L’outil de versionning Git

***

PLAN DU COURS

- Introduction
- Installation et configuration
- Premier commit
- Création manipulation des branches
- Inspecter l’état du dépôt
- Le remisage
- Naviguer entre les versions
- Le Remote
- Github: Fork & Pull Request

Introduction

Git est un système de contrôle des versions. Son objectif est de garder une trace des différentes évolutions sur un projet et de permettre de faire des retours vers des états antérieurs.

Il utilise une architecture DVCS(Distributed Version Control System). Cela veut dire que, contrairement au système décentralisé, avec Git chaque copie de travail du code du développeur est également un référentiel.

Il est de loin, le système de versionning le plus utilisé.

Installation

Pour installer Git sur notre system linux, nous allons utiliser les commandes suivantes,

```
sudo apt update

sudo apt install git
```

Une fois l'installation terminée, nous allons nous placer sur le dossier de notre projet pour configurer notre profil Git.

```
cd ~/www/labo/dossier_projet
```

Il faut d'abord initialiser Git.

```
git init
```

Nous pouvons maintenant ajouter un nom et une adresse email.

```
git config --global user.name mon_nom

git config --global user.email mon_email@email.com
```

Premier Commit

Un commit consiste à enregistrer les dernières modifications en les liant à un profil et un message qui va expliquer quelles sont les modifications qui ont été apportées.

Mais avant de faire un commit nous devons d'abord dire à Git quels sont les fichiers que nous voulons inclure dans le commit. Cette action s'appelle "faire un stage" ou "stagé". Elle se fait avec la commande suivante:

`git add nomfichier.extension` Nous pouvons stager tous les fichiers d'un coup avec la commande

`git add --all` ou `git add .`

Nous pouvons maintenant faire un commit

`git commit -m "le message du commit qui explique ce qu'on a fait comme modifications"`

Les branches

La commande suivante va afficher l'état de notre dépôt local:

```
git status.
```

le résultat de cette commande nous précise que nous sommes sur la branche ‘master’.

Les branches sont des espaces virtuelles ou le développeur peut faire ses modifications sans affecter la branche principale. Elles permettent une meilleure organisation entre différents collaborateurs.

Pour créer une branche nous utilisons la commande suivante:
`git branch nom_de_la_branche`
nous pouvons maintenant nous positionner sur la branche:
`git checkout nom_de_la_branche`
Ces deux commandes peuvent être unies en une seule:
`git checkout -B nom_de_la_branch`

La commande `git branch` va afficher une liste des branches de notre dépôt.

Les branches: le merge

Une bonne utilisation des branches consiste à définir toutes les fonctionnalités d'un projet et de créer une nouvelle branche pour chaque fonctionnalité.

La branche 'master' ne doit jamais être utilisé pour faire des modifications.

Une fois que la fonctionnalité est conçue, on pourra ensuite utiliser le concept de merge pour "déverser" tout le travail fait sur notre branche dans la branche master.

Le concept de 'merge' consiste à copier toutes les modifications sur une branche vers la branche principale (très souvent la branche master).

Il se fait avec la commande:

```
git merge nom_de_branche
```

Il faudra penser à se positionner sur la branche principale avant d'utiliser cette commande.

Inspecter l'état du dépôt

Nous allons maintenant nous familiariser avec quelques commandes qui vont nous permettre de connaître l'état de notre dépôt local.

- git status cette commande va afficher les fichiers à stager ou les fichiers stagés en attente d'un commit. Elle précise aussi la branche sur laquelle nous travaillons
- git diff cette commande est utilisé pour voir les modifications apporte sur un fichier. Elle affiche l'ancienne version du fichier et les modifications apportées.
- git log cette commande va afficher les détails sur les commits. elle va lister tous les commits que nous avons fait depuis l'initialisation du projet.

  Elle va afficher le nom du profil et la date de création du commit et aussi son identifiant.

  Elle prend un attribut --oneline qui sera très utiles pour naviguer entre les commits.

Le remisage

Le remisage consiste à enregistrer l'état actuel de notre projet en mémoire pour pouvoir revenir sur l’état qui précède nos dernières modifications.

Il permet d'interrompre une tâche, pour travailler sur une autre sans perdre les dernières modifications.

La commande pour faire un remisage est:

```
git stash
```

La commande `git stash list` permet de voir la liste des stash enregistrés en mémoire.

Pour récupérer les modifications d’un stash, nous utilisons la commande

`git stash pop` ou `git stash apply`.

Manipulation de l'historique: amend, checkout & revert

L'historique sur git est un registre qui contient les différents commits faits sur le projet, de manière chronologique. Il garde une trace des différentes actions avec le nom de leur auteur et la date.

Manipuler cet historique consiste donc à éditer ce registre de différentes manières.

- `git commit --amend` cette commande va permettre de modifier le message du dernier commit.

- `git checkout id_comit`: cette commande permet de revenir en arrière vers un commit. Il permet un retour vers le passé sans la possibilité de faire des modifications.

  Les éditions réalisées alors qu'on est dans cet état ne seront pas enregistrés par git.

- `git checkout commit_id nom_fichier`: cette commande va mettre le fichier spécifié à l'état où il était lors du commit. Elle ne va cependant pas nous positionner vers ce commit.
- `git revert` va supprimer les modifications faites sur le commit choisi. Elle ne supprime pas toutes les modifications depuis le commit, mais va plutôt défaire que les actions sur ce commit.

