Les commandes Git de base//avancées

Created By: IMHAMED BOUJEMAA

February 21, 2023

1 Les commandes Git de base

Commande	Description
`git clone`	Cloner un dépôt distant
`git pull`	Récupérer et fusionner les changements d'un dépôt distant
`git add`	Ajouter des modifications à l'index
`git commit`	Enregistrer les modifications dans le référentiel
`git push`	Envoyer les modifications locales vers un dépôt distant
`git status`	Afficher l'état des fichiers
`git branch`	Afficher et gérer les branches
`git checkout`	Basculer entre les branches ou les versions
`git merge`	Fusionner les modifications de deux branches

Table 1: Les commandes Git de base

2 Les commandes Git avancées

Commande	Description
`git log`	Afficher l'historique des modifications
`git blame`	Afficher qui a modifié chaque ligne d'un fichier
`git diff`	Afficher les différences entre les fichiers
`git reset`	Annuler les modifications dans l'index
`git stash`	Enregistrer les modifications en cours pour plus tard
`git tag`	Marquer un commit avec un tag
`git remote`	Gérer les dépôts distants
`git cherry-pick`	Appliquer un commit spécifique à une branche

Table 2: Les commandes Git avancées

3 Explication des commandes

Voici une brève explication de chaque commande :

- `git clone` : créer une copie locale d'un dépôt distant.
- `git pull` : récupérer et fusionner les modifications d'un dépôt distant.
- `git add` : ajouter des modifications à l'index.

- `git commit` : enregistrer les modifications dans le référentiel.
- `git push` : envoyer les modifications locales vers un dépôt distant.
- `git status` : afficher l'état des fichiers.
- `git branch` : afficher et gérer les branches.
- `git checkout` : basculer entre les branches ou les versions.
- `git merge` : Fusionne une branche dans la branche courante. Cette commande combine les modifications de deux branches et crée un nouveau commit de fusion.

3.1 Les commandes Git avancées

Voici une liste de commandes Git avancées :

Commande	**Utilisation**	**Description**
git stash	git stash save	Permet de sauvegarder temporairement les modifications locales sans avoir à les valider.
git cherry-pick	git cherry-pick ¡commit¿	Permet de récupérer un ou plusieurs commits d'une branche pour les intégrer dans une autre branche.
git bisect	git bisect start	Permet de chercher un commit qui a introduit un bogue en faisant une recherche binaire entre deux points d'une branche.
git clean	git clean -f	Permet de supprimer les fichiers non suivis par Git.
git reset	git reset ¡commit¿	Permet de déplacer le HEAD et la branche courante vers un commit spécifié sans perdre les modifications.

Table 3: Les commandes Git avancées

Extension	**Description**
-a	Ajoute automatiquement les fichiers modifiés et supprimés avant de valider les changements.
-m "message"	Utilise le message donné comme message de validation.
-u	Ajoute les modifications de fichiers mis sous suivi à la validation.
-f	Force l'opération à s'exécuter même si elle peut causer la perte de modifications ou d'historique.
–amend	Modifie le commit précédent pour y inclure les modifications actuelles.
-b nom_branche	Crée une nouvelle branche nom_branche et s'y positionne.
-d nom_branche	Supprime la branche nom_branche.

Table 4: Exemples d'extensions de commandes Git

4 Exemple d'utilisation de Git

Supposons que nous avons un dossier de travail contenant le code source d'un projet, et que nous voulons le suivre avec Git. Nous commençons par initialiser un référentiel Git dans ce dossier en utilisant la commande suivante :

```
git init
```

Maintenant, nous pouvons ajouter les fichiers de notre projet à l'index Git en utilisant la commande suivante :

```
git add .
```

Cela ajoutera tous les fichiers du répertoire de travail à l'index. Nous pouvons maintenant valider ces modifications en utilisant la commande suivante :

```
git commit -m "Initial commit"
```

Cela va créer un nouveau commit dans l'historique Git contenant tous les fichiers que nous avons ajoutés à l'index.

Supposons maintenant que nous avons travaillé sur notre code et que nous avons effectué des modifications dans certains fichiers. Nous pouvons voir l'état actuel de ces fichiers en utilisant la commande suivante :

```
git status
```

Cela nous montrera quels fichiers ont été modifiés depuis la dernière validation.

Nous pouvons ajouter ces modifications à l'index Git en utilisant la commande suivante :

```
git add nom_du_fichier_modifié
```

Cela ajoutera les modifications du fichier spécifié à l'index Git. Nous pouvons ensuite valider ces modifications en utilisant la commande suivante :

```
git commit -m "Modification dans le fichier nom_du_fichier_modifié"
```

Nous pouvons également fusionner des branches en utilisant la commande suivante :

```
git merge nom_de_la_branche
```

Cela fusionnera la branche spécifiée avec la branche courante. Si des conflits surviennent pendant la fusion, Git nous avertira et nous devrons les résoudre manuellement.

Enfin, nous pouvons pousser nos modifications vers un référentiel distant en utilisant la commande suivante :

```
git push
```

Cela poussera les modifications locales vers le référentiel distant spécifié.

4.1 Conclusion

Git est un outil puissant et flexible pour la gestion de version de code source. Nous avons vu les commandes Git les plus utilisées pour un développement de base, ainsi que certaines commandes moins utilisées mais tout aussi utiles dans des situations spécifiques. En fin de compte, il est important de comprendre comment Git fonctionne pour pouvoir utiliser pleinement ses fonctionnalités. J'espère que ce document aidera à approfondir votre connaissance de Git. `https://github.com/IMHAMEDBOUJEMAA/Master_Course_1_2/`.