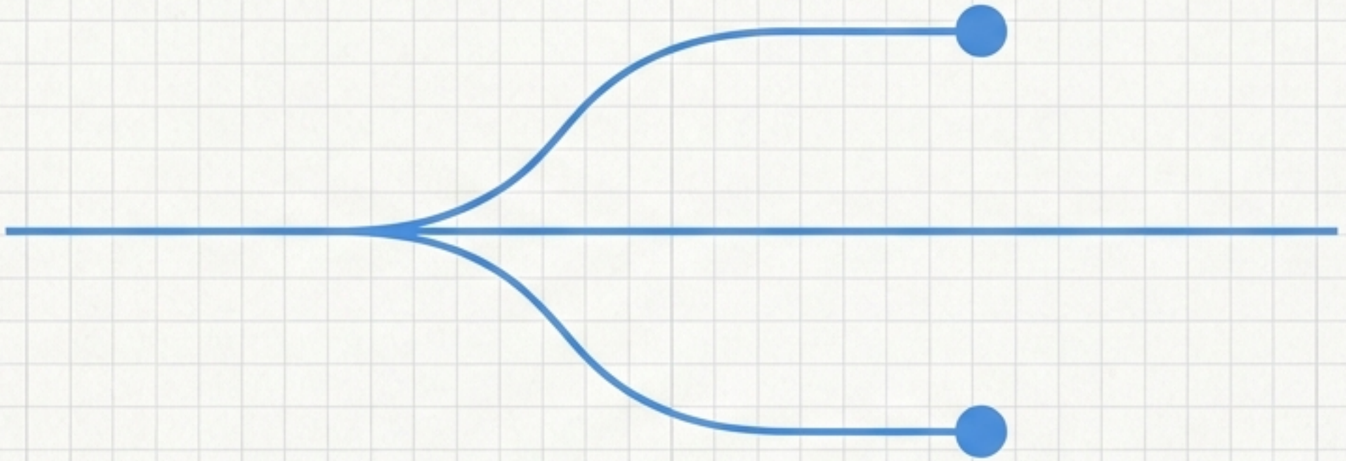
Git en Local : Maîtrise du Versionning et du Voyage Temporel

Un guide pratique pour initialiser un dépôt, naviguer dans l'historique et gérer des branches de développement.



Objectifs Pédagogiques

- Initialiser un dépôt Git pour un nouveau projet.
- Créer et enregistrer plusieurs versions d'un fichier.
- Consulter l'historique des modifications de manière claire.
- Revenir temporairement sur une ancienne version pour inspection.
- Créer une nouvelle ligne de développement (branche) à partir d'un point passé.
- Comprendre et identifier les commandes potentiellement destructrices.

Périmètre

Ce guide se concentre exclusively sur l'utilisation de Git en local. Aucune interaction réseau (GitHub, GitLab, etc.) n'est requise ou abordée.

Préparation de l'Environnement de Travail

Vos Outils

Git installé (Vérifier avec `git --version`)

Un terminal ou une ligne de commande

Un éditeur de texte (ex: VS Code, Sublime Text, Vim)

Votre Identité Numérique

La première fois que vous utilisez Git sur une machine, il est essentiel de définir votre identité. Ces informations seront associées à toutes vos contributions.

```
# À n'exécuter qu'une seule fois par machine
git config --global user.name "Votre Nom"
git config --global user.email "vous@exemple.com"
```

Étape 1 & 2 : La Naissance du Projet et son Initialisation

Créer un répertoire de travail et le placer sous le contrôle de version de Git.

Création de la structure

```
mkdir tp-git-local
cd tp-git-local
echo "Version 1 du projet" > projet.txt
```

Activation de Git

```
git init
git status
```

La commande `git init` crée un sous-dossier caché `.git`. C'est le 'cerveau' de votre dépôt, contenant toute l'histoire. `git status` confirme que `projet.txt` existe mais n'est pas encore suivi (`untracked`).

