Maîtriser Git & GitHub

Des Fondamentaux à la Collaboration Efficace

Objectifs d'Apprentissage

1. **Comprendre** la nécessité de la gestion de version pour tout projet de développement.
2. **Distinguer** l'outil (Git) de la plateforme (GitHub) et leur interaction.
3. **Maîtriser** le flux de travail local essentiel : initialiser, ajouter, valider.
4. **Utiliser** les branches pour un développement parallèle, isolé et sans risque.
5. **Collaborer** efficacement sur des projets en utilisant le modèle Fork & Pull Request.
6. **Appréhender** les bonnes pratiques et les outils pour un usage professionnel.

Pourquoi la Gestion de Version est-elle Cruciale ?

Le Développement sans Filet de Sécurité

Imaginez deux développeurs, Alice et Bob, travaillant sur le même projet. Ils rencontrent rapidement deux problèmes majeurs :

Problème 1 : Perte de travail.

> "L'ordinateur de Bob est vieux et peu fiable. Il redémarre beaucoup et Bob ne veut *vraiment* pas avoir à refaire tout son travail tout le temps." Comment sauvegarder et restaurer le code de manière fiable ?

Problème 2 : Conflits de collaboration.

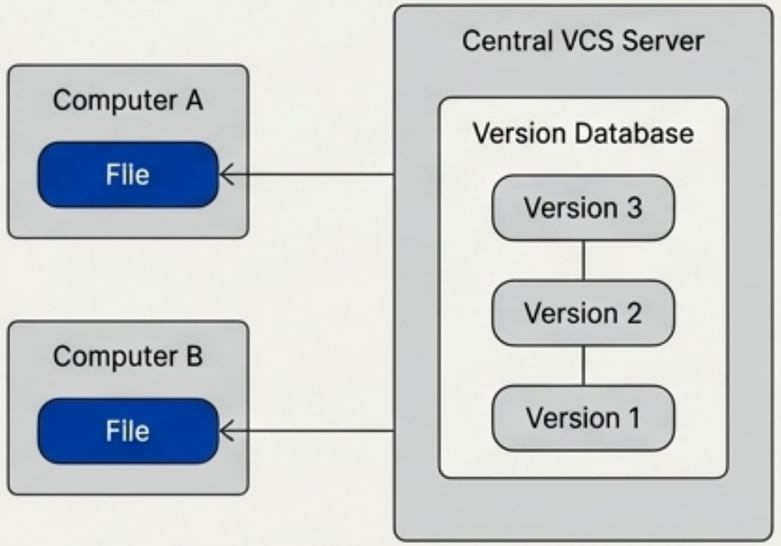
> Alice et Bob modifient le même fichier. Alice utilise une valeur, Bob une autre. Comment fusionner leurs modifications sans écraser le travail de l'autre ? Le risque est une "lutte acharnée où Alice change [la valeur], puis Bob la rétablit, puis Alice la modifie à nouveau".

Conclusion : Sans un système dédié, le développement devient chaotique, risqué et inefficace.

L'Évolution : Des Systèmes Centralisés aux Systèmes Distribués

CVCS (Système de Gestion de Version Centralisé) - Ex: Subversion

Un serveur unique contient tous les fichiers versionnés. Les clients extraient les fichiers depuis ce dépôt central.

