# Aide-mémoire pour `git`

César Almecija
étudiant en première année aux MINES ParisTech
cesar.almecija@mines-paristech.fr

6 février 2021

# Chapitre 1

# Les bases de `git`

## 1.1 Quelques éléments de vocabulaire

Voici quelques notions de base, nécessaires pour comprendre ce document, et plus généralement, l'environnement `git` :

- Un **dépôt** regroupe les fichiers sur lesquels on peut travailler, mais aussi tout l'historique de modification. Il est souvent appelé **repo**.
- Un **commit**, c'est une modification du dépôt. Un commit ne peut jamais être supprimé. Il est référencé de manière unique par son **SHA-1**. Un commit connaît son ou ses commits parents (*ie* la ou les modifications précédentes dont il est la conséquence directe), mais ne connaît pas ses commits enfants (*ie* les modifications suivantes)
- **HEAD** fait référence au **commit courant**, celui sur lequel on travaille actuellement.
- Le **stage** ou **index** correspond à un ensemble de modifications effectuéees depuis le dernier commit. Par défaut, un fichier modifié n'est pas inclus dans le stage, il doit y être ajouté manuellement. Les fichiers dans le stage sont ceux qui sont ajoutés au commit suivant.
- Une **branche** est un ensemble de commits successifs.
- Un **dépôt propre**, c'est un dépôt où les fichiers du dernier commit, les fichiers de l'index et les fichiers sur le disque sont identiques (*ie* un dépôt qui n'a subi aucune modification locale depuis le dernier commit)
- Le **remote** correspond au dépôt distant. En général, c'est de là que le projet a été cloné au départ.
- Un **merge** correspond à la fusion de deux branches qui n'ont pas subi les mêmes commits, par exemple dans le cas où l'on met à jour les fichiers locaux par les commits situés sur le serveur distant. Dans ce cas, `git` opère une fusion, qui peut se matérialiser de trois manières différentes :
  - sous la forme d'un *fast-forward* : le dernier commit d'une des branches est le fils du dernier commit de l'autre branche. Aucun nouveau commit est créé, et la fusion se passe bien

— sous la forme d'un véritable merge, lorsque les deux branches ont subi des commits (et donc que le cas précédent ne s'applique pas), mais qu'il n'y a aucun conflit (pas de fichier modifié en même temps par les deux branches). Dans ce cas, il y a création d'un commit et la fusion se passe bien également
— sous la forme d'un véritable merge lorsque les deux branches ont subi des commits, mais qu'il y a un conflit. On se reportera à **??** pour plus d'informations.

## 1.2 Les commandes de base

### 1.2.1 Créer un nouveau dépôt

- `git clone url` clone le dépôt distant un dossier portant le nom du projet
- `git clone url folder` clone le dépôt existant dans le dossier `folder`
- `git init` initialise `git` dans ce dossier pour le transformer en dépôt (sans cloner de projet existant)

### 1.2.2 Gérer l'index et faire des commits

- `git add file` ajoute le fichier au stage.
- `git reset fichier` retire le fichier du stage.
- `git commit -m"Message"` fait un commit, avec le message spécifié.
- `git reset --hard` rend le dépôt propre en revenant au commit précédent. Pour revenir à un commit qui ne soit pas le dernier, le spécifier sous forme d'argument avant `--hard`.

### 1.2.3 Afficher les modifications

- `git status` affiche un résumé de ce qui a été modifié depuis le dernier commit, en distinguant selon si les modifications font partie de l'index ou non.
- `git diff file` affiche les différences d'un fichier entre sa version dans l'espace de travail et sa version sur le stage. **Permet donc de voir les modifications qui n'ont pas encore été mises sur le stage**.
- `git diff --cached file` affiche les différences d'un fichier entre sa version dans le dernier commit et sa version sur le stage. **Permet donc de voir les modifications qui seront apportées par le prochain commit, relativement au commit précédent**.
- `git log` affiche l'historique des commits (les parents de celui-ci) L'argument `--oneline` fait un affichage condensé des commits (recommandé). L'argument `--graph` fait un affichage 2D pour expliciter la succession des commits L'argument `--all` force l'affichage de tous les commits.

### 1.2.4 Faire le lien avec le dépôt distant

- `git push` envoie les commits effectués en local au serveur distant. Si le dépôt distant possède des modifications qui ne sont pas sur le dépôt local, les modifications sont rejetées. Il faut alors effectuer un `git pull`, résoudre les conflits le cas échéant, pour pouvoir envoyer les commits au serveur distant.
- `git pull` récupère les commits effectués sur le dépôt distant, et modifie donc les fichiers locaux. Si aucun changement n'a été effectué localement, `git` procède à un *fast-forward*. Autrement, `git` procède à un merge et créé un nouveau commit
- `git fetch` récupère les commits effectués sur le dépôt distant, sans modifier les fichiers locaux (les modifications sont stockées dans un autre dossier)

### 1.2.5 Faire des merge

- `git merge branch` fusionne la branche actuelle (`HEAD`) avec la branche spécifiée. Pour cela, **le dépôt doit être propre**. La fusion créé un nouveau commit, avec comme premier parent `HEAD` et comme deuxième parent le dernier commit de la branche spécifiée. Si jamais la branche est un enfant de `HEAD`, le merge ne créé pas de commit car il s'agit d'un *fast-forward*.

### 1.2.6 Revenir à des versions antérieures

-

### 1.2.7 Gérer des branches