Gestion des Sources

Maîtriser le Contrôle de Version

De la théorie à la pratique avec Git

Pourquoi utiliser un VCS/SCM ?

Suivi des modifications

Historique complet du projet permettant de suivre l'évolution du code et de comprendre les décisions prises.

Collaboration facilitée

Permet à plusieurs développeurs de travailler simultanément sur le même projet sans conflit.

Restauration facile

Possibilité de revenir à n'importe quelle version antérieure en cas de problème.

Développement parallèle

Création de branches pour développer de nouvelles fonctionnalités sans affecter la version principale.

Le Langage du Contrôleur de Version

Dépôt (Repository)

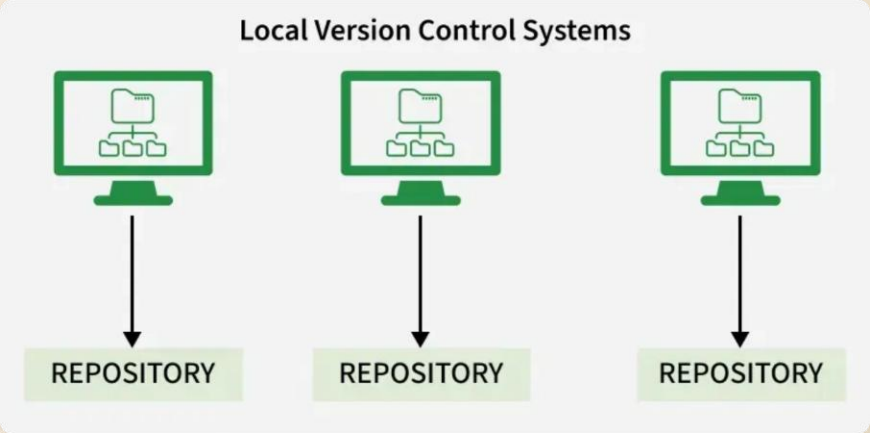
Le "dossier" central contenant tout le projet et son historique complet de modifications.

Commit

Un "instantané" des modifications à un moment donné, avec un message descriptif et un identifiant unique.

Branche (Branch)

Une ligne de développement indépendante permettant de travailler sur des fonctionnalités sans affecter la branche principale.



Le Langage du Contrôleur de Version (Suite)

Fusion (Merge)

L'action de combiner les modifications de différentes branches pour intégrer les changements.

Tag

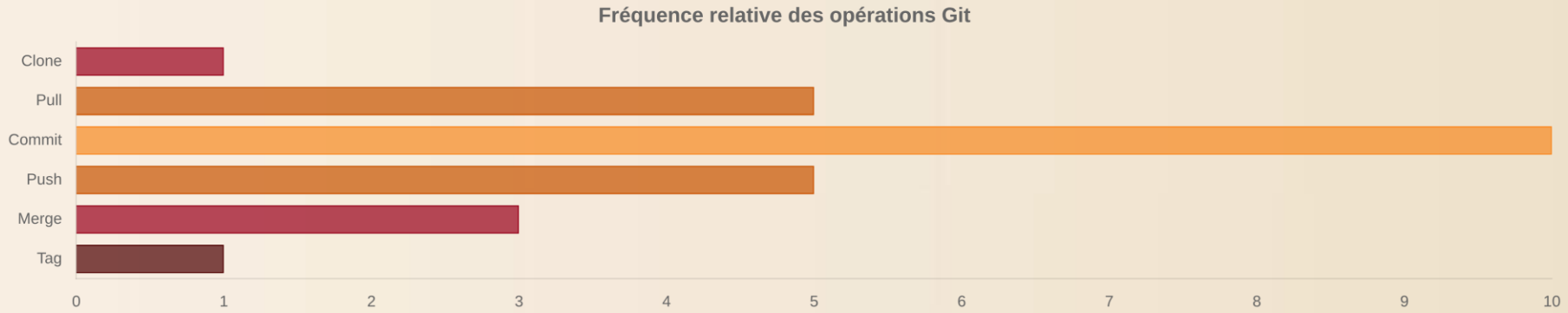
Un marqueur pour identifier une version spécifique du code (ex: v1.0, v2.1.3).

