Manipuler GitHub et les Commandes Git

Plan de présentation:

- **C’est quoi Git ?**
- **Comment fonctionne Git ?**
- **Commandes de Git (Local Repository)**
- **Manipuler les branche en Git**
- **GitHub/GitLab**
- **Commandes de Git (Remote Repository)**

C’est Quoi Git ?

Définition

Git est un système de gestion de versions distribué qui permet de suivre les modifications de fichiers et de faciliter le travail collaboratif, en offrant un historique complet et une gestion avancée des branches.

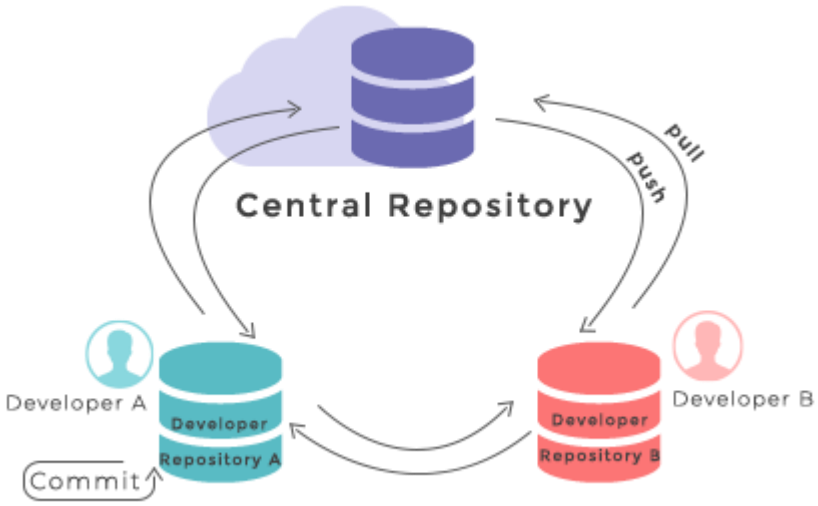
Comment Fonctionne Git ?

Creation d’une Repository

git init: Lorsque vous exécutez cette commande dans un répertoire, Git initialise un nouveau dépôt Git local dans ce répertoire.

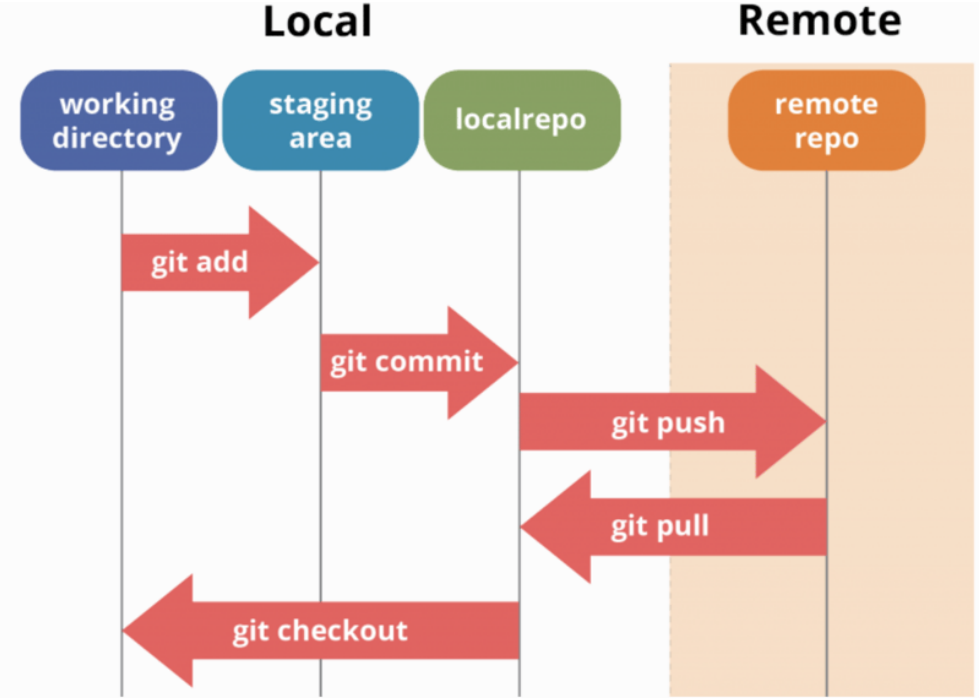
Un dossier caché nommé .git est créé dans le répertoire actuel.

C’est le dossier que Git utilise pour suivre gérer le controle de version.



