Petit tutoriel sur Git Documentation

Version 1.0.0

Aurelazy

sept. 29, 2017

Table des matières

Contents :

CHAPITRE 1

Installation

Pour installer git sur une Debian, il suffit de taper la commande :

```
apt-get install git-core gitk
```

A quoi correspond ces 2 paquets :

- git-core : c'est git
- gitk : une interface graphique qui aide à visualiser les logs. Facultatif.

Pour activer la couleur, à ne faire qu'une seule fois :

```
git config --global color.diff auto
git config --global color.status auto
git config --global color.branch auto
```

Pour configurer le nom :

```
git config --global user.name "aurelazy"
```

Pour configurer le mail :

```
git config --global user.email mon@ema.il
```

On peut vérifier toutes ces infos en éditant le fichier `~/.gitconfig`.

On peut créer des alias pour les commandes, dans le fichier `.gitconfig` :

```
[alias]
        ci = commit
        co = checkout
        st = status
        br = branch
```

Donc au lieu d'écrire :

```
git commit
```

On peut écrire :

```
git ci
```

CHAPITRE 2

Création de son premier dépôt

Nous allons voir 2 possibilités pour créer un dépôt. La première nous créerons un dépot depuis notre PC, et la seconde en clonant un dépôt Github.

Création d'un dépôt

Se placer dans le dossier du projet ou en créer un. Puis on tape la commande suivante :

```
git init
```

Le dépôt est créé, un fichier `.git` est créé dans ce dossier. Il faudra ensuite créer des fichiers sources et les faire connaitre à Git. La commande "commit" sera utilisé, nous verrons cela un peu plus loin.

Il faut créer un fichier `README.md` qui sera un fichier d'information sur le projet.

Cloner un dépôt

Lorsque nous clonons un dépot, nous récupérons l'historique et le code source d'un projet. Nous pouvons nous rendre sur des serveurs contenant des dépôt comme Github En se rendant sur le site et en se connectant sur un projet, nous pouvons récupérer un lien que nous pourrons utiliser pour cloner ce projet. Par exemple, nous récupérons le projet domopi sur notre PC :

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

Git va créer un dossier caché `.git` dans le dossier téléchargé, celui-ci comprend l'historique des modifications, et la configuration de Git pour ce projet

Création d'un dépôt servant de serveur

Pour un serveur de dépôt, il n'y a besoin que du dossier .git, car les fichiers seront modifiés sur les clients distant et non sur le serveur ! Donc pour la création il suffit de taper :

```
git init --bare
```

c'est l'option `--bare` qui permet de ne créer que le dossier `.git`. Pour récupérer le dépot depuis un poste client, il suffit d'utiliser SSH :

```
git clone ssh://utilisateur@serveur/chemin/git
```

Lier notre dépôt à GitHub

Pour envoyer ensuite notre projet sur Github, rien de plus simple. Après création d'un compte, puis d'un dépôt sur GitHub, on lance la commande :

```
git remote add origin https://github.com/<username>/<mon-projet>.git
git push -u origin master
```

CHAPITRE 3

Modification de notre code et sauvegarde

Dans ce chapitre, nous allons voir les différentes commandes utiles à la modification de nos fichiers de code ainsi que la sauvegarde de ceux-ci, `commit`.

Voir le status de notre *code*

Pour voir les fichiers modifiés :

```
git status
```

Méthode de travail :

1. Modifier le code
2. Tester le programme pour vérifier s'il fonctionne
3. Faire un commit pour "enregistrer" les changements et les faire connaitre à Git ;
4. Recommencer à partir de l'étape 1

Si l'on change un code dans un fichier, ex :`status`, et que l'on refait un :

```
git status
```

nous voyons que le retour n'est pas le même, nous avons, entre autre, la ligne suivante :

```
modifié:          status
aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git␣
↪commit -a")
```

Pour voir les changements fait dans le fichier :

```
git diff
```

Les lignes ajoutées sont précédées d'un +, les lignes supprimées d'un −. Si plusieurs fichiers avaient été modifiés, nous aurions vu la totalité des fichiers, par contre si nous ne voulions voir les modifications que d'un seul fichier, nous pouvions faire :

```
git diff status # Où status est le nom du fichier
```

Sauvegarder notre *code*

Lorsque nous modifions un fichier de notre code, il faut dire à *Git* de prendre les nouveaux changements dans celui-ci. Cette sauvegarde s'appelle un *commit*.

