Table of Contents

Git

Mise à niveau

Introduction

Définition

Git est un logiciel de gestion de **version décentralisé** créé par Linus Torvald

- **version**

> L'objectif est de gérer l'évolution du contenu sur une arborescence de fichier. C'est le même fonctionnement qu'une base de données.

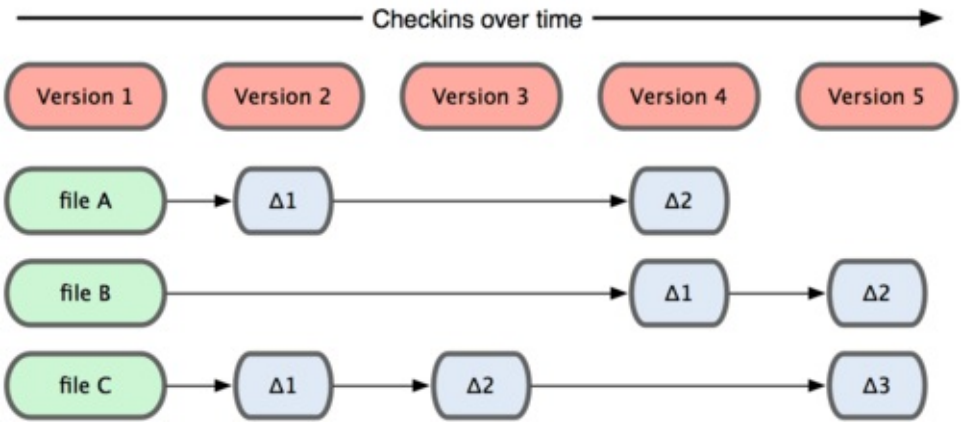
- **décentralisé**

> Ne repose pas sur un serveur centralisé. Cela permet de pouvoir partager du code à n'importe quel moment. Pas besoin d'authentification ou de compte. Chacun des utilisateurs a son propre historique.

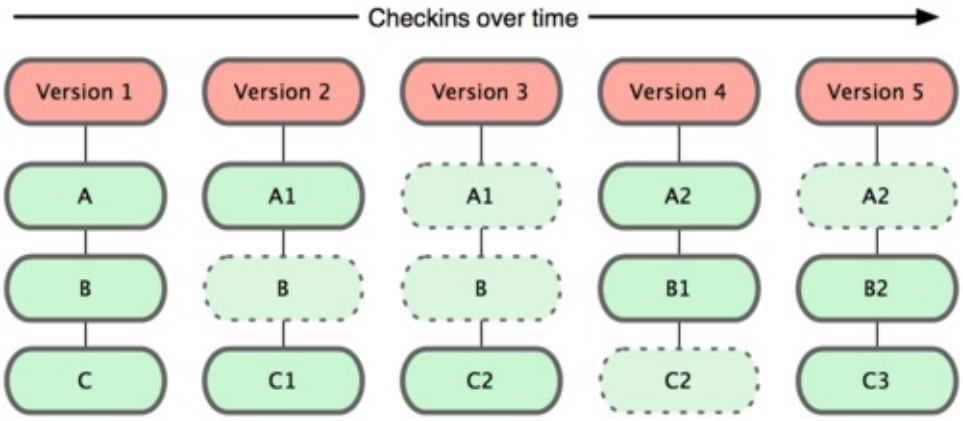
Git gère des **snapshots** (instantanés) en **local**

- **snapshots**

> La plus part des gestionnaires de versions tels que SVN enregistre un fichier dans l'état initiale puis ne sauvegarde que les modifications effectuées



> A l'inverse, Git stocke la dernière version indexée. Le terme snapshots est utilisé pour dire que l'on sauvegarde une image de notre code à un instant T (commit).A l'inverse des autres, Git enregistre l'intégralité de nos fichiers. Lorsque nous créons un nouveau commit et si le fichier n'a pas été modifié, Git n'enregistre pas à nouveau le fichier mais stocke un pointeur vers la dernière version du fichier.



- **local**

> L'utilisation de Git peut être réaliser de façon intégrale sur votre poste. Aucune connexion internet n'est nécessaire pour utiliser Git.
> **Attention** : Il ne faut surtout pas confondre Git, Github, Gitlab, SVN

Petit rappel sur les différents termes de votre quotidien

Objet	Définition
Git	Logiciel de gestion de version local
Github	Site internet communautaire permettant de publier son repository
Gitlab	Application web permettant de partager son repository
SVN	Logiciel de gestion de version par différence centralisé

Et le travail collaboratif dans tout ça ?!

Comme vous avez pu le constater, pour l'instant nous n'avons jamais travailler en dehors notre propre machine. Je ne me suis jamais connecté à un serveur distant ni demandé du code à une autre personne. Il est très important de comprendre que la gestion de version à travers Git se fait en local avant tout.