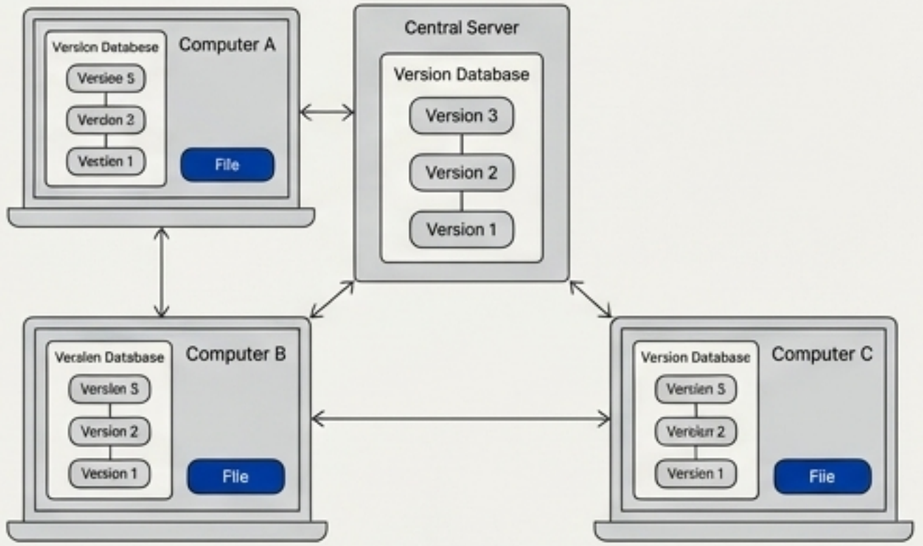


⚠ Inconvénients

- Représente un point unique de défaillance ("single point of failure"). Si le serveur est en panne, personne ne peut collaborer.
- Le travail hors ligne est très limité.

DVCS (Système de Gestion de Version Distribué) - Ex: Git

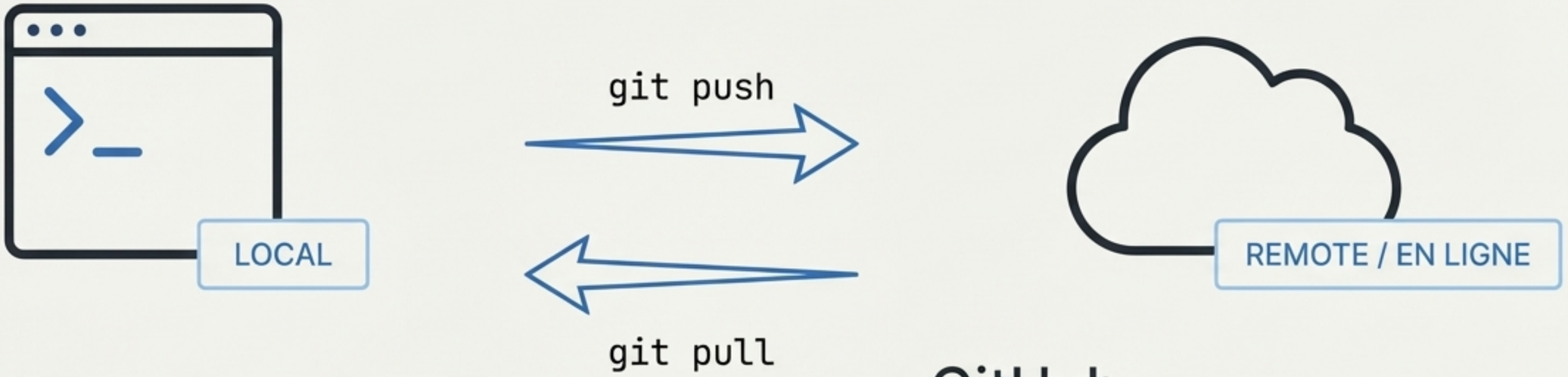
Les clients n'extraient plus seulement la dernière version d'un fichier, mais ils dupliquent complètement le dépôt.



✓ Avantages

- Chaque client possède une sauvegarde complète de l'historique.
- La plupart des opérations (commit, branch, merge) sont locales et donc extrêmement rapides.
- Collaboration et travail hors ligne fluides.

Distinction Essentielle : Git n’est pas GitHub



Git

Git est le logiciel de gestion de version distribué qui s’exécute sur votre machine locale. C’est l’outil qui suit les changements de votre code.

Analogie : Le moteur.

