Dépôts à distance avec Git

Steve Kossouho

Contents

1 Concepts de base de Git

1.1 Qu'est-ce qu'un dépôt Git ?

Un dépôt Git, ou "repository", est essentiellement l'ensemble de votre projet suivi par Git. Il comprend tous les fichiers de votre projet ainsi que l'historique de toutes les modifications que vous avez effectuées. Un dépôt Git est créé en exécutant la commande `git init` dans le répertoire racine de votre projet.

1.1.1 Répertoire de travail

Le répertoire de travail est l'endroit où vous effectuez les modifications sur vos fichiers. Il s'agit de l'endroit où vous pouvez voir et modifier les fichiers présents dans le dépôt Git. Toutes les modifications effectuées dans le répertoire de travail ne sont pas automatiquement suivies par Git. Pour que Git suive ces modifications, vous devez les ajouter à la "zone de préparation" (stage) avec la commande `git add`.

```
# Ajouter un fichier au stage
git add mon_fichier.txt
```

1.1.2 Stage

Le "stage", ou "zone de préparation", est l'étape intermédiaire entre le répertoire de travail et le dépôt Git. Après avoir apporté des modifications à vos fichiers dans le répertoire de travail, vous pouvez les ajouter à la zone de préparation avec la commande `git add`. C'est ici que vous rassemblez toutes les modifications qui seront incluses dans le prochain commit.

```
# Ajouter tous les fichiers modifiés au stage
git add .
```

1.2 Le répertoire `.git`

Le répertoire `.git` est un répertoire caché à la racine de votre projet qui contient toutes les informations dont Git a besoin pour gérer votre dépôt. Ce répertoire est créé lorsque vous initialisez un nouveau dépôt avec `git init`. Voici quelques-uns des éléments qu'il contient :

- `config` : Le fichier de configuration du dépôt.
- `HEAD` : Un pointeur vers la branche ou le commit actuellement check-out.
- `objects` : Le stockage de tous les commits et autres objets Git.
- `refs` : Les pointeurs vers les commits, comme les branches et les tags.
- `index` : La zone de préparation (ou "stage") pour le prochain commit.

Il est important de ne jamais modifier ou supprimer manuellement les fichiers dans le répertoire `.git`, car cela pourrait corrompre votre dépôt.

1.3 Qu'est-ce qu'un commit ?

Un commit dans Git représente une "photographie" de votre projet à un moment précis. Chaque commit contient :

- Un ou plusieurs parents : Les commits qui l'ont précédé.
- Un ensemble de modifications : Les différences entre ce commit et ses parents.
- Un message de commit : Une description de ce qui a changé dans ce commit.
- Un auteur : La personne qui a effectué le commit.
- Une date : Quand le commit a été effectué.

Un commit est identifié par un hash SHA-1 unique, qui est basé sur le contenu du commit. Vous pouvez voir l'information d'un commit spécifique avec la commande `git show <commit-hash>`.

Un commit est un élément clé du modèle de données de Git. Il permet de suivre l'évolution de votre projet dans le temps et facilite la collaboration entre plusieurs développeurs.

1.3.1 Historique du dépôt

L'historique du dépôt est une liste des commits précédents, permettant de suivre l'évolution de votre projet dans le temps. Chaque commit est accompagné d'un message décrivant les modifications effectuées. Pour consulter l'historique du dépôt, utilisez la commande `git log`.

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

1.3.2 Fichiers ignorés : `.gitignore`

`.gitignore` est un fichier spécial qui indique à Git les fichiers ou répertoires à ignorer. Parfois, certains fichiers, comme les fichiers de logs ou les fichiers temporaires, n'ont pas besoin d'être suivis par Git. Vous pouvez ajouter ces fichiers ou répertoires au fichier `.gitignore` et Git les ignorera.

```
# Exemple de .gitignore
*.log
node_modules/
```

