git

script linux

Git est un système de gestion de versions qui permet de suivre les modifications apportées à des fichiers et de collaborer sur des projets informatiques de manière organisée. Il enregistre l'historique de toutes les modifications, ce qui permet de revenir en arrière en cas d'erreur et de travailler en équipe sur un même code source sans conflits majeurs. En résumé, Git est un outil essentiel pour les développeurs et d'autres personnes travaillant sur des projets informatiques pour gérer et suivre les modifications apportées aux fichiers.

1. Commandes Git de base

Lorsque vous travaillez uniquement en local avec Git, vous avez généralement besoin d'un ensemble de commandes Git de base pour gérer vos dépôts locaux. Voici quelques-unes des commandes Git les plus couramment utilisées dans ce contexte :

- **git init** : Cette commande initialise un nouveau dépôt Git local dans le répertoire courant. Vous l'utiliserez une seule fois au début du projet pour créer un nouveau dépôt.
- **git clone** : Si vous avez déjà un dépôt distant et que vous souhaitez créer une copie locale de ce dépôt, vous pouvez utiliser la commande **git clone**.
- **git status** : Cette commande vous permet de vérifier l'état de votre répertoire de travail par rapport au dépôt Git local. Elle vous montre les fichiers modifiés, non suivis, en attente de commit, etc.
- **git add** : Utilisez cette commande pour mettre des fichiers sous suivi Git (*staging area*) en vue de les inclure dans le prochain commit.
- **git commit** : Crée un instantané (commit) des modifications enregistrées dans la *staging area*.
- **git log** : Cette commande affiche l'historique des commits du dépôt local, y compris les messages de commit et les informations sur les auteurs et les dates.
- **git branch** : Permet de lister les branches disponibles dans votre dépôt local, et de voir sur quelle branche vous vous trouvez actuellement.
- **git checkout** : Vous permet de passer d'une branche à une autre.
- **git merge** : Utilisé pour fusionner une branche avec une autre.
- **git diff** : Vous montre les différences entre deux commits, deux branches ou deux fichiers.

2. Commandes Git avec un dépôt distant

Lorsque vous travaillez avec un dépôt distant en plus de votre dépôt local, vous devez utiliser quelques commandes supplémentaires pour synchroniser votre travail avec le dépôt distant. Voici les commandes Git les plus couramment utilisées dans ce contexte :

- **git remote** : Cette commande vous permet de voir la liste des dépôts distants associés à votre dépôt local.
- **git fetch** : Utilisez cette commande pour récupérer les dernières modifications du dépôt distant sans les fusionner dans votre branche actuelle. Elle met à jour vos références locales avec les modifications distantes.
- **git pull** : Cette commande récupère les dernières modifications du dépôt distant et les fusionne automatiquement dans votre branche locale. C'est équivalent à exécuter **git fetch** suivi de **git merge**.
- **git push** : Utilisez cette commande pour pousser vos commits locaux vers le dépôt distant. Vous devez spécifier la branche locale que vous souhaitez pousser et la branche distante vers laquelle vous voulez la pousser.
- **git clone** (déjà mentionné) : Utilisez cette commande pour cloner un dépôt distant et créer une copie locale de celui-ci.
- **git branch** (déjà mentionné) : Vous permet de voir les branches locales et distantes. Utilisez **git branch -r** pour voir les branches distantes.
- **git checkout** (déjà mentionné) : Vous permet de basculer entre les branches locales et de créer de nouvelles branches.
- **git merge** (déjà mentionné) : Utilisé pour fusionner les branches locales. Vous pouvez également utiliser git merge pour fusionner des branches distantes dans votre branche actuelle après avoir récupéré les modifications avec **git fetch**.
- **git remote add** : Si vous souhaitez ajouter un nouveau dépôt distant à votre dépôt local, vous pouvez utiliser cette commande pour l'associer.
- **git remote remove** : Cette commande vous permet de supprimer un dépôt distant associé à votre dépôt local.

3. init

Que vous travailliez en collaboration avec d'autres développeurs ou que vous travailliez uniquement sur votre propre projet, vous effectuerez systématiquement une commande `git init` pour initialiser la construction de votre dépôt **Git**. Pour rappel, **Git** sait aussi bien travailler en collaboration à distance qu'en local, car il permet de gérer efficacement les versions de votre code, de suivre l'historique des modifications et de fusionner les contributions de différents collaborateurs, que ce soit sur un dépôt distant ou sur votre propre machine.

4. gitignore

Le fichier `.gitignore` est utilisé pour spécifier des règles sur les fichiers et les répertoires que Git doit ignorer lorsqu'il suit les modifications. Les règles définies dans ce fichier indiquent à Git de ne pas inclure certains fichiers ou dossiers dans les commits, ce qui peut être utile pour exclure des fichiers de configuration locaux, des fichiers de génération automatique, des fichiers temporaires, etc.

