Exercices sur git

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'ouvre [1].

```
$ 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 celà, 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 permit 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 `master`	goodby world
Branche `test`	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
```