GitHub

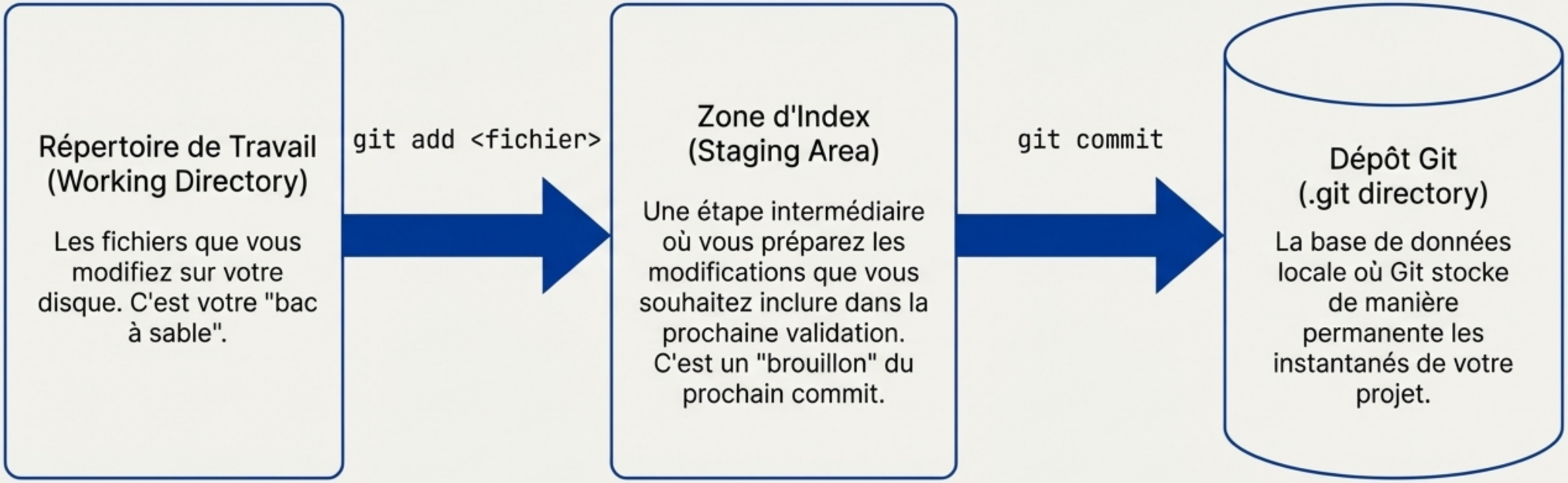
GitHub est un service d’hébergement en ligne pour les dépôts Git. C’est la plateforme pour stocker, partager et collaborer sur le code.

Analogie : Le garage et le club social.

Alternatives : GitLab, Bitbucket.

Le Cycle de Vie d'un Fichier : Les Trois États de Git

Pour comprendre Git, il est primordial de connaître les trois états principaux dans lesquels un fichier peut se trouver.



Le Quotidien du Développeur : Le Cycle de Travail Local

1. Démarrer un projet

```
git init
```

Crée un nouveau dépôt Git dans le répertoire courant.

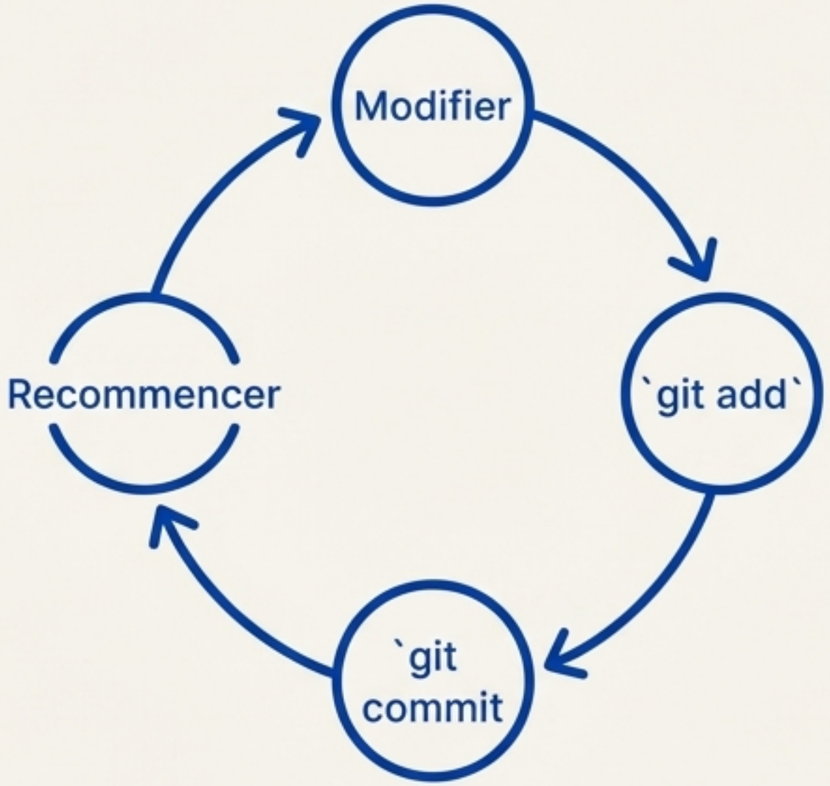
```
git clone [url]
```

Crée une copie locale d'un dépôt distant existant.