Le Workflow de Git

- **Working Directory:** Le dossier local où se trouvent les fichiers de votre projet. Ce sont les fichiers que vous modifiez directement.
- **Staging Area:** Une zone temporaire où les modifications sont préparées avant d'être validées (commit) dans le dépôt.
- **Local Repo:** Le dossier "**.git**" dans votre projet qui contient l'historique complet des modifications et des commits du dépôt
- **Remote Repo:** Une version de votre dépôt hébergée sur un serveur ou une plateforme cloud (ex. : GitHub, GitLab, Bitbucket

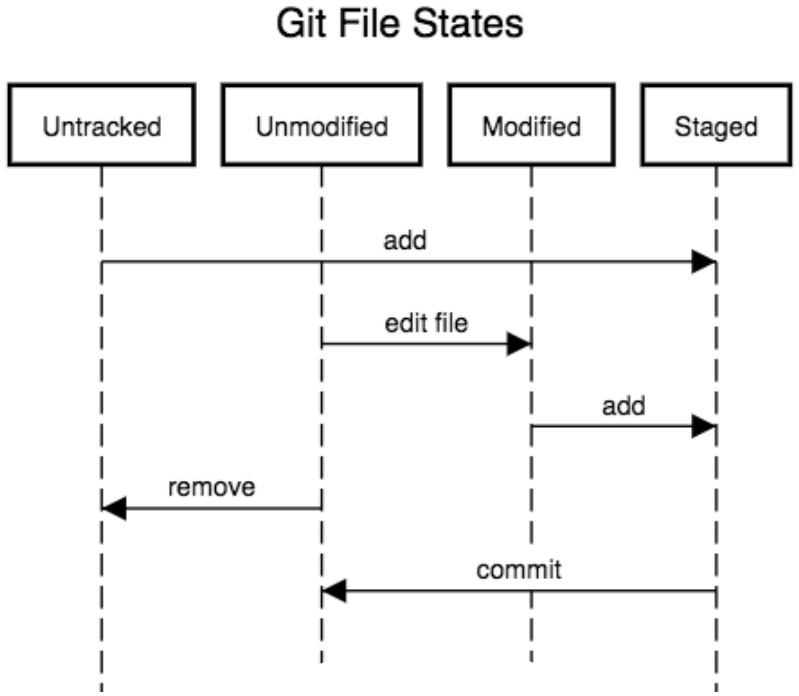


Un Dépot Git (Repository)

Git rassemble dans un dépot (Repository) l'ensemble des données associées au projet. Il vous permet d'enregistrer les versions de votre code

Les États d’un fichier en Git

- **Untracked:** Le fichier existe dans (Working directory) mais n'est pas encore suivi par Git.
- **Unmodified:** Le fichier est suivi par Git et correspond à la version du dernier commit.
- **Modified:** Le fichier est suivi par Git, mais des modifications ont été apportées depuis le dernier commit.
- **Staged(préparé):** Le fichier est suivi et les modifications ont été ajoutées pour le prochain commit.



Commandes de Git (Local Repository)

Configurations en Git

git config management

- **git config [--global] –l:** Afficher la liste des configurations en Git.
- **git config [--global] configName:** Afficher la valeur d'une config spécifique.
- **git config [--global] configName value:** Affecter une valeur a une config spécifique.
- **Git config [--global] --unset configName:** Supprimer un config spécifique.
- **Remarque:** Avec le mot **--global** on peut specifier si la config va etre appliqué sur le dépot uniquement ou sur tous le systéme git

```
naidi@Naiidiine MINGW64 /test (master)
$ git config --global -l
user.email=naidine13015@gmail.com
user.name=naiidiine

naidi@Naiidiine MINGW64 /test (master)
$ git config --global user.name
naiidiine

naidi@Naiidiine MINGW64 /test (master)
$ git config --global --unset user.name

naidi@Naiidiine MINGW64 /test (master)
$ git config --global user.name naiidiine

naidi@Naiidiine MINGW64 /test (master)
$ git config --global -l
user.email=naidine13015@gmail.com
user.name=naiidiine
```

https://www.wildcodeschool.com/blog/comment-mettre-en-place-son-git-guide-complet-pour-d%C3%A9