Placer le fichier `.gitignore` à la racine du dépôt est courant car cela permet de spécifier des règles d'ignorance qui s'appliquent à l'ensemble du projet. Cependant, il est également possible d'avoir des fichiers `.gitignore` dans des sous-répertoires si vous avez besoin de règles d'ignorance spécifiques à ces sous-répertoires.

5. clone

Ce terme désigne l'action de copier ou télécharger le contenu d'un dépôt dans un dossier de travail. L'adresse d'un dépot Git peut-être par exemple https://git.abonnel.fr/cedricAbonnel/scripts-bash.

Dans le contexte de la gestion de versions et du contrôle de code source, un dépôt (ou repository en anglais) est un endroit où sont stockées toutes les informations liées à un projet, y compris les fichiers source, l'historique des versions, les branches de développement, etc.

Donc, lorsque vous souhaitez **cloner** ce lien avec la commande `git clone`, vous obtenez une copie locale de ce dépôt sur votre machine, ce qui vous permet de travailler sur le projet et de suivre les modifications localement.

Par exemple :

```
cd ~/projets
git clone https://git.abonnel.fr/cedricAbonnel/scripts-bash
```

Ces commandes créent un dossier appelé “scripts-bash” dans le répertoire “~/projets” et téléchargent les fichiers et dossiers du dépôt, en préservant la structure de l'arborescence telle qu'elle est définie dans le dépôt. Le dossier “~/projets/scripts-bash” devient un dépôt Git local.

6. remote

Pour collaborer avec d'autres sur **Git**, vous devez initialement créer un dépôt sur la plateforme de votre choix, que ce soit **GitHub**, **GitLab**, ou un dépôt géré par un service tel que **Gitea**. Une fois que votre dépôt distant est prêt, vous devez configurer votre dépôt local pour qu'il puisse interagir avec le dépôt distant en utilisant la commande `git remote`. Cette commande permet d'établir le lien entre votre dépôt local et le dépôt distant en spécifiant son emplacement et un nom symbolique pour référencer le dépôt distant.

Voici quelques exemples d'utilisation de la commande `git remote` pour configurer des dépôts distants dans Git :

1. Ajouter un dépôt distant nommé “origin” avec une URL :

```
git remote add origin https://exemple.com/votre-utilisateur/votre-depot.git
```

2. Voir la liste des dépôts distants configurés dans votre dépôt local :

```
git remote -v
```

3. Modifier l'URL d'un dépôt distant existant (par exemple, changer l'URL du dépôt “origin”) :

```
git remote set-url origin https://exemple.git
```

4. Supprimer un dépôt distant spécifique (par exemple, supprimer le dépôt “origin”) :

```
git remote remove origin
```

5. Renommer un dépôt distant (par exemple, renommer “origin” en “new-origin”) :

```
git remote rename origin new-origin
```

6. Voir les informations détaillées sur un dépôt distant spécifique (par exemple, “origin”) :

```
git remote show origin
```

7. origin

Dans **Git**, “origin” est généralement un nom symbolique utilisé pour faire référence par défaut au dépôt distant à partir duquel vous avez cloné votre dépôt local. Il est important de noter que “origin” n'est pas un terme réservé ou prédéfini par **Git**, mais c'est une convention couramment utilisée.

Plus précisément, “origin” est un alias que **Git** utilise pour simplifier les opérations de communication avec un dépôt distant. Lorsque vous clonez un dépôt distant avec la commande `git clone`, **Git** configure automatiquement “origin” pour pointer vers l'URL du dépôt distant que vous avez cloné. Cela vous permet d'accéder facilement au dépôt distant sans avoir à spécifier son URL à chaque fois que vous effectuez des opérations telles que `git fetch` ou `git push`.

Par exemple, après avoir cloné un dépôt depuis **GitHub**, votre dépôt local aura par défaut “origin” configuré pour pointer vers l'URL du dépôt **GitHub**. Vous pouvez ensuite utiliser des commandes comme `git pull origin` pour tirer les mises à jour du dépôt distant ou `git push origin` pour pousser vos modifications vers le dépôt distant, en utilisant simplement l'alias “origin”.

Cependant, vous pouvez également configurer d'autres dépôts distants avec des noms différents si vous travaillez avec plusieurs dépôts distants dans votre projet. “origin” est simplement le nom par défaut pour le dépôt distant d'origine à partir duquel vous avez cloné.