Étape 3 : Le Premier Jalon (Commit)

Enregistrer la première version officielle de notre fichier dans l'historique du projet.

Le Processus en Deux Temps

1. Préparer la "Photo" (git add) : Sélectionner les fichiers dont les modifications seront incluses dans le prochain point de sauvegarde.

```
git add projet.txt
```

2. Prendre la "Photo" (git commit) : Créer un instantané permanent des fichiers préparés, avec un message descriptif.

```
git commit -m "Version 1 : création du fichier projet"
```

Vérification

```
git log --oneline
```

Affiche une ligne unique avec un identifiant (hash) et le message du commit, confirmant que notre jalon a été posé.

Étape 4 & 5 : Construire la Chronologie du Projet

Objectif : Ajouter de nouvelles versions au projet pour enrichir son historique.

Le Cycle de Développement

Modifier -> Ajouter -> Valider.

Création de la Version 2

`echo "Version 2 du projet" >> projet.txt`

```
git add projet.txt
git commit -m "Version 2 : ajout d'une ligne"
```

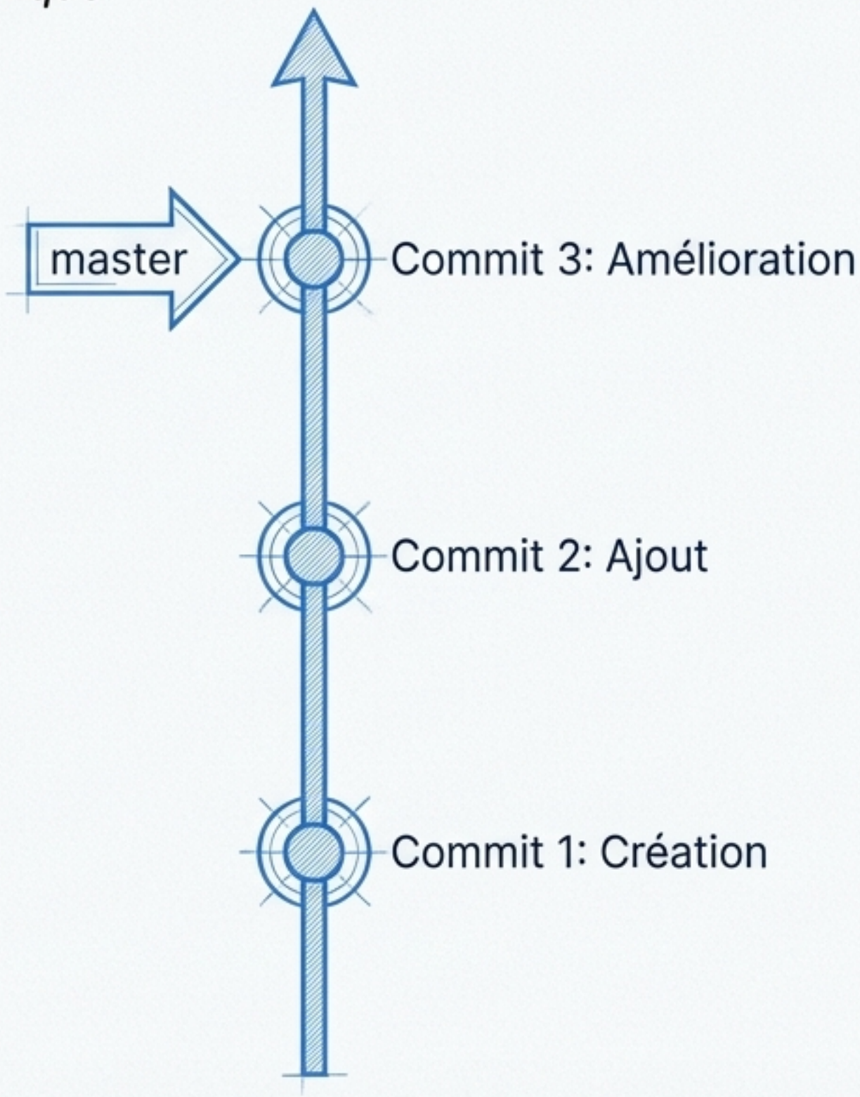
Création de la Version 3

`echo "Version 3 du projet" >> projet.txt`

```
git add projet.txt
git commit -m "Version 3 : amélioration du contenu"
```