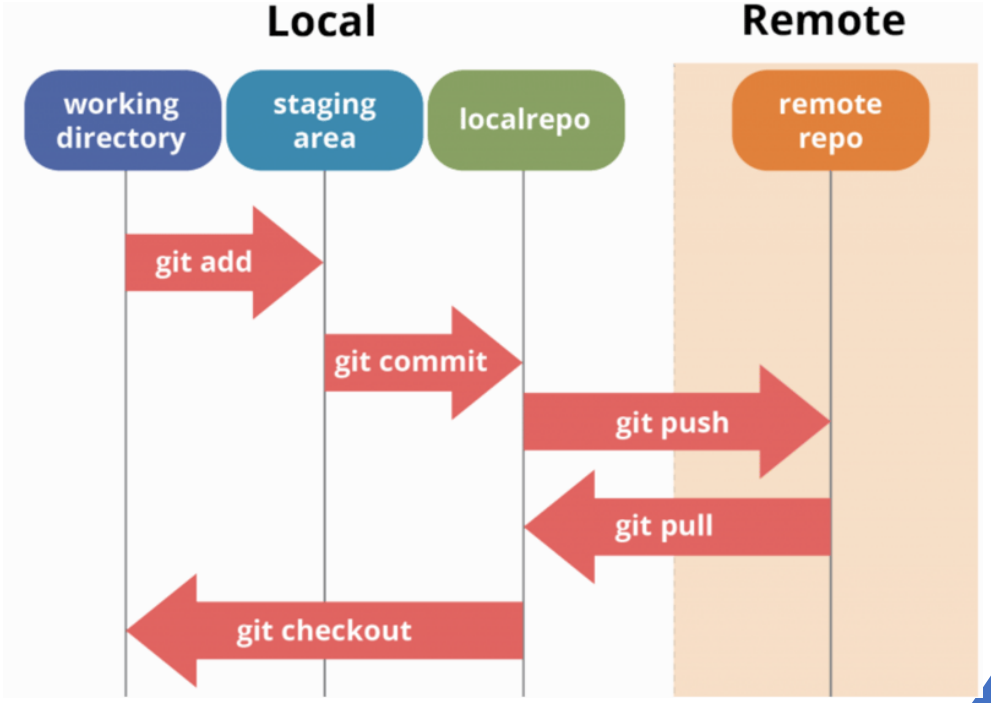
Statut d'un dépôt

- **git status:** Affiche le statut actuel de repository avec des informations detaillées.
- **git status –s:** Affiche en bref le statut de repository

```
naidi@Naiidiine MINGW64 /test (master)
$ touch demofile.txt

naidi@Naiidiine MINGW64 /test (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        demofile.txt
        test.txt

nothing added to commit but untracked files present (use "git add" to
track)

naidi@Naiidiine MINGW64 /test (master)
$ git status -s
?? demofile.txt
?? test.txt
```

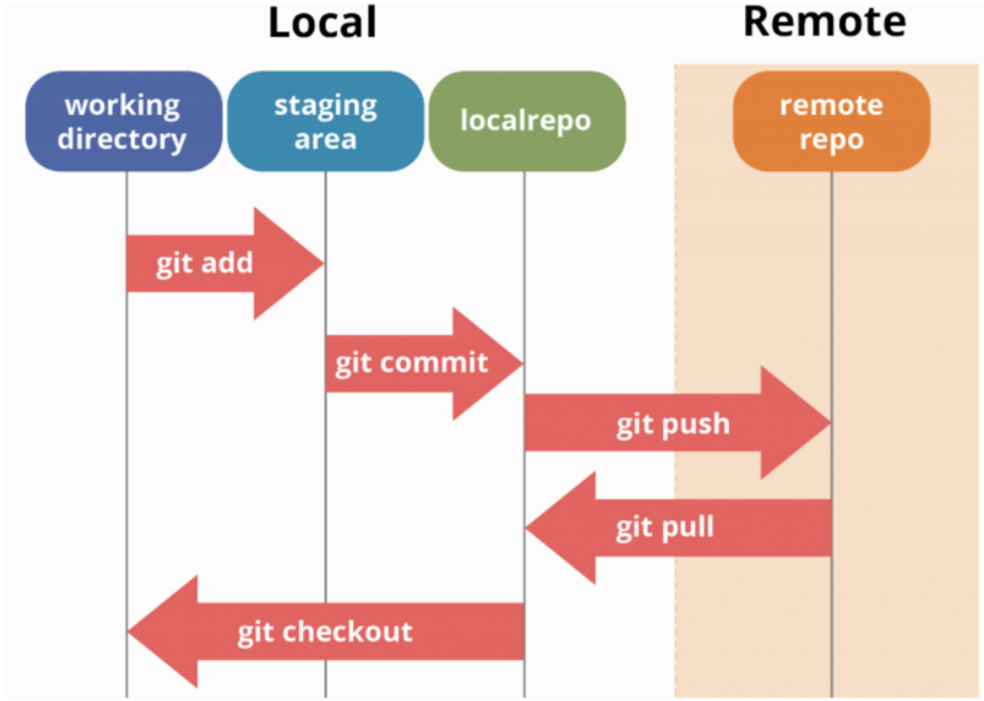
Commandes de preparation (Staging)

- **git add fileName:** transférer le fichier non suivi du “Working Directory” vers “Staging Area”.
- **git add . :** transférer tous les fichiers non suivi vers “Stagin Area”.
- **git diff fileName :** Affiche les modifications apportées.
- **git restore --staged (filename)/(.):** Récupérer les fichiers de “Staging Area”



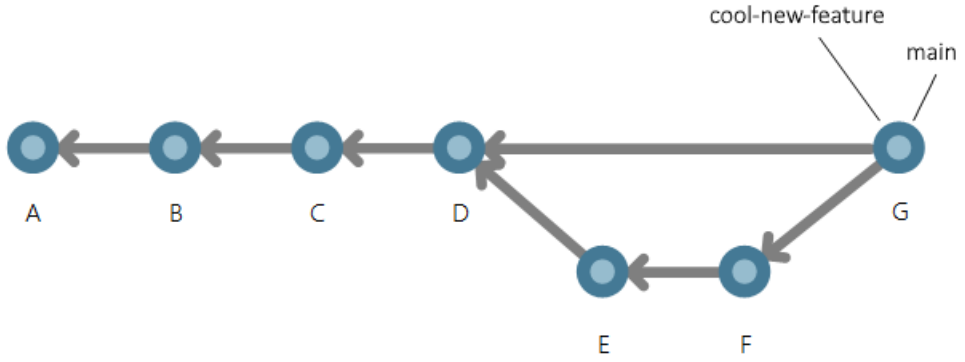
Commande de validation (Committing)

