

Antoine  
Charmeau

# LO43

## Compte Rendu TP1

# **Git**

Git est un outil de versioning qui facilite le travail à plusieurs sur un même projet de développement. Il dispose de nombreuses commandes permettant d'effectuer des tâches simples comme très complexes pour gérer les différentes versions.

Durant le TP nous avons utilisé des commandes « basiques » tel que « git push » ou « git pull ». Nous allons voir quelques commandes plus complexes :

## **I) Les commandes basiques**

### **Git clone**

```
$ git clone <url>
```

Cette commande permet de cloner un repository à partir de son URL.

### **Git pull**

```
$ git pull
```

Cette commande permet de récupérer les dernières versions en ligne des différents fichiers du projet.

### **Git add**

```
$ git add <file>
```

Cette commande permet d'ajouter les fichiers modifiés pour les préparer à être commit.

### **Git commit**

```
$ git commit -m "message"
```

Cette commande permet de valider les fichiers ajoutés avec « git add » et de les regrouper sous une même « mise-à-jour » dans le but d'être push.

### **Git push**

```
$ git push
```

Cette commande permet d'envoyer/pousser ses commits en ligne.

## **Git branch**

```
$ git branch <branch>
```

Cette commande permet de créer une nouvelle branche de travail à partir de la branche actuelle.

Il est cependant important de publier cette branche avant de faire les premiers push dessus. Pour cela, on utilise la commande suivante :

```
$ git push --set-upstream origin <branch>
```

## **Git checkout**

```
$ git checkout <branch>
```

Cette commande permet de basculer d'une branche à l'autre.

## **II) Quelques commandes complexes**

### **Git config**

Permet de voir et modifier les variables de configuration qui contrôlent tous les aspects de l'apparence et du comportement de Git.

On peut par exemple changer son identité :

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

Pour voir ses paramètres, il suffit de taper la commande :

```
$ git --list
```

### **Git init**

```
$ git init
```

Cette commande est indispensable pour démarrer un nouveau projet git. Cela crée un nouveau sous-répertoire nommé « .git » qui contient tous les fichiers nécessaires au dépôt / un squelette de dépôt Git.

## **Git status**

```
$ git status
```

Cette commande affiche la liste des fichiers modifiés ainsi que les fichiers qui doivent encore être ajoutés ou validés.

## **Git merge**

```
$ git merge <branch>
```

Cette commande permet de fusionner une branche dans la branche actuelle (où est le « HEAD »).

## **Git diff**

```
$ git diff
```

Cette commande permet de voir les conflits qu'il existe actuellement. On peut aussi voir les conflits d'un fichier spécifique avec :

```
$ git diff --base <file>
```

Ou alors les conflits entre deux branches à fusionner (avant de faire la fusion)

```
$ git diff <source> <target>
```

## **Git blame**

```
$ git blame <file>
```

Cette commande permet d'obtenir l'historique des modifications d'un fichier. Elle nous donne l'ensemble des commits dans lequel ce fichier a été modifié ainsi que les lignes modifiées et l'auteur de ces modifications.

### III) Récapitulatif

Il est possible d'obtenir rapidement une brève explication de chaque commande disponible avec git en tapant simplement la commande :

```
$ git
```

Et l'on obtient le résultat suivant :

```
loyle@DESKTOP-HADLO3R:~$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one


work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  reset      Reset current HEAD to the specified state
  rm         Remove files from the working tree and from the index


examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status


grow, mark and tweak your common history
  branch     List, create, or delete branches
  checkout   Switch branches or restore working tree files
  commit     Record changes to the repository
  diff       Show changes between commits, commit and working tree, etc
  merge      Join two or more development histories together
  rebase     Reapply commits on top of another base tip
  tag        Create, list, delete or verify a tag object signed with GPG


collaborate (see also: git help workflows)
  fetch      Download objects and refs from another repository
  pull       Fetch from and integrate with another repository or a local branch
  push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
loyle@DESKTOP-HADLO3R:~$
```

Des explications plus poussées sont disponibles avec les commandes :

```
$ git help -a
$ git help -g
$ git help <command>
$ git help <concept>
```