Clone

Copie complète d'un dépôt distant sur votre machine locale.

Pull / Push

Récupérer (pull) ou envoyer (push) des modifications entre dépôts locaux et distants.



L'Art de la Ramification (Branching)

Branche principale

La branche **main** ou **master** représente la version stable et de production du code.

Branches de fonctionnalité

Les **feature branches** sont créées pour développer de nouvelles fonctionnalités isolément.

Branches de correction

Les **hotfix branches** permettent de corriger rapidement des bugs critiques en production.

Collaborer Efficacement

1. ## Créer une branche
   Créez une branche dédiée à votre tâche à partir de la branche principale.

2. ## Effectuer des commits réguliers
   Enregistrez vos modifications avec des messages clairs et descriptifs.

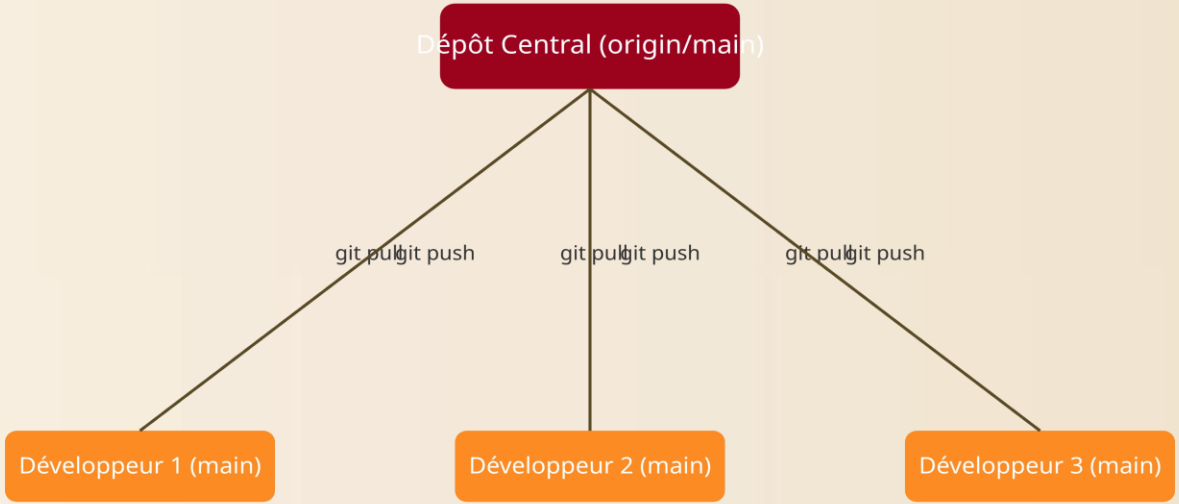
3. ## Mettre à jour votre branche
   Synchronisez régulièrement avec la branche principale pour éviter les conflits.

4. ## Ouvrir une Pull Request
   Soumettez vos modifications pour revue de code par l'équipe.

5. ## Fusionner la branche
   Une fois approuvée, intégrez votre branche dans la branche principale.

