

Exercices sur git

Benoit Daloze

Sébastien Wilmet

16 avril 2012

1 Échauffement

Ouvrez un terminal (sous Windows, lancez "Git bash" depuis le menu Démarrer). Commencez par vous identifier à git (à ne faire qu'une fois) :

```
$ git config --global user.name "Your Name"
$ git config --global user.email "foo@bar.be"
$ git config --global color.ui auto
```

Puis créez un dépôt git :

```
$ mkdir exercices
$ cd exercices
$ git init
```

Le premier but est de créer un *commit*. Comme il nous faut quelque chose à mettre dedans, créez un fichier :

```
$ echo "I'm learning git"> README
```

Faites un `git status` pour voir où vous en êtes. Git liste votre fichier comme *untracked* et vous indique qu'il faut utiliser `git add` pour prendre en compte ce fichier :

```
$ git add README
```

`git status` indique maintenant le fichier dans la partie *Changes to be committed*. Créez donc ce *commit*. L'option `-m` permet de spécifier le titre du *commit* directement dans la ligne de commande. Si l'option `-m` n'est pas utilisée, un éditeur de texte s'ouvre¹.

```
$ git commit -m "first commit"
```

Vous pouvez maintenant vérifier que votre dossier (*working directory*) est propre avec `git status`. Vérifiez que le *commit* a bien été créé, en utilisant `git log`, avec l'option `-p` pour voir les changements.

Modifiez le fichier README et regardez la *diff* :

```
$ git diff
```

La modification n'est pas encore dans la *staging area*, c'est-à-dire qu'il n'y a encore rien pour le prochain *commit*. Si vous voulez commiter le changement, nous avons vu qu'il est possible de faire un `git add` suivi d'un `git commit`. Cependant, il est possible de le faire en une seule étape, grâce à :

```
$ git commit -a
```

Quand vous faites cela, vérifiez bien avant avec `git status` ainsi que `git diff` que tous les changements sont OK pour faire le *commit*.

1. En console l'éditeur de texte peut être vim, nano, etc. Sous Unix cela dépend de la variable d'environnement EDITOR, que vous pouvez modifier si besoin : `export EDITOR=nano`

2 Branches, *merges* et autres *rebases*

Créez une branche `test` :

```
$ git branch test
$ git branch
```

Créez un *commit* sur `master` qui modifie le `README`.

Passez sur la branche `test` :

```
$ git checkout test
```

Créez un *commit* qui rajoute un *nouveau* fichier (pour ne pas qu'il y ait de conflits plus tard).

Voyez maintenant où vous en êtes avec :

```
$ git log --graph --decorate --oneline --all
```

Comme c'est une commande pratique, vous pouvez en faire un alias :

```
$ git config --global alias.lola "log --graph --decorate --oneline --all"
$ git lola
```

Il est temps de faire un *merge* pour intégrer la branche `test` dans la branche `master` :

```
$ git checkout master
$ git merge test
$ git lola
```

Un *rebase* aurait permis d'avoir un historique linéaire. Annulez d'abord le dernier *commit*, qui est le *merge* (attention commande dangereuse, vérifiez bien avec `git lola` ou `git log` que vous vous trouvez bien sur le bon *commit*!) :

```
$ git reset --hard HEAD~1
```

Vous êtes normalement revenu à l'état précédent, comme si le *merge* n'avait pas eu lieu.

Faites maintenant le *rebase* :

```
$ git checkout test
$ git rebase master
$ git lola
```

La branche `test` se trouve maintenant juste au-dessus de `master`, le *merge* se fera en « *fast-forward* » :

```
$ git checkout master
$ git merge test
```

OK toujours vivant ? Compliquons un peu les choses : faites un *commit* sur `master` qui modifie une certaine ligne d'un fichier. Allez sur la branche `test` et créez un autre *commit* qui modifie exactement la même ligne. Le *merge* donnera un conflit. En voici un exemple :

Avant	hello world
Branche <code>master</code>	goodby world
Branche <code>test</code>	Hello, world!
Merge	Goodby, world!

Éditez le fichier pour régler le conflit, et suivez les instructions données par la commande `merge`. Une fois terminé, la branche `test` ne sert plus à rien, vous pouvez la supprimer :

```
$ git branch -d test
```

3 Pour aller plus loin

Faites des modifications, puis imaginez que vous voulez changer de branche. Il vous faut pour cela un dossier propre. Utilisez `git stash save`, changez de branche et puis revenez et faites un `git stash pop`.

Changez votre dernier *commit* en ajoutant une autre modification à l'aide de `git commit --amend`.

Essayez d'appliquer un *commit* dont vous trouverez la référence avec `git log` sur la branche courante en utilisant `git cherry-pick`.

Voyez un peu ce que vous avez fait avec `git reflog`, et profitez-en pour satisfaire votre curiosité en inspectant une référence avec `git show`.

Notez que `git show` permet facilement de voir un fichier à une référence donnée avec la syntaxe `git show ref:file`.

Comparez la taille de votre répertoire avec du `-ks` avant et après `git gc`. `git` possède un *garbage collector*, qui permet de mieux arranger le contenu du dépôt.

Faites plusieurs modifications dans le même fichier, pas forcément contiguës. Essayez d'en ajouter qu'une partie avec `git add -p`. Pouvez-vous même le faire lorsque les lignes se suivent ? (indice : e)

Créez quelques commits dans une branche. Inversez l'ordre de certains commits avec `git rebase -i HEAD~n`, qui sélectionne les *n* derniers commits et vous propose différentes actions à faire sur ceux-ci, par exemple renommer le message, fusionner plusieurs commits, changer l'ordre, etc. Regardez le résultat grâce à `git lola`.

Faites quelques modifications et faites un `git add`. Vous aimeriez bien voir la diff de ces modifications prêtes à être incluses dans le prochain *commit* (la *staging area*). Trouvez l'option de `git diff` permettant de faire cela.

Imaginez que vous vous rendez compte qu'il y a un bug quand vous testez le dernier *commit*, et qu'auparavant ce bug n'existait pas. Vous regardez l'historique et vous trouvez un *commit* où vous êtes certain que le bug n'existait pas, le plus récent possible. Avec `git bisect`, il est alors facile de trouver le *commit* qui a introduit le bug. Pour démarrer la recherche dichotomique, faites :

```
$ git bisect start HEAD <bad-commit>
```

Pour l'exercice, assurez-vous qu'entre HEAD et le « bad-commit », il y ait au moins une petite dizaine de commits. `git` va maintenant vous emmener à travers plusieurs *commit* que vous allez devoir marquer comme bon ou mauvais. Utilisez :

```
$ git bisect bad
$ git bisect good
```

Une fois terminé, `git` vous donne le premier *commit* qui introduit le bug. `git lola` vous permet de voir les *commits* que vous avez marqués. Terminez par :

```
$ git bisect reset
```

Pour les slides et un exemple de fichier `$HOME/.gitconfig` : <http://eregon.me/git/>