Git
Gestion de l'historique

Sylvain Bouveret, Grégory Mounié

2017

Ce document peut être téléchargé depuis l'adresse suivante :

http://recherche.noiraudes.net/resources/git/TP/tp2-historique.pdf

1 Introduction

La clarté de l'historique est un préalable à la production de logiciel de qualité. Dans cette session nous allons :
— nettoyer d'abord un petit historique ;
— rechercher un commit par bissection dans un historique de grande taille.
Pour cette session, les différents exercices peuvent être réalisé seuls.

2 Nettoyage d'un historique

2.1 Mise en place

Cloner le dépot suivant contenant un historique perfectible :

```
git clone https://github.com/moy/dumb-project.git
```

Observer l'historique, par exemple avec :

```
git log --graph -c --pretty=full
```

2.2 Reconstruire l'historique

Vous pouvez observer dans cet historique un certain nombre de problèmes : coquilles dans les messages de commit, commits qui sont visiblement mal séparés ou redondants, etc.

Tout ce que vous avez à faire est d'utiliser la commande `git rebase -i` pour reconstruire l'historique à votre convenance. Procédez par petits pas plutôt que de tout faire d'un coup. Git vous permet de faire plusieurs reconstructions d'historique à la suite. Pour remonter au commit d'origine, vous pouvez utiliser :

```
git rebase -i --root
```

Sinon, à la place de l'option `--root`, vous pouvez également utiliser un numéro de commit particulier.

Pour séparer un commit en plusieurs morceaux, le plus simple est d'arrêter le rebase sur le commit à séparer pour une édition (edit), de le défaire (avec reset), puis d'enregistrer les nouveaux commits avant de continuer.

```
git rebase -i
# mettre le commit à séparer à "edit"
git reset HEAD^
git add ...
git commit ...
git add ...
git commit ...
git rebase --continue
```

3 Recherche par bissection

Afin d'illustrer le principe de la recherche par bissection, nous allons récupérer un dépôt de grande taille : celui de Git lui-même. Récupérez le dépôt du projet en le clonant :

```
git clone https://github.com/git/git.git
```

L'une des fonctionnalités qui a été ajoutée à git est la possibilité de synchroniser un dépôt avec Mediawiki. Mais quand donc cette fonctionnalité a-t-elle été introduite ? Nous allons faire une recherche dans l'historique.

Tout d'abord, vérifiez que cette fonctionnalité est bien présente dans la version actuelle du projet. Le plus simple pour cela est de tester l'existence du répertoire `contrib/mw-to-git`.

```
[ -d contrib/mw-to-git ] && echo "Le répertoire existe" || echo "Le répertoire n'
    existe pas"
```

Nous allons débuter la recherche par bissection :

```
git bisect start
```

Git bissect est un outil plutôt utilisé pour remonter à l'origine de problèmes dans les programmes (plutôt que pour savoir quand telle ou telle fonctionnalité a été introduite). Le raisonnement à adopter est donc le suivant : la fonctionnalité Mediawiki est nocive (*bad*) ; nous cherchons à remonter au premier commit dans lequel cette fonctionnalité nocive est apparue.

Nous partons de la tête de la branche master. Comme nous avons pu le vérifier préalablement, le répertoire `contrib/mw-to-git` existe bien dans l'état actuel. C'est donc qu'il faut remonter dans l'historique pour trouver le premier commit dans lequel ce répertoire a été versionné. Pour cela, nous disons à git que ce commit est mauvais :

```
git bisect bad
```

Ensuite, il faut remonter (manuellement pour cette fois-ci) à un commit antérieur. Remontons à la version 1.0.0 de git, et testons si la fonctionnalité est déjà présente ou non. Normalement, elle ne devrait pas l'être, donc nous marquons ce commit comme `good` :

```
git checkout v1.0.0
[ -d contrib/mw-to-git ] && echo "Le répertoire existe" || echo "Le répertoire n'
    existe pas"
git bisect good
```

La suite est semi-automatique. Après avoir réfléchi un petit moment, git va vous proposer un commit intermédiaire. À vous de déterminer si le répertoire est présent ou pas. S'il est présent, il faut marquer ce commit comme `bad`. Sinon, il faut le marquer comme `good`.

Après quelques itérations, vous devriez tomber sur le commit ayant permis l'apparition de cette fonctionnalité. Qui en est responsable ? Est-ce un bon exemple de message de commit ?

Terminez l'exercice en revenant à la position initiale :

```
git bisect reset
```

4 Pour conclure…

Vous avez appris dans ce TP à maîtriser l'historique de vos dépôts. Vous avez vu comment on peut utiliser la bissection pour rechercher à quel moment une fonctionnalité a été introduite. Vous avez également vu comment réécrire l'histoire (*sur un dépôt local uniquement*) en utilisant `rebase`. Attention, en rebasant, vous modifiez l'historique de votre dépôt, donc vous pouvez potentiellement perdre des informations sur les états intermédiaires [1].

1. En fait, en général rien n'est vraiment perdu dans git, tout peut être retrouvé lorsque l'on connaît les bons numéros de commits ou alors les noms des références permettant de s'y retrouver. Mais ça peut être plus compliqué...