Git en Action : Workflow Centralisé

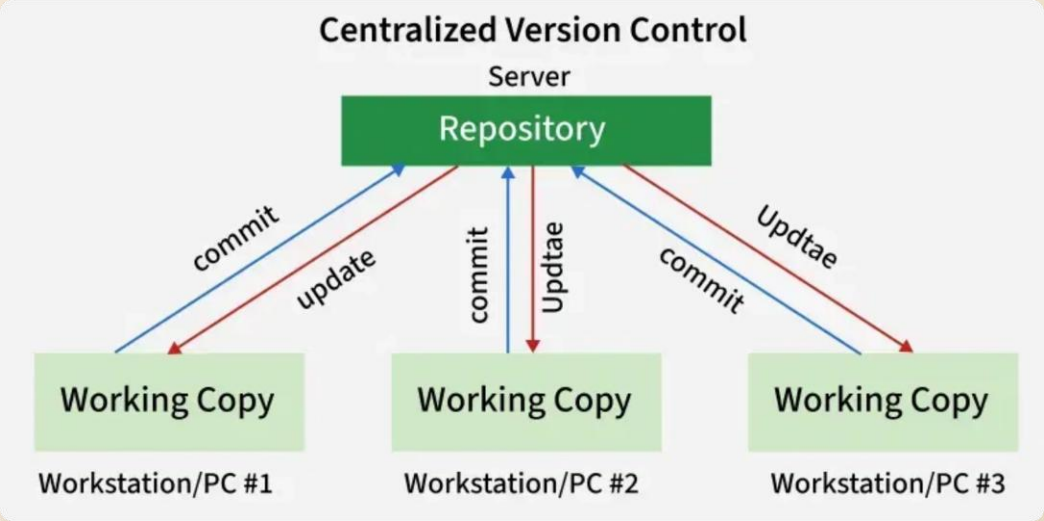
Le workflow centralisé est le plus simple à comprendre et à mettre en œuvre. Tous les développeurs travaillent sur une seule branche principale (généralement `main` ou `master`).



Note : Ce workflow est simple mais présente des limitations : risques de conflits fréquents et difficulté à gérer plusieurs fonctionnalités en parallèle. Recommandé uniquement pour les petites équipes ou les projets simples.

Git en Action : Feature Branch Workflow

Le **Feature Branch Workflow** est le modèle le plus recommandé pour les équipes.
Chaque nouvelle fonctionnalité est développée dans une branche dédiée, puis fusionnée dans la branche principale après Revue.



Avantages

- Isolation complète des fonctionnalités
- Facilite la revue de code
- Branche principale toujours stable
- Déploiement continu possible
- Historique de projet clair et organisé

Quelques commandes Git à connaître

git clone

Copie un dépôt distant sur votre machine locale.

```
git clone https://github.com/user/repo.git
```

git branch

Crée, liste ou supprime des branches.

```
git branch feature-login
```

git checkout

Change de branche ou restaure des fichiers.

```
git checkout feature-login
```

git add

Ajoute des fichiers à l'index pour le prochain commit.

```
git add index.html
```

git commit

Enregistre les modifications dans l'historique.

```
git commit -m "Ajout de la page de connexion"
```

git pull

Récupère les modifications du dépôt distant.

```
git pull origin main
```

Quelques commandes Git à connaître

git push

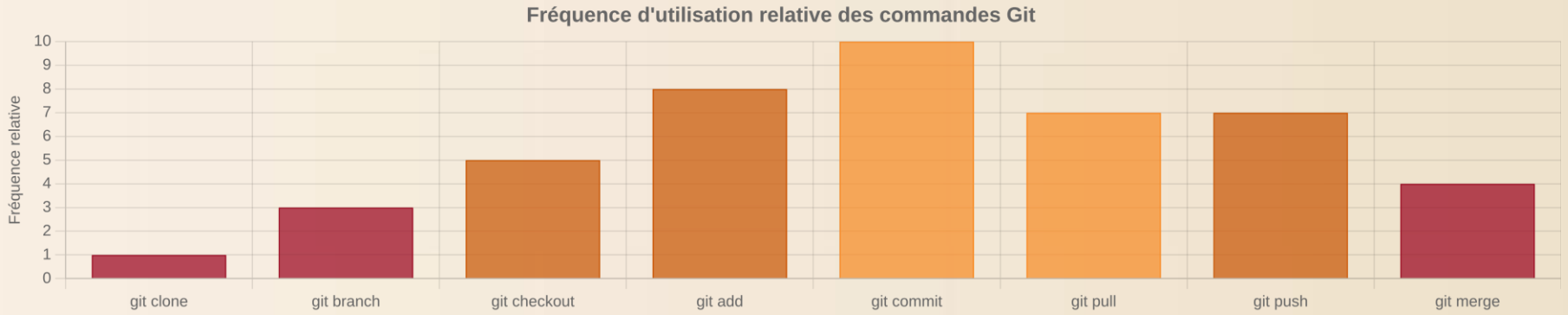
Envoie les commits locaux vers le dépôt distant.

```
git push origin feature-login
```

