

**Ckevin – Anis – Cedric**

# Versionning avec GIT

### Projet final - Rapport

Table des matières

[Versionning avec GIT 1](file:///C:\Users\GUEULE%20DANGE\Documents\docs.docx#_Toc189151710)

[Projet final - Rapport 1](file:///C:\Users\GUEULE%20DANGE\Documents\docs.docx#_Toc189151711)

[Introduction 3](file:///C:\Users\GUEULE%20DANGE\Documents\docs.docx#_Toc189151712)

[I - Historique et utilité des VCS 4](#_Toc189151713)

[I.1. Définition d’un système de gestion de versions (VCS) 4](#_Toc189151714)

[I.2. Explication et différence entre VCS centralisé (SVN) et décentralisé (Git) 4](#_Toc189151715)

[I.3. Évolution des VCS 5](#_Toc189151716)

[II - Principes fondamentaux de Git 7](#_Toc189151717)

[II.1 Les bases de Git 7](#_Toc189151718)

[II.2 Les commandes de base 7](#_Toc189151719)

[II.3 Fonctionnalités avancées de Git 8](#_Toc189151720)

[III - Présentation de GitHub et ses alternatives 9](#_Toc189151721)

[Conclusion 11](#_Toc189151722)



# Introduction

**Le versioning (ou gestion de versions) est un processus qui permet de suivre, gérer et conserver un historique des modifications apportées à un fichier ou un ensemble de fichiers dans un ordre chronologique. Il est principalement utilisé en développement logiciel, mais aussi dans d'autres domaines nécessitant un suivi rigoureux des modifications (documents, bases de données, etc.).**

**Imaginez que vous travaillez sur une nouvelle fonctionnalité de votre projet depuis des heures, totalement concentré sur votre code. Tout semble bien se passer… Jusqu’au moment où vous testez une autre partie de votre application et là, catastrophe : ça ne fonctionne plus ! Mais qu’est-ce qui s’est passé ?!**

**À quel moment ça a commencé à bugger ?!"**

**Difficile de le savoir si vous n’avez pas pris l’habitude de sauvegarder votre travail à chaque étape. C’est là que versionning entre en jeu : un outil indispensable qui permet d’enregistrer différentes versions de votre code, de suivre chaque modification et même de revenir à un état stable si un problème survient.**

# I - Historique et utilité des VCS

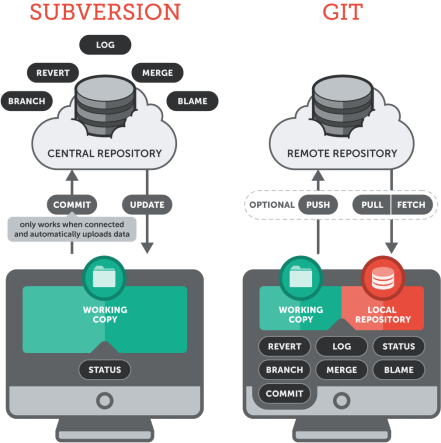
### I.1. Définition d’un système de gestion de versions (VCS)

Les logiciels de gestion de version (VCS) permettent aux équipes de sauvegarder et d'archiver le code source de leurs projets. Cette solution permet d'effectuer des révisions et des modifications dans le dépôt plus facilement ou de restaurer des anciennes versions si une erreur de conception survient.

Les outils de versionning permettent de gérer les modifications de fichiers en enregistrant chaque version avec horodatage et identifiant unique. Ils offrent la possibilité de restaurer d’anciennes versions et d’identifier les auteurs des modifications. Leur principal objectif est de faciliter le travail collaboratif en assurant un accès coordonné aux fichiers et en permettant le développement simultané via des branches distinctes. Ces systèmes sont largement utilisés en développement logiciel et en gestion de contenu. Parmi les plus connus figurent Apache Subversion (SVN) et Git.

### I.2. Explication et différence entre VCS centralisé (SVN) et décentralisé (Git)

On trouve deux grands types de VCS, les gestionnaires de versions centralisées ou décentralisées, voyons en détail les deux : Les VCS centralisés : ils sont caractérisés par un dépôt privilégié géré par un serveur. Chaque développeur travaille sur une copie et synchronise ses évolutions avec le dépôt central, le serveur détectant les conflits. Subversion (SVN) fonctionne sur ce principe. Les VCS décentralisés : contrairement au VCS centralisé, il n’impose pas techniquement de dépôt de référence. Chaque développeur travaille avec son dépôt qu’il synchronise avec ceux des autres et selon le modèle de développement, avec un dépôt de référence. Git fonctionne de façon décentralisée. En d’autres termes, imaginez, vous avez un dépôt GIT en local, vous partez en week-end chez Mamie (désolé pour l’exemple :3) mais malheureusement, cette dernière rechigne toujours à se mettre à internet. Avec git vous allez pouvoir commiter vos changements et les push sur votre dépôt local, pouvant commencer alors une nouvelle fonctionnalité sans tout casser. Au retour du week-end, vous allez pouvoir merge le tout sur le dépôt distant.