- **git commit –m “message”:** valider les modifications et les transférer de « Staging Areas » vers « Local Repository »



Historique de branche (Log)

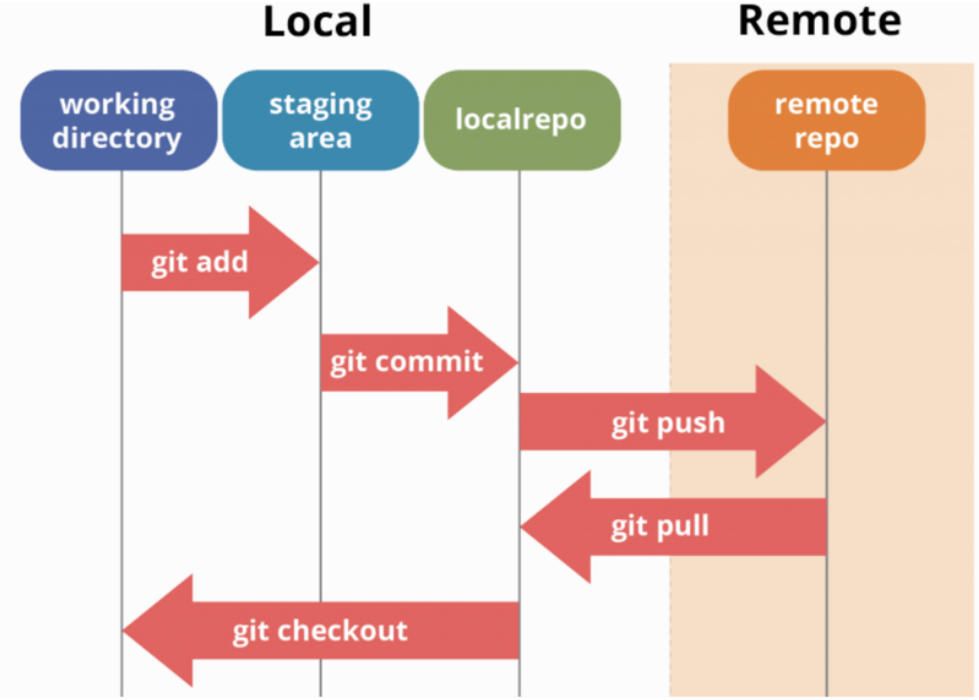
- **git log:** Afficher l'historique de tous les commits.
- **git log--oneline:** Afficher l'historique de tous les commits de manière synthétique.




```
naidi@Naididine MINGW64 /test (dev)
$ git log
commit b7387f11251d3c45a21bd28679d6860c4d08c833 (HEAD -> dev)
Author: naididine <naidine13015@gmail.com>
Date:   Fri Nov 21 09:41:34 2025 +0100

    ajout csv

commit 52e04936a54c272b75e5f575a69adbd5657a88c9 (master)
Author: naididine <naidine13015@gmail.com>
Date:   Fri Nov 21 09:39:30 2025 +0100

    ajout test
```

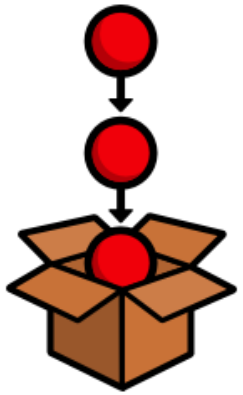
Nettoyage des dépôts

- **git clean –n:** Afficher une liste des fichier non préparé (Not staged).
- **git clean –f:** Supprimer tous les fichiers non préparé



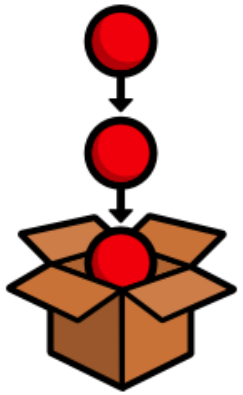
Stashing

Dans Git, **Stashing** (stockage) est un moyen de sauvegarder temporairement les modifications sans les valider **(Commit).** C'est utile lorsque vous souhaitez changer de branche ou travailler sur autre chose, mais que vous n'êtes pas prêt à valider vos modifications actuelles. Stashing enregistre votre travail et laisse votre répertoire de travail **(Workign Directory)** propre, ce qui vous permet de revenir et d'appliquer les modifications ultérieurement.



