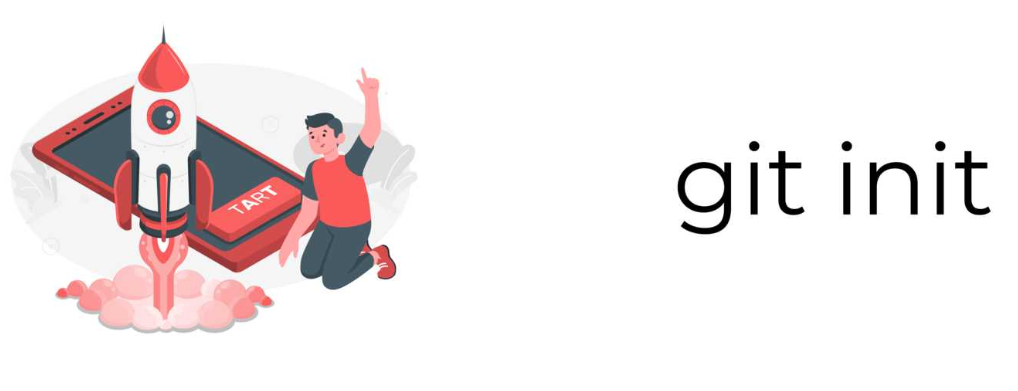
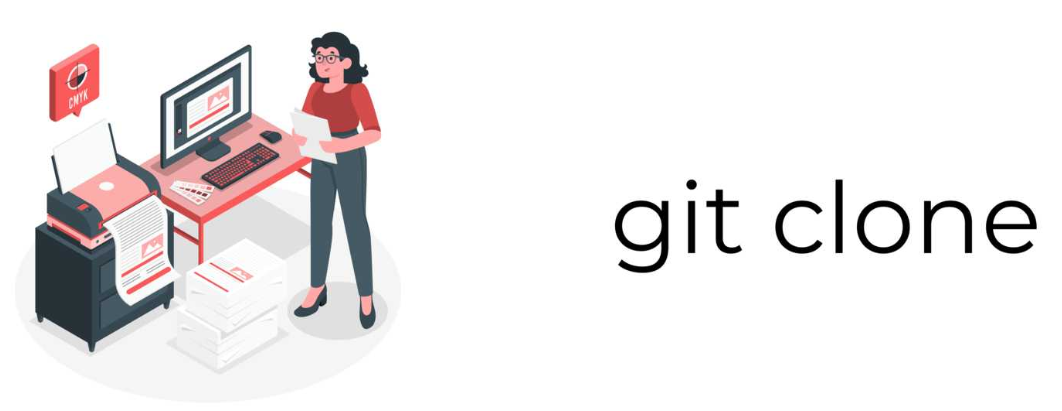
**Les commandes de Git :**



