Introduction à git

Olivier Lafleur

4 mai 2015

Table des matières

1 Introduction

Ce document se veut une introduction au système de gestion de versions décentralisé git. Nous utiliserons l'exemple de git, mais les principes présentés pourraient aussi bien être appliqués à d'autres systèmes du genre comme Mercurial ou BitKeeper.

2 Pourquoi utiliser git ?

Lorsque l'on développe du code, il n'est pas rare que l'on soit plusieurs à vouloir travailler sur un même bout de code. Cependant, cela peut devenir problématique lorsque l'on veut changer des bouts de code à des endroits différents.

En utilisant le courriel ou une clé USB on peut s'échanger des bouts de code, mais cela devient rapidement très complexe et il est facile de faire des erreurs lors de l'intégration du code d'un collègue dans le sien.

Une technique est de travailler chacun son tour : la personne A commence le code, envoie le résultat à une personne B, qui ajoute un autre bout de code et renvoie le tout à la personne A. Ce va-et-vient peut être laborieux et ralentir de beaucoup ce qu'il serait possible de réaliser en travaillant en parallèle.

Quels seraient donc les avantages d'utiliser la gestion de versions ? Cela est mieux puisque :

- Rien qui est sauvegardé (on dit faire un *commit*) n'est perdu. Cela veut dire que l'on peut l'utiliser comme la fonctionnalité pour revenir en arrière (*undo*) d'un éditeur. Par ailleurs, puisque toutes les anciennes versions sont sauvegardées, il est toujours possible de revenir dans le temps et de se replacer dans un état passé.
- Nous avons une liste des changements qui ont été faits, par qui et quand. On sait donc à qui poser nos questions plus tard.
- C'est difficile (mais pas impossible) de réécrire par-dessus les changements de quelqu'un. Le système de gestion de versions avertit automatiquement l'usager lorqu'il y a un conflit entre deux changements effectués sur la même ligne par deux personnes différentes.

La gestion de versions est essentielle pour tous les projets de développement logiciel significatifs, et la plupart des programmeurs l'utilisent pour leurs petits projets aussi. Ce n'est pas non plus que pour du logiciel : les livres, les notes de cours (comme celles-ci), les petits jeu de données et tout ce qui change à travers le temps et a besoin d'être partagé peut (et devrait) être stocké dans un système de gestion de versions.

3 Mise en place de git

Nous commencerons par explorer comment la gestion de versions peut être utilisée pour garder une trace de ce qu'une personne a fait, et quand. Même si vous ne collaborez pas avec d'autres personnes, la gestion de versions est beaucoup mieux pour ça. On peut donc éviter de se retrouver avec des noms de fichiers laborieux incluant des VERSION_FINALE_2_olivier ou autres noms du genre, comme illustré dans une bande dessinée du *comic* en ligne Piled Higher and Deeper.

Nous avons tous déjà été dans cette situation par le passé. Il est ridicule de se retrouver avec de multiples versions presque identiques d'un même document.

Les systèmes de gestion de version commencent avec une version de base du document et sauvegardent seulement les changements (les *diff*) que vous avez faits à chaque étape.

FIG. 1 : Le document de base et ses changements

À partir du moment où on commence à voir les changements comme étant séparés du document lui-même, on peut penser à "appliquer" des changements différents sur le document de base et ainsi obtenir des versions différentes du document. Par exemple, deux utilisateurs peuvent faire des changements indépendants sur le même document.

3.1 Comment git fonctionne

Un système de gestion de versions est un outil qui garde une trace de ces changements pour nous et nous aide à amalgamer tous les changements ensemble. Un système comme git est conçu pour garder de multiples changements synchronisés sur différents ordinateurs et serveurs. C'est pourquoi on dit qu'il

s'agit d'un système *distribué* (par opposition à SVN ou TFS qui sont des systèmes dits centralisés).

Un dépôt (*repository* en anglais) est un ensemble de fichiers que nous voulons versionner.

Avec git, chaque utilisateur qui veut faire un changement à un dépôt a sa propre copie des fichiers dans ce dépôt, ainsi que sa copie des changements (les *commits*) qui ont été faits à ces fichiers. Git garde les *commits* dans un répertoire caché avec les copies des fichiers.

3.2 Première utilisation

La première fois que l'on utilise git sur une nouvelle machine, nous avons besoin de configurer quelques paramètres. Voici comment on procède (dans le terminal) :

```
$ git config --global user.name "Olivier Lafleur"
$ git config --global user.email "olivier.lafleur@gmail.com"
```

(Utilisez évidemment votre nom et votre adresse courriel à la place de mes informations.)

Les commandes git sont écrites sous la forme `git verbe`, où `verbe` est ce que nous voulons faire. Dans ce cas, nous donnons à git notre nom et notre adresse courriel, en lui disant que nous voulons utiliser ces paramètres de façon globale (pour tous les projets sur cet ordinateur).

Les commandes précédentes ont seulement besoin d'être exécutées une seule fois : le *flag* `--global` dit à git d'utiliser ces paramètres pour tous les projets.

4 Créer un dépôt

Une fois que git est configuré, nous pouvons commencer à l'utiliser. Créons un répertoire pour nos fichiers : $ mkdir planètes $ cd planètes et disons à git de créer un dépôt - un endroit où git peut stocker les anciennes versions de nos fichiers : $ git init Si nous utilisons `ls` pour montrer le contenu du répertoire, on dirait que rien n'a changé. Par contre, si nous ajoutons le *flag*

`-a` pour ton montrer, on voit que git a créé un répertoire caché appelé `.git` :
$ ls -a . .. .git

5 Auteur

Olivier Lafleur (`olivier.lafleur@gmail.com`)

Par ailleurs, il est important de mentionner que le contenu de ce document est fortement inspiré du contenu des formations de l'organisme *Software Carpentry*, organisme à but non lucratif qui met disponible en ligne son contenu sous licence Creative Commons.