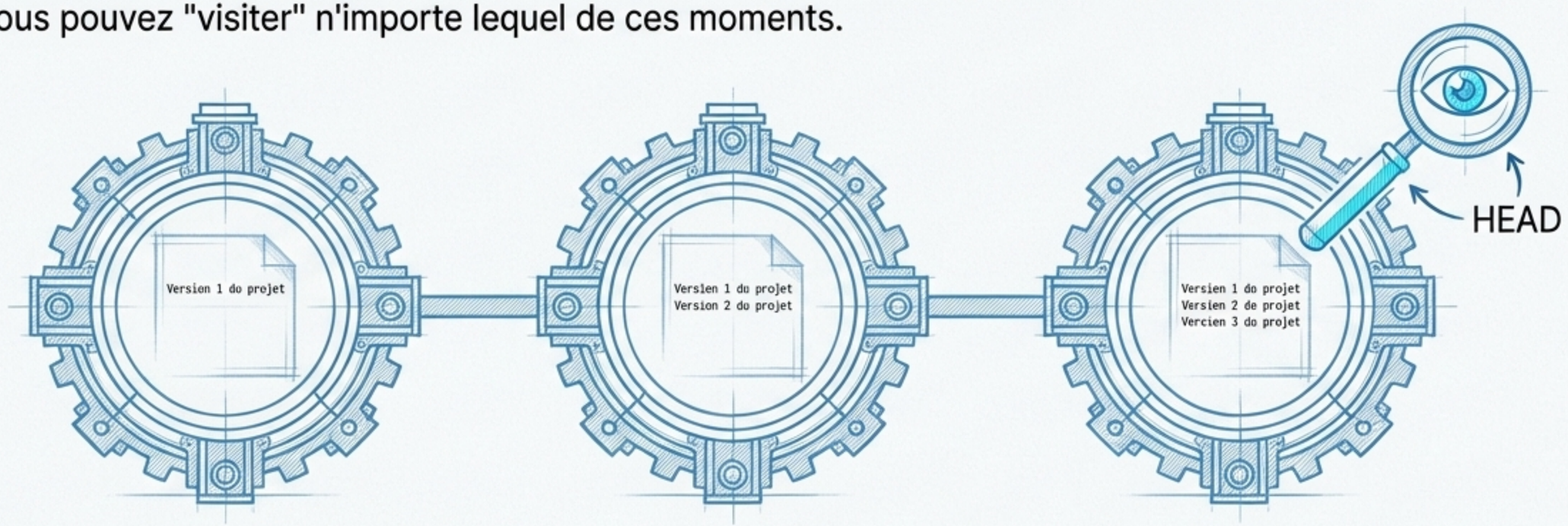
Vue d'ensemble de l'historique

`git log --oneline`



L'Exploration Temporelle : Naviguer dans l'Historique

Concept Clé : Chaque commit dans Git est un portail vers une version passée complète de votre projet. Vous pouvez "visiter" n'importe lequel de ces moments.



L'outil d'exploration : La commande `git checkout` permet de déplacer votre 'tête de lecture' (HEAD) vers un commit spécifique.

Métaphore Visuelle : Pensez à l'historique comme à une bibliothèque de sauvegardes. `git checkout` vous permet de consulter un ancien volume en lecture seule.

Étape 6 : Visite Temporaire d'une Ancienne Version

Objectif : *Consulter le contenu du projet tel qu'il était lors d'un commit passé.*

Procédure

1. **Identifier la destination :** Listez les commits pour trouver le hash (identifiant) de la version que vous souhaitez visiter.

