📘 Synthèse Git : Commandes et Concepts Essentiels

1. Configuration Git

- `git config --global user.name "Nom"`
- `git config --global user.email "email@example.com"`
- `git config --local user.name "NomLocal"`
- `git config --local user.email "emailLocal@example.com"`
- `git config --list` pour voir la configuration
- `git config --global merge.conflictstyle diff3` pour afficher les conflits avec la base commune

2. Dépôt distant

- `git init` → initialise un dépôt
- `git remote add origin URL` → ajoute un dépôt distant
- `git clone [URL]` → cloner un dépôt distant
- `git fetch --all` → met à jour toutes les références distantes
- `git push` → pousse la branche actuelle (les commits) au dépot distant
- `git pull` → récupère les modifications dans la branche actuelle depuis le dépot distant
- `git fetch --all` → met à jour toutes les références distantes

3. Les étapes du code dans Git

1. 📁 Répertoire de travail (Working Directory)

C'est le dossier où tu modifies tes fichiers. Toute modification d'un fichier commence ici.

Exemple : tu modifies `main.py` → il est modifié mais pas encore prêt à être sauvegardé.

---

2. 📥 Zone de staging (Index)

C'est une zone temporaire où tu sélectionnes les fichiers que tu veux inclure dans le prochain commit.

Commande :

`git add main.py`

Le fichier est maintenant "préparé" pour être enregistré dans un commit.

3. ✅ Commit (Historique Git)

Le commit est un instantané des fichiers préparés. Il entre dans l'historique du projet.

Commande

`git commit -m "Ajout de la fonction principale"`

Le fichier est maintenant enregistré de manière permanente dans l'historique Git.

Vue d'ensemble du cycle

```
[ Working Directory ]
        |
        | git add
        v
 [ Staging Area ]
        |
        | git commit
        v
 [ Historique Git ]
```

Commandes utiles associées

Étape	Commande	Description
Suivi des modifications	`git status`	Affiche les fichiers modifiés ou non suivis
Annuler les changements non commités	`git checkout -- fichier`	Restaure la version du fichier depuis le dernier commit
Retirer un fichier du staging	`git reset fichier`	Enlève le fichier de la zone de staging
Voir les modifications	`git diff`	Compare le répertoire de travail avec le staging
Voir les fichiers prêts à être commités	`git diff --staged`	Compare la zone de staging avec le dernier commit

4. Branches

- `git branch` → branches locales
- `git branch -r` → branches distantes
- `git branch -a` → toutes les branches
- `git checkout -b nom-branche` → créer une branche
- `git branch -d nom` → supprimer une branche locale

- `git push origin --delete nom` → supprimer une branche distante
- `git diff commit1 commit2` → comparer 2 versions par exemple HEAD~2 et HEAD ou bien par id de commits

5. Merge vs Merge Request

Type	En local	Sur GitHub/GitLab
`git merge`	✔	✘
Merge Request	✘	✔ (review + tests CI/CD)

6. Fast-Forward dans Git

Un merge en **fast-forward** est une fusion sans commit de merge : Git avance simplement le pointeur de la branche cible si elle n'a pas divergé.

Avantages

- Pas de commit de merge inutile
- Historique plus linéaire

Limites

- Impossible si la branche cible a des commits en plus
- Moins de visibilité sur les branches fusionnées

Commandes utiles

- `git merge --ff-only` : autorise uniquement un fast-forward (pas de commit de merge)
- `git merge --no-ff` : force un commit de merge même si un fast-forward est possible

7. Rebasage

Le **rebase** permet de rejouer les commits d'une branche comme s'ils avaient été faits à partir d'une autre. Cela rend l'historique plus linéaire.

```
 git rebase branche-cible
```

Avantages

- Historique propre et linéaire
- Pas de commit de merge

Attention

- Réécrit l'historique → à éviter sur des branches partagées

Commandes utiles

- `git rebase --continue` : poursuivre après un conflit
- `git rebase --abort` : annuler le rebase

Rebase interactif

Le rebase interactif permet de modifier l'historique des commits : réorganiser, fusionner, supprimer ou modifier des commits. (en réalité il ajoute les commits réorganisés après les commits initiaux)

- `git rebase -i HEAD~3` → lance un rebase interactif pour les 3 derniers commits dans la branche actuelle
- `git rebase -i --root` → lance un rebase interactif pour tous les commits dans la branche actuelle

Commande	Effet
pick	Garde tel quel
reword	Modifie le message
edit	Permet de modifier le commit
squash	Fusionne avec le commit précédent
drop	Supprime le commit

- `git commit --amend` → modifier un commit lors d'un rebase
- `git rebase --continue` → poursuivre le rebase

8. Supprimer ou modifier un commit

- `git reset --hard HEAD~1` → supprime le dernier commit (et les fichiers modifiés)
- `git reset --soft HEAD~1` → garde les modifs mais supprime le commit
- Avec rebase interactif : `git rebase -i HEAD~1` puis `drop`

9. Rôles et permissions

GitLab

Rôle	Créer MR	Fusionner

Developer	✔	✘ (sauf autorisé)
Maintainer / Owner	✔	✔

GitHub

Rôle	Peut merger ?
Write	✔ (sauf règles spéciales)
Admin / Maintain	✔

10. Autres commandes utiles

- `git log --oneline --graph` → affiche l'historique simplifié des commits + branches + merges
- `git commit --amend` → permet de modifier le dernier commit (message ou contenu)
- `git diff --name-only commit1 commit2` → liste les fichiers modifiés entre deux commits