https://jj-vcs.github.io/jj/latest/

Jujutsu (jj)

Successeur de Git
ou nouvel allié ?

Présenté par David Darras & Samuel Lepin
VTSI 2025

GRENOBLE INP UGA

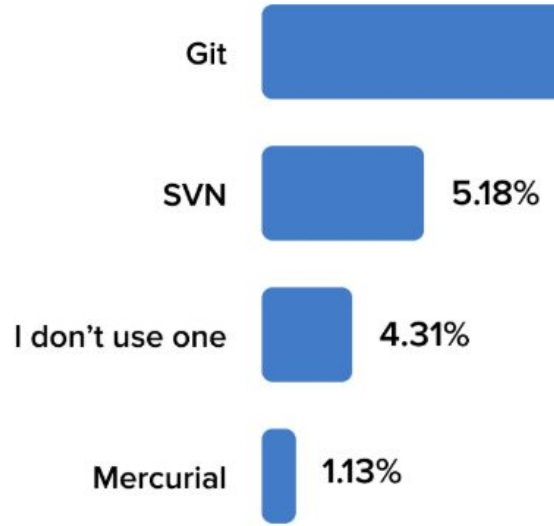
POLYTECH GRENOBLE

Au sommaire

1. Introduction aux VCS (*Version Control System*)
2. Présentation générale de jj
3. Principales différences entre Git et jj
4. Démonstration : apprentissage de l'utilisation de jj en solo et en équipe
5. Conclusion

Popularité des systèmes de contrôle de version en 2023

https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/, *January 9, 2023*

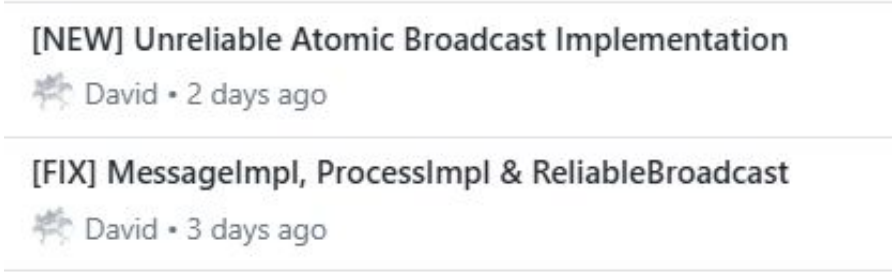


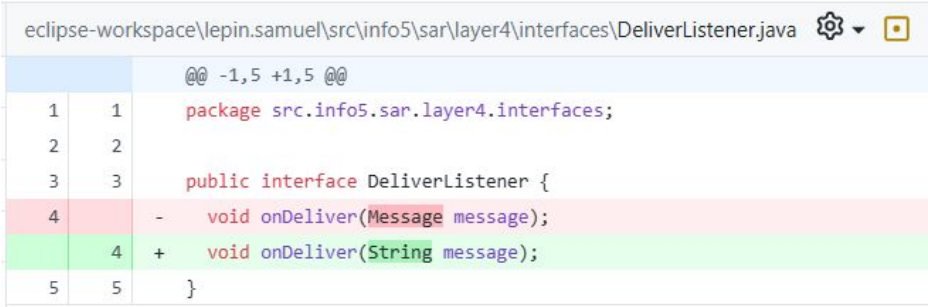
GRENOBLE INP UGA

POLYTECH GRENOBLE

Utilité d'un système de gestion de versions

- **Faciliter le travail en équipe**
- **Revenir en arrière** en cas d'erreur
- **Utiliser des branches** pour créer des features sans toucher au code stable
- **Documenter** les changements apportés au code via les commits
- **Suivre** l'historique des modifications (qui a modifié quoi et quand ?)





