

Documenter les TP, screen de la command et screen de l'output (si existante)

Toujours avoir un .md dans projet

MD accepte les balises HTML (interchangeable)

SCM Source Control Management (logiciel ressemblant à git)

Snapshot: État du code à l'instantané

Logiciel distribué : Peer 2 Peer (blockchain)

En torrent on télécharge depuis des sources qui contiennent la copie exacte du logiciel, plus il y a de sources plus cela va vite, on appelle ça des noeuds

Git est un contrôle de version distribué, décentralisé

La faiblesse d'un serveur centralisé est le nombre élevé de participants

La force d'un système décentralisé est le nombre élevé de participants

Revenir à des versions antérieurs ou à une version actuelle en évitant les conflits

Travailler sur le même logiciel sur des fonctionnalités différentes en même temps

Traçabilité: Chaque changement est enregistré avec un id souvent accompagné d'un message descriptif.

Collaboration: Les devs peuvent travailler sur le même code source

Nomenclatures de versionning :

Majeure: Indique une version qui fait des changements incompatibles avec les versions antérieures

Mineures: Indique l'ajout de nouvelles fonctionnalités de manière rétrocompatible

Correctif: Indique des corrections de bugs rétrocompatible

Checkout/Update/Commit: Opérations de base de git

Branching et Merging: Opérations de branchement et de fusion

Checkout: Clone le dépôt distant en local

Pourquoi Git et non pas d'autres SCM ?

Les autres SCM gèrent le projet comme une suite de modifications, ils ne vont sauvegarder uniquement les modif d'un point A à B.

Git fonctionne avec des snapshots et sauvegarde l'ensemble du dépôt

Conflit: Git ne sais pas choisir entre 2 modifs, nous choisissons la partie que l'on souhaite et le conflit sera résolu au prochain commit

Remote : le repo avec lequel on synchronise le travail

Origin: nom donné à remote en local (alias du repo)

Head: l'endroit où l'on se trouve (souvent dernier commit sur la branche) pointeur

Git Config

Les paramètres de Git peuvent être stockés dans 3 endroits différents :

- [chemin]/etc/gitconfig : spécifique à l'utilisateur. On peut forcer Git à lire et écrire dans ce fichier en passant l'option `--global`
- Fichier config dans le répertoire Git d'un dépôt en cours d'utilisation (`.git/config`) : spécifique au seul dépôt. Chaque niveau surcharge les valeurs du niveau précédent donc les valeurs dans `.git/config` écrasent celles de `etc/config`.

Ajouter éditeur préféré à git

`git config --global core.editor phpstorm`

Définir nom branche par défaut

`git config --global init.defaultBranch main`

Afficher les paramètres :

`git config --list`

Comment les autres SCM fonctionnent :

Git en details

La principale différence entre Git et d'autres SCMs, réside dans la façon dont Git considère les données.

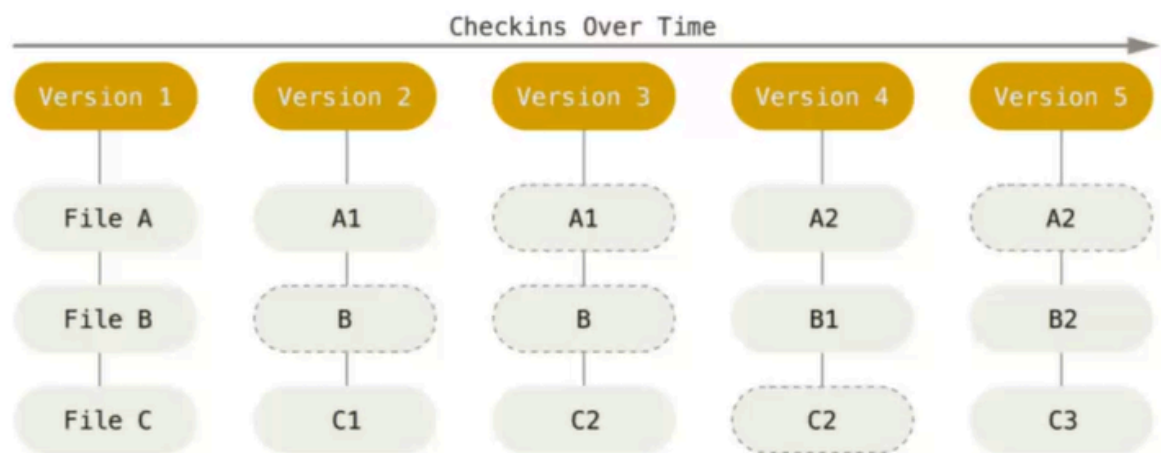
Conceptuellement, la plupart des autres SCMs stockent les informations sous la forme d'une liste de modifications apportées à chaque fichier au fil du temps.