Lors de la commande :

```
git status
```

nous avons vu que le fichier modifié était en rouge, ce qui veut dire que lors du commit il ne sera pas pris en compte. Il faudra lui préciser le nom lors du commit, il y a 3 possibilités :

1. `git add fichier1 fichier2` # pour ajouter les fichier au commit puis `git commit`
2. `git commit -a` # pour tout commiter
3. `git commit fichier1 fichier2`

Lorsque la commande *commit* est lancé, un éditeur de texte s'ouvre. Cela permet d'insérer un texte pour décrire le changement. Ce texte doit être court, par défaut, chaque commit ne devrait faire l'objet que d'un seul changement dans le code. Pour changer l'éditeur par défaut de git il suffit de faire :

```
git config --global core.editor "vim"
```

Le *commit* enregistre les changements dans sont historique local, si nous voulons que ces changements soient pris en compte par pour tout le monde, il faudra envoyer ces changements sur le serveur.

Annuler un commit

Il peut arriver que nous ayons besoin d'annuler un commit que nous avons fait. Pour cela, nous avons la possibilité de vérifier tous les commits fait sur un projet :

```
git log
```

Avec cette commande, on peut voir tous les commits que nous avons effectué sur ce projet.

Pour voir également les changements faits, on peut lancer la commande avec l'option `-p` :

```
git log -p
```

On peut également avoir les changements effectué lors d'un commit mais plus court, en lançant la commande suivante :

```
git log --stat
```

Nous allons voir maintenant les commandes pour annuler un commit ou un message de commit.

Si nous devons changer le message du dernier commit, il suffit de faire :

```
git commit --amend
```

On reviens sur l'éditeur de texte. On ne peut pas modifier un message déjà envoyé. Pour annuler un commit de manière soft, voici plusieurs commandes possibles :

```
git reset HEAD
```

On peut également utiliser plusieurs notations :

- HEAD : dernier commit
- HEAD^ : avant dernier commit
- HEAD^^ : avant-avant dernier commit
- HEAD~2 : pareil
- d6d98923868578a7f38dea79833b56d0326fcba1 : le numéro de commit.

Le commit sera retiré, par contre, les fichiers restent modifiés. Pour annuler un commit de manière hard, qui annulera le dernier commit ainsi que les changements du fichier.

```
git reset --hard HEAD^
```

Pour annuler les changements sur un fichier avant un commit il suffit d'envoyer :

```
git checkout
```

Ceci va également restaurer le fichier.

Annuler/Supprimer un fichier avant un commit

Supposons que vous veniez d'ajouter un fichier à Git avec git add et que vous vous apprêtiez à le *commiter*. Cependant, vous vous rendez compte que ce fichier est une mauvaise idée et vous voudriez annuler votre `git add`. Il est possible de retirer un fichier qui avait été ajouté pour être *commité* en procédant comme suit :

```
git reset HEAD -- fichier_a_supprimer
```

CHAPITRE 4

Ce qu'il reste à faire

Télécharger les nouveautés et partager votre travail depuis le site http://openclassrooms.com/courses/gerez-vos-codes-source-avec-git

Liens utiles :

- http://stackoverflow.com/questions/16330404/how-to-remove-remote-origin-from-git-repo
- http://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes
- http://dont-be-afraid-to-commit.readthedocs.org/en/latest/documentation.html

CHAPITRE 5

Indices and tables

— genindex
— modindex
— search