MÉMO TECHNIQUE

Gestion de versions · Forges · DevOps

Git, GitHub & GitLab

Comprendre l’outil, son écosystème et son rôle dans la CI/CD

Objet	Fiche de synthèse pédagogique
Public	Développeurs, DevOps, profils techniques
Auteur	OSD — M. Slimani
Version	v1.0 — 07/06/2026

1. Git, c'est quoi ?

Git est un logiciel libre de gestion de versions décentralisée. Sa tâche : suivre l'évolution du contenu d'une arborescence de fichiers dans le temps.

Concrètement, Git enregistre des **instantanés** (commits) de votre projet à chaque étape importante. Chaque commit garde une trace de qui a changé quoi, quand et pourquoi, et permet de revenir à n'importe quel état antérieur.

Créé en 2005 par Linus Torvalds pour le développement du noyau Linux, Git est devenu en quelques années le gestionnaire de sources le plus utilisé au monde. Il est **multiplateforme** et fonctionne entièrement en local.

> ℹ️ **Décentralisé = chacun a tout**
>
> Contrairement aux outils centralisés (comme SVN), chaque développeur possède une **copie complète** du dépôt et de son historique. On peut travailler, committer et consulter l'histoire **sans connexion réseau** ; la synchronisation avec les autres se fait quand on le décide.

Git repose sur trois zones mémorisables : le **répertoire de travail** (vos fichiers), l'**index / staging** (ce qui est prêt à être validé) et le **dépôt** (l'historique des commits).

2. Pourquoi Git ?

Git répond à des besoins concrets du développement logiciel :

- **Traçabilité** : un historique complet et horodaté de chaque modification.
- **Filet de sécurité** : revenir en arrière, comparer, retrouver l'origine d'un bug.
- **Travail en parallèle** : les **branches** isolent les fonctionnalités sans bloquer le reste de l'équipe.
- **Collaboration** : plusieurs personnes travaillent sur le même projet et fusionnent leurs apports.
- **Performance & robustesse** : rapide, fiable, et intègre les données par empreintes cryptographiques (SHA-1/SHA-256).
- **Standard du marché** : maîtriser Git est aujourd'hui un prérequis professionnel.

> ✓ **L'idée clé**
>
> Git ne se contente pas de « sauvegarder » : il transforme l'historique du code en un outil de travail — pour comprendre, expérimenter et collaborer sans peur de tout casser.

3. L'écosystème : GitHub & GitLab

Git est l'outil. **GitHub** et **GitLab** sont des **forges logicielles** : des plateformes web qui hébergent des dépôts Git distants et ajoutent une couche de collaboration (revue de code, suivi des tâches, automatisation…). On peut utiliser Git sans elles, mais elles démultiplient sa valeur en équipe.

GitHub

Lancé en 2008 et racheté par Microsoft en 2018, **GitHub** est la plus grande plateforme d'hébergement de code au monde. C'est le cœur de l'open source : communauté immense, écosystème d'intégrations très riche, et **GitHub Actions** pour l'automatisation (CI/CD). La collaboration s'articule autour des **pull requests**.

GitLab

Créé en 2011, **GitLab** se positionne comme une **plateforme DevOps intégrée** couvrant tout le cycle de vie applicatif. Sa CI/CD native est l'une de ses forces historiques. Point distinctif : GitLab peut être **auto-hébergé** facilement (Omnibus, Docker, cloud), ce qui séduit les organisations soucieuses de maîtriser leur infrastructure. La collaboration passe par les **merge requests**.

4. GitHub vs GitLab : les différences

Les deux reposent sur **le même Git** : le code et les commandes sont identiques. Les différences portent sur la plateforme qui entoure le dépôt.

Critère	GitHub	GitLab
Proposition	Écosystème & communauté open source	Plateforme DevOps tout-en-un
Demande de fusion	Pull Request (PR)	Merge Request (MR)
CI/CD native	GitHub Actions	GitLab CI/CD (.gitlab-ci.yml)
Auto-hébergement	GitHub Enterprise Server	Natif et répandu (CE/EE)
Point fort	Visibilité, intégrations, adoption	Intégration du cycle complet, contrôle
Usage typique	Projets open source, portfolios	Équipes/entreprises, chaîne DevOps interne

> ℹ️ **Comment choisir ?**
>
> Pour la visibilité open source et un écosystème maximal → **GitHub**. Pour une chaîne DevOps intégrée et/ou de l'auto-hébergement → **GitLab**. Dans les deux cas, les compétences Git acquises restent les mêmes et sont transférables.

5. L'utilité de Git dans un processus DevOps / CI/CD

Dans une démarche DevOps, Git n'est pas qu'un coffre à code : il est le déclencheur de toute la chaîne d'automatisation.

La **CI/CD** (Intégration Continue / Déploiement Continu) consiste à automatiser la construction, les tests et la mise en production du logiciel. Le **point de départ** de cette automatisation est presque toujours un **événement Git** : un `push`, l'ouverture d'une merge/pull request, ou la création d'un tag.

Le dépôt comme source unique de vérité

- Le code, mais aussi la configuration et les pipelines (`.gitlab-ci.yml`, workflows) vivent dans Git : c'est l'approche **« everything as code »**.
- Chaque changement est traçable et réversible : on sait exactement quelle version est déployée.
- Les branches structurent les environnements (ex. `develop` → recette, `main` → production).

Le commit qui déclenche le pipeline

Un cycle CI/CD typique déclenché par Git :

1. Un développeur **pousse** un commit ou ouvre une merge request.
2. La forge déclenche automatiquement un **pipeline** (build).
3. Les **tests** automatisés s'exécutent ; un échec bloque la fusion.
4. Après revue et fusion, l'artefact est **déployé** (staging puis production).
5. Un **tag** de version (`v1.0`) marque une release traçable.

.GITLAB-CI.YML

```
# Exemple minimal de pipeline déclenché par un push (GitLab CI)
stages: [build, test, deploy]

build:
  stage: build
  script: [ "npm ci", "npm run build" ]

test:
  stage: test
  script: [ "npm test" ]

deploy:
  stage: deploy
  script: [ "./deploy.sh" ]
  only: [ main ]        # ne déploie que depuis main
```

✓ **Bonnes pratiques DevOps autour de Git**

Branches **protégées** (pas de push direct sur main), **revue de code** obligatoire via MR/PR, **messages de commit normalisés** (Conventional Commits), et tests verts exigés avant fusion. Ces règles rendent la CI/CD fiable.

En résumé

Git fournit la **traçabilité**, la **collaboration** et le **déclencheur** ; la forge (GitHub/GitLab) fournit la **plateforme** et l'**automatisation**. Ensemble, ils forment la colonne vertébrale d'une chaîne DevOps moderne : du commit local jusqu'à la production, sans rupture.