Dépôts à distance avec Git

Steve Kossouho

Contents

1 Dépôts à distance avec Git **1**
- 1.1 Serveur simple de dépôts Git . . . 1
- 1.2 Solutions installables de serveurs d'usine logicielle . . . 2
- 1.3 Solutions non open-source d'usine logicielle (ex. Github) . . . 2
- 1.4 GitHub : Étapes de création d'un dépôt . . . 2
  - 1.4.1 Configurer les clés SSH (Windows et Linux) . . . 2
  - 1.4.2 Enregistrer les clés SSH dans GitHub . . . 3
  - 1.4.3 Création d'un nouveau dépôt . . . 3
  - 1.4.4 Cloner un dépôt (HTTPS et SSH) . . . 3
- 1.5 Rebase interactif pour les branches de travail . . . 5
- 1.6 Stratégies de fusion des Merge Requests (Gitlab) / Pull Requests (GitHub) . . . 5
  - 1.6.1 Meilleures stratégies de fusion (Github Flow, Gitflow) . . . 5
  - 1.6.2 Créer des gabarits de texte pour les tickets . . . 6
  - 1.6.3 Configurer les accès aux branches, notamment `main/master` . . . 6
- 1.7 Créer un dépôt localement, puis le gérer avec Gitlab ou GitHub . . . 6
  - 1.7.1 Si vous voulez synchroniser en SSH . . . 6
  - 1.7.2 Si vous voulez synchroniser votre dépôt en HTTPS . . . 7
  - 1.7.3 . . . 7
  - 1.7.4 Pousser toutes les branches vers le dépôt à distance (origin) . . . 7

1 Dépôts à distance avec Git

1.1 Serveur simple de dépôts Git

Un serveur Git est un emplacement centralisé où vos dépôts Git sont stockés. Le serveur sert de point de vérité où tous les membres de votre équipe peuvent synchroniser leurs modifications. Un dépôt hébergé sur un serveur est souvent appelé un dépôt "bare" parce qu'il ne contient pas de répertoire de travail visible.

Un exemple de serveur Git simple est un serveur SSH avec quelques dépôts Git "bare".

```
# Sur le serveur
mkdir mon-projet.git && cd mon-projet.git
git init --bare
```

1.2 Solutions installables de serveurs d'usine logicielle

Il existe plusieurs solutions installables pour héberger vos propres serveurs Git. Ces solutions offrent plus de contrôle et sont souvent accompagnées d'outils supplémentaires pour aider à gérer le cycle de vie du développement logiciel.

- **Gitea** : Une solution légère écrite en Go. Facile à installer et à configurer.
- **GitLab** : Une solution robuste avec des fonctionnalités intégrées pour la CI/CD, le suivi des problèmes, et bien plus encore.
- **Bitbucket Server (anciennement Stash)** : Offert par Atlassian, intégré avec d'autres outils populaires d'Atlassian comme JIRA.

1.3 Solutions non open-source d'usine logicielle (ex. Github)

Il existe également des solutions Git hébergées non open-source, parmi lesquelles :

- **GitHub** : L'une des plateformes Git les plus populaires. Elle offre de nombreuses intégrations et est le lieu d'accueil de nombreux projets open-source.
- **Bitbucket** : Offert par Atlassian, il s'intègre bien avec d'autres outils d'Atlassian.
- **GitLab.com** : La version hébergée de GitLab.

Ces solutions sont pratiques car elles ne nécessitent aucune configuration de serveur de votre part. Elles offrent également des fonctionnalités supplémentaires comme les demandes de fusion (pull requests), l'intégration continue/déploiement continu (CI/CD), et les outils de gestion des problèmes.

1.4 GitHub : Étapes de création d'un dépôt

1.4.1 Configurer les clés SSH (Windows et Linux)

Avant de pouvoir interagir avec un dépôt sur GitHub, vous devez configurer vos clés SSH. SSH (Secure Shell) est un protocole de réseau sécurisé qui vous permet de vous connecter à une machine distante. Dans le contexte de Git, nous utilisons SSH pour communiquer en toute sécurité avec GitHub.

