GIT EXPERT

Maîtrise absolue du contrôle de version : Architecture interne, diagnostics, hooks et DevOps.

DISTRIBUTED VCS • CONTENT-ADDRESSABLE FS • HIGH SCALE

01. Architecture Interne

Voyage sous le capot du système de fichiers adressable par contenu.

VCS CENTRALISÉ VS DÉCENTRALISÉ

Modèle Centralisé (ex: SVN)

Dépendance directe d'un serveur unique pour l'historique.

⚠ Point de défaillance unique (SPOF) et requêtes lentes sur le réseau.

Modèle Décentralisé (Git)

Chaque clone local contient une base de données de l'historique complet.

✔ Opérations locales instantanées, résilience extrême et travail hors-ligne complet.

LES TROIS ZONES DE GIT

Working Directory

Votre espace de travail local avec les fichiers physiques. Les modifications y sont dans un état non suivi ou modifié.

Staging Area (Index)

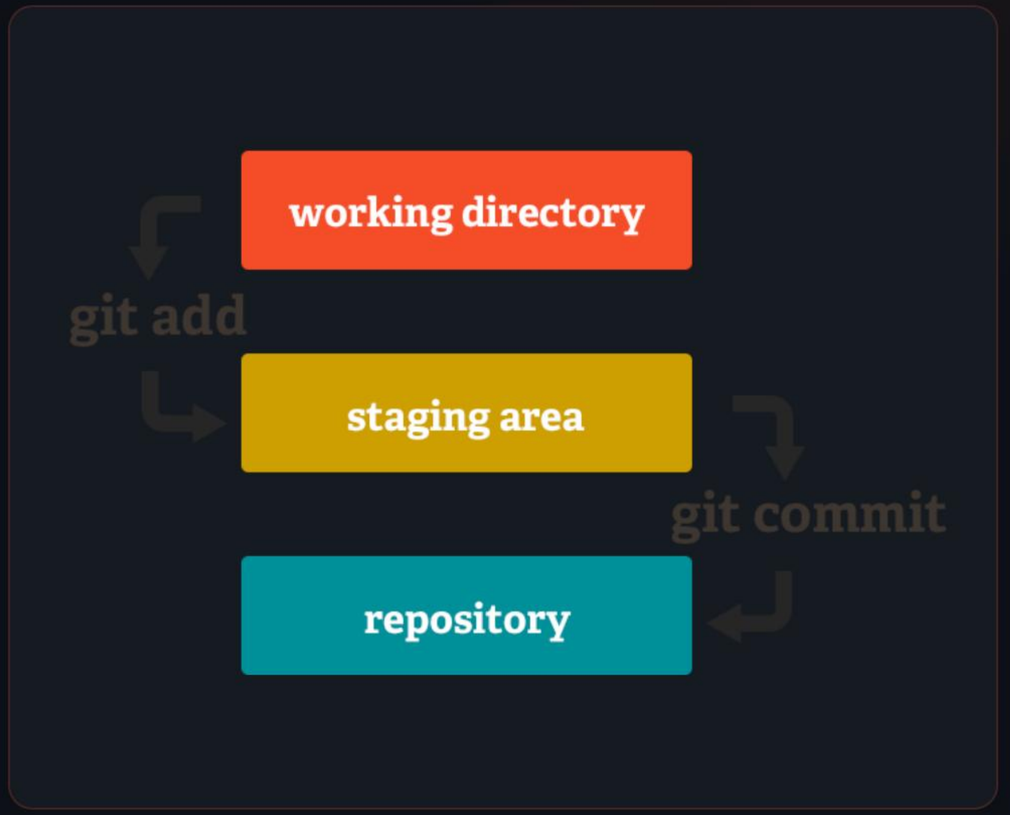
L'index de préparation. Un fichier binaire contenant l'arbre du prochain commit, servant de zone de transition.

Local Repository

La base de données interne de votre projet (dossier `.git/objects`) contenant l'historique immuable et validé.

LE CYCLE DE VIE DES FICHIERS

- **Untracked :** Fichier nouvellement créé, ignoré par Git.
- **Modified :** Fichier déjà suivi mais modifié localement.
- **Staged :** Fichier préparé dans l'index via la commande `git add`.
- **Committed :** Fichier définitivement encapsulé dans l'historique via `git commit`.



LA TRINITÉ DES OBJETS GIT

Blobs & Trees

Un **Blob** stocke le contenu brut d'un fichier (sans nom ni droits), adressé par son hash SHA-1.

Un **Tree** représente un dossier. Il référence d'autres Blobs et Trees avec leurs noms et permissions réelles.

Commits & Tags

Un **Commit** pointe vers un Tree racine. Il contient l'auteur, la date, le message et un pointeur vers le commit parent.

Les pointeurs de branches et de tags ne sont que des fichiers textes légers contenant des empreintes SHA-1.

02. Branches & Workflows

Organiser la collaboration d'équipe en production sans frictions.

GIT FLOW VS GITHUB FLOW

Git Flow (Classique/Lourd)

Structure rigide basée sur 5 types de branches : `main`, `develop`, `feature/*`, `release/*`, `hotfix/*`.