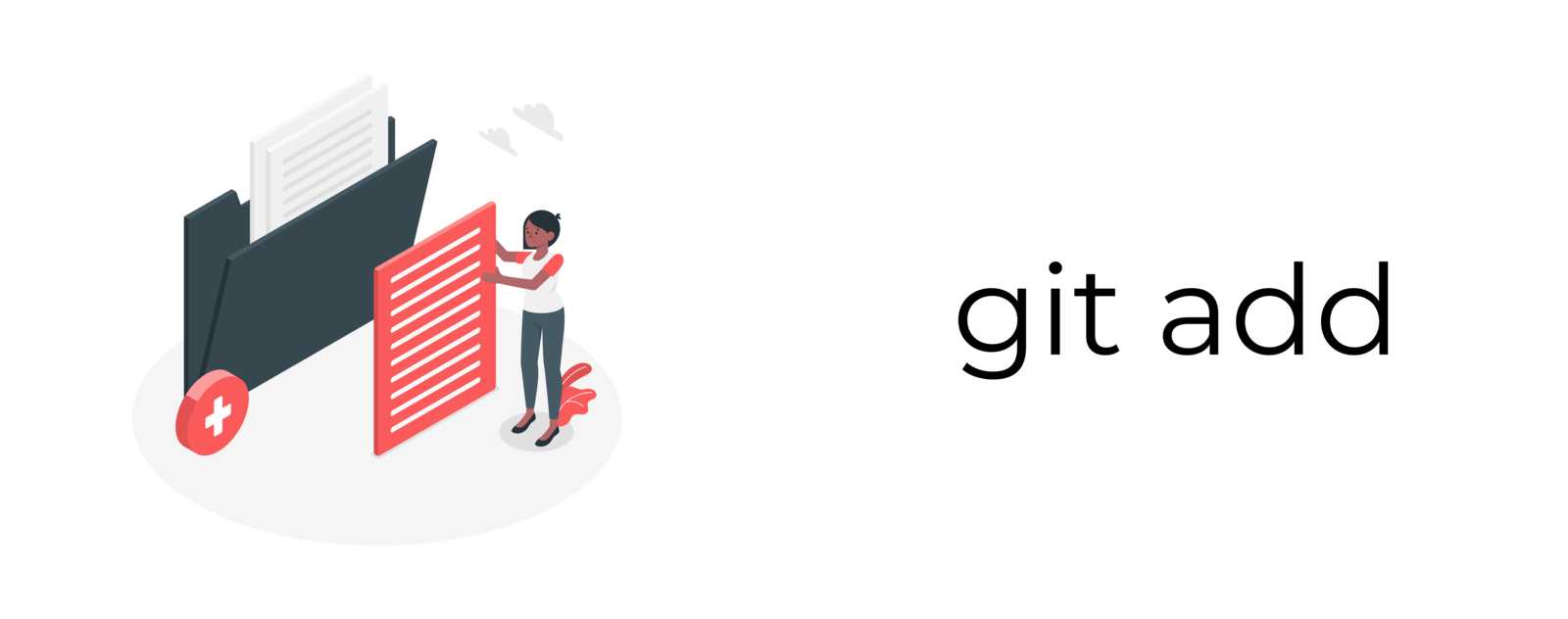
git init <le\_nom\_de\_votre\_repo\_git>

Cette commande initialise un nouveau dépôt Git local. Elle crée un nouveau sous-répertoire .git qui contient tous les fichiers nécessaires au dépôt Git.



git clone <url\_du\_depot\_git>

crée une copie locale d'un dépôt distant. Cette opération inclut non seulement les fichiers du projet mais aussi toutes les branches et tous les historiques de commits.



git add <nom\_du\_ou\_des\_fichier(s)>

Ajoute les fichiers spécifiés à la zone de préparation (staging area), les préparant ainsi pour le prochain commit.

git add \*.extension

sert à ajouter tous les fichier d’extension x

***exemple***: git add \*.html

Pour ajouter tous les fichiers modifiés :

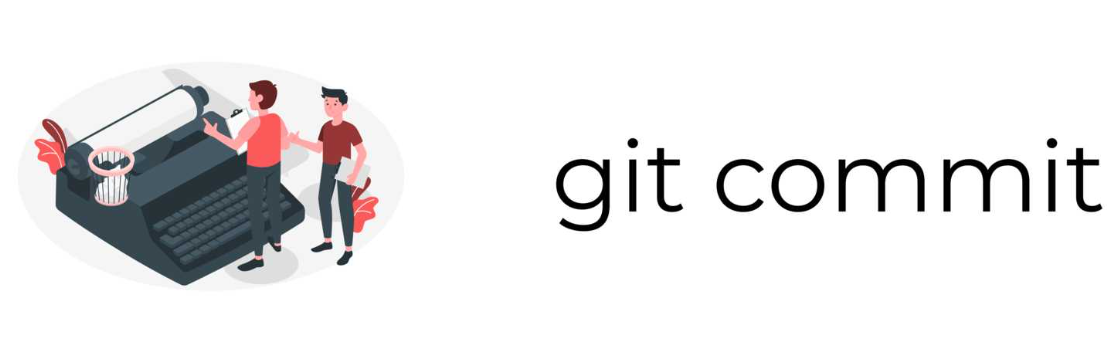
git add .