Les commandes de stashing

- **git stash:** Stocker tous les fichiers non validé **(Uncommitted)** dans un stash
- **git stash list:** Afficher la liste de tous les stashes.
- **git stash pop:** Extraire les fichiers du stash récente.
- **git stash show stash@{ID}:** Afficher details d'un stash spécifique.
- **git stash pop stash@{ID}:** Extraire les fichiers d'un stash spécifique.
- **git stash drop stash@{ID}:** Supprimer un stash spécifique.
- **git stash clear:** Supprimer tous les stash.



Ignorer le suivi d’un fichier/dossier

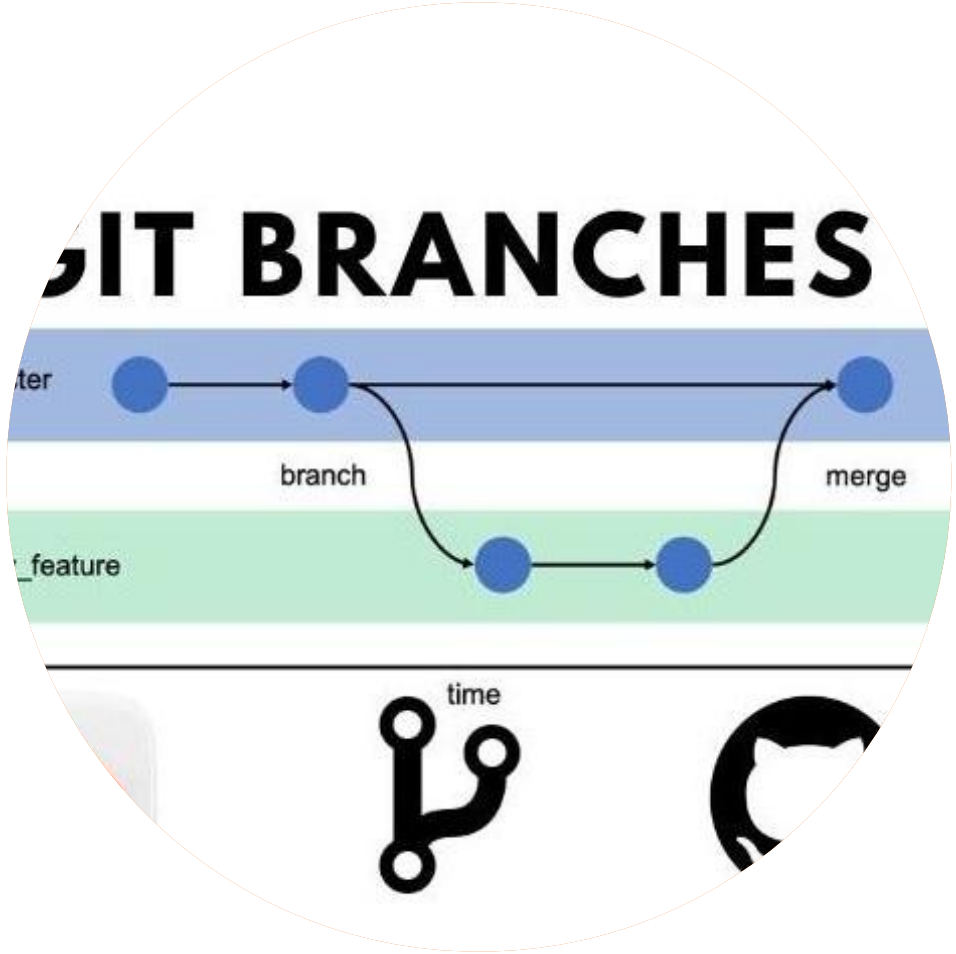
Pour ignorer le suivi d’un fichier ou dossier par , on crée un fichier **“.gitignore”** qui indique à Git quels fichiers ou répertoires ignorer et ne pas suivre dans le dépôt. Il est utilisé pour éviter de suivre des fichiers inutiles ou sensibles.

- **Remarque:** Seuls les fichiers non suivis seront ignorés. Pour ignorer un fichier déjà suivi on utilise la commande suivante: **git rm --cached fileName** puis on l’ajoute dans le fichier **.gitignore** et ensuite on valide les changement (commit).

Pour mieux comprendre le contenu du fichier .gitignore

```
# Example of what a ".gitignore" file contains:
    # *.log       => Ignore all the files with ".log" extension
    # !vip.log    => Ignore all files with ".log" extension except the file "vip.log"
    # index.html  => Ignore every "index.html" file in the project
    # node_packs/ => Ignore every "node_packs" directory in the project
```