🚢 Idéal pour les cycles de livraison longs et releases planifiées.

GitHub Flow (Agile/Léger)

Modèle simplifié : une branche `main` stable en production, des branches éphémères de fonctionnalités, fusionnées via Pull Request.

🚚 Idéal pour l'intégration et le déploiement continus (CI/CD).

LE CYCLE D'UNE PULL REQUEST

1. Isolation

Création d'une branche locale propre à partir de la branche principale stable.

2. Commit

Ajouts itératifs et structurés avec des commits atomiques explicites.

3. Review

Publication sur le serveur, revue de code par les pairs et exécution de la CI.

4. Merge

Fusion validée dans la branche principale, suivie du déploiement en production.

L'ART DU COMMIT CONVENTIONNEL

- **Sémantique Standardisée :** Utiliser le format `():` pour automatiser la génération du Changelog (ex: `feat(auth): add OAuth2 provider`).
- **Atomicité :** Un commit doit représenter un seul changement logique cohérent. Facilite le debug et le retour en arrière.
- **Message de corps :** Décrire le *pourquoi* et non le *comment* dans la description étendue du commit.

03. Diagnostics & Commandes Avancées

Manipuler l'historique et réparer les erreurs de manière chirurgicale.

MERGE VS REBASE : STRATÉGIES

Intégration via Merge

Combine les historiques en créant un "commit de fusion" (Merge Commit). Conserve la chronologie réelle des événements.

Peut encombrer l'arbre d'historique en équipe.

Intégration via Rebase

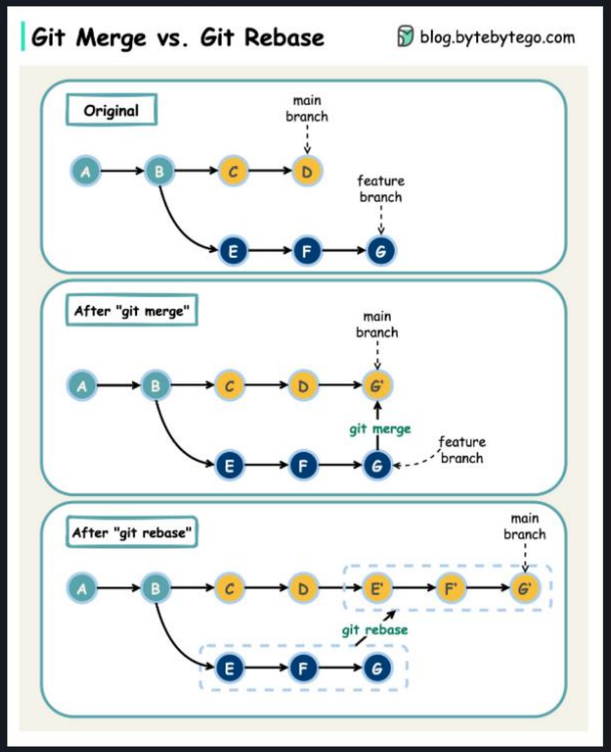
Déplace les commits locaux au sommet de la branche cible. Réécrit l'histoire pour un rendu parfaitement rectiligne.

Historique propre, mais interdit sur les branches publiques.

VISUALISATION DU REBASE

Le rebase débranche vos modifications locales temporairement, applique les nouveaux commits de la branche parente (origin/main), puis réapplique vos modifications par-dessus.

⚠ Règle d'or : Ne jamais appliquer de rebase sur des branches partagées avec d'autres développeurs au risque de désynchroniser leurs dépôts.



SAUVER SON CODE AVEC LE REFLOG

Le **Reflog** est la boîte noire locale de Git. Il enregistre chaque déplacement de la tête de lecture (HEAD), même les commits écrasés ou supprimés.

```
# Consulter l'historique complet des actions git reflog # Restaurer un commit perdu (ex: HEAD@{5}) git reset --hard HEAD@{5}
```

Le **Cherry-Pick** permet d'isoler un commit spécifique d'une autre branche et de l'appliquer proprement sur votre HEAD courante.

```
# Appliquer uniquement le commit spécifié git cherry-pick
```

GÉRER LES CONFLITS SANS PEUR

- **Comprendre la cause :** Un conflit survient lorsque deux commits modifient la même ligne d'un fichier et que Git ne peut décider de la priorité.
- **Configuration Rerere :** Activer l'enregistrement des résolutions de conflits pour éviter de résoudre deux fois la même erreur : `git config --global rerere.enabled true`.
- **Outils Visuels :** Privilégier un outil tridirectionnel (3-way merge tool) via `git mergetool` pour une intégration claire et sans erreur manuelle.

04. DevOps & Administration

Sécuriser, automatiser et administrer l'infrastructure Git en entreprise.