Un aperçu rapide de jj

- Nouveau **VCS** (Version Control System)
- **Compatible avec Git**
- **Créateur :** Martin von Zweigbergk (ingénieur chez Google)
- **Première version :** 2020
- **Langage** : Rust (performance + sécurité mémoire)
- **Licence :** Apache 2.0 (open source)

GRENOBLE INP UGA

POLYTECH GRENOBLE

Philosophie et Conception de jj

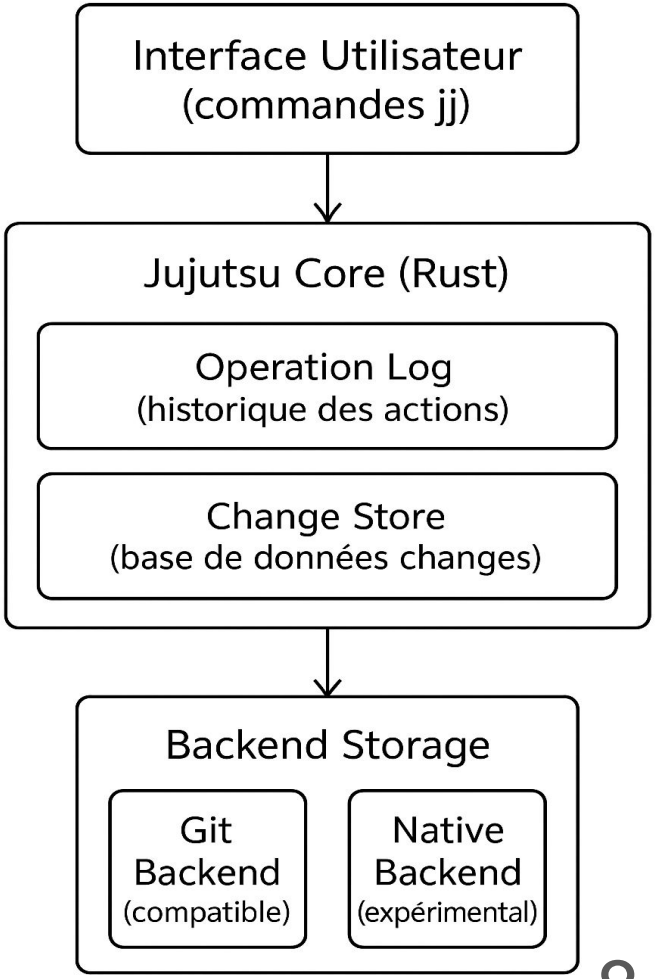
- **Sécurité d'abord** : Toutes les opérations sont réversibles
- **Simplicité** : Moins de concepts à apprendre
- **Expérimentation** : Encourager l'essai-erreur sans risque
- **Pragmatisme** : Compatible avec Git pour adoption graduelle

Objectif principaux de jj

- **Éliminer la Staging Area**
  - Tracking automatique des modifications, plus besoin de "git add"
- **Undo Universel**
  - Annuler n'importe quelle opération facilement avec "jj op undo"
- **Historique Mutable**
  - Modifier les commits passés sans rebase interactif complexe
- **Conflits Commitables**
  - Traiter les conflits comme des objets de première classe
- **Working Copy = Commit**
  - Le répertoire de travail EST un commit, pas une zone à part
- **Compatibilité Git**
  - Coexister avec Git, utiliser les remotes existants

Architecture de Jujutsu

- **Backend Git :** Stockage compatible, interopérabilité totale
- **Change Store :** Gestion des changes avec IDs stables
- **Operation Log :** Journal de toutes les modifications