Manipuler Les Branches En Git



C’est Quoi le branching

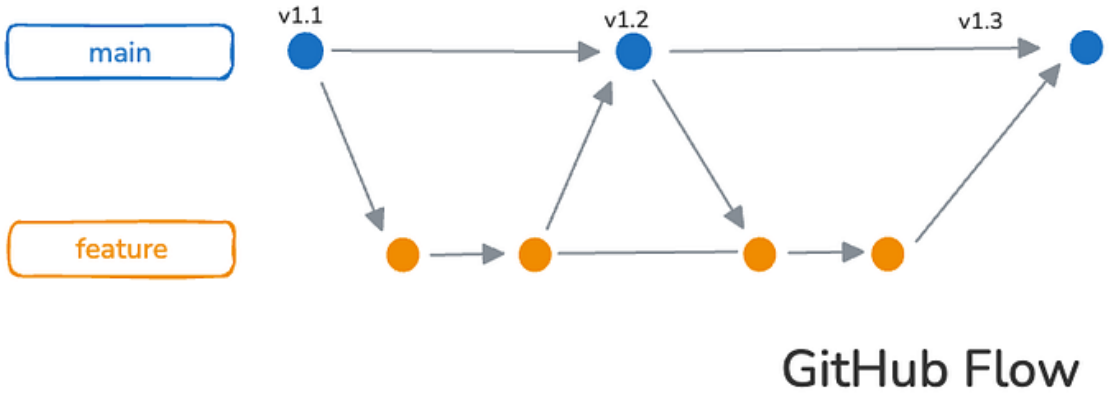
Le branching Git permet de travailler sur différentes versions d’un projet en même temps. Vous pouvez créer des branches pour de nouvelles fonctionnalités, des corrections de bugs ou des expérimentations sans affecter le code principal.

Exemple

Stratégies de branching

https://blog.prateekjain.dev/the-ultimate-guide-to-git-branching-strategies-6324f1aceac2

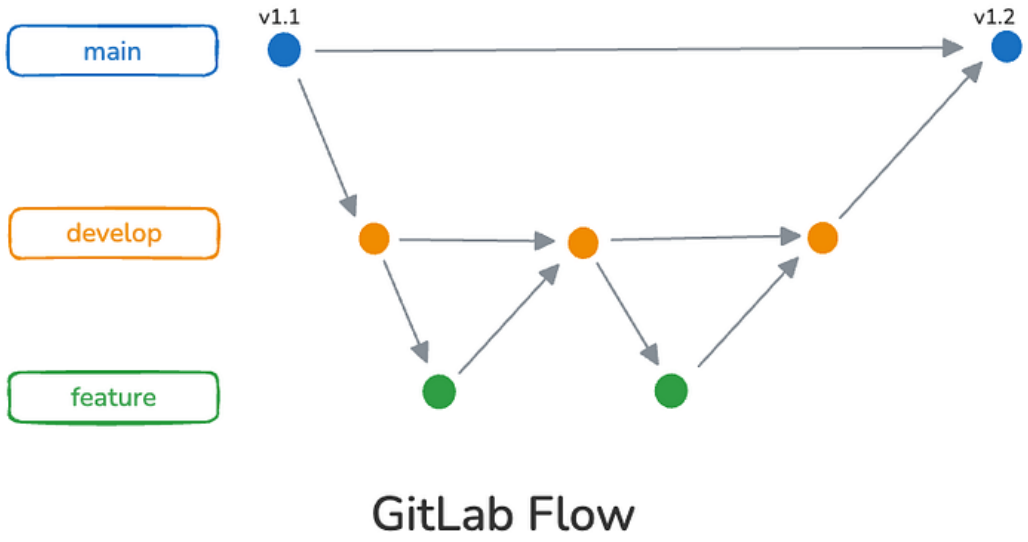
Github flow



GitHub Flow

- **main**— Contient du code prêt pour la production. Chaque commit ici correspond à une version stable.
- **feature/***— Pour développer de nouvelles fonctionnalités. Se ramifie à partir de la branche principale develop et s'y réintègre.