SERVEUR GIT PRIVÉ SOUS LINUX

Créer un serveur Git privé bare minimal sous Debian/Ubuntu pour un contrôle total de la propriété intellectuelle :

```
# Créer l'utilisateur système dédié sudo adduser git su - git # Initialiser un dépôt nu (Bare repository) mkdir project.git cd project.git git init --bare
```

Sécuriser les accès en interdisant le shell standard pour l'utilisateur git (uniquement via les clés SSH autorisées) :

```
# Restreindre l'utilisateur au shell Git sudo chsh -s /usr/bin/git-shell git # Cloner depuis le client distant git clone git@serveur-ip:project.git
```

GIT HOOKS : AUTOMATISATION LOCALE

Pre-Commit

Exécuté localement avant la validation. Parfait pour lancer des Linters, valider le formatage automatique, ou scanner des secrets (clés API) accidentels.

Pre-Push

Bloque l'envoi vers le serveur distant si les tests unitaires locaux échouent ou si la structure sémantique des commits n'est pas respectée.

Post-Receive

Script côté serveur exécuté après réception. Déclenche des webhooks de CI/CD, redémarre des serveurs applicatifs ou envoie des notifications d'équipe.

PLATEFORMES GIT EN ENTREPRISE

Solution	Hébergement	Fonctionnalités Clés	Ressources système
GitHub Enterprise	SaaS / Sur site	Écosystème Actions, sécurité avancée (Dependabot)	Négligeable (SaaS) / Élevée (On-Prem)
GitLab Self-Hosted	Sur site (On-Prem)	CI/CD intégrée extrêmement robuste, Container Registry	Élevée (Requis minimum 4 Go de RAM)
Gitea (Léger)	Sur site (On-Prem)	Interface ultra-rapide écrite en Go, empreinte minimale	Très Faible (Idéal pour micro-serveurs / IoT)

GIT LFS : FICHIERS VOLUMINEUX

LFS

LARGE FILE STORAGE

Éviter l'asphyxie du dépôt .git

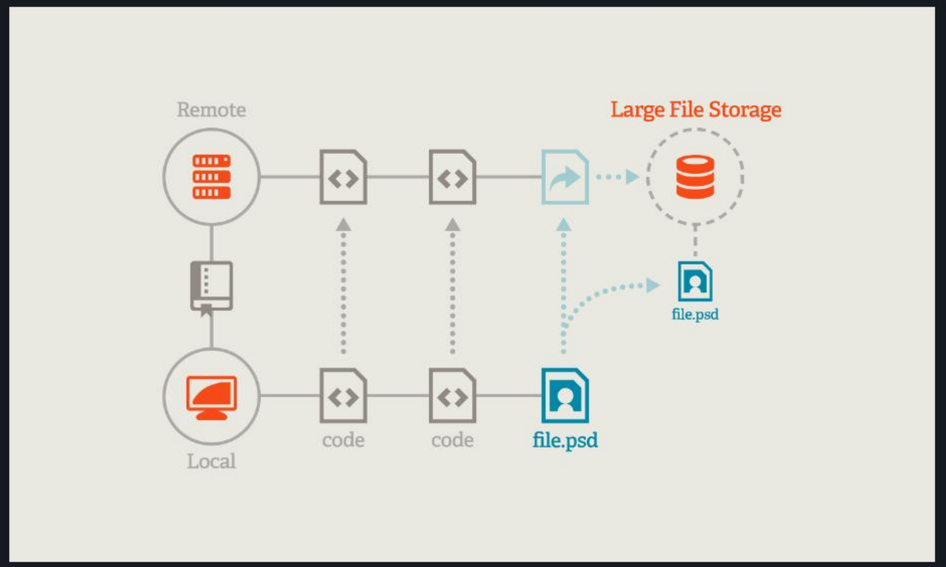
Git n'est pas conçu pour stocker l'historique d'images HD, de vidéos, ou de bases de données binaires lourdes.

Git LFS (Large File Storage) remplace ces fichiers par de simples pointeurs textuels légers de quelques octets dans le dépôt, déportant le stockage physique réel vers un stockage objet cloud.

ARCHITECTURE GIT LFS

Lors d'un clone ou d'un checkout, les fichiers d'images ou d'assets binaires sont téléchargés à la demande depuis un serveur de stockage Git LFS externe dédié.

- **Légèreté :** Clones et Pulls infiniment plus rapides.
- **Externalisation :** Stockage stable sur AWS S3 ou équivalent.



OPTIMISER LES MONOREPOS

Clone Partiel (Shallow Clone)

Permet de récupérer le projet en tronquant l'historique de commits à une profondeur donnée.

```
git clone --depth 1
```

Idéal pour les conteneurs éphémères de CI/CD.

Sparse Checkout

Permet de cloner un monorepo massif en ne matérialisant physiquement que certains répertoires ciblés.

```
git sparse-checkout set
```

Évite de surcharger les disques des développeurs.

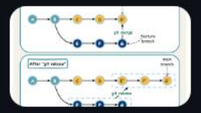
Questions ?

git-scm.com/doc | git@expert-dev.fr

IMAGE SOURCES

https://miro.medium.com/1*diRLm1S5hkVoh5qeArND0Q.png

Source: medium.com



https://assets.bytebytego.com/diagrams/0203-git-merge-git-rebase.jpg

Source: bytebytego.com

Thumbnail for git-lfs.com

https://git-lfs.com/images/tweet-promo.png

Source: git-lfs.com