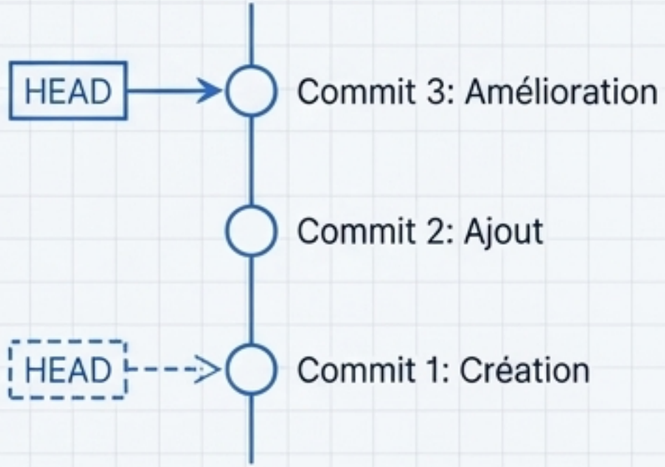
```
git log --oneline
```

2. **Voyager dans le temps :** Utilisez `checkout` avec le hash désiré (ex: celui de la 'Version 1').

```
git checkout <hash_du_commit_version_1>
```

3. **Inspecter le fichier :** Vérifiez que le contenu est bien celui de la version 1.

```
cat projet.txt \
# Affiche: "Version 1 du projet"
```

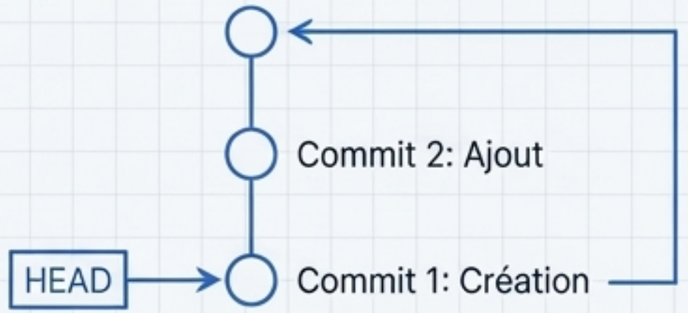


⚠ ****État 'HEAD détachée'**** : Git vous informe que vous êtes dans un état de consultation. Les modifications ici ne font partie d'aucune branche.

Retour au présent

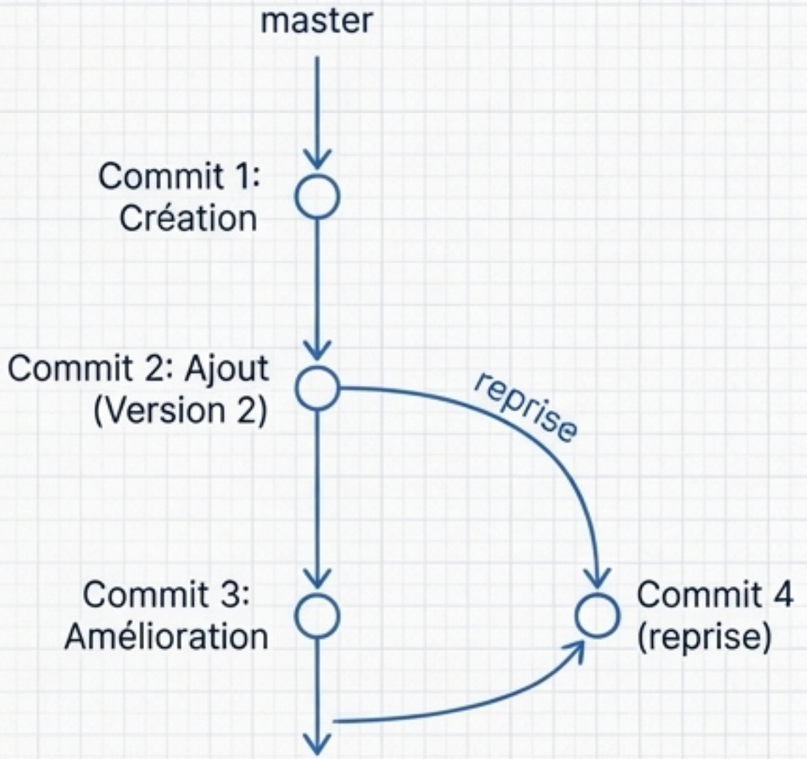
Pour revenir à la dernière version de votre branche de travail principale, exécutez :

```
git checkout master # ou main selon votre configuration
```