git merge

Fusionne une branche dans la branche courante.

```
git merge feature-login
```



Pour aller plus loin

Messages de commit clairs

Rédigez des messages descriptifs qui expliquent le "pourquoi" et non seulement le "quoi". Utilisez un format cohérent comme "type: sujet" (ex: "fix: correction du bug d'authentification").

Commits atomiques

Chaque commit doit représenter une seule modification logique et cohérente. Évitez de mélanger plusieurs fonctionnalités ou corrections dans un même commit.

Protégez la branche principale

Ne commitez jamais de code non fonctionnel sur la branche principale. Utilisez des branches de fonctionnalités et des pull requests avec revue de code.

Utilisez .gitignore

Configurez un fichier .gitignore pour exclure les fichiers non pertinents (fichiers temporaires, dépendances, configurations locales, etc.) du dépôt.

"Un bon historique Git est comme un bon livre d'histoire : il raconte clairement l'évolution du projet."

Synthèse

Les VCS sont indispensables

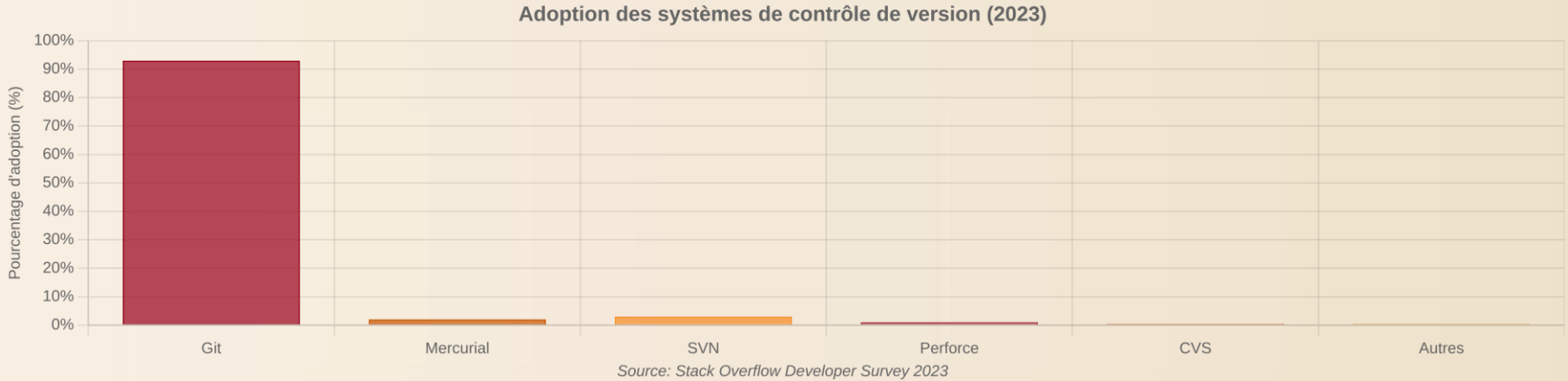
Les systèmes de contrôle de version sont devenus un outil fondamental pour tout projet de développement moderne, quelle que soit sa taille ou sa complexité.

Git comme standard

Git s'est imposé comme l'outil de facto dans l'industrie, mais les concepts fondamentaux sont transférables à d'autres systèmes de contrôle de version.

Workflow structuré

Une bonne gestion des branches et un workflow clair sont les clés du succès pour une collaboration efficace et un développement fluide.



"La maîtrise d'un système de contrôle de version n'est pas une compétence optionnelle, mais une nécessité pour tout développeur professionnel."

Questions & Discussion

Merci pour votre attention !

contact@example.com

github.com/example