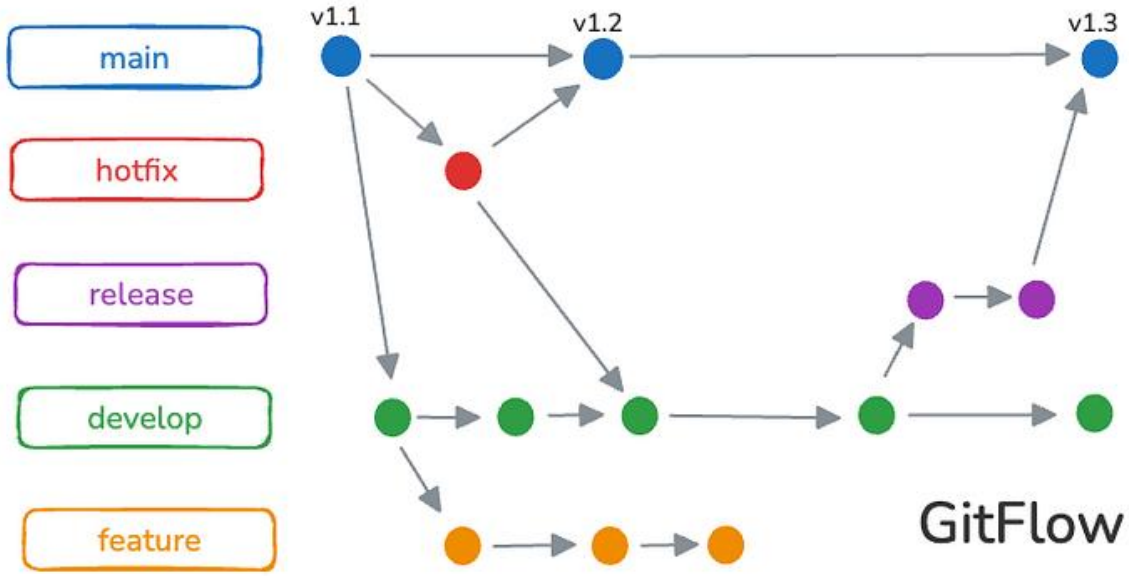
Gitlab flow



GitLab Flow

- **main**— Contient du code prêt pour la production. Chaque commit ici correspond à une version stable.
- **develop**— La branche d'intégration où les nouvelles fonctionnalités sont fusionnées avant leur mise en production.
- **feature/***— Pour développer de nouvelles fonctionnalités. Se ramifie à partir de la branche principale developet s'y réintègre.

Git flow

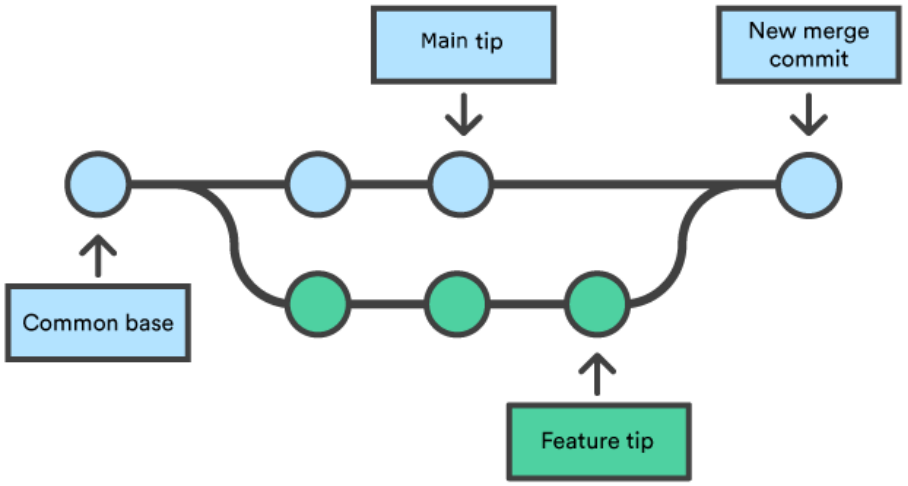


- **main**— Contient du code prêt pour la production. Chaque commit ici correspond à une version stable.
- **develop**— La branche d'intégration où les nouvelles fonctionnalités sont fusionnées avant leur mise en production.
- **feature/***— Pour développer de nouvelles fonctionnalités. Se ramifie à partir de la branche principale developet s'y réintègre.
- **release/***— Utilisé pour préparer une nouvelle version pour la production. Créé à partir de developet finalement fusionné dans mainet develop.
- **hotfix/***— Pour les correctifs urgents en production. Créé à partir de main, puis fusionné dans mainet develop.

Commandes de Branching

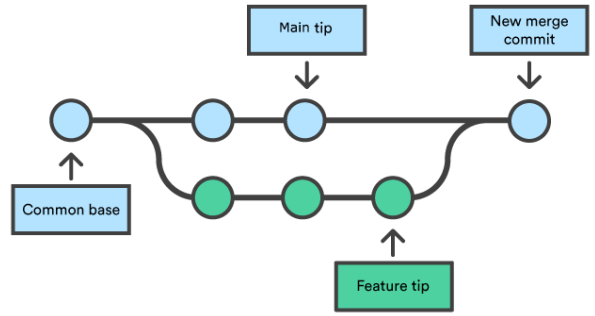
- **git branch:** Afficher la liste des branches
- **git branch branchName:** Créer une branche.
- **git branch checkout branchName:** Changer la branche vers une autre branche.
- **git branch –m newName:** Renommer une branche.
- **git branch -D branchName:** Supprimer une branche spécifique.
- **git checkout –b Branchname :** créer et se déplacer dans la branche
- **git switch :** changer de branche

Branches Merging



Fusion de branches (Branches Merging)

- **git merge branchName:** Fusionner la branche spécifié avec la branche actuelle.
- **Remarque 1:** En cas d'absence de conflits, les modifications sont directement validées.
- **Remarque 2:** En cas de conflits, nous devons les résoudre puis valider les modifications.



Conventions

Branches :

Essayez d'adopter une convention comme :

- feature/<nom_fonctionnalité>
- bugfix/<correction>
- hotfix/<urgence>

Messages de commit :

Essayez de suivre une structure comme :

- feat: ajout du script d'extraction
- fix: correction d'un problème sur le nettoyage
- docs: mise à jour du README

https://www.conventionalcommits.org/fr/v1.0.0/

https://blog.prateekjain.dev/the-ultimate-guide-to-git-branching-strategies-6324f1aceac2

GitHub

C’est Quoi GitHub ?