Git commit **VS** jj change

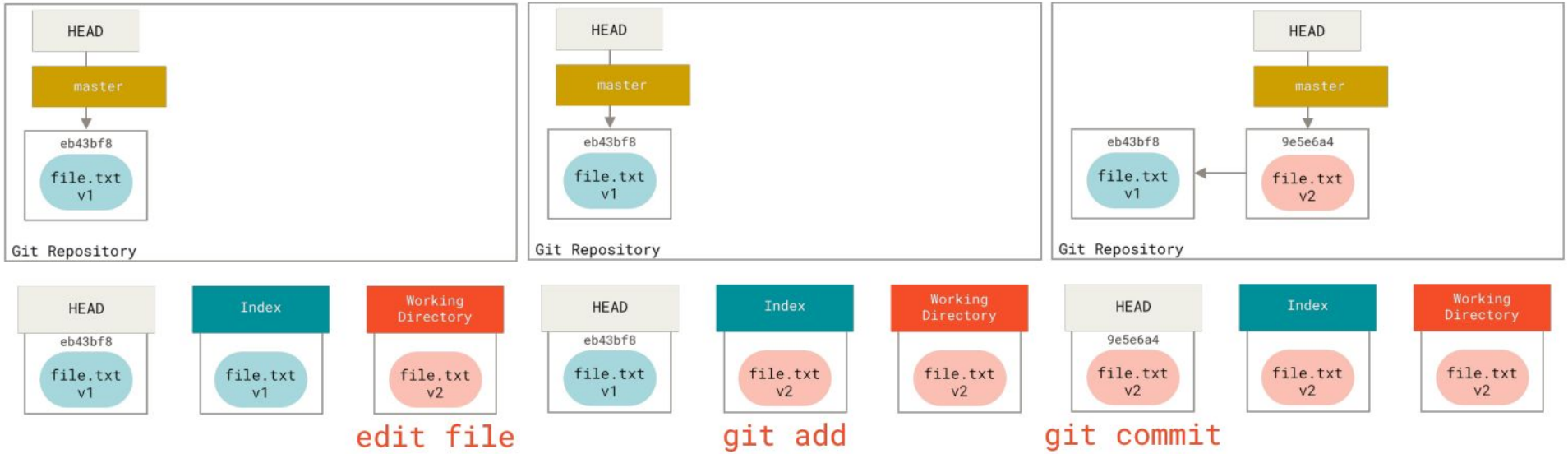
Git commit

- Enregistre un état figé du projet
- Chaque commit est définitif (comme une photo)
- Modifier un commit passé → compliqué (rebase)

JJ change

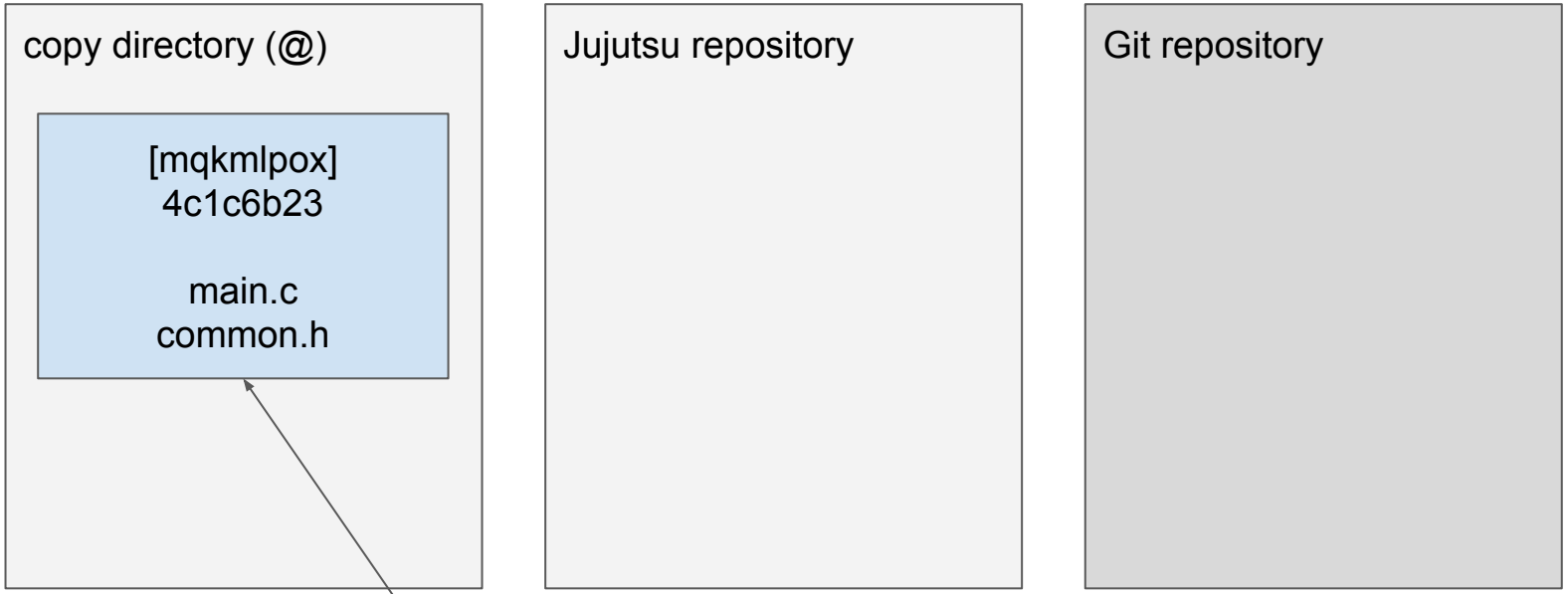
- Regroupe une idée / une tâche (ex : “ajouter une feature”)
- On peut le modifier sans que cela soit trop complexe
- JJ met automatiquement à jour le change actif

