

# Veille Git, PR et CI/CD

## Commandes Git à connaître

Git est un outil de **gestion de version**. Voici les principales commandes à maîtriser pour collaborer efficacement sur un projet.

## 1. Les principaux Workflows Git

Git permet de gérer différentes façons de collaborer. Chaque équipe choisit le workflow adapté à sa taille et à son rythme.

Workflow	Principe	Avantages	Limites / Usage
<b>Feature Branch</b>	Chaque fonctionnalité a sa branche et fusion via PR	Simple, clair, adapté aux petites équipes	Peut créer beaucoup de branches
<b>Git Flow</b>	Branches permanentes : main, develop, feature, release, hotfix	Organisation rigoureuse, idéal pour équipes nombreuses	Complexe pour CI/CD rapide
<b>Trunk Based Development</b>	Développement rapide sur une seule branche, branches courtes	Intégration continue fluide, compatible CI/CD	Nécessite tests solides
<b>Fork Workflow</b>	Chaque contributeur fork le dépôt principal puis propose PR	Idéal open-source, sécurisé	Moins fluide pour équipes internes

## 1. Initialisation et configuration

```
# Créer un nouveau dépôt Git
git init

# Cloner un dépôt existant
git clone https://github.com/utilisateur/nom-du-projet.git

# Configurer ton nom et ton email (identité des commits)
git config --global user.name "TonNom"
git config --global user.email "ton.email@example.com"
```

```
# Vérifier la configuration
git config --list
```

## 2. Suivi et validation des changements

```
# Vérifier les fichiers modifiés
git status

# Ajouter un fichier spécifique à l'index (staging area)
git add fichier.py

# Ajouter tous les fichiers modifiés
git add .

# Enregistrer les changements avec un message
git commit -m "Message clair décrivant la modification"

# Voir l'historique des commits
git log
```

## 3. Gestion des branches

Pull Requests (PR)

**Définition :** Une PR est une **demande de fusion** de code d'une branche vers une autre (souvent main ou develop).

### Étapes

1. Push de ta branche locale :

```
git push origin feature/ma-feature
Créer une PR sur GitHub/GitLab.
```

Rédiger un titre et description claire (fonctionnalité, test, objectif). Les reviewers commentent et approuvent. Après validation → fusion dans la branche cible. Déclenchement automatique des tests CI/CD.

### Bonnes pratiques

- PR petite et ciblée.
- Messages de commit clairs.
- Exiger au moins une review.
- Automatiser tests et linting.
- Utiliser checklists (tests unitaires passés, doc mise à jour, etc.).

## ⚡ Astuces avancées

Rebase pour un historique linéaire :

```
git checkout feature
git rebase main
Multiple approvals : configurer plusieurs reviewers avant fusion.
```

Checklists PR pour garantir qualité et tests.

```
# Créer une nouvelle branche
git branch nom-branche

# Se déplacer sur une autre branche
git checkout nom-branche

# Créer et basculer en une seule commande
git checkout -b feature/nouvelle-fonctionnalite

# Lister toutes les branches
git branch

# Fusionner une branche dans la branche actuelle
git merge nom-branche

# Supprimer une branche locale
git branch -d nom-branche
```

## ⚙ 3. CI/CD

### 🔄 CI – Continuous Integration

- Tests automatiques à chaque commit ou PR.
- Objectif : code toujours intégrable.
- Le code doit être toujours intégrable et fonctionnel.

Exemple GitHub Actions :

```
name: CI
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Installer Python
        uses: actions/setup-python@v4
        with:
```



```
python-version: 3.10
- name: Installer dépendances
  run: pip install -r requirements.txt
- name: Lancer les tests
  run: pytest
```

## CD – Continuous Delivery / Deployment

Automatisation du déploiement après succès des tests.

**Peut être :** - Continuous Delivery → déploiement prêt mais manuel. - Continuous Deployment → déploiement automatique après validation.

## Outils populaires

 Outil	 Fonction principale
GitHub Actions	Automatisation CI/CD intégrée à GitHub
GitLab CI/CD	Pipelines puissants et simples à configurer
Jenkins	Serveur open-source pour automatiser builds/tests
Azure DevOps	Pipelines professionnels intégrés à Azure
CircleCI / Travis CI	Automatisation cloud simple et rapide

## Enjeux

- Qualité du code : revues + tests automatisés.
- Fiabilité : éviter les régressions.
- Collaboration efficace : chacun travaille sans bloquer les autres.
- Livraison rapide : moins d'attente entre le code et la production.

## Types de tests dans CI/CD

- Unitaires : test d'une fonction/méthode.
- Intégration : test de plusieurs composants ensemble.
- End-to-End (e2e) : test complet comme l'utilisateur final.