1.4 Informations sur le dépôt

1.4.1 Voir l'état du dépôt : `status`

La commande `git status` affiche l'état actuel de votre dépôt. Elle vous indique quels fichiers ont été modifiés et sont en attente d'être commités, quels fichiers ne sont pas suivis par Git, et l'état de votre branche actuelle. C'est une commande très utile pour obtenir un aperçu rapide de ce qui se passe dans votre dépôt.

```
# Afficher l'état du dépôt
git status
```

1.4.2 Voir l'historique du dépôt : `log`

La commande `git log` permet de voir l'historique des commits du dépôt. Chaque entrée dans le log représente un commit, avec son auteur, la date du commit et le message du commit. C'est un outil puissant pour comprendre l'historique de votre projet.

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

1.4.3 Afficher les différences depuis le dernier commit : `diff`

La commande `git diff` affiche les modifications apportées aux fichiers depuis le dernier commit. Elle montre les lignes qui ont été ajoutées ou supprimées. C'est un excellent moyen de vérifier vos modifications avant de les commiter.

```
# Afficher les différences depuis le dernier commit
git diff
```

1.4.4 Afficher le différentiel entre deux commits : `diff`

La commande `git diff` peut également être utilisée pour voir les différences entre deux commits spécifiques. Vous devez spécifier les identifiants des deux commits entre lesquels vous voulez voir les différences. C'est une fonctionnalité très utile pour comprendre les modifications effectuées entre deux points précis de l'historique du projet.

```
# Afficher le différentiel entre deux commits
git diff <id_commit1> <id_commit2>
```

1.5 Commencer à travailler dans un dépôt

1.5.1 Prendre en compte des modifications : `add`

La commande `git add` est utilisée pour ajouter des modifications dans le répertoire de travail à la zone de préparation, également appelée "stage". C'est une étape préparatoire pour le commit. Vous pouvez ajouter un fichier spécifique, plusieurs fichiers ou tous les fichiers modifiés à la fois.

```
# Ajouter un fichier spécifique
git add mon_fichier.txt

# Ajouter tous les fichiers modifiés
git add .
```

1.5.2 Créer une nouvelle étape d'historique : `commit`

Après avoir ajouté vos modifications à la zone de préparation avec `git add`, vous pouvez créer un nouveau commit avec la commande `git commit`. Un commit est une "photographie" de votre projet à un moment donné. Chaque commit est accompagné d'un message décrivant les modifications apportées.

```
# Créer un nouveau commit avec un message
git commit -m "Description des modifications"
```

1.5.3 Messages de commit : règles d'écriture, grammaire

Le message de commit devrait décrire clairement ce qui a été modifié et pourquoi. Utilisez le temps présent et soyez bref et concis. La première ligne devrait être un résumé de moins de 50 caractères, suivi d'une ligne vide, puis d'une description plus détaillée si nécessaire.

```
# Exemple de bon message de commit
git commit -m "Ajoute la fonctionnalité X

Cette fonctionnalité permet à l'utilisateur de faire Y et Z. Cela a été demandé dans l'iss
```

1.6 Faire des retouches

1.6.1 Revenir en arrière

1.6.1.1 Annuler des étapes d'historique mais garder mes modifications : `reset`

La commande `git reset` permet de revenir en arrière dans l'historique des commits tout en gardant les modifications dans le répertoire de travail. C'est utile si vous voulez réorganiser vos commits ou modifier vos messages de commit.

```
# Annuler le dernier commit mais garder les modifications
git reset HEAD~1
```

1.6.1.2 Revenir en avant dans les étapes d'historique : `reset`

La commande `git reset` permet de revenir en avant dans l'historique des commits, et c'est utile si vous êtes revenu en arrière dans l'historique et que vous avez immédiatement changé d'avis.

```
# Revenir n commits en avant dans l'historique
# toujours sans modifier mes fichiers.
git reset HEAD@{n}
```

1.6.1.3 Annuler des étapes d'historique et perdre mes modifications : `reset`

Si vous voulez annuler des commits et perdre toutes les modifications associées, utilisez `git reset` avec l'option `--hard`.

```
# Annuler le dernier commit et perdre les modifications
git reset --hard HEAD~1
```