ou

git add -A

ou

git add -all



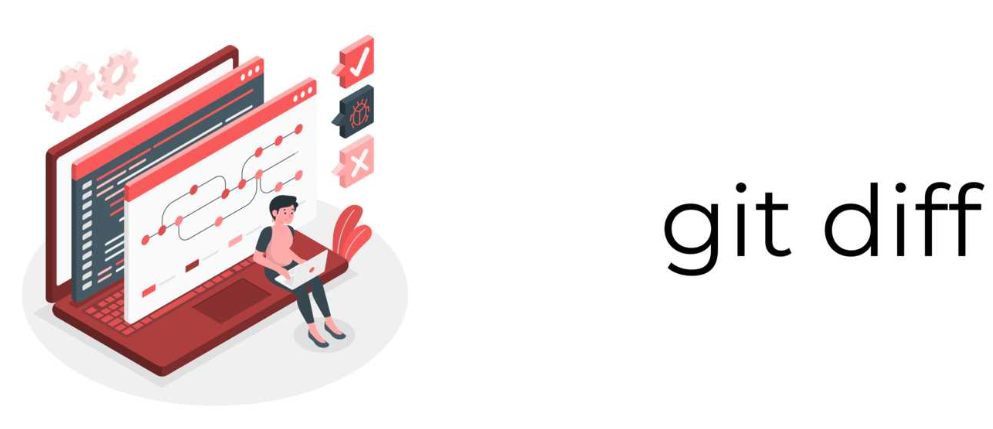
git commit -m “message”

La commande git commit permet d’enregistrer les changements effectués sur votre code dans l’historique git.

Avoir un message de commit clair permet à vos collaborateurs (ou votre “vous” du futur) de s’y retrouver facilement.

git commit -am “message”

La commande est une combinaison pratique qui permet d'ajouter des modifications à la zone de préparation (staging area) et de faire un commit en une seule étape.



git diff

Pour afficher les différences de fichiers qui ne sont pas encore staged.

git diff –staged

Pour afficher les différences entre les fichiers dans staging et la version présente.

git diff <branche\_une> <branche\_deux>

Voir les différences entre les deux branches mentionnées.



git reset <fichier>

Pour retirer le fichier de staging, mais préserve son contenu.

git reset option <hash\_du\_commit>

Cette commande est utilisée pour revenir en arrière dans l'historique de vos commits. Elle a trois options principales qui déterminent à quel point les changements sont annulés :

**--soft** : Annule le commit, garde les modifications prêtes à recommiter.

**--mixed** : Annule le commit et la préparation, laisse les fichiers modifiés disponibles pour re-préparation.

**--hard** : Efface tout jusqu'au commit spécifié, retourne à un état propre.

git reflog

visualiser historique en cas de reset hard.

git revert <commit>

est utilisé pour annuler les modifications d'un commit spécifique tout en conservant l'historique existant en ajoutant un nouveau commit d'annulation.



git status

Pour lister tous les fichiers qui doivent être commit et leur états : untracked et to be committed.

git rm <fichier>

Pour supprimer le fichier de votre répertoire de travail et stage la suppression

git log

Pour lister l’historique des versions de la branche courante.