8. add, commit, push

Les commandes **add**, **commit** et **push** sont des commandes essentielles dans Git qui vous permettent de gérer et de versionner vos fichiers et modifications.

git add : Cette commande est utilisée pour mettre des fichiers sous suivi Git (staging area). En d'autres termes, elle permet de préparer les modifications que vous souhaitez inclure dans votre prochain commit. Vous pouvez spécifier les fichiers individuellement ou utiliser des motifs pour ajouter plusieurs fichiers à la fois. Par exemple :

```
git add fichier1.txt
git add dossier/
git add .
```

La première commande ajoute un fichier spécifique, la deuxième ajoute tous les fichiers dans un dossier, et la troisième ajoute tous les fichiers modifiés ou nouveaux dans le répertoire de travail.

git commit : Une fois que vous avez ajouté les fichiers à la staging area avec git add, vous pouvez utiliser git commit pour créer un instantané (commit) des modifications. Chaque commit est accompagné d'un message descriptif qui explique les changements apportés. Par exemple :

```
git commit -m "Ajout de fonctionnalité XYZ"
```

Cette commande crée un commit contenant les fichiers ajoutés à la staging area avec un message qui décrit la modification effectuée.

git push : Cette commande est utilisée pour pousser vos commits vers un dépôt distant, comme celui sur GitHub, GitLab ou un autre serveur Git. Lorsque vous effectuez des commits localement, ils ne sont pas automatiquement disponibles pour d'autres collaborateurs ou pour sauvegarde sur le serveur distant. git push permet de transférer vos commits locaux vers le dépôt distant. Par exemple :

```
git push origin nom_de_la_branche
```

Cette commande envoie les commits de la branche locale vers la branche correspondante sur le dépôt distant.

–

Si vous souhaitez modifier l'un des scripts ou fichier dans le dépôt cloné (par exemple, le “scripts-bash” que vous avez cloné), suivez ces étapes :

1. Naviguez vers le répertoire où vous avez cloné le dépôt. Vous avez mentionné que vous l'avez cloné dans “~/projets/scripts-bash”. Utilisez la commande `cd` pour vous déplacer vers ce répertoire :

```
cd ~/projets/scripts-bash
```

2. Une fois dans le répertoire du projet, vous pouvez éditer le script que vous souhaitez modifier à l'aide de l'éditeur de texte de votre choix. Par exemple, si vous utilisez l'éditeur de texte “nano”, vous

pouvez l'ouvrir en spécifiant le nom du fichier à éditer :

```
nano nom_du_script.sh
```

3. Effectuez les modifications nécessaires dans le script à l'aide de l'éditeur de texte.

4. Enregistrez les modifications et quitter l'éditeur de texte.

5. Une fois les modifications enregistrées, vous pouvez les valider en utilisant **Git**. Voici comment cela peut être fait :

- Si vous avez modifié un fichier existant, utilisez la commande `git add` pour ajouter les modifications au suivi Git :

```
git add nom_du_script.sh
```

- Ensuite, utilisez la commande `git commit` pour enregistrer les modifications avec un message descriptif :

```
git commit -m "Description de la modification"
```

- Enfin, utilisez `git push` pour pousser les modifications vers le dépôt distant (si vous avez les autorisations nécessaires) :

```
git push origin nom_de_la_branche
```

Assurez-vous de remplacer “nom_de_la_branche” par le nom de la branche sur laquelle vous souhaitez pousser les modifications.

–

En résumé, **git add** prépare les modifications, **git commit** crée un instantané des modifications avec un message, et **git push** envoie ces commits vers un dépôt distant. Ensemble, ces commandes permettent de gérer efficacement les versions de votre code source. Elles permettront de modifier un script dans votre dépôt Git local et de mettre à jour le dépôt distant avec vos modifications.

9. Enregistrer ces identifiants durant un laps de temps en mémoire

Comme vous l'aurez remarqué, les identifiants sont systématiquement requis lorsque vous effectuez des modifications sur le serveur distant avec la commande `git push`. Vous avez la possibilité de mémoriser temporairement les identifiants de connexion Git localement en utilisant la commande `git credential`. Voici comment procéder :

1. Ouvrez votre Terminal

2. Exécutez les commandes suivantes pour définir votre nom d'utilisateur et votre mot de passe pour le dépôt distant :

```
git config --global credential.helper 'cache --timeout=3600'
git credential approve <<EOF
protocol=https
host=github.com
username=VOTRE_NOM_D_UTILISATEUR
password=VOTRE_MOT_DE_PASSE
EOF
```

Assurez-vous de remplacer les valeurs de `protocol`, `host`, `username` et `password` par les valeurs appropriées pour votre dépôt distant.

_

Une fois que vous avez exécuté ces commandes, Git stockera vos informations d'identification en mémoire de manière sécurisée localement pendant un certain temps (dans cet exemple, pendant 3600 secondes, soit 1 heure). Vous n'aurez pas besoin de saisir vos informations d'identification à chaque opération Git pendant cette période.

Source : https://git-scm.com/docs/git-credential-cache/fr

10. Enregistrer ces identifiants dans les gestionnaires de secrets

GNOME Keyring

Source : https://pkgs.org/search/?q=git-credential-libsecret

11. Enregistrer ces identifiants dans keepassXC

Source : https://github.com/Frederick888/git-credential-keepassxc