## 4. Travailler avec un dépôt distant

```
# Lier un dépôt distant
git remote add origin https://github.com/utilisateur/nom-du-projet.git

# Vérifier les dépôts distants
git remote -v

# Envoyer les commits sur le dépôt distant
git push origin nom-branche
```

```
# Récupérer les changements depuis le dépôt distant
git pull origin main

# Mettre à jour la branche locale depuis le dépôt distant
git fetch origin
```

## 5. Synchronisation et correction

```
# Annuler le dernier commit (sans supprimer les modifications)
git reset --soft HEAD~1

# Supprimer complètement le dernier commit
git reset --hard HEAD~1

# Annuler les modifications d'un fichier
git checkout -- fichier.txt

# Comparer les changements entre commits
git diff

# Voir la différence entre ta branche et le dépôt distant
git diff origin/main

# Sauvegarder modifs temporairement
git stash

# Réappliquer modifs
git stash apply
```

## 6. Collaboration et Pull Requests

```
# Créer une nouvelle branche pour ta fonctionnalité
git checkout -b feature/ma-feature

# Commits locaux, puis push sur ta branche distante
git push origin feature/ma-feature
```

## Sur GitHub ou GitLab :

→ Crée une Pull Request pour fusionner la branche dans main

## 7. Git Flow (pour projets structurés)

### b) Git Flow

**Principe :** Workflow complet basé sur plusieurs branches permanentes : - main → code en production - develop → code de développement - feature/\* → nouvelles fonctionnalités - release/\* → préparation de version - hotfix/\* → correction d'urgence

```
# Initialiser Git Flow dans le dépôt
git flow init

# Démarrer une nouvelle fonctionnalité
git flow feature start nom-feature

# Terminer la fonctionnalité (fusion dans develop)
git flow feature finish nom-feature

# Créer une release
git flow release start 1.0.0
git flow release finish 1.0.0
```

---

### Avantages :

- Très structuré, adapté aux cycles de release longs.
- Idéal pour des équipes nombreuses.

### Inconvénients :

- Complexe à gérer pour des projets rapides ou CI/CD.

## c) Trunk Based Development

**Principe** : - Tout le monde travaille sur une seule branche (main ou trunk). - Les branches sont courtes et fusionnées très vite. - Tests automatiques obligatoires avant fusion.

**Avantages** : - Intégration continue fluide. - Compatible avec CI/CD.

**Inconvénients** : - Nécessite une grande discipline et des tests solides.

---

## d) Workflow de Fork

**Principe** : Chaque contributeur forke (duplique) le dépôt principal sur son propre compte, travaille dessus, puis propose une Pull Request vers le dépôt d'origine.

### Commandes typiques :

```
# Cloner le fork
git clone https://github.com/mon-compte/nom-du-projet.git

# Ajouter le dépôt principal comme "upstream"
git remote add upstream https://github.com/original/nom-du-projet.git

# Synchroniser son fork
git fetch upstream
git merge upstream/main
```

---

### Avantages :

- Idéal pour projets open-source.
- Sécurisé : pas d'accès direct au dépôt principal.

### Inconvénients :

- Moins fluide pour les équipes internes.

## 2. Les Pull Requests (PR)

Une Pull Request (PR) est une demande de fusion de code vers la branche principale.

### Étapes d'une PR

**Tu pushes ta branche sur le dépôt distant :** - git push origin feature/nouvelle-fonctionnalite  
- Sur GitHub ou GitLab → cliquer sur "New Pull Request".

**Rédiger un titre et une description claire :** - Ce que fait la modification - Pourquoi elle est utile - Comment la tester - Un collègue relit et commente ton code. - Une fois validée → fusion dans main. - GitHub peut ensuite lancer un pipeline CI/CD.

### Bonnes pratiques

- Une PR = une seule fonctionnalité.
- Rédiger des messages de commit clairs.
- Exiger au moins une review avant fusion.
- Automatiser les tests et le linting sur les PR.


## 8. Commandes pratiques

```
# Voir un résumé des commits récents
git log --oneline --graph --decorate --all




# Ignorer certains fichiers (via .gitignore)
# Exemple de contenu dans .gitignore :
# __pycache__/
# .env
# node_modules/

# Voir qui a modifié une ligne dans un fichier
git blame nom_du_fichier
```

### Astuce bonus

 Utilise git status très souvent : C'est ta meilleure commande pour savoir où tu en es (fichiers modifiés, non suivis, prêts à être commit, etc.).





## Résumé des commandes clés

 Catégorie	 Commande	 Description
Initialisation	<code>git init</code>	Crée un dépôt Git
Clonage	<code>git clone URL</code>	Copie un dépôt distant
Suivi	<code>git add .</code>	Ajoute tous les fichiers modifiés
Commit	<code>git commit -m "message"</code>	Valide les changements
Push	<code>git push origin branche</code>	Envoie les commits
Pull	<code>git pull origin main</code>	Récupère les changements
Branches	<code>git checkout -b nom</code>	Crée et change de branche
Fusion	<code>git merge branche</code>	Fusionne une autre branche
Historique	<code>git log</code>	Affiche les commits
Réinitialisation	<code>git reset --hard HEAD~1</code>	Annule le dernier commit

## 6. Bonnes pratiques à retenir

- Faire des commits petits et clairs.
- Messages de commit à l'infinitif : "Ajouter fonction X", "Corriger bug Y".
- Toujours pull avant push pour éviter conflits.
- Utiliser `.gitignore` pour fichiers sensibles ou volumineux.
- Tester localement avant PR.
- PR = petite, claire, avec description et checklist.

## Ressources utiles

-  Git - Documentation officielle
-  GitHub Docs - Pull Requests
-  GitLab CI/CD Guide
-  Trunk Based Development