dav_svn.so
LoadModule authz_svn_module
                      modules/mod_
                      authz_svn.so
```

Listing 2. *SVN Apache Konfiguration*

```
<VirtualHost 192.168.1.2:80>
    ServerAdmin webmaster@dummy-host.example.com
    ServerName svn.visoco.local
    ServerAlias svn
    ServerAlias svn.server
    <Location />
        DAV svn
        SVNPath  RepositoryPfad
    </Location>
    ErrorLog /var/log/httpd/vhosts/svn/error.log
    CustomLog /var/log/httpd/vhosts/svn/access.log Combined
</VirtualHost>
```

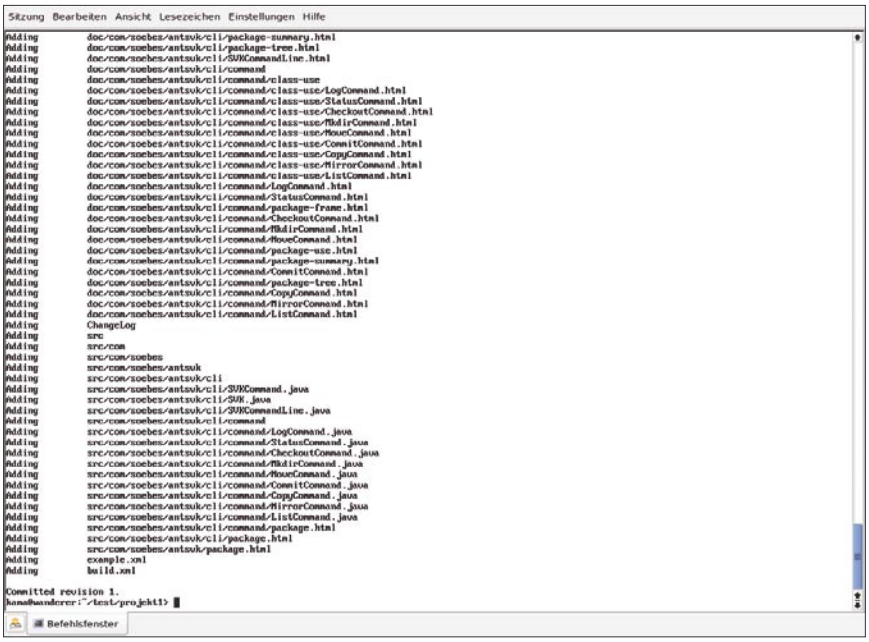

Figure 2. *Importation du premier pojet*

il nous faut retirer une copie du Repository, avec laquelle nous pouvons travailler. Nous faisons cela par la commande *vn checkout http://svn.visoco.local/projekt1*

Sur l'écran l'affichage ressemble à la Figure 3.

Le répertoire dans lequel nous avons importé s'appelle Working-Copy (WC) ou Bac à sable. Dans le Bac à sable, on peut maintenant commencer à apporter des modifications au projet.

Supposons maintenant qu'une collaboratrice (appelons la Nadine) commence à apporter des modifications au projet. Nadine a importé une instance de Subversion et l'a déposée dans son bac à sable. C'est généralement le répertoire Home de la personne, mais pas nécessairement.

Listing 3. *Falsche Rechte für das Repository*

```
svn: PROPFIND request failed on '/'
svn:
Could not open the requested SVN
                        filesystem
```

Listing 4. *Einchecken von Änderungen*

```
--This line, and those below, will
                        be ignored--

M    src/com/soebes/antsvk/cli/
SVKCommand.java
M    src/com/soebes/antsvk/cli/
                        SVK.java
M    src/com/soebes/antsvk/cli/
SVKCommandLine.java
M    ChangeLog
```

Listing 5. *Einchecken abbrechen*