Git Workflow - Trois états



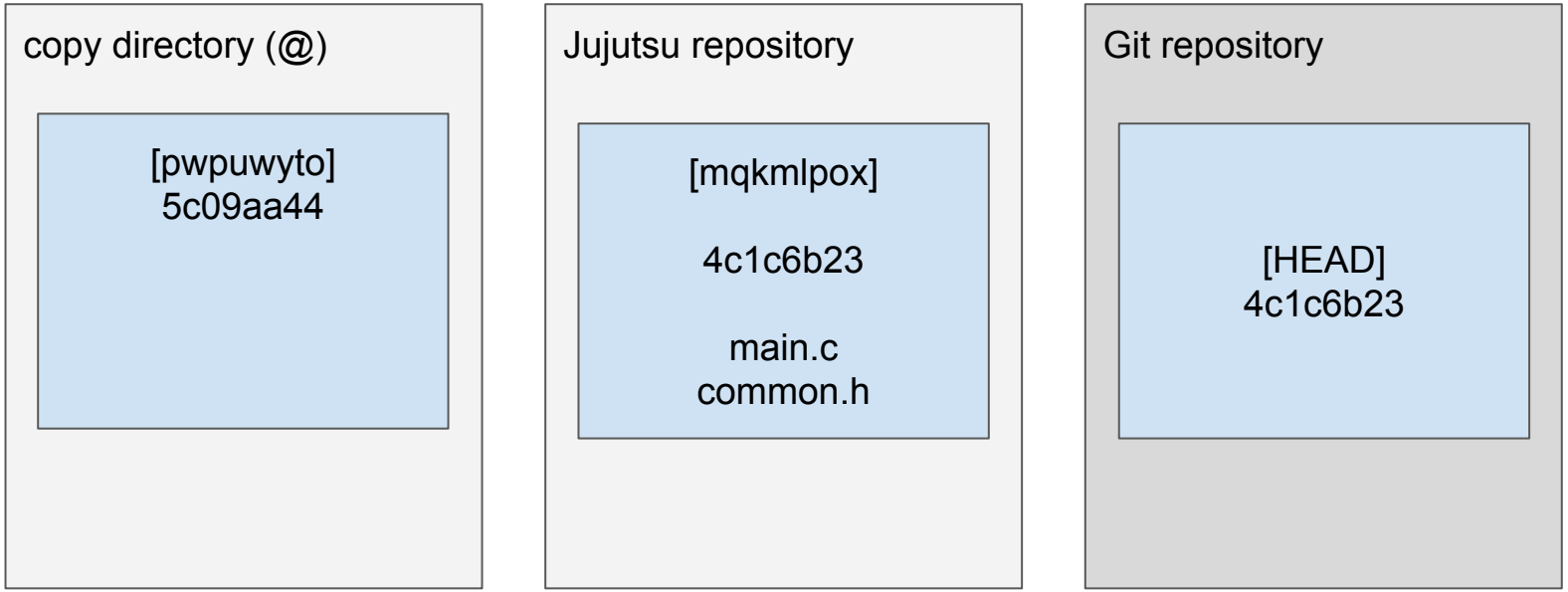
https://github.com/progit/progit2/releases/download/2.1.448/progit.pdf