2. Vérifier l'état

```
git status
```

Montre l'état des fichiers (modifiés, indexés, non suivis). C'est votre commande la plus utilisée.



3. Préparer la validation

```
git add [fichier]
```

Ajoute les modifications du fichier à la zone d'index (Staging Area).

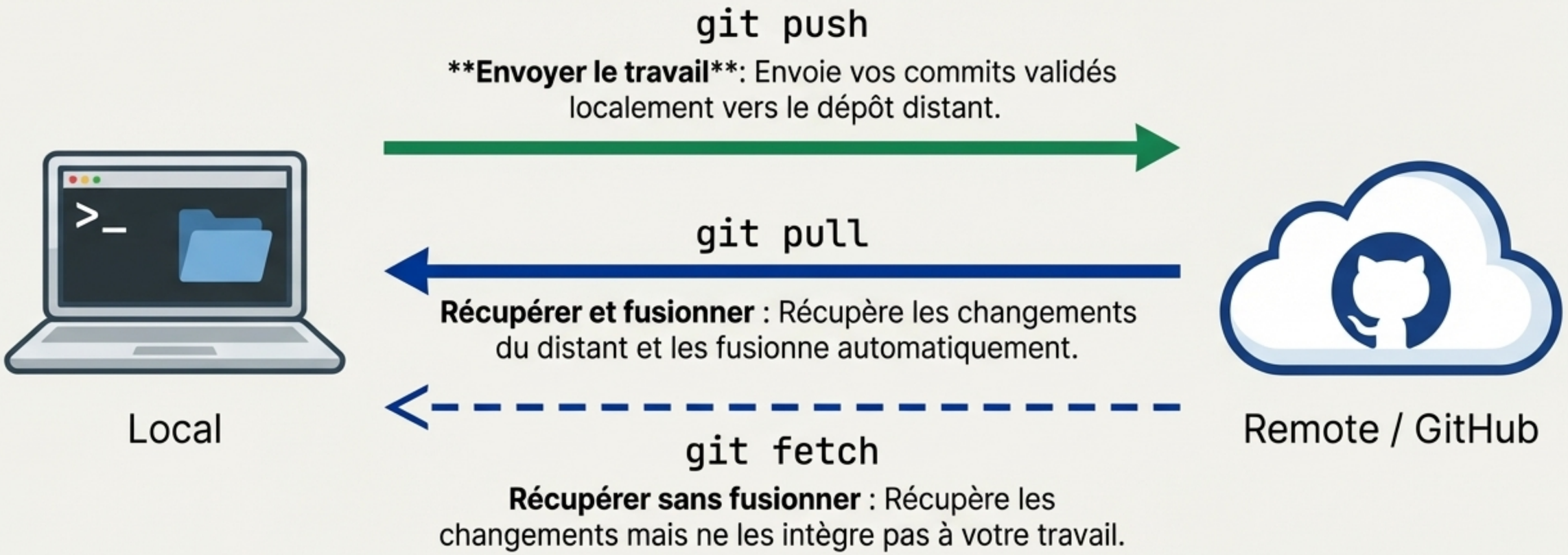
4. Enregistrer l'instantané

```
git commit -m "Message de commit descriptif"
```

Enregistre de manière permanente l'instantané de la zone d'index dans le dépôt local.

Synchronisation : Interagir avec un Dépôt Distant

Une fois le travail validé localement, il faut le partager ou récupérer les mises à jour des collaborateurs.



