Outils de développement et compilation

Git : gestion de source et versionnement

Timothée Ravier, LIFO, INSA-CVL, LIPN

1re année cycle ingénieur STI, 2013 – 2014

Objectifs

— Découvrir les outils patch et diff ;
— Découvrir Subversion ;
— Découvrir Git ;
— Proposer des améliorations sur un projet GitHub.

1 Patch et diff

Récupérons le code source d'un projet quelconque :

```
$ curl https://tim.siosm.fr/downloads/cours/git-td-example.tar.gz -o git-td-example.tar.gz
```

Extraire l'archive :

```
$ tar xf git-td-example.tar.gz
```

En faire une copie de travail :

```
$ cd devtools/git/
$ cp -a insa-cvl-test-project{,.original}
```

Fixez / changez ce que vous voulez dans le code. Vous pouvez ensuite comparez les différences entre les deux versions avec diff :

```
$ diff -u insa-cvl-test-project/fichier insa-cvl-test-projetct.original/fichier
```

Ou :

```
$ diff -ur insa-cvl-test-project insa-cvl-test-project.original
```

Pour créer un patch il suffit de rediriger la sortie de `diff` dans un fichier :

```
$ diff -u chemin/main.c chemin.original/main.c > fix-bug1.patch
```

- **Question 1 :** A quoi correspondent toutes les informations affichées par `diff` ?

Envoyez votre patch à votre voisin par email (en pièce jointe de préférence pour éviter les problèmes de conversion des espaces en tabulations par exemple). Testez ensuite le patch de votre voisin :

```
$ cd insa-cvl-test-project.original
$ patch -p1 -i fix-bug1.patch
```

- **Question 2 :** A quoi sert l'argument `-p1` passé à la commande `patch` ?
- **Question 3 :** Que se passe-t-il lorsque vous essayer d'appliquer un patch déjà ajouté au projet ?

2 SVN

Nous allons utiliser le dépôt SVN disponible à l'adresse : https://code.google.com/p/insa-cvl-test-project/.

Récupérez le contenu du projet :

```
$ svn checkout https://insa-cvl-test-project.googlecode.com/svn/trunk/ insa-cvl-test-project --username ↙
    ↳ timothee.romain.ravier@gmail.com
```

Vous disposez alors d'une copie de travail de la dernière version du projet. Commencez par regarder l'historique du projet :

```
$ svn log
```

On peut obtenir plus d'informations avec :

```
$ svn log -v -r2
```

Modifiez maintenant le contenu des fichiers à votre guise. Une fois les modifications sauvegardées, vous pouvez « commiter » vos changements pour les rendre disponibles à tout le monde. Il est en revanche possible que d'autres aient fait des changements avant vous. Vous devez donc d'abord récupérer ces changements avant de proposer les votre :

```
$ svn update
```

Si vous avez modifié des lignes que d'autres personnes ont aussi modifié, svn vous demandera quel changement il doit conserver. Une fois tous les conflits résolus, vous pourrez commiter vos changements :

```
$ svn commit
```

3 Git

La syntaxe des commandes Git est similaire à SVN :

```
$ git <commande> <arg>
```

On obtient ainsi la page de manuel correspondant à une commande avec man ou git help :

```
$ man git <commande>
$ git help <commande>
```

3.1 Utilisation locale

Configurez votre identité Git :

```
$ git config --global user.name "Bugs Bunny"
$ git config --global user.email bugs.bunny@insa-cvl.fr
```

Déplacer vous dans le dossier que l'on a utilisé tout à l'heure :

```
$ cd git-td-orig
```

Initialisez un dépot local Git dans ce dossier :

```
$ git init .
```

Ajoutez des fichiers au dépôt ainsi qu'au prochain commit :

```
$ git add main.c Makefile
```

Créez votre premier commit :

```
$ git commit
```

Git présente alors le message récapitulatif des changements qui vont être enregistrés lors du commit. On peut alors en profiter pour ajouter un commentaire pour décrire les modifications effectuées plus en détail.

Affichez l'historique de votre projet :

```
$ git log
```

Et si j'ai fait une erreur dans le message de mon dernier commit ?

```
$ git commit --amend
```

Utilisez les patchs que vous avez précédemment créé pour modifier le contenu du projet.

```
$ git apply
```

Constater les changements avec :

```
$ git status
$ git diff
```

Ajoutez seulement une partie des changements en utilisant :

```
$ git add --patch
ou
$ git add -p
```

Créez un alias git pour ne pas avoir à taper tout le temps `status` et `add` par exemple :

```
$ git config --global alias.a add
$ git config --global alias.s status
```

Constater les changements dans le fichier `~/.gitconfig`.

Et si j'ai `git add` des fichiers par erreur ?

```
$ git reset <fichier>
$ git reset --patch <fichier>
```

Modifiez quelques fichiers, faites des commits.