VIM et éditeur Git :

- I pour mode insertion
- Esc pour quitter l'édition
- Quitter sans sauvegarder `Q!` / `:qa!`
- Sauvegarder et quitter `:WQ` `wwwwwwwwww`

Comment Git fonctionne :



Git peut :

- La quasi totalité des opérations sont locales
- Se charge de gérer l'intégrité des données

LA checksum est le résultat d'une fonction, opération, qui prend en argument la totalité des fichiers P2P, si l'on obtient pas le même résultat, nous n'avons pas l'intégrité des données.

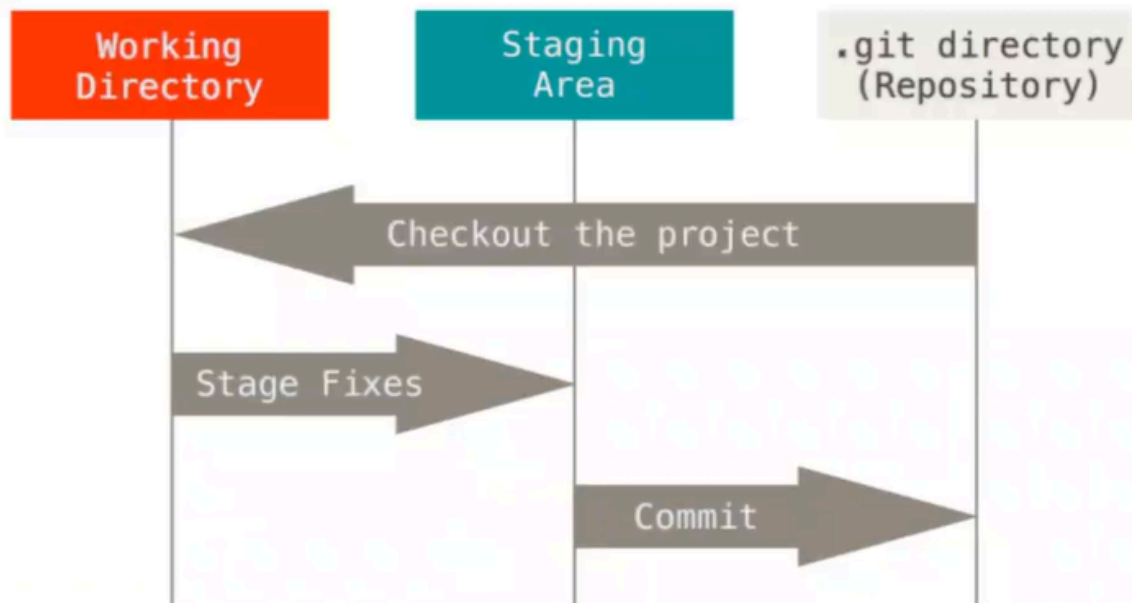
Le mécanisme utilisé par Git est appelé une empreinte SHA-1, une string composée de 40 caractères hexadécimaux.

Git se charge de gérer l'intégrité des données

Avant la plupart des opérations effectuées avec Git, il effectue une "somme de contrôle" et obtient une signature unique qui sert de référence. On peut ainsi vérifier l'intégrité des données. Il est donc impossible de modifier le contenu d'un fichier sans que Git ne le sache.