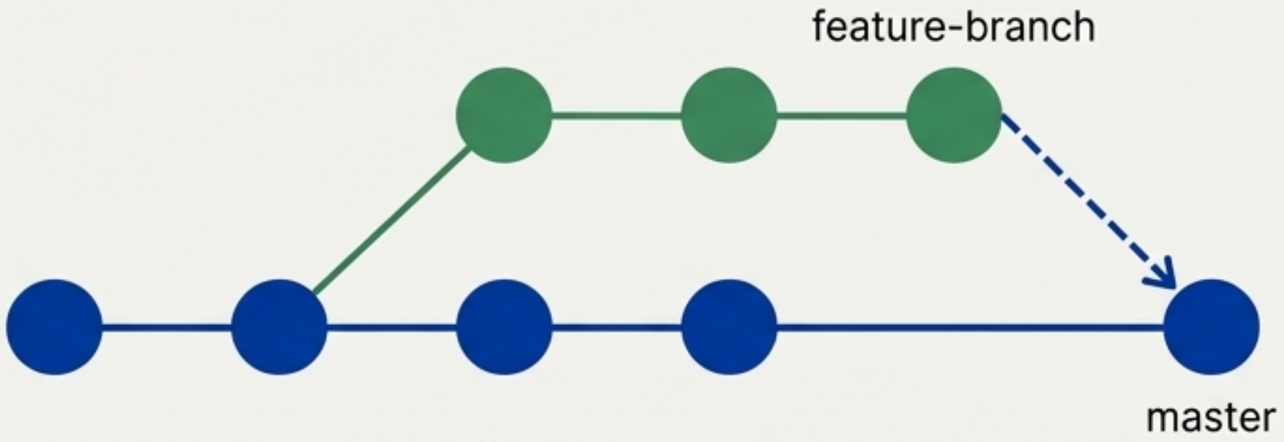
Le Développement Parallèle Simplifié : Le Concept des Branches

Qu'est-ce qu'une branche ?

- C'est un simple pointeur mobile vers un commit.
- La branche par défaut est nommée `master` ou `main`.

Pourquoi utiliser des branches ?

- ****Isolation :*** Permet de développer une nouvelle fonctionnalité ou de corriger un bug sans affecter la ligne de développement principale (la branche `main`).
- ****Expérimentation :*** On peut explorer des idées sans craindre de "casser" le code stable.



La puissance de Git :

- Dans Git, créer, utiliser et fusionner des branches est une opération extrêmement **légère et rapide**.
- Il est courant de créer plusieurs "branches thématiques" (feature branches) par jour.

Le Cycle de Vie d'une Branche Thématique

1. Créer une nouvelle branche :

```
git branch nom-de-la-fonctionnalite
```

2. Se déplacer sur la nouvelle branche pour commencer à travailler :

```
git checkout nom-de-la-fonctionnalite
```

Raccourci (créer et se déplacer en une commande) :
`git checkout -b nom-de-la-fonctionnalite`

3. Travailler sur la branche :
Modifier les fichiers, puis utiliser **`git add`** et **`git commit`** comme d'habitude. Tous les commits sont enregistrés sur cette branche isolée.

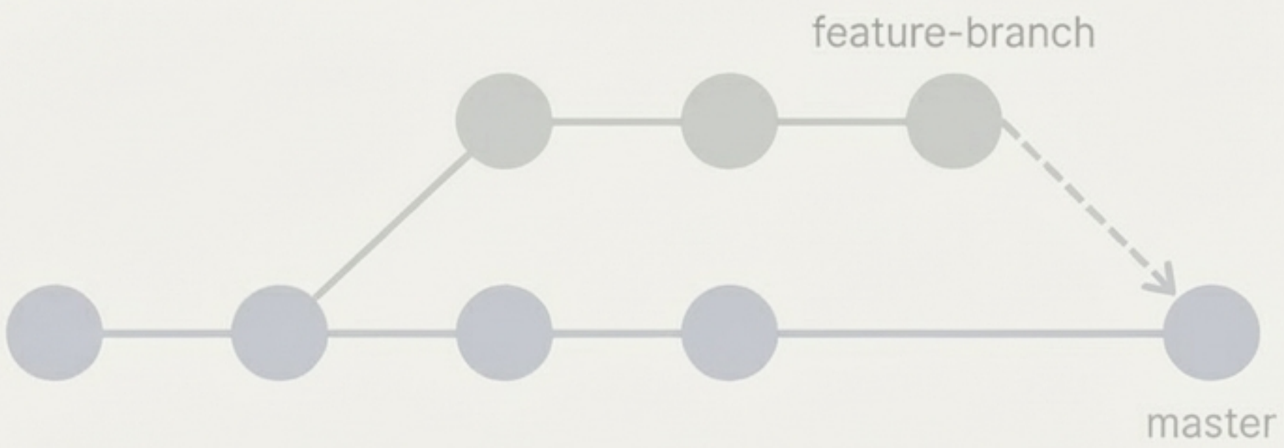
4. Retourner sur la branche principale :

```
git checkout main
```

5. Fusionner le travail terminé :

```
git merge nom-de-la-fonctionnalite
```

(Intègre les commits de la branche thématique dans **`main`**).