Étape 7 : Repartir du Passé via une Nouvelle Branche

Objectif : *Créer une nouvelle ligne de développement alternative à partir d'une ancienne version, sans perturber la branche principale.*



La Méthode Recommandée

La commande `git switch -c` est l'outil moderne et sûr pour créer et basculer sur une nouvelle branche en une seule étape.

Mise en Œuvre

1. **Créer et basculer**
   Placez-vous sur l'ancien commit et créez la branche reprise simultanément.

```
# Remplacer <hash_ancien> par le hash de la "Version 2" par exemple
git switch -c reprise <hash_ancien>
```

2. **Créer une nouvelle histoire**
   Modifiez le fichier et validez. Ce commit existera *uniquement* sur la nouvelle branche reprise.

```
echo "Version 4 (reprise)" >> projet.txt
git add projet.txt
git commit -m "Reprise : nouvelle version depuis un ancien commit"
```

Vous avez créé un 'univers parallèle' où le projet a évolué différemment à partir de la Version 2.

Étape 8 : Visualiser les Univers Parallèles

Objectif : *Obtenir une représentation graphique de l'historique, montrant toutes les branches et leur point de divergence.*

La Commande Magique : `git log --graph` enrichi de quelques options utiles.

```
git log --oneline --graph --decorate --all
```

```
* ◯ (HEAD -> reprise) Reprise : nouvelle version depuis un ancien commit
|\ \
| | |\
* | | ◯ (master) Version 3 : Amélioration
| | | |
| |_|/
|/| |
  * | ◯ Version 2 : Ajout
  | |/|
  |/| |
  * | ◯ Version 1 : Création
```

Analyse du Résultat Attendu

- Le terminal affichera un diagramme en art ASCII.
- Vous verrez clairement la branche `master` (ou `main`) continuant jusqu'à la **'Version 3'**.
- Vous verrez une nouvelle branche `reprise` qui a divergé après la **'Version 2'** et qui possède son propre commit **'Reprise'**.

Commande Dangereuse : Le Point de Non-Retour

```
git reset --hard <hash>
```

- Replace la branche actuelle sur le commit spécifié (<hash>).
- **Supprime définitivement tous les commits qui ont eu lieu après ce** <hash> **sur la branche actuelle.**
- Modifie les fichiers de votre répertoire de travail pour qu'ils correspondent exactement à l'état du <hash>. Les modifications non validées sont perdues.

À utiliser avec une extrême prudence. C'est un outil puissant pour annuler des erreurs, mais il réécrit l'histoire de manière destructive. Il n'y a généralement pas de retour en arrière possible.

Bilan : Votre Boîte à Outils Git Essentielle

Fondation & Historique

`git init`
Initialise un dépôt.

`git add <fichier>`
Ajoute un fichier à la zone de préparation.

`git commit -m "..."`
Crée un nouvel instantané (commit).

Exploration & Navigation

`git log --oneline`
Affiche l'historique de manière concise.

`git status`
Montre l'état actuel du répertoire.

`git checkout <hash>`
Visite un ancien commit (lecture seule).

Branches & Réécriture

`git switch -c <nom> <hash>`
Crée et bascule sur une nouvelle branche.

`git branch`
Liste les branches existantes.

`git reset --hard <hash>`
⚠️ Réinitialise l'historique de manière **destructive**.

La maîtrise de ces commandes constitue une base solide pour toute utilisation de Git, en local comme en équipe.