Sur Linux et Mac :

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Sur Windows, vous pouvez utiliser Git Bash ou un autre client SSH pour générer une clé.

1.4.2 Enregistrer les clés SSH dans GitHub

Une fois votre clé SSH générée, vous devez l'ajouter à votre compte GitHub. Pour ce faire, ouvrez les paramètres de votre compte GitHub, cliquez sur "SSH and GPG keys", puis sur "New SSH key". Copiez et collez votre clé publique SSH dans le champ "Key".

1.4.3 Création d'un nouveau dépôt

Pour créer un nouveau dépôt sur GitHub, allez sur la page d'accueil de votre compte et cliquez sur le bouton "+",

puis sur "New repository". Donnez un nom à votre dépôt, choisissez s'il doit être public ou privé, puis cliquez sur "Create repository".

1.4.4 Cloner un dépôt (HTTPS et SSH)

Cloner un dépôt est le processus de téléchargement d'une copie d'un dépôt Git existant sur votre machine locale.

Pour cloner via HTTPS :

```
git clone https://github.com/username/repository.git
```

Pour cloner via SSH :

```
git clone git@github.com:username/repository.git
```

1.4.4.1 Notion de branches locales, distantes et homologues distants

---

En Git, vous travaillez souvent avec trois types de branches : locales, distantes et "tracking branches" (ou homologues distants).

- Les branches **locales** existent uniquement sur votre machine locale.
- Les branches **distantes** existent sur le dépôt distant (par exemple, sur GitHub).
- Les **tracking branches** sont des branches locales qui ont une branche distante associée. Elles permettent à Git de savoir à quel endroit du dépôt distant synchroniser vos modifications.

1.4.4.2 Notion de tirer une branche

"Tirer" une branche avec Git signifie télécharger les modifications du dépôt distant et les fusionner dans votre branche locale. La commande typique pour ce faire est `git pull`.

```
git pull origin master
```

Dans cet exemple, `origin` est le dépôt distant et `master` est la branche que vous voulez tirer.

1.4.4.3 Rebase avec un dépôt distant

Rebasing est un moyen de garder votre branche locale à jour avec les modifications apportées au dépôt distant. Plutôt que de fusionner les commits du dépôt distant dans votre branche locale, rebase "déplace" vos commits locaux pour qu'ils apparaissent comme s'ils avaient été créés après les derniers commits du dépôt distant.

```
git fetch origin
git rebase origin/master
```

Ici, `origin` est le dépôt distant et `master` est la branche que vous voulez rebaser.

1.4.4.4 Stratégie lorsqu'on finit de travailler sur une branche

---

Lorsque vous avez fini de travailler sur une branche, il y a plusieurs choses que vous pourriez vouloir faire. Vous pouvez fusionner votre branche dans la branche principale (généralement `master` ou `main`), en utilisant `git merge`. Si votre branche a été poussée sur un dépôt distant, vous voudrez probablement aussi la supprimer du dépôt distant après la fusion. Vous pouvez le faire avec `git push origin --delete branch-name`.

```
git checkout master
git merge feature-branch
git branch -d feature-branch
git push origin --delete feature-branch
```

Dans cet exemple, nous supposons que `feature-branch` est la branche sur laquelle vous avez fini de travailler.

1.5 Rebase interactif pour les branches de travail

Le rebase interactif est un outil puissant qui vous permet de modifier l'histoire de vos commits. Cela peut être particulièrement utile lorsque vous travaillez sur une branche de fonctionnalités ou de corrections de bugs. Vous pouvez utiliser le rebase interactif pour nettoyer votre historique de commit avant de le fusionner dans la branche principale.

Pour démarrer un rebase interactif, utilisez 'git rebase -

i HEAD~n, où n' est le nombre de commits que vous voulez examiner.

```
1 git rebase -i HEAD~5
```

Cela ouvrira un éditeur de texte avec une liste de vos `n` derniers commits, et vous pourrez choisir quoi faire avec chacun (par exemple, les réordonner, les modifier, les supprimer, les écraser).