Jujutsu Workflow - Deux états



la zone de travail est un commit !
Il n'y a plus besoin de staging area

Jujutsu Workflow - Deux états



jj commit -m “[FIX] main.c”

Points fort et limites de jj

Points Forts

- Interface simplifiée (30 commandes vs 400)
- Opérations sûres par défaut
- Undo universel (operation log)
- Rebase automatique intelligent
- Performance (Rust)

Limites Actuelles

- Documentation en développement
- Écosystème naissant
- Intégration IDE limitée
- Version 0.x (pas 1.0 stable)
- Support enterprise inexistant

Conclusion : Faut-il adopter Jujutsu ?

Recommandé si :

- Nouveau projet sans Git
- Équipe petite et agile
- Workflow avec rebases fréquents
- Un environnement enclin à tester de nouveau outils

Déconseillé si :

- Infrastructure Git établie
- Grande organisation
- Dépendance forte aux outils Git
- Équipe réticente au changement

Démonstration

```
~$ jj git init demo
Initialized repo in "demo"
Hint: Running `git clean -xdf` will remove `.jj/`!

~/demo$ jj log
@  rltztzun David-Darras@email.fr 2025-11-19 16:20:20 c4fced2a
│  (empty) (no description set)
◆  zzzzzzzz root() 00000000

~/demo$ echo def sub(a, b): return a - b > math.py
@  rltztzun David-Darras@email.fr 2025-11-19 16:26:31 3546b622
│  (no description set)
◆  zzzzzzzz root() 00000000

~/demo$ jj jj commit -m "[ADD] sub method in math.py" && jj log
@  lwyoxyqy David-Darras@email.fr 2025-11-19 16:31:49 c76e9e84
│  (empty) (no description set)
○  rltztzun David-Darras@email.fr 2025-11-19 16:31:49 git_head()
24f1349d
│  [ADD] sub method in math.py
◆  zzzzzzzz root() 00000000
```

Après initialisation, le dossier demo contient deux dossiers : .git et .jj

Le **change ID** reste inchangé même après la modification du commit, contrairement au **commit ID**. (C'est normal puisqu'une modification crée en réalité un nouveau commit, lui aussi immuable.)

@ correspond au **copy directory** (là où on se trouve actuellement un peu comme HEAD avec Git)

◆ indique que le commit est immuable

○ indique que le commit est muable (on peut changer la description du commit, les fichiers, etc.)

Démonstration

```
~$ jj edit rltztzun
~$ jj desc -m “[NEW] sub function” && jj log
@  rltztzun David-Darras@email.fr 2025-11-19 16:42:10 19e86ded
|  [NEW] sub function
◆  zzzzzzzz root() 00000000

~$ jj undo

~$ jj bookmark create main -r rltztzun
@  nvyvmwus David-Darras@email.fr 2025-11-19 16:47:57 e8b306b1
|  (empty) (no description set)
○  rltztzun David-Darras@email.fr 2025-11-19 16:42:10 main
git_head() 19e86ded
|  [NEW] sub function
◆  zzzzzzzz root() 00000000

~$ jj git remote add origin https://monrepo.fr
~$ jj bookmark track main@origin
~$ jj git push --bookmark main
```

On peut modifier un commit même après plusieurs commits : le **change ID** reste le même, mais le **commit ID** change.