```
Log message unchanged or not
                        specified
a)bort, c)ontinue, e)dit
```

Donnez nous notre modification quotidienne

Nadine modifie alors différents fichiers du projet et teste ses modifications. Après un test rigoureux, elle décide de sauvegarder à nouveau les modifications. Elle doit donc reporter les modifications dans le Repository, en allant dans le répertoire racine du projet et en lançant un svn commit. Subversion teste alors quels fichiers ont été modifiés et propose alors un éditeur pour saisir un rapport des modifications. (tableau 4: envoi des modifications). Dans les lignes qui suivent „ - This line, and those below, will be ignored--“; elle peut décrire les modifications faites. À ce niveau, il est important d'écrire un résumé des modifications ou éventuellement une indication concernant un bug, par ex. - Bug #12456 fixed.“. Après l'avoir fait, on peut quitter l'éditeur, la façon de faire dépendant de l'éditeur; ici vi ayant été appelé, on peut le quitter par Shift-ZZ“ ou : wq“. Le rapport est ainsi mémorisé et Subversion sait désormais que nous pouvons retrouver ce texte en tant que rapport (Tableau 6: envoi correct). Si vous vous êtes trompés et si vous n'avez pas l'intention d'effectuer le commit, vous pouvez aussi interrompre l'éditeur. Dans le cas de vi, ce sera par „:q!“. Ce faisant, vous indiquez à Subversion que vous ne voulez pas rendre les modifications permanentes et vous pouvez alors l'interrompre par „a“ (tableau 5:commande annulation) ou „c“ continuer malgré tout, mais avec un rapport vide (ce qu'il ne faudrai pas faire!) ou par „e“ pour appeler à nouveau l'éditeur. Une fois les modifications faites et inscrites dans le Repository, Nadine peut effectuer de nouvelles modifications.

En fin de semaine

Vendredi, Nadine a effectué quelques modifications et est partie l'esprit tranquille en week-end. Elle revient le lundi et suite à un week-end chargé, elle a oublié ce qu'elle a précisément modifié. Là aussi Subversion peut être utile. En premier lieu, elle voudrait préciser les fichiers qu'elle a modifiés. Cela est obtenu simplement et rapidement par la commande *svn status* (Tableau 7 : *svn status*). La lettre „M“ signifie „Modified“, bien évidemment. Bien, Nadine sait quels fichiers ont été modifiés, mais pas encore exactement ce qui a été modifié. Cela est obtenu immédiatement par *svn diff*. Et on peut voir (Tableau 4) que les lignes marquées par un „+“ out été ajoutées, tandis que celles marquées par „-“ ont été enlevées. Il faut dire que ce n'est pas la manière unique ni la la plus simple de visualiser les différences entre la copie de travail et le Repository. Cela peut être fait par TortoiseSVN (Windows) ou par ex. Dans Eclipse (voir tableau 5) pour une présentation plus lisible des modifications.

Le grand jour

Le développement avance avec le temps. Nadine a effectué des milliers de modifications et d'extensions et vient alors l'échéance de la livraison au client de son logiciel. Nadine doit fixer l'étape exacte qui lui sera fournie. Cela peut être fait simplement par Subversion à l'aide de ce que l'on appelle un tag. Pour ce faire, il faut un répertoire *tags* qu'on aura créé auparavant par *svn mkdir http://svn.visoco.local/projekt1/tags -m“- Tags“*. L'option „-m“ et le texte correspondant est une notification des modifications; elle peut être aussi utilisée lors des nécessaires prises en compte des modifications; Nadine créé le tag pour la release 1.0.0 par *svn copy*

Listing 6. *Einchecken erfolgreich*

```
Sending        ChangeLog
Sending        src/com/soebes/antsvk/cli/SVK.java
Sending        src/com/soebes/antsvk/cli/SVKCommand.java
Sending        src/com/soebes/antsvk/cli/SVKCommandLine.java
Transmitting file data ....
Committed revision 2.
```

Listing 7. *svn status*