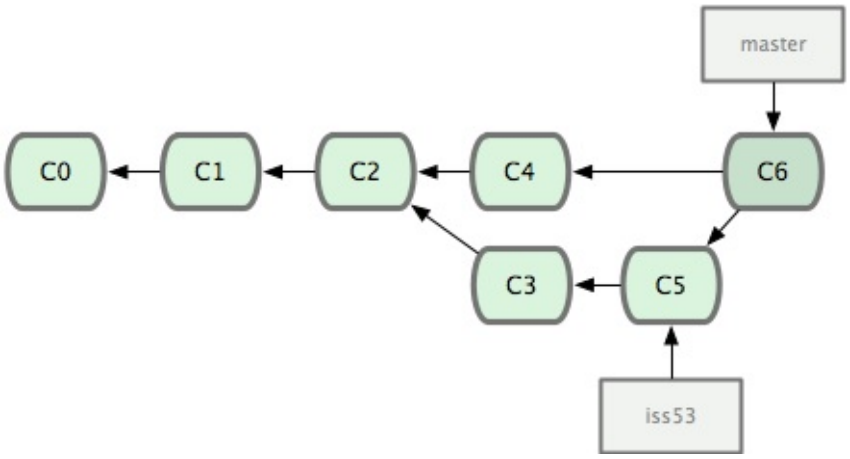
Mais il est vrai que dans notre métier nous sommes rarement amené à travailler tout seul dans notre coin sur notre application. De plus, les informaticiens sont réputés pour leur générosité et adorent partager leurs codes avec les autre :)

Il existe donc des sites sur internet qui permettent de diffuser/partager nos repository (dépôts). Le plus connu est sans aucun doute Github.

L'idée est de diffuser l'ensemble des snapshots afin que n'importe qui puisse reprendre le projet pour contribuer à l'amélioration du projet en cours (clone).

Les différents objets & états sur Git

Objet	Définition
Blob	Contenu d'un fichier
Tree	Liste de blobs
Commit	Message de log + tree + lien Commit Parent
Tag	Commit + Numéro de version
Branche	Pointeur sur un commit. Ceci nous permets de récupérer un ensemble de commit. Le but des branches est de vous permettre de travailler sur des modifications sans pour autant impacter le travail des autres. Cela permet notamment de cloisonner les développements, travailler sur des correctifs tout en maitrisant limitant les impacts.
Master	Lorsque l'on créé un nouveau repository, deux branches par défaut sont créés. L'une d'entre elle est la branche master. Etant donné que cette branche existe dans tous les repository, par convention, c'est celle qui doit contenir le code le plus propre possible
HEAD	A l'image de master, HEAD est lui aussi un pointeur créé lors de la création de notre repository. C'est l'endroit sur lequelle je me trouve actuellement. Toutes les modifications effectuées seront donc faite à partir de ce point.



Etat	Définition
Non indexé	Nouveau fichier dans le répertoire
Validé	Les données ont été sauvegardées dans notre base de données locale (git add)
Modifié	Un fichier passe au statut modifié lorsqu'il est connu de notre base de données locale mais il est différent de celui présent dans l'historique
Indexé	Le fait d'indexer les modifications permet de figer notre/nos fichiers dans une état donné (git commit)

J'aime la ligne de commande

Pré-requis : Installation de Git sur votre environnement.

Initialisation d'un repository

```
cd /my/personal/directory
git init
```

Clone d'un repository distant

```
cd /my/personal/directory
git clone https://github.com/olivernight/MAN_Git.git
```

Indexation d'un nouveau fichier

```
# Pour un fichier
touch mytralala.txt
git add mytralala.txt

# Pour tous les fichiers du répertoire courant
touch mytralala2.txt
touch mytralala3.txt
git add .
```

Création d'un commit

```
# Création d'un commit. L'option -m permet de saisir un commentaire.
git commit -m "Création de mon premier commit"
```

Création d'une nouvelle branche

```
# Création de maNouvelleBranche puis replacement de HEAD
git branch maNouvelleBranche
git checkout maNouvelleBranche

# All in one
git checkout -b branch1

# Création d'une nouvelle branche mBranche à partir du commit C2
# et positionne HEAD dessus.
git checkout -b maBranche C2
```

Tag

```
# Création un tag T1 pour pointer sur le commit C2
git tag T1 C1

# Il est possible d'utiliser le tag plutôt qu'un hash de commit
git checkout -b maBranche T1
```

Merge

```
# Création d'un nouveau commit C de fusion de la branche maNouvelleBranche
# vers la branche master.
# Le dernier commit de la branche ne changera pas.
git merge maNouvelleBranche

# Création d'un nouveau commit C de fusion de la branche maNouvelleBranche vers
# la branche master.
# Le dernier commit de la branche sera C.
git checkout maNouvelleBranche
git merge master
```

Parcours des commits parents

```
# Il est possible de remonter la chaine de commit sans pour autant saisir un numéro
# de hash.
# L'idée est de dire par exemple : "Je veux récupérer le 3e commit parent"
# Pour ce faire, il existe deux fonctions : ~ et ^
# ~ permet de remonter sur le commit parent. Il peut être coupler avec
# un nombre n pour remonter de n noeuds.
git branch test HEAD~3

# Lorsque nous avons plusieurs parents, il existe l'oéprateur ^ qui permet
# de choisir le chemin que l'on souhaite prendre. ^ peut être coupler avec un
# entier n pour choisir le nieme plus vieux parent.
# Les opérateurs peuvent être couplés entre eux
git branch test HEAD~3^2~
```

Rebase

```
# Déplacement de la branche
git rebase maNouvelleBranche

# Rebase interractif
git rebase -i HEAD~4
```

Cherry-pick

```
# Création de nouveaux commits à partir d'une liste de commits existant
# passée en paramètre
git cherry-pick C5 C2 C3
```

Remote / origin

Lorsque l'on travaille avec un repo distant (en utilisant la commande clone ou remote) Git donne un nom par défaur au serveur le nom origin

```
# Récupération du contenu d'une branche distante en local sans pour autant modifier
# mon répertoire courant
git fetch brancheDistante

# Récupération du contenu d'une branche distante en local et tente de
# fusionner avec la branche locale
git pull brancheDistante
```