Le "GitHub Flow" : Contribuer à n'importe quel projet

Comment proposer des modifications à un projet que vous ne possédez pas ? Le flux de travail standard est basé sur les "Forks" et les "Pull Requests".

1 Fork

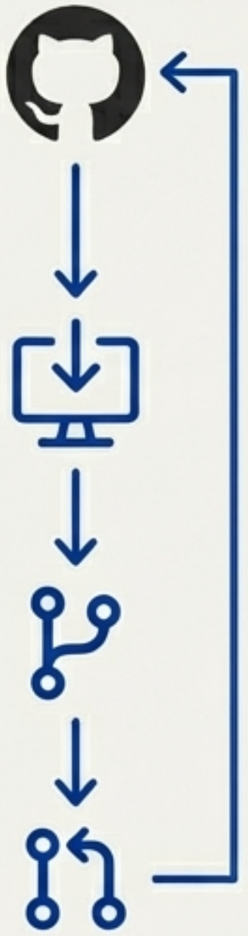
Fork

Sur GitHub, cliquez sur le bouton "Fork". Cela crée une copie personnelle (un "fork") du dépôt sur votre propre compte GitHub.

2 Clone

Clonez **votre fork** sur votre machine locale pour pouvoir y travailler.

```
git clone [url-de-votre-fork]
```



3 Branch & Code

Clone **votre fork** sur votre machine locale pour pouvoir y travailler.

