Gestion des documents structurés dans un VCS : exemple avec Subversion

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 26 septembre 2011. Dernière mise à jour le 27 septembre 2011

https://www.bortzmeyer.org/vcs-documents-structures.html

Ceux qui utilisent un VCS pour stocker des documents structurés (par exemple du XML) ont souvent le problème d'éditeurs trop intelligents qui reformattent brutalement le document, lui conservant son sens sous-jacent (encore heureux !) mais changeant complètement la représentation textuelle. Les VCS typiques, conçus pour du texte, croient alors que le document est différent et annonçent des changements qui ne sont pas réellement importants. Une des solutions à ce problème est de **canonicaliser** les documents et de demander au VCS de vérifier que cette canonicalisation a bien été faite. Comment on fait avec le VCS Subversion ?

Le problème n'est pas spécifique avec XML. Si une équipe de développeurs C utilise indent (ou le vieux logiciel cb) pour **pretty-printer** leur code, mais qu'ils ont mis des options différentes dans leur `/.indent.pro` (au passage, voici le mien (en ligne sur `https://www.bortzmeyer.org/files/my-tilde-indent.pro`)), chaque "*commit*" va engendrer des "*diff*" énormes mais non significatifs. Mais le problème est plus fréquent avec XML, où les auteurs utilisent rarement des éditeurs de texte et passent en général par un logiciel spécialisé, qui n'hésite pas à changer le fichier considérablement. C'est typiquement le cas avec le format SVG, qu'on édite rarement à la main. Si un membre de l'équipe utilise Inkscape et l'autre Sodipodi, les "*diffs*" entre leurs "*commits*" ne seront guère utilisables. C'est un problème que connait le projet CNP3 <https://www.bortzmeyer.org/cnp3.html>, enregistré dans la bogue #24 <https://scm.info.ucl.ac.be/trac/cnp3/ticket/24>. Ce problème a aussi été discuté sur StackOverflow <http://stackoverflow.com/questions/769950/manage-xml-documents-in-s

Bien sûr, le problème n'est pas purement technique. Il y a aussi une part de politique : mettre d'accord les développeurs, fixer des règles. Mais la technique permet-elle ensuite de s'assurer qu'elles sont respectées ?

Avec le VCS Subversion, oui, c'est possible, et c'est même documenté <http://svnbook.red-bean.com/en/1.6/svn.ref.reposhooks.pre-commit.html>. Le principe est d'avoir un script dit `pre-commit`, que le serveur Subversion exécutera **avant** le "*commit*" et qui, si le code de retour indique une erreur, avortera le "*commit*". Voici un exemple, avec un `pre-commit` qui teste qu'on soumet du XML bien formé :

```
% svn commit -m 'Bad XML' bad
Sending        bad
Transmitting file data .svn: Commit failed (details follow):
svn: Commit blocked by pre-commit hook (exit code 1) with output:
/tmp/bad:1: parser error : Opening and ending tag mismatch: x line 1 and y
<x a="3">toto</y>
                ^
```

Vous pouvez voir ce script en (en ligne sur `https://www.bortzmeyer.org/files/pre-commit-check-xml.sh`). Il est très inspiré du script d'exemple (très bien documenté) fourni avec Subversion. Il utilise xmllint pour déterminer si le fichier XML est bien formé. Sinon, xmllint se termine avec un code de retour différent de zéro, et, à cause de `set -e`, le script entier se termine sur une erreur, ce qui empêche le "*commit*" d'un fichier incorrect.

Si le dépôt Subversion est en `/path/example/repository`, le script doit être déposé dans le sous-répertoire `hooks/` du dépôt, sous le nom de `pre-commit`. Il doit être exécutable (autrement, vous verrez une mystérieuse erreur sans message, avec un code de retour de 255).

Plus ambitieux, un script qui teste que le XML soumis est sous forme canonique (cette forme est normalisée dans un texte du W3C <http://www.w3.org/TR/xml-c14n>). Pour cela, il va canonicaliser le fichier soumis (avec `xmllint --c14n`) et, si cela donne un fichier différent de celui soumis, c'est que ce dernier n'était pas sous forme canonique. Le "*commit*" est alors rejeté :

```
% svn commit -m 'Not canonical XML' bad
Sending        bad
Transmitting file data .svn: Commit failed (details follow):
svn: Commit blocked by pre-commit hook (exit code 3) with output:
File "bad" is not canonical XML
```

Voici un exemple du travail de xmllint pour canonicaliser du XML :

```
% cat complique.xml
<?xml version="1.0" encoding="utf-8"?>
<toto   >
    <truc      a="1" >Machin &#x43; </truc >café</toto>

% xmllint --c14n complique.xml
<toto>
    <truc a="1">Machin C </truc>café</toto>
```

Le script de `pre-commit` est en (en ligne sur `https://www.bortzmeyer.org/files/pre-commit-check-xml.sh`).

