Bref aperçu de `git`

Hiver 2011

Table des matières

1 Introduction

Bref aperçu de l'outil `git`, un outil de contrôle des versions, donc même genre d'outil que `CVS` et `svn` (SubVersion), mais avec une approche quelque peu différente : distribuée plutôt que centralisée.

2 Caractéristiques

- Développé par Linus Torvald, le créateur de Linux!
- Dépôt centralisé vs. distribué :
  - `CVS` et `svn` : dépôt centralisé ⇒ tous les *checkout*, *branch* et *commit* entraînent des communications réseaux pour accéder au dépôt centralisé.
  - `git` : dépôt distribué ⇒ on peut faire des *checkout*, *commit*, *branch*, etc. sans accéder au dépôt central.

  Ceci signifie qu'on peut préserver un historique détaillé de l'évolution locale d'un projet sans interférer avec le code dans le dépôt central. Ce n'est que lorsqu'on est certain que tout est correct qu'on peut alors faire un `push` pour transmettre les changements appropriés au dépôt central.
- Avec `git`, la création de *tags* et de *branches* est très efficace, donc il ne faut pas s'en priver.
- Avec `git`, on garde la trace non pas des noms de fichiers... mais bien de leurs contenus.
- Avec la commande `bisect`, il est possible de trouver automatiquement, dans un historique de *commits*, la version qui a introduit un bogue.

Commande `git`	Commande `svn`	Commande `CVS`
`git init`	`svnadmin create`	`cvs -d<repo> init`
`git clone`	`svn checkout`	`cvs -d<repo> co <module>`
`git pull`	`svn update`	`cvs update -dP`
`git add`	`svn add`	`cvs add`
`git add; git commit`	`svn commit`	`cvs commit`
`git status`	`svn status`	`cvs status`
`git checkout <branch>`	`svn switch <branch>`	`cvs co -r <branch>`
`git merge <branch>`	`svn merge <branch>`	`cvs update -j`
`git checkout <file>`	`svn revert <file>`	`cvs update -C`

Tableau 1: Comparaisons entre `git`, `svn` et `CVS`.

3 Comparaisons avec svn et CVS

La table 1 présente une brève comparaison entre les outils `git`, `svn` et `CVS`.

4 Stratégie d'utilisation pour le devoir 1

- Obtenez une copie locale du code : 1 :

  ```
  $ git clone http://www.labunix.uqam.ca/~tremblay/git/Fractions.git
  ```

- Définissez vos paramètre d'identification :

  ```
  $ git config --global user.email "tremblay.guy@uqam.ca"
  $ git config --global user.name "Guy Tremblay"
  ```

- Ensuite, répétez jusqu'à ce que le devoir soit terminé :
  - Faites des modifications au fichier `fractions-body.mpd`.
  - Lorsque vous atteignez un *point clé* (*milestone*), par exemple, l'une des versions à développer est complétée et semble fonctionnelle, faites un `commit` :

    ```
    $ git commit -m "Explications du changement" -a
    ```

- Si vous faites `git status`, vous devriez obtenir quelque chose comme suit :

  ```
  $ git status
  # On branch master
  # Your branch is ahead of 'origin/master' by 2 commits.
  #
  # Untracked files:
  #   (use "git add <file>..." to include in what will be committed)
  #
  #       MPDinter/
  #       tests3.out
  nothing added to commit but untracked files present (use "git add" to track)
  ```

  C'est tout à fait correct : le seul fichier à modifier pour le devoir est `fractions-body` et, de toute façon, les fichiers binaires et exécutables n'ont pas à être mis sous le contrôle des versions. De plus, ce que vous avez dans votre compte est une copie locale du dépôt (`origin/master`, auquel vous ne pouvez évidemment pas accéder en mode écriture!

- Pour voir les différents *commits* effectués, vous pouvez utiliser la commande suivante :

  ```
  $ git log
  ```

- Pour donner un nom symbolique plus explicite, vous pouvez utiliser la commande suivante :

  ```
  $ git tag VERSION_SEQUENTIELLE_OK
  ```

  Par la suite, pour revenir à cet état :

  ```
  $ git checkout VERSION_SEQUENTIELLE_OK
  ```