Tutorial GIT

Installation https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Installation-de-Git

Overview

Dans nos projets on a globalement 3 branches importantes :

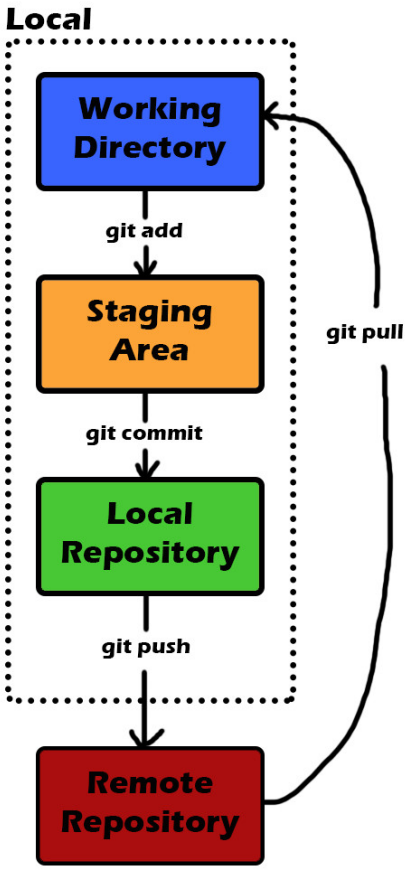
- **master** : La branche par défaut qui n'est pas trop mise à jour. Elle devrait être mise à jour à chaque nouvelle grosse version de l'application.
- **develop** : La branche sur laquelle on peut *merge* nos différentes modifications *fonctionnelles* (càd, qu'on a vérifié que ces modifications marchent correctement, qu'elles ne produisent pas de bug et qu'elles ne contiennent aucun *console.log()*...).
- **prod** : La branche qui représente la version déployée sur le serveur pour les clients. Cette branche à un rôle lors du déploiement d'une nouvelle version mais aussi entre les déploiements pour tester la version actuellement déployé. L'APPLICATION DOIT TOUJOURS ÊTRE DÉPLOYÉE DEPUIS CETTE BRANCHE ET CETTE BRANCHE DOIT RESTER INTOUCHÉE ENTRE 2 DÉPLOIEMENTS.

De plus on peut trouver d'autres branches de type *feature* avec le nom du développeur qui travaille dessus plus un rapide descriptif de la feature à dev dans le nom de la branche.

Tuto c'est quoi git : https://www.jesuisundev.com/comprendre-git-en-7-minutes/

Les commandes principales

A exécuter dans un repo git et toujours garder à l'esprit la branche sur laquelle on se trouve.



Afficher et prendre des infos

- `git status` Afficher le status de la *staging area* locale.
- `git log` Afficher l'historique de la branche actuelle. (`--oneline` pour un historique plus succinct)
- `git branch` Afficher les branches locales.
- `git branch -a` Afficher toutes les branches (locales & remotes).

Sauvegarder des modifications en local

- `git add <path_du_fichier>` Ajoute les modifications du fichier spécifié à la *staging area*.
- `git add .` Ajoute toutes les modifications locales à la *staging area*.
- `git rm --cached <path_du_fichier>` Enlève les modifications du fichier spécifié de la *staging area*.
- `git commit -m "<message_du_commit>"` Créé un commit local avec les modifications de la *staging area*.
- `git commit --amend` Ajoute les modifs de la *staging area* au dernier commit (Attention si dernier commit déjà push sur le repo distant).
- `git stash` Met de côté les modifications locales (tout ce qui n'est pas commité).
- `git stash pop` Ressort les dernières modifications locales misent de côtés.

Se déplacer dans l'historique (attention aux modifications locales non commitées)

- `git checkout <id_commit>` Se déplace sur le commit spécifié.
- `git checkout <nom_de_branche>` Se déplace sur la branche spécifiée.
- `git checkout -b <nom_nouvelle_branche>` Créé une nouvelle branche locale à partir du point actuel de l'historique et s'y déplace.
- `git branch <nom_nouvelle_branche>` Créé une nouvelle branche locale à partir du point actuel de l'historique.
- `git branch -d <nom_branche_à_supprimer>` Supprime une branche locale.

Updates avec le repo distant (toujours avoir un historique local clean : *commited* ou *stashed*)

- `git pull origin <nom_de_branche>` Update la branche actuelle avec les modifications du repo distant. *Toujours pull les modifications de la branche sur laquelle on se trouve.*
- `git rebase <nom_de_branche>` Update la branche actuelle avec les modifications de la branche spécifiée.
- `git push origin <nom_de_branche>` Envoi les modifications locales sur le repo distant. *Toujours push les modifications de la branche sur laquelle on se trouve.*

Revenir en arrière dans l'historique DANGER ZONE

- `git revert <id_commit>` Créé un nouveau commit qui annule les modifications du commit spécifié.
- `git reset <id_commit>` Reset l'historique jusqu'au commit spécifé. (Toutes les modifications aprés ce commit seront perdues)

Workflows

Quelques examples d'utilisation des différentes commandes vues au dessus.

Je veux sauvegarder mes modifications locales

- J'ai des modifications locales que je veux commit.
- *Best Practices* : Commit souvent et commit fonctionnel (dés que quelque chose marche).
- J'ai vérifié que j'étais sur la bonne branche (`git branch`)
- `git add .` ou `git add <nom_du_fichier>`
- `git commit -m <message_du_commit>`
- *Best Practices* : Avoir un message bien explicatif du commit c'est toujours un plus.
- Facultatif : `git push origin <nom_de_ma_branche>` si je veux push mes modifications sur MA branche distante.

Je veux mettre à jour ma branche avec les dernière modifications de *develop*

- J'ai des modifications **commitées** sur ma branche et je souhaite la mettre à jour ma branche avec *develop*.
- `git checkout develop` je vais sur ma branche locale develop
- `git pull origin develop` je récupère la dernière version de la branche develop.
- `git checkout <nom_de_ma_branche>` je reviens sur ma branche.
- `git rebase develop` je mets à jour ma branche avec la dernière version de develop.
- **Merge Conflicts** :
    - Aller voir la personne responsable des modifications qui pose problème.
    - Résoudre les conflits avec elle (VSCode marche bien pour la résolution)
    - `git add .` pour sauvegarder les résolutions.
    - `git rebase --continue` pour valider toutes les résoltions et continuer la mise à jour.
    - (`git rebase --abort` pour annuler la mise à jour et revenir sur le haut de sa branche)

Je veux merge mes modifications sur *develop*

- J'ai des modifications **commitées** sur ma branche que je souhaite envoyer sur la branche *develop*.
- *Best Practices* : J'ai fait plusieurs commits sur ma branche feature pour bien expliquer mes modifications.
- Je mets à jour ma branche avec develop *cf juste au dessus*.
- `git push origin <nom_de_ma_branche>` j'envoi mes modifications sur le repo distant.
- Une fois les modifications sur le repo distant je vais sur GitLab et je créé une **Merge Request** depuis ma branche sur develop.