Manipulation de l'historique: reset

La commande git reset permet de défaire l'action de la commande git add. C'est-à-dire de supprimer du stage tous les fichiers en attente d'un commit.

On peut préciser le nom d'un fichier si c'est juste un fichier que l'on souhaite supprimer des stages; git reset nom_fichier.

une autre manière d'utiliser la commande reset est en le combinant avec l'identifiant d'un commit.

git reset id_commit va remettre notre projet à l'état du commits avec l'identifiant spécifié. Il va aussi supprimer tous les commits faits depuis ce commit. Et en fonction du mode, va supprimer ou pas nos dernières modifications.

Pour préciser les modes nous utilison trois options: --soft, --mixed, --hard.

- git reset id_commit --soft: soft est le mode par défaut. Cette commande va donc nous ramener en arrière vers le commit et va aussi stager les dernières modifications.

- git reset id_commit --mixed: cette commande va faire comme la précédente à l'exception que, avec le mode --mixed, elle ne va pas mettre les dernières modifications en stage.

- git reset id_commit --hard: cette commande est à éviter. Son action est irréversible. Elle va ramener le projet vers le commit précisé, supprimer tous les commits qui ont suivi le commit précisé et supprimer toutes les dernières modifications.

Remote

Pour compléter notre utilisation de git, nous devons lier notre dépôt local à un dépôt distant.

Ces dépôts distants sont le plus souvent créés dans des applications tierses tels que GitHub, BitBucket ou GitLab. Ici nous utiliserons GitHub qui est le plus populaire.

D'abord il faut créer un compte sur GitHub; à l'adresse http://github.com. Et ensuite, créer un repo pour le projet.

En local nous devons créer un 'remote' qui va lier notre projet au repo sur GitHub avec la commande suivante:

```
git remote add nom_remote url_repo_github
```

Une convention veut que l'on nomme le remote 'origin'. La commande sera alors

```
git remote add origin url_repo_github
```

Remote: clone, pull & push

Nous allons voir maintenant trois commandes qui vont nous permettre d'interagir avec un dépôt distant.

`git pull origin master`: cette commande va mettre à jour notre dépôt local. Son action consiste à modifier notre projet en local pour qu'il soit identique à la branche master sur le dépôt github.

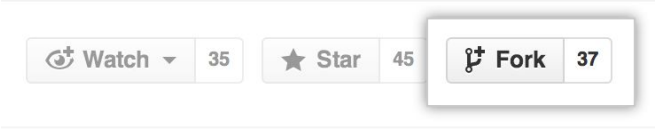
`git push origin master`: cette commande va envoyer l'état de la branche master au dépôt distant. Elle est utilisée pour envoyer les dernières modifications vers le remote.

`git clone url_depot_github`: cette commande va permettre de copier un projet dans notre environnement local. Elle est souvent utilisée quand un développeur rejoint le groupe. Ou quand on souhaite récupérer un dépôt public.

GitHub: fork & pull request

Les dépôts publics sont des espaces sur lesquels tout le monde peut apporter sa contribution. Ils sont open source. Cependant, pour gérer l'ordre et la sécurité dans ces dépôts, GitHub dispose de deux concepts fork et pull request, qui sont essentiels pour ces dépôts.

Un fork consiste à copier le répertoire d'un autre profil dans son propre profil pour y apporter des contributions. Ce concept permet de faire toutes les modifications souhaitées sans affecter le dépôt original. Pour faire un fork, il faut se rendre sur le dépôt original et cliquer sur le bouton de fork.



Cette action va copier le repo sur notre profil. Nous pouvons ensuite cloner notre version du projet sur notre environnement local pour faire nos modifications.

Le pull request est la suite logique d'un fork. Il consiste à proposer nos modifications vers le dépôt original pour une intégration. Une fois les modifications enregistrées en local, il faut faire un push vers notre dépôt. Cela affichera alors un bouton pour faire un pull request. Ce bouton va envoyer notre branche vers le dépôt original. Le propriétaire du dépôt pourra alors tester nos modifications et décider s'il est pertinent de les intégrer.

Une précaution importante à prendre avec un fork est de ne jamais faire les modifications sur la branche. Et aussi de ne jamais pusher un master pour un pull request.

Conclusion

Nous avons fait un bref tour des possibilités qu'offre Git. C'est un outil très puissant qui, de nos jours, est utilisé dans presque tous les grands projets de programmation.

La plupart des commandes que nous avons vues ont des options qui offrent des manipulations avancées. Je vous invite donc à faire des recherches pour mieux se familiariser avec Git.

LIENS UTILS

https://git-scm.com/docs/git-branch

https://git-scm.com/docs/git-config

https://git-scm.com/docs/git-log

https://git-scm.com/docs/git-status/fr

https://git-scm.com/docs/git-diff/fr

https://git-scm.com/docs/git-stash

https://git-scm.com/docs/git-revert

https://git-scm.com/book/fr/v2/Les-bases-de-Git-Annuler-des-actions

http://www.responsive-mind.fr/git-15-points/

https://buzut.net/git-bien-nommer-ses-commits/