Commandes git fréquentes

Initialisation

- `git init` est la commande qui initialise un dépôt dans le dossier actuel. Le dossier peut être vide ou avoir déjà des fichiers/dossiers dedans.
- Pour ajouter une "remote" par la suite :

  ```
  git remote add origin git@github.com:Laykel/git-lesson.git
  git push -u origin master
  ```

- `git clone git@github.com:Laykel/git-lesson.git .` pour cloner un dépôt dans le dossier courant.

Récupérer infos sur l'arbre / les commits

- `git status` nous donne une variété d'informations sur l'état du dépôt.
  - Premièrement, elle nous dit sur quelle branche nous nous trouvons. (On reparlera des branches.)
  - Ensuite, s'il y en a, elle nous montre les fichiers qui ont été modifiés depuis le dernier `commit`, et qui n'ont pas été ajoutés à l'index (l'index est là où vont les fichiers lorsqu'on les `add`).
  - Pour finir, s'il y en a, elle nous montre les fichiers qui sont dans l'index et qui n'ont pas encore été `commit`.
- `git log` permet de voir l'historique des `commits`.
  - Un commit est identifié par un hash (SHA1) unique. Nous utiliserons les premiers caractères de ce hash pour revenir à ce `commit` ou faire d'autres manipulations.
- `git diff` montre les changements effectués entre le dernier `commit` et les fichiers "non-committés".
- `git show <commit_hash>` montre le `diff` d'un commit spécifique par rapport à la copie de travail (les fichiers en cours).

Gérer l'index

- `git add <file>` permet d'ajouter des fichiers à l'index. On peut en spécifier autant qu'on veut, séparés par des espaces.
- `git add -p` nous donne les modifications une par une et nous permet de les accepter ou pas, pour pouvoir les `commit` individuellement.
- `git reset <file>` permet de retirer un fichier de l'index.

Gérer les commits

- `git commit -m "Commit message"` crée un nouveau `commit` avec les fichiers indexés.
  - Un `commit` contient toutes les modifications apportées aux fichiers.
  - On peut revenir à un précédent `commit`, ou comparer nos fichiers modifiés avec un précédent `commit`.
  - C'est pourquoi il est important de faire des `commits` souvent, et de les décrire précisément.
- `git checkout -- <file>` permet de "reset" le fichier à son état lors du précédent `commit`. Cela ne fonctionne que lorsque le fichier n'est pas encore dans l'index.
- `git revert <commit_hash>` crée un nouveau `commit` identique au `commit` spécifié, de sorte que l'état des fichiers soit remis à leur état de ce moment-là.
  - Comme un nouveau `commit` est créé, un message de `commit` va vous être demandé.
- `git reset --hard <commit_hash>` supprime complètement les `commits` qui suivent le `commit` spécifié. (Manipulation déconseillée.)

Déplacer la HEAD

- `git checkout <commit_hash>` détache la **HEAD** pour la faire pointer sur un commit plutôt qu'une branche.
- `git checkout <branch>` attache la **HEAD** à `master` ou une autre branche.

Gérer les branches

- `git branch <branch_name>` crée une nouvelle branche. Pour aller sur cette branche, il faut y attacher la **HEAD**.
- `git checkout -b <branch_name>` crée une nouvelle branche et y attache immédiatement la **HEAD**.

`merge` et `rebase`