3.2 Les branches

Les branches permettent de tester des idées sans avoir à avoir à se soucier des conséquences car les modifications ne seront pas répercutées sur les autres branches. Il sera aussi toujours possible de retourner en arrière. Par défaut, nous travaillons sur la branche nommée master. Nous allons donc créer une nouvelle branche et nous y « déplacer » :

```
$ git branch nom-de-la-branche
$ git checkout nom-de-la-branche
```

Ces deux opérations peuvent se faire en une seule fois :

```
$ git checkout -b nom-de-la-branche-2
```

Il faut noter que tous les fichiers présents dans la branche d'origine (ici master) sont maintenant présents dans cette branche. Il est possible de faire des modifications et de commiter :

```
$ vim test.c
$ git comit -am "Salut, c'est moi le lapin"
```

On a utilisé ici plusieurs options pour commiter (-a pour add et -m pour message). L'option m permet de définir dans la ligne de commande le message correspondant au commit. L'option a permet d'ajouter tous les fichiers au prochain commit. Il est possible que des fichiers inutiles, par exemple `main.c~` ou `a.out`, soient ajoutés. On peut les supprimer pour le prochain commit avec la commande :

```
$ git rm main.c~ a.out
```

Il est parfois fastidieux de choisir entre les fichiers lesquels ajouter au prochain commit. C'est ici qu'intervient le fichier `.gitignore`, qui permet de spécifier à git les fichiers, dossiers, expressions régulières, qu'il ne doit pas ajouter dans le dépôt. Les règles définient dans un fichier `.gitignore` s'appliquent au dossier courant ainsi que d'aux sous-dossiers. Un exemple de .gitignore :

```
*~
a.out
bin/
BUILD/
tmp/
```

Pour revenir sur la branche principale, vous pouvez utiliser :

```
$ git checkout master
```

Il n'est pas possible de changer de branche si vous avez des modifications non commitées. Vous pouvez soit créer un commit, soit les stocker temporairement dans une zone appellée `stash` :

```
$ git stash push
```

Vous pouvez afficher le contenu de votre `stash` avec :

```
$ git stash list
$ git stash show
```

Vous pourrez alors les récupérer à tout moment (pas uniquement dans la branche d'origine) avec :

```
$ git stash pop
```

Chaque dépôt possède l'intégralité de l'historique des modifications. Il est notament possible d'obtenir un diff entre deux commits, ici la tête et son parent, le commit précédent, ou encore son grand parent :

```
$ git diff HEAD^ HEAD
$ git diff HEAD^^ HEAD
$ git diff <hash-commit-1> <hash-commit-2>
```

Il est aussi possible de comparer des branches (ici la branche courante par rapport à la branche master) :

```
$ git diff master
```

3.3 Utilisation distante

Récupérez la clé SSH de test qui vous permettra de discuter avec le serveur de projet de façon sécurisé :

```
$ cd ~/.ssh
$ wget https://tim.siosm.fr/downloads/cours/git-test-ssh
$ wget https://tim.siosm.fr/downloads/cours/git-test-ssh.pub
$ mv git-test-ssh id_rsa
$ mv git-test-ssh.pub id_rsa.pub
```

Récupérez un dépôt git depuis le serveur de projet. On utilise ici git via ssh pour se connecter au serveur :

```
$ git clone gitolite@projetsti.ensi-bourges.fr:td-git.git
```

On récupère ainsi un dossier testgit contenant l'ensemble des fichiers du projet `td-git` ainsi qu'un répertoire .git, où sont stockés les informations concernant le dépôt, notamment l'intégralité de l'historique, et la configuration du dépôt :

```
$ cd testgit
$ ls -a
$ ls .git/
```

On peut maintenant regarder l'historique des modifications déjà effectuées dans ce projet :

```
$ git log
```

Vous pouvez aussi installer le logiciel `tig` (ou `qgit` ou `giggle` ou `gitg`...), qui permet d'obtenir une vue graphique plutôt claire des modifications effectuées sur le projet.

Pour ne pas géner les autres, nous allons créer un dossier à notre nom et faire nos modification à l'intérieur de celui-ci :

```
$ mkdir nom-prenom
$ cd nom-prenom
$ echo "J'aime les carottes" > lapin.c
```

Commitez vos changements.

Avant d'envoyer nos modifications sur le serveur, il faut d'abord récupérer les modifications qui ont été faites depuis notre dernière mise à jour (ici c'est l'opération git clone) :

```
$ git pull
```

Une fois les changements pris en compte par git, on peut envoyer nos propres modifications, si personne n'en a envoyé entre temps !

```
$ git push
```

Si l'on veut maintenant envoyer une branche sur le serveur, il faut préciser où (origin correspond à l'adresse entrée lors du git clone) :

```
$ git push origin nom-de-la-branche
```

3.4 Merge et conflict

Essayez de récupérer la branche de votre voisin. Pour récupérer une branche du serveur, il faut préciser :

```
$ git pull nom_branche_distante nom_branche_locale
```

Lorsque l'on récupère une branche distante, il est possible qu'une même ligne ai été modifiée différemment par chacun des développeurs. Git ne peut alors pas choisir pour nous laquelle conserver. C'est un conflit. Il se présente de cette manière sous git (dans un fichier) :

```
<<<<<<<<<<<<<<<<<<< votre version
blablabla
===================
bla bla bla
>>>>>>>>>>>>>>>>>>> la version du serveur
```

Si vous n'obtenez pas de conflits, essayez de récupérer la branche "test" du serveur et de l'enregistrer dans master.

Pour résoudre un conflit, il suffit de modifier le ou les fichiers incriminés pour ne garder que la version voulue, puis de commiter :

```
$ vim le(s)-fichier(s)-qui-a(ont)-des-conflits
$ git add -p <fichiers-modifies>
$ git commit
```

4 Références

— Pro Git book : http://git-scm.com/book/
— Une vue partisane mais intéressante sur git : http://thkoch2001.github.io/whygitisbetter/
— Le tutoriel officiel : http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html