Remarque : on peut simplement indiquer la première lettre du change ID au lieu de tout écrire, et cela fonctionnera aussi :

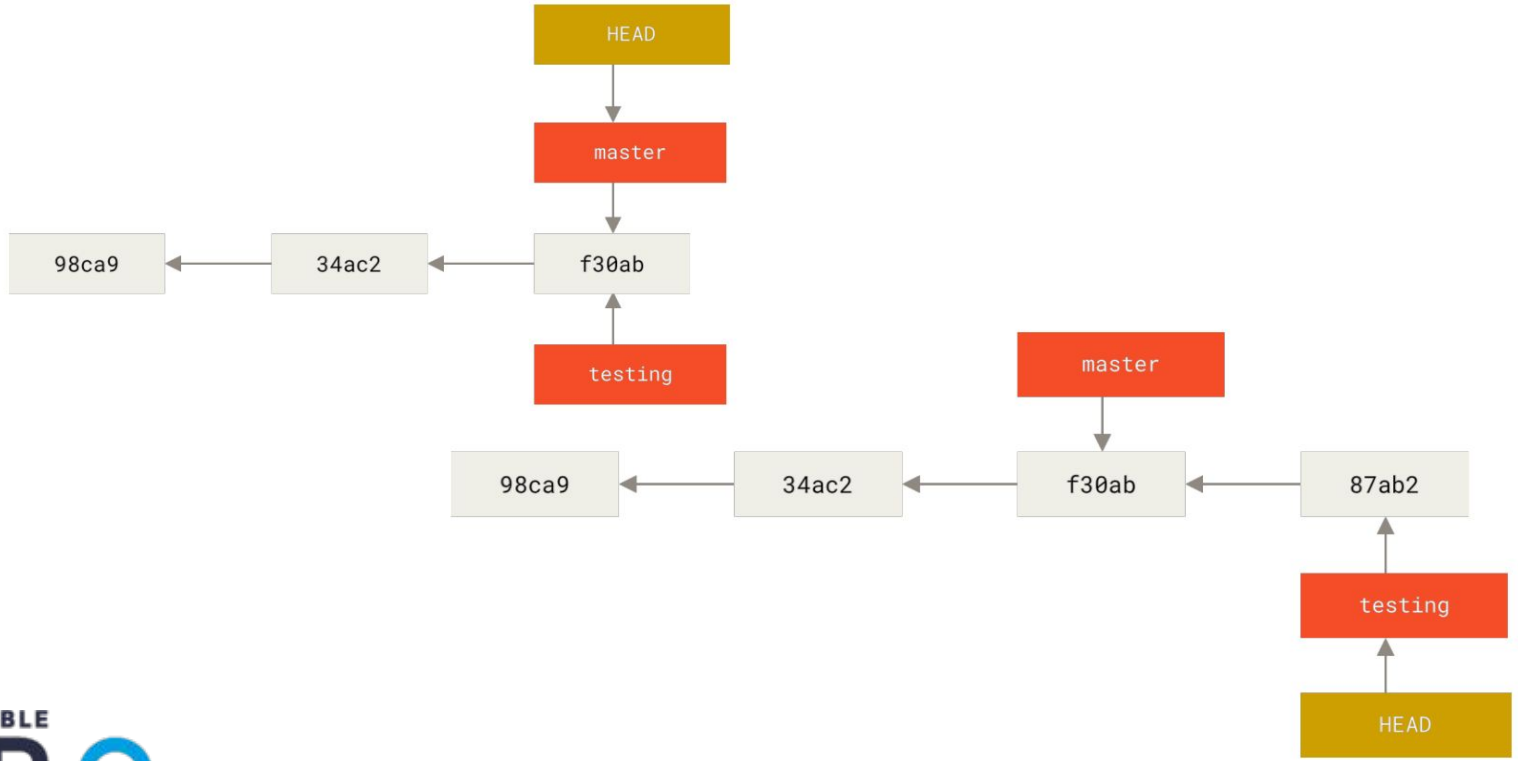
```
jj edit r
```

ici on annule la modification de la description et on restaure l’ancien commit

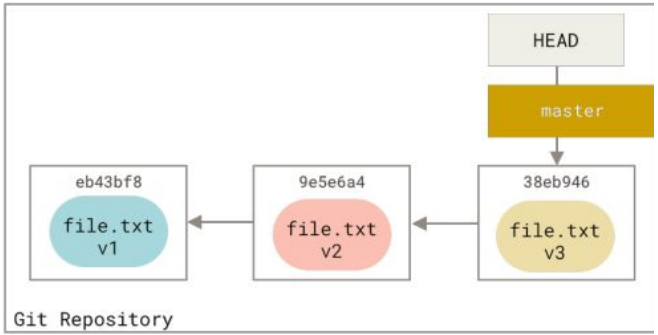
jj n'utilise pas le concept de branche.
On attribue plutôt une étiquette (*bookmark*) au commit que l'on souhaite pousser sur le dépôt distant.