git log –follow <nom\_du\_fichier>

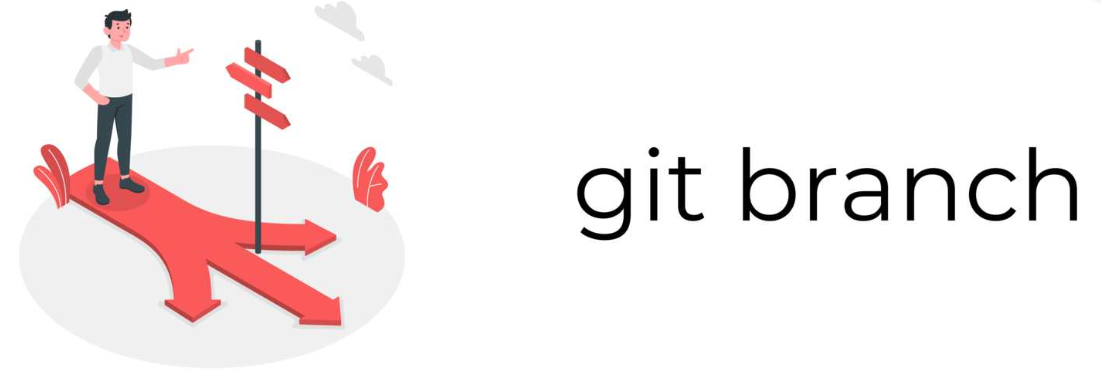
Pour lister l’historique des versions d’un fichier, y compris le renommage des fichiers.

git show <hash\_du\_commit>

Pour afficher les métadonnées et les modifications de contenu du commit spécifiée.

ls

afficher tous les fichiers .



git branch

Pour lister toutes les branches locales dans le dépôt actuel.

git branch <branche>

Pour créer une nouvelle branche.

git branch -d <branche>

Pour supprimer une certaine branche.

git checkout <branche>

Pour passer d’une branche à l’autre.

git checkout -b <branche>

Pour créer une nouvelle branche et switcher directement à celle-ci.

git merge <branche>

Pour fusionner l’historique de la branche spécifiée dans la branche actuelle.

git rebase <branche>

Réapplique les commits d'une branche sur une autre de manière linéaire, gardant l'historique propre et ordonné

git squash<branche>

Combine plusieurs commits en un seul, permettant de nettoyer l'historique des commits et de le rendre plus lisible.

git remote add <nom\_du\_remote (origin ou autre)><lien\_du\_serveur>

Pour lier votre repo local a votre repo remote.

git remote -v

affiche les repos distants.

git tag <hash\_du\_commit>

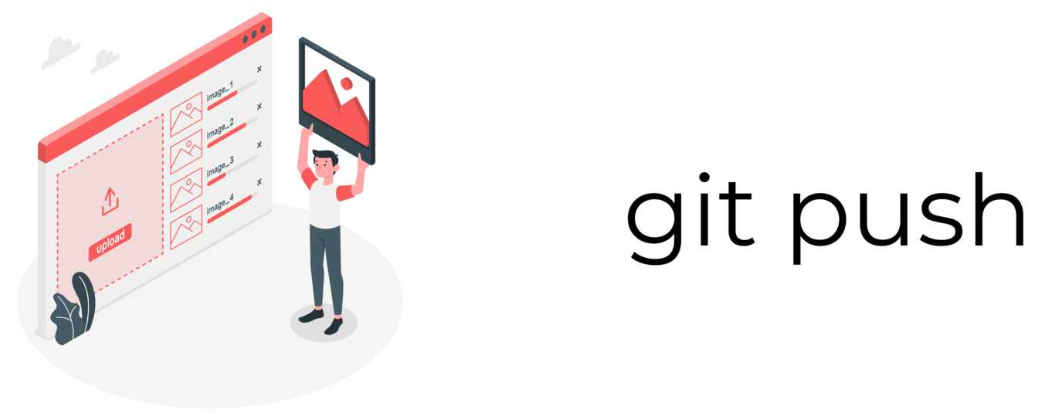
Pour donner un tag au commit spécifié.

git tag --list

afficher liste des tags.

git tag -d <hash\_du\_commit>

supprimer un tag.



git push

Pousse la branche courante vers la branche distante configurée pour le suivi.

git push -u <remote> <branch>

Pousse les modifications et configure la branche sur <remote> comme la branche distante par défaut pour le suivi. Utile pour la première configuration ou pour réinitialiser la configuration de suivi.

git push <nom\_du\_remote> <branche>

Pour push les commits à votre dépôt distant sur de la branche spécifiée.

git push –all <remote>

Pour pousser toutes les branches vers votre repo remote.