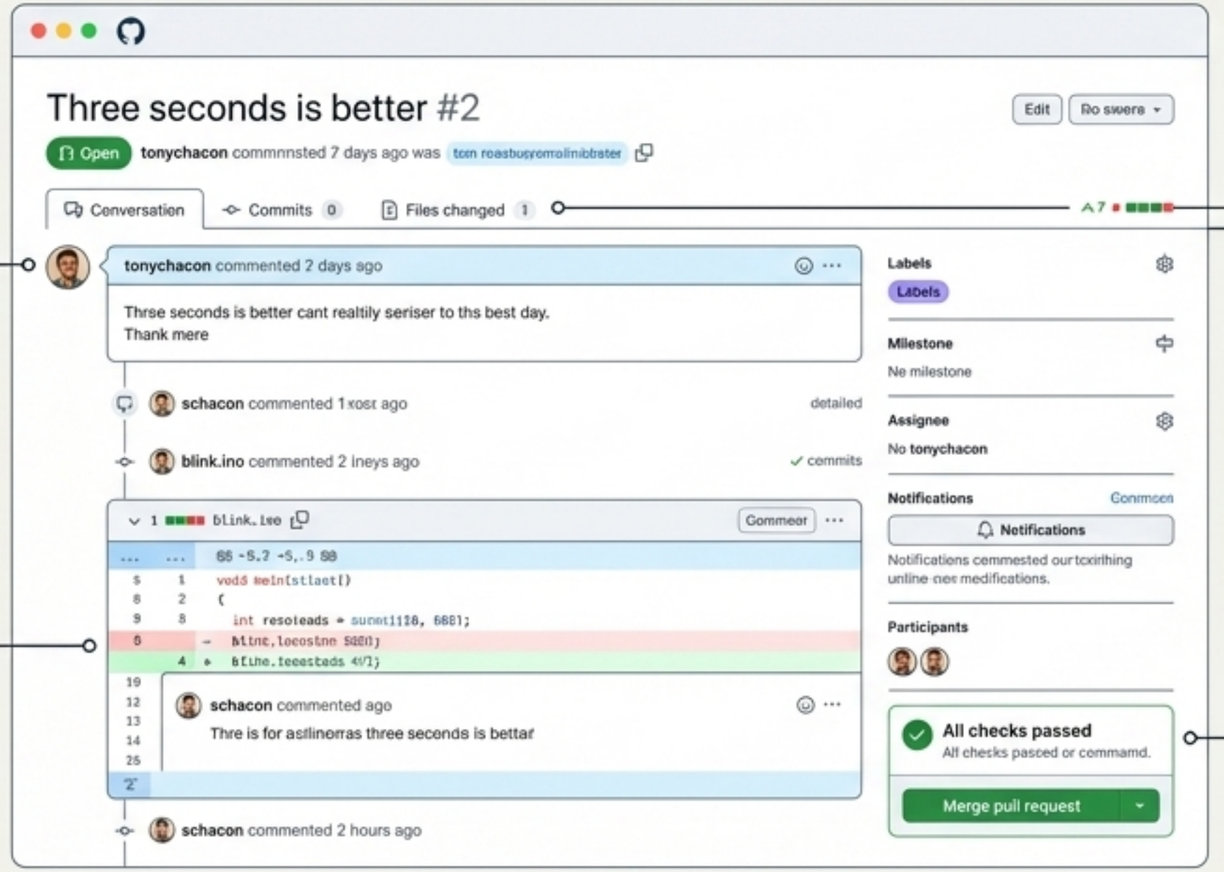
```
git clone [url-de-votre-fork]
```

4 Pull Request (PR)

Depuis votre fork sur GitHub, ouvrez une "Pull Request" (demande de tirage) vers le dépôt original. Cela lance le processus de proposition de vos modifications.

Anatomie d’une Pull Request : Un Espace de Discussion et de Revue

Une PR n'est pas un simple mécanisme technique. C'est le lieu central de la collaboration autour d'une modification.



Conversation

Un fil de discussion général pour discuter des objectifs, des choix de conception et des retours.

Revue de Code

Permet de commenter chaque ligne de code modifiée, posant des questions ou suggérant des améliorations.

Liste des Commits

Un historique clair de toutes les validations incluses dans la PR.

Fichiers Modifiés (“Diff”)

Une vue claire et unifiée de tous les changements (ajouts et suppressions).

Vérifications d’Intégration Continue (CI)

Des statuts automatiques indiquent si les tests passent, si le code respecte les standards, etc.

Flux de Travail d’Équipe Avancés

Pour les très grands projets, des flux de travail plus structurés sont nécessaires pour gérer des centaines de contributeurs.

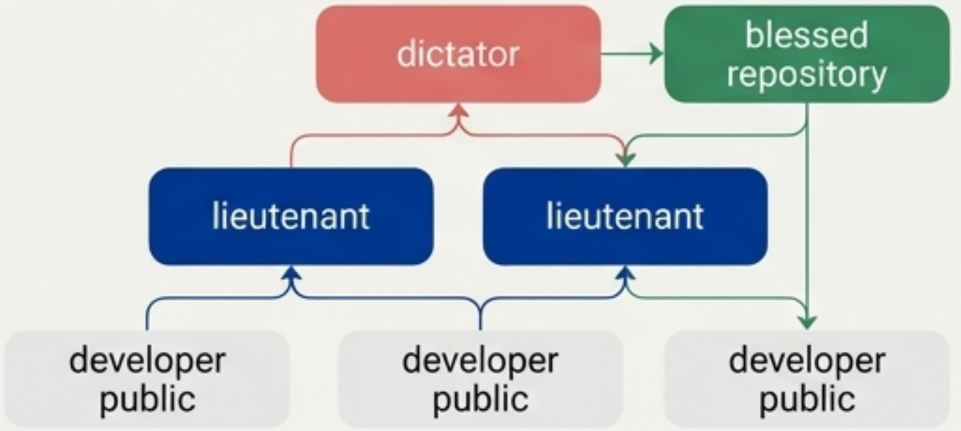
Modèle 1 : Le Gestionnaire d’Intégration

- **Principe :**
  Un petit groupe de personnes (les "mainteneurs") a le droit d'écrire sur le dépôt principal.

- **Flux :**
  Les contributeurs externes “forkent” le projet, proposent leurs modifications via des Pull Requests, et les mainteneurs sont responsables de l'intégration (la fusion) de ces contributions. C'est le modèle de la plupart des projets sur GitHub.

Modèle 2 : Dictateur et Lieutenants (Ex: Noyau Linux)

- **Principe :**
  Une structure hiérarchique pour gérer les contributions à très grande échelle.



1. Les développeurs envoient leur travail à un “lieutenant” responsable d'un sous-système.
2. Chaque lieutenant fusionne et teste les contributions pour son sous-système.
3. Le “dictateur” (le mainteneur principal) tire les modifications des branches des lieutenants pour les intégrer dans le projet principal.

Ligne de Commande (CLI) vs. Interface Graphique (GUI) : Quel Outil pour Quel Usage ?

