Engineer

Git

gestion de versions qui conserve l'historique de ce site

Ce que c’est

Git est un système de gestion de versions distribué : un programme qui enregistre l’état d’un ensemble de fichiers sous forme d’une chaîne d’instantanés. Tout état antérieur peut ainsi être consulté, comparé au présent ou rétabli. Le projet écrit son nom Git et sa commande `git` ; les deux désignent le même outil.

Chaque instantané est un commit. Un commit conserve l’état de chaque fichier suivi à cet instant, l’auteur et la date du changement, un message expliquant pourquoi et un pointeur vers le ou les commits précédents. Git ne stocke qu’une fois un fichier déjà rencontré : lorsqu’il ne change pas entre deux commits, le nouvel instantané pointe vers la copie déjà sur le disque au lieu de l’écrire à nouveau. Un long historique coûte donc beaucoup moins de place qu’une copie complète par commit.

Tout est nommé d’après son contenu. Avant de stocker un fichier, la liste du contenu d’un répertoire ou un commit, Git calcule une somme de contrôle sur les octets : dans le format classique, un hachage SHA-1 écrit sur 40 caractères hexadécimaux. Cette somme devient le nom de l’objet. Un seul octet changé modifie le nom, ainsi que celui de chaque commit qui repose sur cet objet. C’est ainsi que Git détecte la corruption des données, et c’est pourquoi un identifiant de commit pointe encore vers exactement le même code des années plus tard.

Distribué signifie que chaque copie est complète. Cloner un dépôt récupère tout l’historique, pas seulement les derniers fichiers : lire le journal, comparer deux versions ou créer un commit se fait donc sur le disque local, sans serveur. Partager le travail est une étape distincte et délibérée : on pousse ses commits vers une autre copie, ou l’on récupère des commits depuis une autre copie pour

les fusionner. Toute copie peut servir de serveur ; dans la plupart des équipes, une copie choisie d'un commun accord, souvent chez un hébergeur, joue ce rôle par convention plutôt que par conception.

Une journée de travail passe par trois lieux. L'arbre de travail contient les fichiers que vous modifiez. La zone de préparation, aussi appelée index, contient ce qui figurera dans le prochain commit : elle permet de ne committer qu'une partie des changements et de garder le reste pour plus tard. Le répertoire Git, `.git` à la racine du projet, contient la base d'objets et toutes les références. Une branche n'est qu'un nom mobile pointant vers un commit, d'où sa création instantanée.

Git a été écrit en 2005 par Linus Torvalds et la communauté du noyau Linux, après la fin de la gratuité de BitKeeper, l'outil propriétaire qu'ils utilisaient. Leurs objectifs étaient la vitesse, une conception simple, une bonne prise en charge de milliers de branches parallèles, un fonctionnement entièrement distribué et la capacité de gérer un projet de la taille du noyau.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que Git ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des changements testés, publiés et réversibles, pour livrer aux clients un mardi après-midi plutôt qu'un vendredi soir tard.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Déploiement SaaS GIS automatisé : GitHub Actions — déploiement, données et reporting automatisés

Déploiement SaaS GIS automatisé : GitHub Actions Contactez-nous →

Pourquoi c'est important

La gestion de versions répond aux questions qui surviennent dès que plusieurs personnes, ou une seule pendant plus d'une semaine, touchent aux mêmes fichiers : qu'est-ce qui a changé, quand, par qui et pourquoi ? À la main, les réponses restent dans la mémoire, dans des noms comme `final-v3` et dans les fils de courriel. Avec Git, elles sont à côté du code, rattachées à chaque changement, où tout le monde peut les chercher.

La conception distribuée change le quotidien d'une équipe. Chaque clone contient tout l'historique : personne n'attend un serveur central pour lire le journal ou créer un commit, et une panne du serveur n'empêche pas de

poursuivre le travail. Les branches coûtent peu : on peut en créer une pour chaque fonctionnalité, correctif ou expérience, l'examiner séparément et la fusionner quand elle est prête, tout en gardant la ligne principale livrable.

Cette même conception rend Git prudent avec les données. Presque chaque opération ne fait qu'ajouter à la base d'objets : il est donc difficile de perdre du travail committé. On peut généralement récupérer une branche supprimée par erreur ou annuler un rebase qui a mal tourné ou une réinitialisation sur le mauvais commit. Le filet de sécurité est le reflog, un journal local des commits vers lesquels chaque branche pointait et des dates correspondantes. Sauf configuration contraire, il est conservé 90 jours pour les commits encore accessibles et 30 jours pour ceux qui ne le sont plus.

Git est aussi le socle commun de la plupart des autres outils d'une chaîne de livraison. Forges, outils de revue, systèmes de compilation, scripts de déploiement et générateurs de sites statiques le lisent tous. L'historique devient une source de faits bien au-delà du code : le jour de la dernière modification d'une page, le commit ayant servi à construire une version ou le changement qui a introduit un bug.