```
kama@wanderer:~/test/projekt1> svn status
M       src/com/soebes/antsvk/cli/command/LogCommand.java
M       src/com/soebes/antsvk/cli/command/CheckoutCommand.java
M       src/com/soebes/antsvk/cli/command/CommitCommand.java
M       src/com/soebes/antsvk/cli/command/MirrorCommand.java
M       src/com/soebes/antsvk/cli/command/CopyCommand.java
```

*http://svn.visoco.local/projekt1/trunk http://svn.visoco.local/projekt1/tags/RELEASE-1.0.0 -m"- Release 1.0.0".*L'étape du développement est marquée et Nadine; rassurée n'a plus qu'à noter quelle Release (tag) a été livrée à quel client. La création d'un tag peut aussi être faite par les outils déjà mentionnés, si la commande parait trop compliquée. Dès que la livraison et l'installation du logiciel a été effectuée chez le client, Nadine peut se consacrer à nouveau à son développement et y ajouter de nouvelles fonctionnalités.

La catastrophe

Après quelques semaines le client se fait connaître et annonce avoir trouvé une erreur. Il est vital de trouver rapidement cette erreur et de livrer la version corrigée au client. Nadine peut retirer du Repository la Release fournie au client en utilisant simplement la commande *svn checkout http://svn.visoco.local/projekt1/tags/RELEASE-1.0.0.* Elle dispose alors dans le répertoire „RELEASE-1.0.0" exactement de la version livrée, ce qui permet de localiser et de corriger l'erreur et bien sûr de réintégrer les modifications. Avant la livraison au client, l'étape doit être fixée avec exactitude. Elle peut effectuer cela directement sur la copie de travail par la commande *svn copy . http://svn.visoco.local/projekt1/tags/RELEASE-1.0.1 -m"- Bug Fixed".* Le point remplaçant la version actuelle.

Nadine peut alors se consacrer à la nouvelle version du logiciel. Mais attention ! Le bug que le client a signalé a été corrigé dans la version livrée, mais quid de la version actuelle de développement ?

A l'évidence, le bug y est encore. On peut résoudre cela à la main c'est à dire insérer la modification manuellement dans le développement actuel ou résoudre le problème par subversion. Il nous faut simplement déterminer à quelle étape, c'est-à-dire à quelle Révision la marque de la „RELEASE-1.0.0" a été fixée. On obtiendra cela par *svn log --verbose --stop-on-copy http://svn.visoco.local/projekt1/tags/RELEASE-1.0.0* qui donne une grande quantité d'informations (Tableau 8 Release 1.0.0 Revision). Ce que nous cherchons se trouve sous „from /projekt1/trunk:4", le „4" indique que la Release 1.0.0 a été obtenu à partir de la Révision 4. Le numéro de révision de la copie de travail peut être obtenu par „svn update" qui l'amène celle-ci au niveau de révision actuel. Le message indique le numéro de révision désiré, 10 par exemple. Pour prendre en compte les modifications du côté des livraisons dans le développement actuel il faut maintenant lancer *svn merge -r4:10 http://svn.visoco.local/projekt1/tags/RELEASE-1.0.* Ce faisant, toutes les modifications qui ont été faites dans la Release 1.0.0 seront intégrées dans la copie de travail. Elle peut alors intégrer directement les corrections, ou si cela n'a pas réussi, faire le chemin inverse par *svn revert -R*.

La croissance