1.6.1.4 Retirer des modifications de fichiers du stage : `reset`

Si vous avez ajouté des modifications au stage mais que vous souhaitez les retirer avant de commiter, vous pouvez utiliser `git reset`.

```
# Retirer un fichier du stage
git reset mon_fichier.txt
```

1.6.1.5 Annuler toutes les modifications d'un fichier : `reset`

Si vous voulez annuler toutes les modifications que vous avez apportées à un fichier depuis le dernier commit, utilisez `git reset` avec l'option `--hard`.

```
# Annuler toutes les modifications d'un fichier
git reset --hard mon_fichier.txt
```

1.6.2 Raccourcis staging

1.6.2.1 Déplacer ou renommer un fichier : `mv`

Git permet de déplacer ou de renommer un fichier avec la commande `git mv`. Cette commande modifie l'emplacement ou le nom du fichier et prépare la modification pour le prochain commit. C'est un raccourci pour déplacer/renommer le fichier manuellement et ensuite utiliser `git add`.

```
# Déplacer un fichier
git mv chemin_vers_le_fichier/ancien_nom_fichier chemin_vers_le_nouvel_emplacement/nouveau
```

1.6.2.2 Supprimer un fichier : `rm`

Pour supprimer un fichier et préparer la suppression pour le prochain commit, vous pouvez utiliser la commande `git rm`. C'est un raccourci pour supprimer le fichier manuellement et ensuite utiliser `git add`.

```
# Supprimer un fichier
git rm mon_fichier.txt
```

1.6.2.3 Marquer un fichier comme supprimé : `rm`

Si vous avez déjà créé un commit contenant un fichier indésirable (par exemple un fichier de cache), vous souhaitez probablement retirer ce fichier des prochains commits (s'il n'est pas critique).

Par défaut, si vous utilisez la commande `rm` de Git sur ledit fichier, ce dernier sera effectivement marqué comme supprimé pour votre prochain commit.

Il est possible de vouloir marquer correctement le fichier comme supprimé, mais de le conserver intact dans le répertoire de travail.

Pour cela, vous pouvez utiliser l'option `--cached` :

```
git rm mauvais_fichier --cached
```

Pensez ensuite à ajouter ce fichier au contenu du fichier `.gitignore` pour éviter à l'avenir d'effectuer un `git add .` et récupérer les modifications de fichiers indésirables dans l'index git.

1.6.3 Options sur les commits

1.6.3.1 Modifier le message du dernier commit

Si vous voulez modifier le message du dernier commit, vous pouvez utiliser la commande `git commit` avec l'option `--amend`. Cela ouvrira un éditeur où vous pourrez modifier le message.

```
# Modifier le message du dernier commit
git commit --amend
```

1.6.3.2 Options supplémentaires pour créer un commit

La commande `git commit` a de nombreuses options qui vous permettent de personnaliser votre commit. Par exemple, l'option `-a` ajoutera automatiquement toutes les modifications des fichiers suivis au commit. L'option `--no-verify` sautera les hooks de pré-commit.

```
# Créer un commit avec toutes les modifications des fichiers suivis
git commit -a -m "Mon message de commit"

# Créer un commit sans exécuter les hooks de pré-commit
git commit --no-verify -m "Mon message de commit"
```

1.6.3.3 Créer un message de commit avec plusieurs lignes

Si vous voulez créer un message de commit avec plusieurs lignes, n'incluez pas l'option `-m` lorsque vous exécutez `git commit`. Cela ouvrira un éditeur où vous pourrez écrire un message sur plusieurs lignes. La première ligne doit être le résumé du commit et les lignes suivantes peuvent fournir des détails supplémentaires.

```
# Créer un message de commit avec plusieurs lignes
git commit
```

Dans l'éditeur qui s'ouvre, vous pouvez alors écrire votre message :

```
Résume le commit en moins de 50 caractères

Fournit des détails supplémentaires sur les modifications apportées. Explique pourquoi les
```