États :

Git a trois états principaux dans lesquels peuvent se trouver vos fichiers :



Quand on git add, les fichiers ajoutés se retrouvent dans la zone staging area

- Modifié (un nouveau fichier ou fichier non suivi est dans le même état qu'un fichier modifié)
- Indexé
- Validé (ce qui va être intégré au dépôt, ajouté lors du commit)

Working directory (projet où est situé .git)

Les fichiers ajoutés avec git add se retrouvent dans staging area

Commandes :

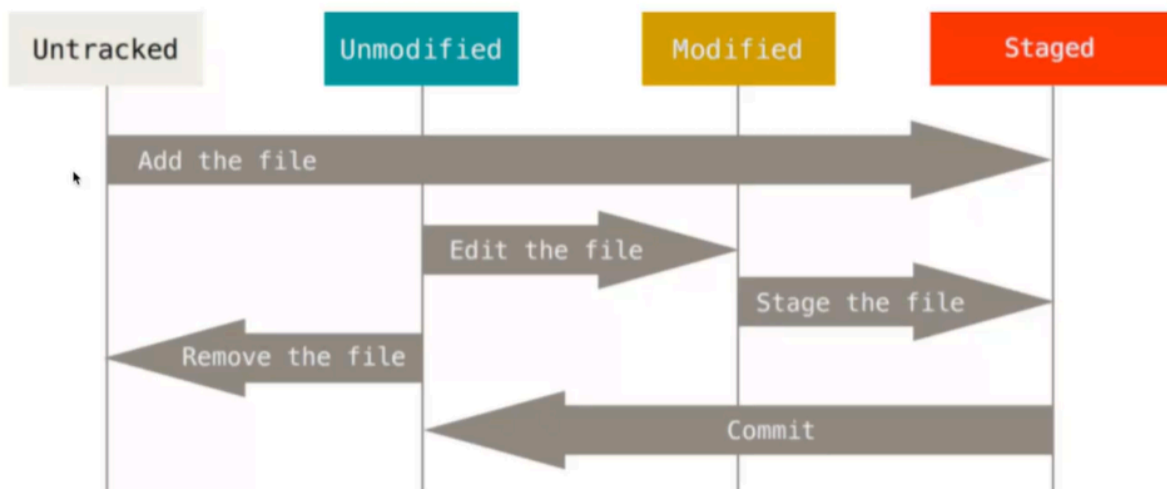
- Le tilde ~ représente la racine du répertoire utilisateur
- git init pour initialiser un dépôt Git
- git status pour obtenir l'état du dépôt
- le . signifie que la commande part du bas et ne remonte pas, de façon récursive, à partir de là où je suis vers la fin, comme gitignore
- push -u origin main informer git que la branche locale main peut être synchronisée avec branche main distante, si elle n'existe pas on crée main sur origin
- git diff présente les modifications apportées aux fichiers, git diff --staged permet d'afficher les mods aux fichiers indexés
- git log pour visualiser l'historique des commits
- git remote show origin ?
- commit --amend annule les modifications

- `git reset HEAD <fichier>`, permet de modifier le dernier commit
- création de tags : `git tag -a v1.0 -m "Version 1.0"`
- `git tags` liste les tags
- `git fetch` met à jour une branche depuis un remote (mettre à jour suivi à distance par rapport à **origin**)

Git est un logiciel qui versionne

Github et Gitlab sont des applications web dont l'interface met à disposition des services autour de Git

De manière générale on peut résumer le cycle de vie des fichiers dans Git comme suit :



Quand on git add, le fichier va dans staged, ensuite dans unmodified puis modified !!!!!!! rien compris

Tips :

Par défaut Git ignore les dossiers vides

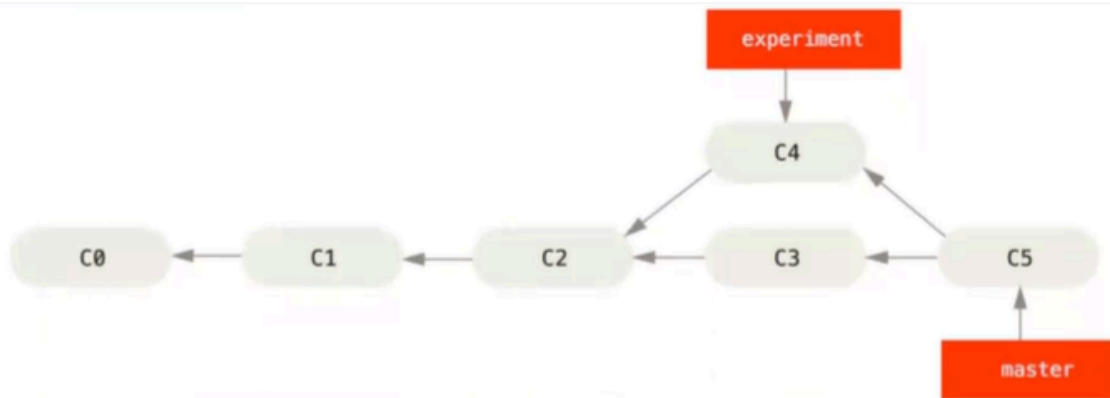
Quand on crée une branche sur un repo qu'on clone, on a directement les droits pour push, pull ...