L'équipe se devait de grandir, le travail devenant trop important pour une personne, il a été décidé de prendre un nouveau développeur dans la société. Frank en est à son premier jour et voudrait commencer de suite. Il peut alors reprendre dans son espace personnel l'état actuel du projet par la commande *svn checkout http://svn.visoco.local/projekt1/trunk projekt1.* L'indication complémentaire *projekt1* à la fin signifie que ce qui est ainsi obtenu sera déposé dans le répertoire *projekt1* et non plus dans dans un répertoire de nom *trunk*, ce qui est plus agréable. Frank explore durant son premier jour les différentes branches du code source; tandis que Nadine poursuit le développement et ayant implémenté une nouvelle fonctionnalité et voulant la valider; elle la mémorise et poursuit son travail. De son côté Frank doit vérifier si de nouvelles modifications se trouvent dans le Repository. Ces der-

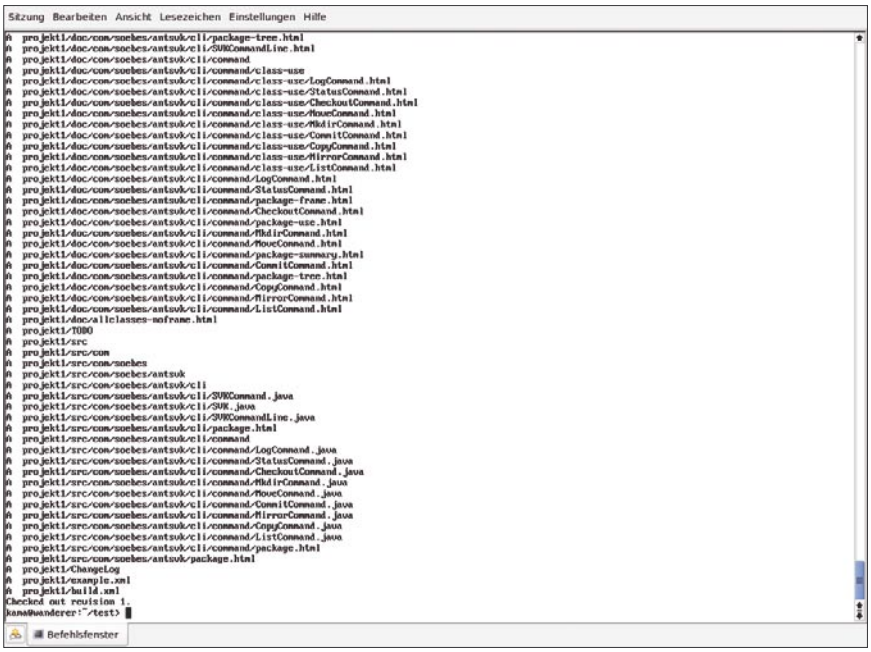

Figure 3. *Checkout du premier projet*

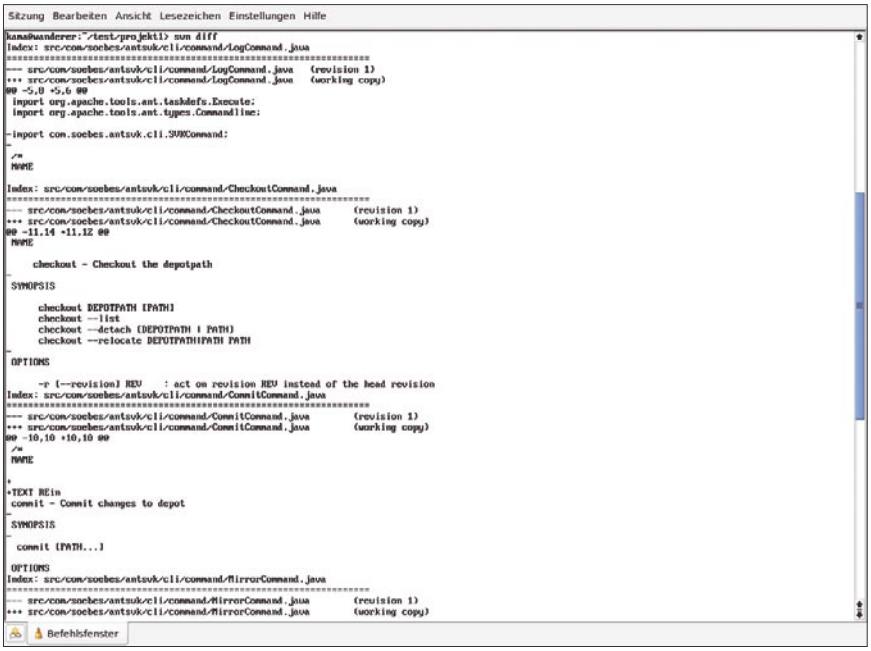

Figure 4. *Visualisation des différences*

nières peuvent être importées par *svn update* dans sa copie de travail. Il peut ensuite effectuer d'autres modifications et mémoriser celles-ci. Lorsque Nadine recommence à travailler, elle doit vérifier si des modifications se trouvent dans le Repository. En résumé, chacun doit faire attention avant de travailler à la présence de modifications dans le Repository.

Gérer les conflits

Si Frank a oublié de faire un *svn update*, il reçoit suivant les circonstances un message (Tableau 9 : *svn update* oublié). Cela signifie que Nadine a modifié les mêmes données. Pour pouvoir continuer, Frank doit auparavant intégrer les modifications de Nadine dans sa copie de travail, par un *svn update.* Des messages apparaissent alors (Tableau 10 conflits), affirmant qu'un conflit („*C*“) est apparu. Cela veut dire que Nadine et Frank ont effectué des modifications concurrentes sur les mêmes fichiers et Subversion ne peut pas décider qui a raison. Il faut résoudre le problème à la main. Examinons encore les fichiers pour lesquels Subversion a indiqué un conflit. (Tableau 11 Indication des conflits): on y trouve une ligne avec <<<<<<<<<*moi* et une ligne avec =========. Là ou les lignes entre celles-ci correspondent aux modifications de Frank sur ces fichiers; celles entre les lignes ========= et „>>>>>>>>.r14“ correspondent aux modifications de Nadine. Frank peut alors décider quelle modification est correcte et laquelle ne l'est pas. Il doit donc éliminer les lignes incorrectes et conserver les autres, et - cela est important- enlever les marques <<<<< et ======. Les différentes étapes des fichiers sont encore dans le répertoire et peuvent être examinées, si on ne craint pas de lire les marqueurs. On y trouve un fichier *TODO.mine*, *TODO.r13* et *TODO.r14*. Au lieu d'éditer le fichier *TODO*, vous pouvez copier une des autres versions de celui-ci. Lorsque le conflit est résolu, il faut l'indiquer à Subversion; cela se fait simplement par *svn resolved TODO* . Frank a ainsi écarté le conflit et peut mémoriser les modifications.

L'équipe s'agrandit

Lorsque l'équipe s'agrandit encore, on peut penser que tout développeur devra commencer par faire un update et résoudre quelques conflits. Cela prend naturellement du temps. L'expérience montre qu'au delà de trois développeurs, la mise à jour constante demande beaucoup de temps; il faut alors explorer

Listing 8. *Release 1.0.0 Revision*

