📖

Les branches de Git

1. Qu'est-ce qu'une branche ?

Une branche est une version parallèle d'un projet.

- Par défaut, Git crée une branche appelée `main` ou `master`.
- Il est possible de créer des branches pour développer de nouvelles fonctionnalités ou corriger des bugs sans affecter le code principal.
- Une branche est simplement un pointeur vers un commit spécifique dans l'historique.

Exemple visuel simplifié :

```
main: A---B---C
feature:         \---D---E
```

- `main` continue sur son chemin principal (commits A, B, C).
- `feature` diverge pour travailler sur une fonctionnalité spécifique (commits D, E).

2. Commandes essentielles pour les branches

2.1. Afficher les branches existantes

```
git branch
```

- Liste toutes les branches locales.
- Ajouter `r` pour lister les branches distantes :

  ```
  git branch -r
  ```

2.2. Créer une nouvelle branche

```
git branch branche_name
```

- Crée une branche sans basculer dessus.

Exemple :

```
git branch feature-login
```

2.3. Basculer sur une autre branche

```
git checkout branch_name
```

- Permet de passer à une autre branche.

Alternative (plus moderne) :

```
git switch branche_name
```

Exemple combiné :
Pour créer une branche et y basculer immédiatement :

```
git checkout -b feature-login
```

ou

```
git switch -c feature-login
```

2.4. Supprimer une branche

```
git branch -d branch_name
```

- Supprime une branche locale.
- Si la branche n'a pas été fusionnée et que l'on veut la supprimer de force :

  ```
  git branch -D branch_name
  ```

3. Fusionner des branches

Une fois que l'on a terminé un travail sur une branche, on peut la fusionner avec une autre (souvent `main`).

3.1. Fusionner une branche dans `main`

1. Basculer sur la branche dans laquelle on veut fusionner (par ex. `main`) :

   ```
   git checkout main
   ```

2. Fusionner la branche cible (par ex. `feature-login`) :

   ```
   git merge feature-login
   ```

3.2. Résolution des conflits

Si Git détecte des modifications incompatibles entre les branches, il signale un conflit.

Étapes pour résoudre un conflit :

1. Git marquera les fichiers en conflit dans l'éditeur ou dans la sortie de `git status`.
2. Ouvrir les fichiers concernés et résoudre **manuellement** les conflits.

   Les conflits sont délimités par des marqueurs comme :

   ```
   <<<<<<< HEAD
   Votre version
   =======
   Version de la branche fusionnée
   >>>>>>> feature-login
   ```

3. Ajouter les fichiers résolus à la staging area :

   ```
   git add conflicted_file
   ```

4. Compléter la fusion avec un commit :

```
git commit
```

4. Rebaser une branche

Le rebase est une autre manière d'intégrer des changements d'une branche à une autre. Il réécrit l'historique pour rendre le flux de commits plus linéaire.

4.1. Exemple :

Avant rebase :

```
main:   A---B---C
feature:        \---D---E
```

Après rebase :

```
main:   A---B---C---D'---E'
```

4.2. Commande :

```
git checkout feature
git rebase main
```

- Cette commande applique les commits de `feature` après `main`.

4.3. Attention :

- N'utiliser `rebase` uniquement que si les commits concernés ne sont pas encore partagés avec d'autres collaborateurs.
- En cas de conflit pendant un rebase, les résoudre comme lors d'une fusion.

5. Stratégies de branches

5.1. Git Flow

Un workflow structuré pour gérer les branches :

- `main` : branche de production.
- `develop` : branche principale pour le développement.
- **Feature branches** : pour développer de nouvelles fonctionnalités (`feature/nom`).
- **Hotfix branches** : pour corriger des bugs critiques sur `main`.

5.2. Trunk-Based Development

Un workflow plus simple :

- Tout le monde travaille sur une seule branche principale (`main`).
- Les fonctionnalités sont développées dans de petites branches temporaires, fusionnées rapidement.

6. Travailler avec des branches distantes

6.1. Pousser une branche vers un dépôt distant

```
git push origin branch_name
```

- Ajouter `u` pour suivre automatiquement la branche distante :

  ```
  git push -u origin branch_name
  ```

6.2. Récupérer une branche distante

Pour lister et créer une branche locale basée sur une branche distante :

```
git fetch
git checkout branch_name
```

6.3. Supprimer une branche distante

```
git push origin --delete branch_name
```

7. Résumé des commandes principales liées aux branches

Action	Commande
Lister les branches	`git branch`
Créer une branche	`git branch nom_branche`
Basculer sur une branche	`git switch nom_branche` ou `git checkout`
Fusionner une branche	`git merge nom_branche`
Rebaser une branche	`git rebase nom_branche`
Supprimer une branche locale	`git branch -d nom_branche`
Supprimer une branche distante	`git push origin --delete nom_branche`
Pousser une branche distante	`git push origin nom_branche`