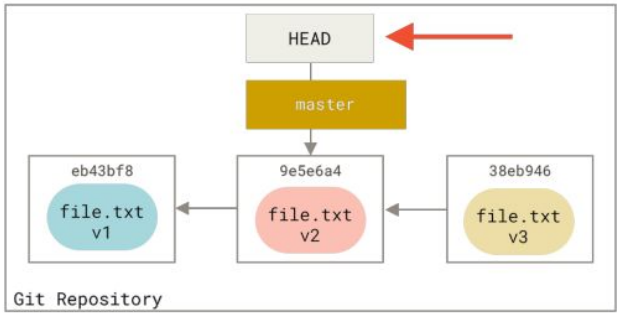
ATTENTION : à partir du moment où un commit est poussé, tous ses descendants, y compris lui-même, deviennent immuables !

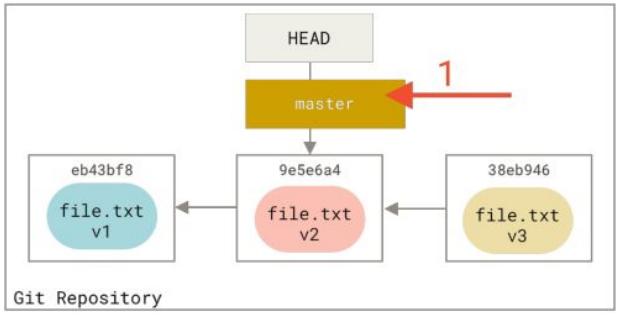
Git Branching vs Jujutsu Bookmark

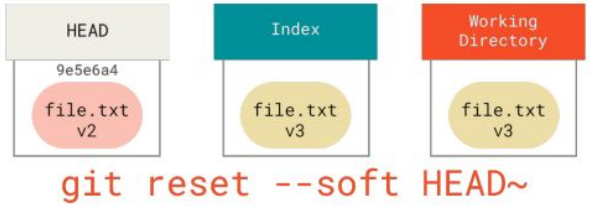


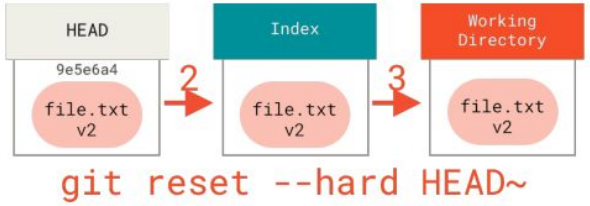
Git Workflow











Resources/Questions