Introduction au gestionnaire de version Git

Dans un système de fichiers, les fichiers vont évoluer : on veut un environnement qui va gérer cette évolution : aller en arrière, contrôler les versions des données manipulées

→ git

Git fonctionne sur un principe bien adapté aux fichiers textes. Pour les binaires, il faut utiliser `git-lfs`. Un git ne doit contenir que du texte !

Définitions

- Une version est le contenu d'un projet à une étape de sa vie
- Un dépôt est l'ensemble des versions
- Une branche est la variation d'un projet

On va utiliser `git` en ligne de commandes. Il suit la syntaxe suivante : `git <commande>`

Par exemple :

- `git help`
- `git help <subcommand>`

Il faut utiliser `git config` pour dire à Git qui sommes nous

Dépôt local

Pour créer un dépôt local, il faut un répertoire (par exemple `~/projet`) où il faut faire la commande `git init`. Cela fait, un répertoire `.git` sera créé dans `~/projet/.git` Le répertoire `.git` contient des méta-données.

Organisation des versions

Ces versions sont organisées en graphe acyclique orienté direct. Ce graphe représente l'ensemble des versions Ces versions sont nommées par un numéro hexadécimal (hashcode)

On peut mettre des étiquettes (tag) sur les versions Il existe des tags dynamiques importants : le bout des branches. Toutes les branches ont des étiquettes

(main, fix, …) Etiquette spéciale : `HEAD` ⇒ C'est la position courante dans le dépôt

Quelques commandes

Comment voir l'état de la branche

`git log` : Montre l'historique de la branche dans laquelle on se trouve.

Pour voir l'historique d'une autre branche : `git log <nom-de-la-branche>`

Comment créer et gérer une version ?

Quand on créé un fichier `~/projet/a.txt`, et qu'on fait `git log`. On ne voit rien Il faut indiquer les fichiers que nous voulons suivre

Pour voir l'état de la version courante, on peut faire `git status`. l'option `-uno` ne montre que les fichiers qui sont suivis

Pour suivre un fichier, on fait `git add <nom-du-fichier>`. On peut désormais le voir avec `git status` Pour le retirer, on fait `git reset <nom-du-fichier>`

Pour voir la différence entre la version et les nouvelles modifications, on fait `git diff`

Pour valider cette modification, on créé une version avec `git commit`. Si on l'écrit comme ça, on aura un éditeur de texte, qui est par défaut `vi` (un peu brutal) L'option `-m "<texte>"` ajoute `<texte>` en commentaire

A chaque modification d'un fichier, on doit le passer dans `git add`

Dans le fichier `.gitignore`, on met tous les fichiers qui seont automatiquement ignorés par Git

Une version ne peut pas être modifiée, sauf si elle n'a pas de fils. Dans ce cas, on écrit `git commit --amend`

Pour changer de version, on fait `git checkout <nom-du-commit>`. On peut aussi faire `git checkout HEAD~n` pour reculer de `n` versions par rapport à `HEAD`

Pour créer une nouvelle branche, on fait `git checkout -b <nom-de-la-branche>`

Pour voir les branches, on fait `git branch`

Pour changer de branche, on fait `git checkout <nom-de-la-branche>`

Pour fusionner deux branches, on se place sur la branche destination, sur laquelle je veux me retrouver, et je fais `git merge <nom-de-branche>`

> Attention cependant aux conflits : Si un même fichier a deux contenus différents, il va y avoir un soucis. Il faut le résoudre

Dépôts distants >:)

A l'université, on utilise GitLab, qui permet de construire des dépôts sur des serveurs

Pour cloner ce dépôt distant sur la machine en local, on fait `git clone <url>` Pour voir l'URL du serveur distant, on fait `git remote -v`

Pour relier un dépôt local à un dépôt distant, on fait `git remote add <nom> <url>` Avec `git pull`, on télécharge tous les fichiers du dépôt distant sur le dépôt local. Avec `git push`, on envoie tous les fichiers du dépôt local sur le dépôt distant

---

Résumé des commandes :

- `git status`
- `git log`
- `git help`
- `git config`
- `git add`
- `git reset`
- `git diff`
- `git commit`
- `git checkout`
- `git branch`
- `git merge`
- `git remote`
- `git clone`
- `git pull`