```
------------------------------------
                                    ------------
r8 | (no author) | 2006-01-05 11:32:
                        22 +0100 (Do,
                        05 Jan 2006)
                        | 1 line
Changed paths:
   M /projekt1/tags/RELEASE-1.0.0/
                        TODO

- Fixed Bug in Release 1.0.0
------------------------------------
                                    ------------
r5 | (no author) | 2006-01-05 11:14:
                        49 +0100 (Do,
                        05 Jan 2006)
                        | 1 line
Changed paths:
   A /projekt1/tags/RELEASE-
                        1.0.0 (from
                        /projekt1/
                        trunk:4)

- Release 1.0.0
```

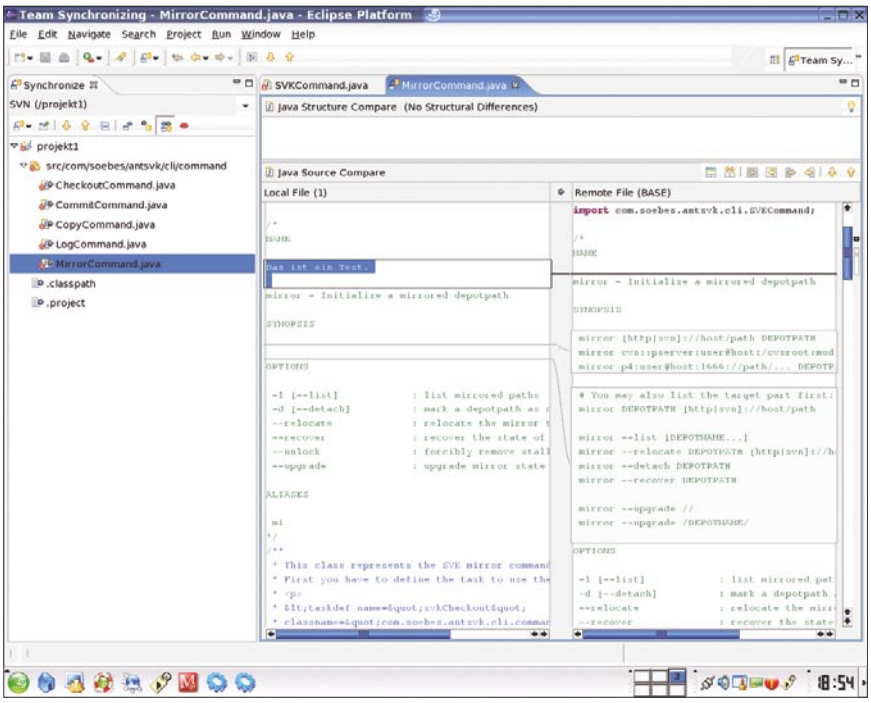

Figure 5. *Différences avec Eclipse*

Listing 9. *Vergessenes svn update*