1.6 Stratégies de fusion des Merge Requests (Gitlab) / Pull Requests (GitHub)

Il existe plusieurs stratégies pour gérer les merge requests (GitLab) ou pull requests (GitHub). Deux des plus courantes sont :

- **Merge Commit** : Crée un nouveau commit qui fusionne la branche de la MR/PR dans la branche cible.
- **Rebase and Merge** : Rebaser d'abord la branche de la MR/PR sur la branche cible, puis fusionner. Cela donne un historique de commits linéaire.

Quelle que soit la stratégie que vous choisissez, assurez-vous de bien tester vos modifications avant de les fusionner.

1.6.1 Meilleures stratégies de fusion (Github Flow, Gitflow)

Il existe plusieurs stratégies de fusion courantes dans l'industrie, dont deux sont particulièrement populaires :

- **GitHub Flow** : Une stratégie simple qui n'utilise qu'une seule branche principale et des branches de fonctionnalités. Lorsqu'une fonctionnalité est prête, elle est fusionnée dans la branche principale.
- **Gitflow** : Une stratégie plus complexe qui utilise deux branches principales (`develop` et `master`), des branches de fonctionnalités, des branches de release et des branches de hotfix.

1.6.2 Créer des gabarits de texte pour les tickets

Les gabarits de texte pour les tickets (également appelés modèles de ticket) peuvent aider à garder vos tickets cohérents et informatifs. Vous pouvez créer des modèles de ticket pour différents types de tickets (par exemple, les bugs, les fonctionnalités, les améliorations) qui guident le rapporteur du ticket sur les informations à inclure.

1.6.3 Configurer les accès aux branches, notamment `main/master`

Il est important de contrôler qui peut pousser sur certaines branches de votre dépôt. Par exemple, vous voudrez peut-être restreindre l'accès à votre branche `main` ou `master` pour vous assurer que seules les modifications approuvées sont fusionnées. Vous pouvez configurer ces restrictions d'accès dans les paramètres de votre dépôt.

1.7 Créer un dépôt localement, puis le gérer avec Gitlab ou GitHub

---

1.7.1 Si vous voulez synchroniser en SSH

Pour synchroniser votre dépôt avec un serveur distant via SSH, vous devez d'abord générer une clé SSH et l'ajouter à votre compte GitLab ou GitHub (comme expliqué ci-dessus).

Ensuite, vous pouvez ajouter votre dépôt distant à votre dépôt local en utilisant `git remote add` :

```
1 git remote add origin git@github.com:username/repo.git
```

1.7.2 Si vous voulez synchroniser votre dépôt en HTTPS

Si vous préférez utiliser HTTPS pour synchroniser votre dépôt, vous pouvez utiliser une URL HTTPS lors de l'ajout de votre dépôt distant :

```
git remote add origin https://github.com/username/repo.git
```

1.7.3

Ajoute au dépôt local un dépôt à distance en lui donnant le nom origin

---

Quelle que soit la méthode que vous choisissez pour synchroniser votre dépôt, vous pouvez utiliser `git remote add` pour ajouter un dépôt distant à votre dépôt local. Dans l'exemple suivant, nous ajoutons un dépôt distant et le nommons `origin` (un nom standard pour le dépôt principal sur lequel vous travaillez).

```
git remote add origin git@github.com:username/repo.git
```

1.7.4 Pousser toutes les branches vers le dépôt à distance (origin)

Une fois que vous avez ajouté votre dépôt distant, vous pouvez pousser vos branches locales vers ce dépôt en utilisant `git push`. Pour pousser toutes les branches vers le dépôt distant, utilisez `git push --all`.

```
git push --all origin
```

Notez que cette commande ne poussera que les branches pour lesquelles un homologue distant existe. Si vous avez des branches locales qui n'ont pas été poussées auparavant, vous devrez les pousser individuellement avec `git push origin branch-name`.