Principe fondamental : Il n'y a pas de "meilleur" outil. L'approche la plus efficace consiste à "utiliser les deux de manière judicieuse". Une IHM (Interface Homme Machine) n'est qu'une surcouche qui exécute **des commandes Git en arrière-plan.**

Interface Graphique (GUI)

Ex: GitHub Desktop, GitKraken

Idéal pour :

- Opérations très visuelles : visualiser l'historique des branches, comparer les versions d'un fichier (`diff`).
- Indexation interactive (`git add --patch`).
- Opérations simples comme le `commit` et le `push`.

Limites :

- Masque la complexité et peut nuire à la compréhension profonde de Git.
- Toutes les commandes et options de Git ne sont pas accessibles.

Ligne de Commande (CLI)

Le Terminal

Idéal pour :

- **Apprendre** le fonctionnement réel de Git.
- Scripts et automatisation des tâches.
- Accès à 100% des fonctionnalités de Git.
- Opérations complexes : `rebase`, `cherry-pick`, etc.

L'Art du Message de Commit : Rédiger un Historique Utile

****Pourquoi est-ce important ?**** Un message de commit clair explique le "pourquoi" d'un changement, pas seulement le "quoi". C'est une documentation essentielle pour vos collaborateurs et votre futur vous.

Le Modèle en 4 Règles (inspiré par Tim Pope)

1. **Sujet et corps séparés :** Séparez le sujet du corps du message par une ligne vide.
2. **Sujet concis :** Limitez la ligne de sujet à 50 caractères.
3. **Sujet à l'impératif :** Rédigez le sujet à l'impératif présent. Ex: "Ajoute la validation" et non "Ajouté la validation". Pensez "Si appliqué, ce commit va... [votre sujet]".
4. **Corps explicatif :** Le corps du message explique le contexte, le problème résolu et la solution apportée. Faites des retours à la ligne à 72 caractères.

Exemple de bon commit

```
Corrige le problème d'authentification des utilisateurs

Le corps du message donne plus de détails. La vérification du token
était effectuée avant la validation du format de l'email, ce qui
pouvait causer une erreur 500 dans certains cas de figure.

Cette modification inverse l'ordre des vérifications pour assurer
que les données sont valides avant de contacter le service d'auth.

- Les listes à puces sont aussi possibles pour détailler les points.
```

Ce qu’il Faut Retenir

La gestion de version est **non-négociable** pour tout projet logiciel sérieux, individuel ou en équipe.

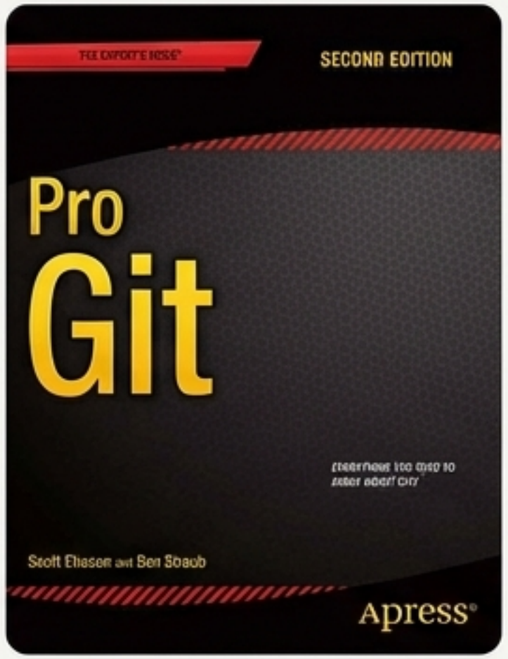
Git est un outil **local, rapide et puissant** basé sur un modèle d'instantanés (snapshots) qui garantit intégrité et performance.

GitHub est la **plateforme de collaboration** qui héberge vos dépôts Git et facilite la revue de code et le travail d'équipe.

Les branches sont votre meilleur outil pour un **développement organisé et sans risque**. Utilisez-les abondamment.

Les Pull Requests sont le cœur de la **revue de code** et de la collaboration. C'est un espace de discussion, pas seulement une fusion.

Pour Aller Plus Loin



Le Livre de Référence : *Pro Git* de Scott Chacon et Ben Straub.

La ressource la plus complète et faisant autorité. Disponible gratuitement en ligne en français.

Documentation Officielle

`git-scm.com`

Pour des recherches précises sur les commandes et les options.

Guides et Tutoriels GitHub

`guides.github.com`

`learn.github.com`

Des guides thématiques et des tutoriels interactifs pour maîtriser le "GitHub Flow".