```
Sending        TODO
svn: Commit failed (details follow):
svn: Your file or directory 'TODO'
is probably out-of-date
svn:
The version resource does not
correspond to the resource within
the transaction.  Either the
requested version resource
is out of date (needs to be
                      updated),
or the requested version resource
is newer than the transaction root
(restart the commit).
```

Listing 10. *Konflikte*

```
C  TODO
Updated to revision 14.
```

Listing 11. *Anzeige Konflikte*

```
<<<<<<< .mine
- Fixed Frank
=======

>>>>>>> .r14
- Fixed Bug in Release 1.0.0
```

d'autres voies et on devrait au delà de deux développeurs établir une méthode convenable. Le mot magique s'appelle la branche. Comment utilise-on cette notion de branche ? Les tâches de développement sont partagées entre les différents membres de l'équipe, c'est à dire que chaque membre reçoit un travail dédié. Nous créons ainsi pour chaque membre et chaque tâche une branche, par *svn copy http://svn.visoco.local/projekt1/trunk http://svn.visoco.local/projekt1/branches/frank_feature1 -m"- Frank Feature 1".*

Le nom de la branche devrait contenir le nom correspondant au membre de l'équipe pour une meilleure vue d'ensem-

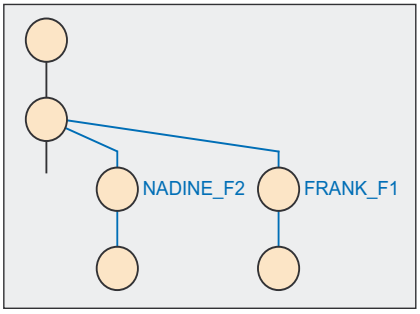

Figure 6. *Branches*

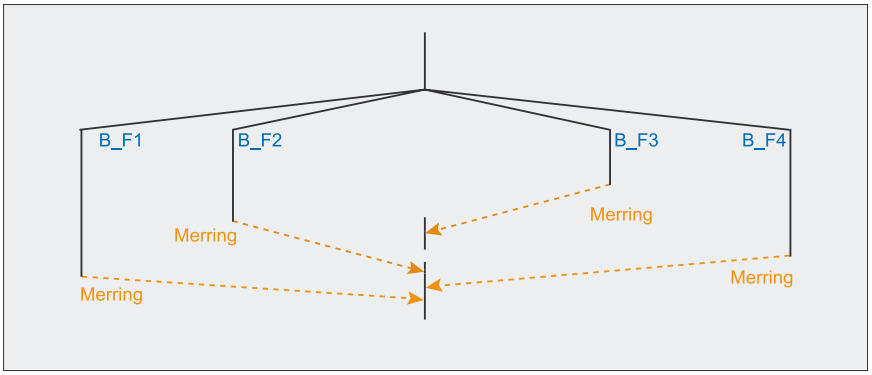

Figure 7. *Branches et merging*

ble. Il est bien sûr important de créer un répertoire de branches (svn mkdir), ce que nous faisons aussi pour Nadine par *vn copy http://svn.visoco.local/projekt1/trunk http://svn.visoco.local/projekt1/branches/nadine_feature2 -m"- Nadine Feature 2"* (Figure 5)

Pour travailler avec une branche, il faut bien sûr se placer dans celle-ci, en utilisant la commande *svn switch* http://svn.visoco.local/projekt1/branches/frank_feature1, pour Frank et pout Nadine respectivement. La copie de travail est alors déposée dans la branche et on peut poursuivre le développement en toute tranquillité, sans être obligé de faire des *svn update* constants. Vient le jour où les fonctionnalités désirées sont terminées et doivent être incorporées dans le logiciel. L'intégrateur a pour rôle d'intégrer les diverses fonctionnalités; pour cela il doit retirer une copie de travail dans le répertoire principal (tronc) par la commande *svn switch* http://svn.visoco.local/projekt1/trunk. La branche de Frank peut alors être intégrée avec la fonctionnalité 1 dans le répertoire principal. Il nous faut préciser à partire de quelle révision la branche a été crée, en utilisant *svn log –verbose –stop-on-copy* http://svn.visoco.local/projekt1/branches/frank_feature1, qui nous donne le numéro de révision à partir duquel la branche a été crée (ex. 16), la révision actuelle etant par ex. 21. Nous pouvons alors procéder à son intégration effective par *svn merge -r16:21* http://svn.visoco.local/projekt1/branches/frank_feature1 qui intégrera dans la copie de travail toutes les modifications effectuées dans cette branche. Ce faisant, il se peut que des conflits surgissent. Ils doivent alors être résolus à la main, comme cela a été décrit. L'intégration de la première branche peut alors être mémorisée. On peut procéder de cette manière pour chaque branche individuelle et intégrer ainsi pas à pas chaque fonctionnalité dans le logiciel (Figure 6).