On peut se dire qu'il serait plus simple que le script canonicalise le fichier, qu'il l'était avant ou pas, épargnant ainsi ce travail aux auteurs. Mais Subversion ne permet pas de modifier un fichier lors d'un "*commit*" (cela impliquerait de changer la copie de travail de l'utilisateur). On ne peut qu'accepter ou refuser le "*commit*".

Rien n'interdit d'utiliser le même principe avec du code, par exemple en C. On peut envisager de tester si le code est « propre » en le testant avec splint dans le `pre-commit`. Ou bien on peut voir si le code C a été correctement formaté en le reformatant avec indent et les bonnes options, et voir ainsi s'il était correct.

Et avec les DVCS ? Par leur nature même, ils rendent plus difficile la vérification du respect de règles analogues à celles-ci. Je ne connais pas de moyen de faire ces vérifications avec les DVCS existants. Il faudrait le faire lors d'un "*push*"/"*pull*" car un dépôt n'a aucune raison de faire confiance aux vérifications faites lors du "*commit*" par un autre dépôt. J'ai inclus à la fin de cet article des explications fournies par des lecteurs, pour d'autres VCS. Il me manque encore git. Un volontaire ?

Enfin, tout ceci ne résout pas tous les problèmes, par exemple pour l'édition SVG citée plus haut : les éditeurs SVG comme Inkscape aiment bien rajouter aussi des éléments XML à eux et, là, le problème du diff est plus délicat. Des outils comme Scour <http://www.codedread.com/scour/> deviennent alors nécessaires, pour nettoyer le SVG.

D'autres exemples de scripts `pre-commit` sont fournis avec Subversion en <http://svn.apache.org/repos/asf/subversion/trunk/contrib/hook-scripts/>. Parmi eux, on note un programme <http://svn.apache.org/repos/asf/subversion/trunk/contrib/hook-scripts/enforcer/> écrit en Python (rien n'oblige les scripts `pre-commit` à être en shell), qui vérifie un certain nombre de propriétés (configurées dans son `enforcer.conf`) sur des sources Java.

Pour le cas du VCS distribué darcs, Gabriel Kerneis propose : « On peut créer un patch qui définit un test. Si les autres développeurs appliquent ce patch, alors darcs exécutera automatiquement le test <http://darcs.net/manual/Darcs_commands.html#SECTION00682000000000000000> à chaque fois qu'ils voudront enregistrer un nouveau patch :

```
darcs setpref test 'autoconf && ./configure && make && make test'
darcs record -m "Add a pre-record test"
```

Mais comme vous le notez, cela ne prémunit pas d'un envoi de patch qui ne passe pas le test. Sur le répertoire central de référence (à supposer qu'il y en ait un), ou chacun sur son propre répertoire de travail, peut aussi ajouter un "*hook*" qui vérifiera que les patchs sont corrects <http://darcs.net/manual/Configuring_darcs.html#SECTION00410000000000000000> avant de les appliquer :

```
# à vérifier que si le prehook échoue, le apply échoue bien - je ne
# l'ai jamais utilisé en pratique
echo 'apply prehook make test' >> _darcs/prefs/defaults
echo 'apply run-prehook' >> _darcs/prefs/defaults
```

Il est même possible d'effectuer la canonicalisation à l'aide de `prehook` :

```
echo 'record prehook canonicalize.sh' >> _darcs/prefs/defaults
echo 'record run-prehook'
```

Voir les détails des options prehook (et posthook) <http://darcs.net/manual/Darcs_commands.html#SECTION00613000000000000000>. Malheureusement (ou heureusement ?), il n'est pas possible de distribuer les prehooks et les posthooks sous forme de patch, comme avec setpref. Mais on peut toujours ajouter un test avec setpref qui vérifie si les prehooks sont présents et échoue sinon... »

Pour Mercurial, André Sintzoff explique : « Il y a aussi des "*hooks*" qui sont notamment décrits dans la documentation <http://hgbook.red-bean.com/read/handling-repository-events-with-hooks.html>. Le "*precommit hook*" permet de contrôler avant le "*commit*" comme avec Subversion Comme tu le fais remarquer, c'est local à l'utilisateur donc pas obligatoire. La solution est donc de se baser sur les hooks `changegroup` et `incoming` qui contrôlent l'arrivée de nouveaux "*changesets*" dans un dépôt via "*pull*" ou "*push*". Il suffit donc de mettre un "*hook*" `changegroup` avec la vérification qui va bien. Bien entendu, il vaut mieux pour l'utilisateur d'utiliser un "*precommit hook*" pour contrôler sur son dépôt local. »

Enfin, pour git, le bon point de départ semble être <http://book.git-scm.com/5_git_hooks.html>.

Merci beaucoup à Gabriel Kerneis et André Sintzoff pour leurs contributions.