À quoi cela ressemble en pratique

Le code propre à cette entreprise en montre trois aspects.

Le code source résidait auparavant chez un hébergeur tiers. Il est maintenant sur un serveur Git que l'entreprise exploite sur son propre matériel : un seul binaire supervisé par le système d'exploitation, choisi pour son approche centrée sur SSH sans interface web d'administration. Clones et pushs passent par SSH vers le port 23231, qui n'écoute que sur l'interface de bouclage ; un client distant extérieur à l'hôte y accède donc par un hôte de rebond. Les sessions anonymes et sans clé sont refusées, et chaque dépôt déclaré est marqué privé. La page en expose clairement le prix : exploiter sa forge implique aussi d'assumer l'authentification, le contrôle d'accès, le stockage et les sauvegardes.

Des copies complètes de l'historique sur plusieurs machines peuvent facilement passer pour une sauvegarde. L'implémentation des sauvegardes et des tests de restauration de l'entreprise a été conçue pour étendre la récupération au-delà d'un instantané du fournisseur. Ses scripts restic et ses unités systemd prennent en charge un export de base de données lorsqu'il est configuré, des instantanés chiffrés et dédupliqués vers le stockage d'un autre fournisseur par SFTP, la

suppression selon les règles de conservation et les vérifications d'intégrité. Un processus mensuel distinct restaure un fichier connu dans un emplacement temporaire et compare ses octets. Ce test doit réellement s'exécuter avant de prouver une capacité de récupération : l'état d'implémentation daté du 2026-10-27 indique que les temporisateurs étaient désactivés faute de dépôt configuré et ne consigne aucune restauration achevée en conditions réelles. Quel que soit leur nombre, les clones sur des ordinateurs portables ne fournissent pas cette preuve manquante.

L'historique alimente aussi le site lui-même. La page des crédits indique que le W3C Feed Validator avertit que la plupart des entrées des flux partagent une date de modification, et que l'avertissement est juste : les dates viennent de l'historique Git, et la plupart de ces pages ont réellement été rédigées en une poignée de commits. Le site préfère afficher l'avertissement plutôt qu'inventer une date. L'historique dit ce qui s'est passé, y compris que beaucoup de choses se sont passées en même temps.

Une commande permet de relire un tel fait. Celle-ci affiche l'identifiant court, la date du commit au format ISO 8601 et l'objet du dernier commit qui a touché un fichier :

```
git log -1 --format='%h %cI %s' -- README.md
```

Où les problèmes surviennent

- **Un secret committé une fois reste dans l'historique.** Supprimer le fichier au commit suivant le retire du dernier instantané, mais le laisse dans le précédent, déjà présent dans chaque clone. Considérez le secret d'authentification comme exposé et renouvelez-le ; réécrire ensuite l'historique limite la propagation, mais les copies faites entre-temps conservent les anciens commits.
- **Un push forcé remplace le travail d'un autre.** `git push --force` désactive la vérification que la branche distante est un ancêtre de ce qui est poussé : les commits poussés entre-temps par un collègue disparaissent donc de la branche. `--force-with-lease` ne met à jour le dépôt distant que s'il pointe toujours vers le dernier état observé.

- **Les clones sont traités comme des sauvegardes.** Une copie sur un ordinateur portable est en retard, ne contient que les branches que son propriétaire a effectivement récupérées et n'a jamais servi délibérément à une restauration. Tickets, règles d'accès et autres données propres à une forge sont entièrement hors du dépôt.
- **Un gros fichier binaire alourdit chaque clone pour toujours.** Git garde chaque version de chaque fichier, et chaque clone les télécharge toutes. Une vidéo committée par accident puis supprimée le lendemain continue donc de voyager vers toute machine qui clone ensuite le projet.
- **Le travail gardé en un seul lieu disparaît avec lui.** Le reflog qui sauve une mauvaise réinitialisation réside sur une machine et expire. Les commits jamais poussés n'existent nulle part ailleurs, et une branche laissée de côté pendant des mois s'éloigne au point que sa réintégration devient un projet à part entière.

Pour aller plus loin

- Documentation Git — le manuel de référence, le livre et les vidéos du projet lui-même.
- Pro Git — le livre complet en ligne, du premier commit à la base d'objets sous-jacente.
- Qu'est-ce que Git ? — instantanés, opérations locales, sommes de contrôle et trois états dans un chapitre.
- Brève histoire de Git — pourquoi la communauté du noyau a écrit son propre outil en 2005 et ce qu'elle voulait accomplir.
- git-reflog — comment retrouver vers quoi pointait une branche avant une erreur et la durée de conservation de cette trace.
- git-push — les options de forçage et les protections propres à chacune.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que Git ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/git/