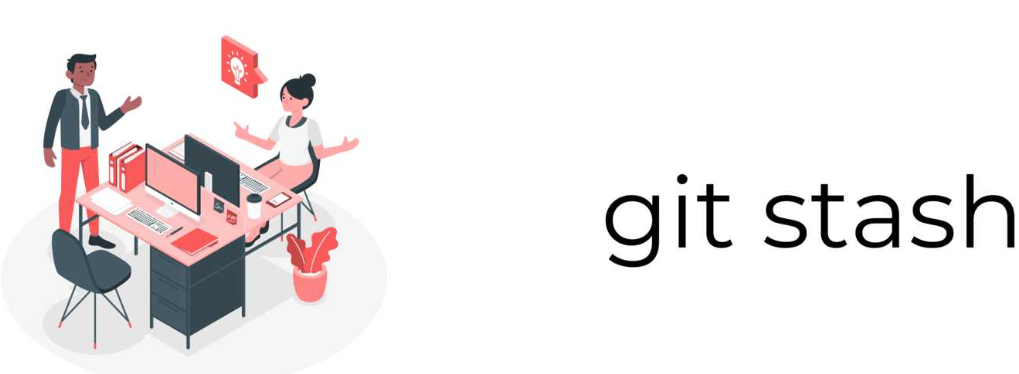
est très utilisée pour récupérer des modifications depuis un dépôt distant et immédiatement mettre à jour le dépôt local à correspondre à ces modifications.

**git pull** est essentiellement un raccourci pour exécuter deux commandes Git :

git pull =git fetch + git merge.

git fetch <remote> <branch>

La commande git fetch permet de récupérer les modifications d’un repo distant sans écraser la version du projet que vous avez localement.



git stash

ou

git stash save

Pour stocker temporairement tous les fichiers tracked modifiés.

**Scénarios d'utilisation :**

**Basculer entre les branches :** Si vous êtes en train de travailler sur une branche et devez rapidement changer pour travailler sur autre chose, vous pouvez utiliser git stash pour mettre de côté vos modifications actuelles, changer de branche, et plus tard revenir et continuer où vous vous étiez arrêté.

**Corriger un bug urgent** : Si vous détectez un bug qui doit être corrigé immédiatement, mais que vous travaillez sur une fonctionnalité sans rapport, git stash vous permet de sauvegarder votre progression, revenir à un état de code stable, corriger le bug, puis reprendre votre travail initial.

git stash pop

Cette commande récupère les modifications stashed les plus récentes, les applique à votre répertoire de travail et supprime cet ensemble de modifications de la pile de stash. C'est utile lorsque vous souhaitez appliquer les modifications stashed et les intégrer immédiatement à votre travail en cours.

git stash list

Pour lister tous vos stashes.

git stash drop

Pour éliminer le dernier ensemble de modifications stashed.

**Git flow :**

git flow init

cette commande est utilisée pour initialiser un nouveau dépôt Git avec le workflow Git Flow.

Gitflow vous guidera à travers le processus d'initialisation en vous posant plusieurs questions. - La première question que Gitflow vous posera concerne le nom de votre branche principale. Par défaut, il est généralement appelé master , choisissez ‘main’ .

Laissez les autres choix par default. - Une fois que vous avez répondu à toutes les questions, Gitflow créera les branches de develop et main dans votre référentiel et initialisera Gitflow avec les paramètres que vous avez choisis.

***Créez votre Branch de fonctionnalité :***

- Assurez-vous d'être sur la branche de développement Exécutez la commande suivante pour créer une nouvelle branche de fonctionnalité :

git flow feature start <nom-de-fonctionnalité>

si on veut ajouter ça dans le dépôt distant :

git push -u <remote> feature/<nom-feature>.

Une fois que vous avez terminé de travailler sur la fonctionnalité, vous pouvez la finaliser en utilisant la commande suivante :

git flow feature finish <nom-de-fonctionnalité>

## Remarque : "Assurez-vous que vous avez terminé la fonctionnalité avant d'exécuter la commande git flow feature finish. Cette commande finalise la fonctionnalité en cours de développement et la fusionne dans la branche de développement principale..