Bien sûr il faut faire suivre une intégration par une phase de tests pour détecter d'éventuelles erreurs. Les test automatiques sont les plus adéquats pour cela. Il existe encore d'autres stratégies de branche pour venir à bout des problèmes les plus variés dans le domaine de la gestion de projet. Cela constitue un tout autre sujet.

Intégration des outils

La plupart des développements ne sont plus effectués avec l'éditeur (vi, emacs etc) et la ligne de commande, mais la plupart du temps avec un IDE, comme Eclipse, Zend Studio, IntelliJ etc. L'intégration actuelle de subversion dans ces IDE permet d'utiliser les méthodes présentées ici, ramenant l'utilisation de la ligne de commande à néant. Si on doit travailler directement avec Subversion, il existe TortoiseSVN ou RapidSVN o.ä. Tools. Je travaille personnellement volontiers avec la ligne de commande même si cela est un peu excentrique.

La responsabilité de la sauvegarde

Il reste encore une chose à faire concernant le Backup. Vous faites bien des sau-

Sur Internet

- *http://subversion.tigris.org*
- *http://svnbook.red-bean.com*
- *http://www.tortoisesvn.org*
- *http://www.sleepycat.com/download/index.shtml*
- *http://httpd.apache.org*
- *http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91*
- *http://www.tortoisesvn.org*
- *http://svnbook.red-bean.com:9000/en/1.1/svn-book.html*

vegardes de vos données ? Alors il faut faire des Backup des données du Repository, c'est à dire que votre repository doit faire partie du Backup. Si on examine les choses, cela veut dire que lors du backup, il faut y lier les Repository, comme pour une banque de données. Si on peut être sûrs que plus personne ne travaille à 04: 00 heures,l'expérience montre toutefois qu'il peut y avoir de l'activité à des heures inhabituelles. Il est donc nécessaire d'arrêter le serveur (Ici le serveur Web Apache), pour en interdire l'accès pendant la durée du Backup. Ce dernier sera lancé par la commande *svnadmin dump Repos >respos.dump* . Cela fera un Backup des Repositories sous forme d'un fichier éditable. Prudence, un tel fichier peut être assez gros (parfois plusieurs MegaOctets).

Resumé

Subversion a atteint actuellement un état de maturité que l'on peut qualifier de très bon. Cela signifie que l'on peut utiliser Subversion pour des travaux de groupe comportant de une à plusieurs centaines de personnes. Une rationalisation systématique de son utilisation est naturellement d'autant plus nécessaire que le groupe est important. Une connaissance exacte des structures; des outils et des techniques concernées en facilitent le processus. On ne peut pas atteindre cela que par quelques essais en fin de semaine. Il faut de l'expérience. En règle générale, on peut dire que l'utilisation de Subversion est toujours à recommander. En outre, si on planifie l'utilisation de Subversion, je recommande de lire non seulement la documentation mais aussi de constituer un jeu d'exemples pour accumuler un peu d'expérience. Il faut noter que l'introduction d'un système de contrôle de version demande un certain temps, l'expérience montrant que environ 1 à 6 mois sont nécessaires pour tirer un profit incontestable d'une telle mesure. ■