GitHub est une plateforme cloud qui permet d’héberger et de gérer des dépôts Git. Elle facilite la collaboration sur le code, le suivi des modifications et la gestion des projets.

Avantages de GitHub

- **Controle de version**
- **Collaboration**
- **Suivi des problémes**
- **Hébergement**
- **Communauté Open source**

Commandes de Git (Remote Repository)

Uploader un projet sur GitHub

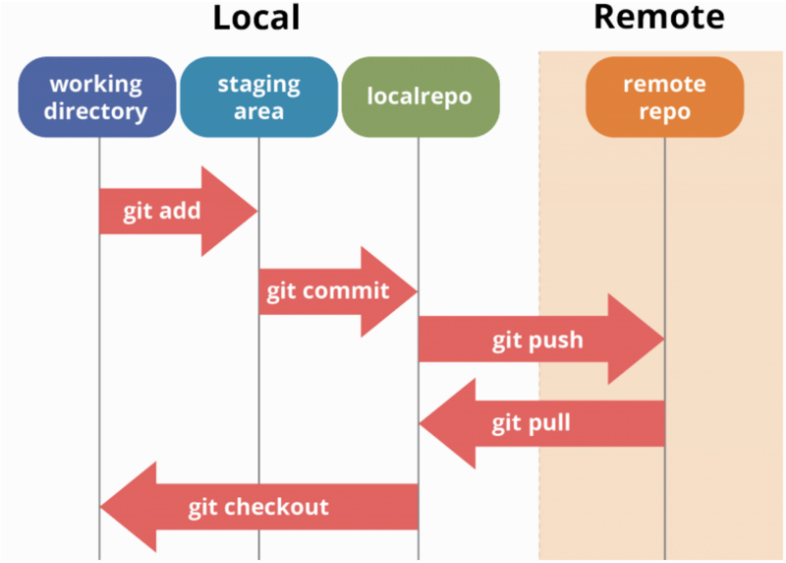
Aprés avoir créer un depot sur GitHub et initialiser un autre sur notre machine, on peut lier entre les deux en utilisant les commandes suivantes:

- **git remote add remoteName remotreURL**
- **git branch –m Main**
- **git push –u remoteName branchName**

Uploader des modifications sur Remote Repository

git remote: Afficher le nom de remote repository

git push [remoteName] [branchName]: Transferer les modifications de “Local Repo” vers ”Remote Repo”.



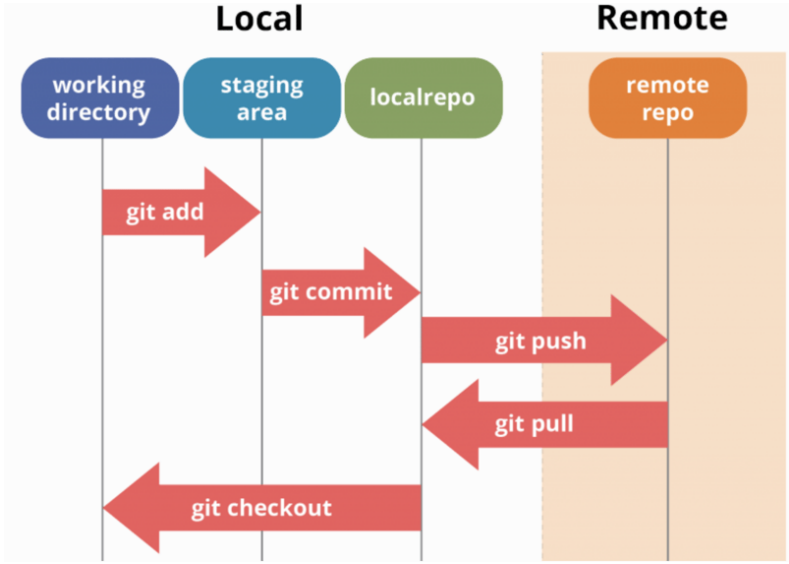
Récupérer des modifications à partir du Remote Repository

Methode 1:

- **git fetch [remoteName] [branchName]:** Obtenez les modifications sans les appliquer sur local repository.
- **git merge remoteName/BranchName:** Fusionner les modifications récuprées avec local repository.

Methode 2:

- **git pull [remoteName] [branchName]:** Elle combine les deux commandes précédentes dans une seul commande.



Ressources en plus

- https://www.atlassian.com/fr/git
- https://rogerdudler.github.io/git-guide/index.fr.html
- https://learnxinyminutes.com/fr/git/
- https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners
- https://www.w3schools.com/git/default.asp
- https://talks.freelancerepublik.com/git-commandes-indispensables-developpeurs/
- https://le-guide-du-secops.fr/2020/09/14/20-commandes-git-incontournables/
- https://www.wildcodeschool.com/blog/comment-mettre-en-place-son-git-guide-complet-pour-débutants
- https://dubreu-data-formations.notion.site/J-aimerais-partager-du-code-avec-ma-collgue-6335efc7ef194d8f9d2882ab7b340bcf

Merci à tous pour votre attention