📖

Projet Typique

Dans un projet typique, l'utilisation de Git suit souvent un flux de travail organisé pour faciliter la collaboration et garder le code propre. Voici les étapes générales d'un flux de travail Git, basé sur un modèle commun appelé **Git Flow** (ou des variantes de celui-ci).

1. Initialisation du projet et branche principale

- Au début du projet, on initialise le dépôt avec `git init` et crée généralement deux branches principales :
  - `main` **(ou** `master`**)** : La branche principale qui contient toujours le code en production, stable et prêt à être livré.
  - `develop` : Une branche de développement où sont fusionnées toutes les fonctionnalités avant de les intégrer dans la branche `main`.

2. Création d'une branche de fonctionnalité (feature branch)

- Pour chaque nouvelle fonctionnalité, correction de bug, ou amélioration, on crée une **branche de fonctionnalité** basée sur `develop` (ou parfois `main` si on utilise ce modèle). Cela permet de travailler de manière isolée sans impacter les autres développeurs ni le code en production.
- **Créer une nouvelle branche** :

```
git checkout develop
git checkout -b feature/nouvelle-fonction
```

- Les branches de fonctionnalité sont nommées de manière descriptive pour faciliter la compréhension du projet (par exemple, `feature/authentification` ou `bugfix/correction-login`).

3. Développement dans la branche de fonctionnalité

- On travaille dans cette branche pour développer la fonctionnalité en faisant des **commits fréquents**. Par exemple :

```
# Ajouter un fichier au staging
git add fichier_modifié.py

# Faire un commit avec un message
git commit -m "Ajout de la fonctionnalité de connexion"
```

- Il est possible de **mettre à jour sa branche avec les changements récents de** `develop` pour éviter les conflits futurs :

```
git checkout develop
git pull # Récupère les derniers changements de develop
git checkout feature/nouvelle-fonction
```

```
git merge develop # Fusionne develop dans la branche feature pour mettre à jo
ur
```

4. Test et revue de code

- Une fois que la fonctionnalité est prête, il est courant de créer une **Pull Request (ou Merge Request)** si on utilise une plateforme comme GitHub, GitLab, ou Bitbucket.
- Cette demande de fusion permet à d'autres membres de l'équipe de **passer en revue le code** et de s'assurer qu'il respecte les standards, avant qu'il soit fusionné.
- Si des modifications sont demandées, on effectue les changements et **on ajoute des commits** supplémentaires à la branche de fonctionnalité.

5. Fusion de la branche de fonctionnalité dans `develop`

- Après la validation du code (et des tests), on peut **fusionner la branche de fonctionnalité dans** `develop`.
- On bascule sur `develop` et effectue la fusion :

```
git checkout develop
git merge feature/nouvelle-fonction
```

- Une fois la fusion terminée, on peut **supprimer la branche de fonctionnalité** pour garder le dépôt propre :

```
git branch -d feature/nouvelle-fonction
```

- Ensuite, on peut **pousser les changements vers le dépôt distant** :

```
git push origin develop
```

6. Préparation et fusion vers `main` pour la production

- Quand toutes les fonctionnalités sont prêtes à être livrées, on fusionne `develop` dans `main` pour une nouvelle version.
- Souvent, on crée d'abord une branche `release` pour effectuer des tests finaux avant de la fusionner dans `main` :

```
git checkout develop
git checkout -b release/1.0
```

- Après les tests, on fusionne dans `main` :

```
git checkout main
git merge release/1.0
```

- On **pousse vers le dépôt distant** et éventuellement **tague la version** :

```
git tag -a v1.0 -m "Lancement de la version 1.0"
git push origin main --tags
```

- On peut également **fusionner la branche de release dans** `develop` pour garder `develop` à jour :

```
git checkout develop
git merge release/1.0
```

7. Résolution des bugs en production

- Si un bug est découvert en production, on crée une **branche de correction (hotfix)** depuis `main` :

```
git checkout main
git checkout -b hotfix/correction-urgent
```

- Après correction, on **fusionne dans** `main` pour un déploiement rapide et dans `develop` pour garder les branches synchronisées :

```
git checkout main
git merge hotfix/correction-urgent
git checkout develop
git merge hotfix/correction-urgent
```

- Ensuite, on peut **supprimer la branche de correction** :

```
git branch -d hotfix/correction-urgent
git push origin develop
git push origin main
```