Git exécute automatiquement des commandes pour tracker la branche ce qui nous évite de le faire.

Rebase / Merge :

Il y a 2 manières d'intégrer les modifications d'une branche à l'autre

- la fusion (merge)
- le rebasage (rebase) linéarise l'historique des commits en les coupants, et les collants au bout



Avec le rebase on aurait entré les commandes suivantes :

```
git checkout experiment
git rebase master
```

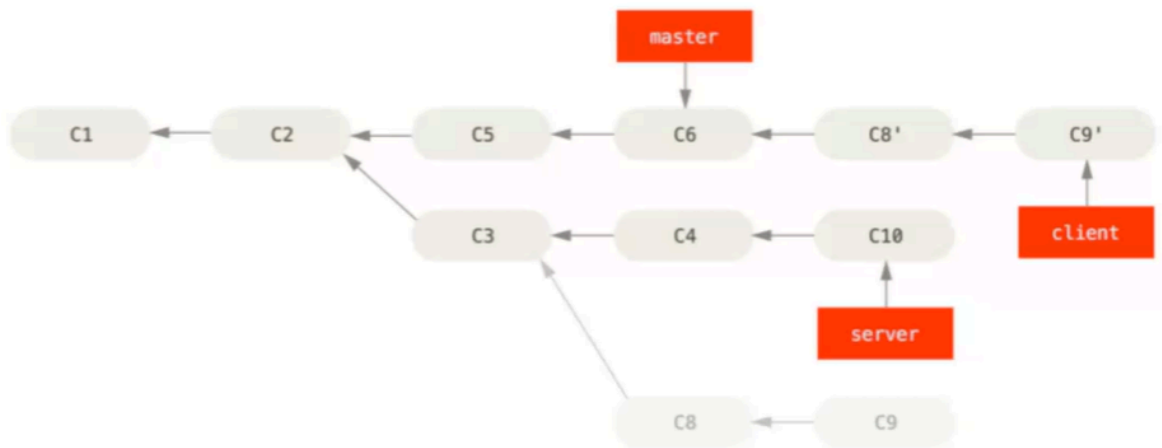
Avec rebase C4 a disparu, c'est la branche experiment qui a été rebase sur master, comme un merge sauf que tout l'historique depuis la divergence a disparu, tous les commits entre sont perdus

Rebase supprime une partie de l'historique git

La commande qui correspond au rebase cité en cours :

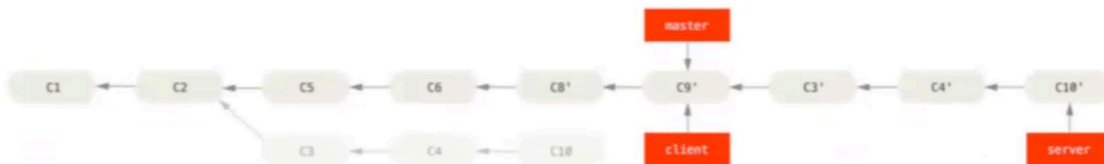
```
git rebase --onto master server client
```

Essaye d'extraire la branche client de la branche server et de la rebaser sur la branche master.



On peut aussi rebaser la branche server sur la branche master :

```
git rebase master server
```



Vous pouvez ensuite merge server dans master:

```
git checkout master
git merge server
```



Ne jamais rebase des modifications qui ont été publiées sur un serveur distant (push)

expliquer ce qui se passe quand on rebase ce qu'on a déjà push et qui sert de bases à nos collègues qui ont commencé à travailler dessus

l'historique des commits ne doit jamais différer des collègues et du repo

rebase ré écrit l'historique