Comment fonctionne SVN (à gauche) ? Dépôt central unique : Toutes les versions du projet sont stockées dans un serveur central. Travail en réseau : Un utilisateur doit être connecté pour committer (envoyer ses modifications) ou mettre à jour son code. Dépendance totale au serveur : Si le serveur tombe en panne, personne ne peut travailler ni récupérer les versions précédentes. Les commits sont directement envoyés sur le dépôt central sans gestion locale.

Comment fonctionne Git (à droite) ?

Dépôt local + dépôt distant : Chaque utilisateur a une copie complète du dépôt, y compris l’historique des modifications. Travail hors ligne possible : On peut committer et gérer les versions localement sans être connecté. Synchronisation manuelle : Les commandes push, pull et fetch permettent de choisir quand envoyer ou récupérer les modifications sur le dépôt distant. Plus rapide et plus flexible : Les utilisateurs peuvent créer des branches, fusionner, revenir à d’anciennes versions sans dépendre d’un serveur central.

### I.3. Évolution des VCS

* CVS (Concurrent Versions System) – Années 1980
* Un des premiers systèmes de gestion de versions largement adoptés.
* Repose sur un modèle centralisé, où un serveur unique stocke l’ensemble du code et ses modifications.
* Permet à plusieurs utilisateurs de travailler simultanément sur un même projet.
* Stocke uniquement les différences entre chaque version pour économiser de l’espace.
* Subversion (SVN) – 2000
* Conçu comme une amélioration de CVS avec une gestion plus efficace des versions et des branches.
* Système centralisé, mais avec des fonctionnalités avancées.
* Permet le versionnement des répertoires et des fichiers individuellement.
* Meilleure gestion des branches et fusions que CVS.
* Git – Créé en 2005 par Linus Torvalds
* Conçu par Linus Torvalds (créateur de Linux), Git est un système décentralisé qui corrige les défauts des VCS centralisés en offrant plus de flexibilité et de robustesse.
* Chaque utilisateur possède une copie complète du dépôt, permettant de travailler hors ligne.
* Gestion avancée des branches et fusions, simplifiant le travail collaboratif.
* Sécurisé : chaque commit est identifié de manière unique grâce au hachage SHA-
* Limites :

SVN

* Toujours dépendant d’un serveur central (risque de panne unique).
* Impossible de travailler hors ligne.
* Moins flexible que les systèmes distribués.

CVS

* Gestion des branches limitée et complexe.
* Risque de perte de données en cas de panne du serveur central.
* Faible support pour la gestion des conflits.
* Pourquoi Git a remplacé SVN et CVS ?
* Pas de point de défaillance unique.
* Meilleure gestion des conflits et des modifications simultanées.
* Possibilité de travailler en local et de pousser les modifications plus tard.
* Git est aujourd’hui le standard de l’industrie et est utilisé par des millions de développeurs à travers le monde.

# II - Principes fondamentaux de Git

### II.1 Les bases de Git

Git est un logiciel de gestion de versions à destination des développeurs. S’il est envisageable de se passer d’un tel logiciel, celui-ci pourrait bien vous tirer de quelques misères, particulièrement si vous travaillez en équipe. Au cours de cet article, nous allons voir comment utiliser Git à l’aide de scénarios pour que cela soit le plus concret possible. Ce que Git va faire pour vous : stocker le code de votre projet sur un serveur en ligne et ainsi vous permettre de reprendre ce projet depuis n’importe quel ordinateur conserver les anciennes versions de chaque fichier du projet vous permettre de travailler à plusieurs dans le code du projet en étant certain de ne pas écraser le code des autres vous permettre de savoir qui a fait telle ou telle modification

### II.2 Les commandes de base

Ces deux commandes permettent de configurer l’identité de l’utilisateur pour Git



* Nom utilisateur qui apparaîtra dans les commits.
* l’adresse email associée aux commits.



* Ajouter le(s) fichier(s) modifié(s) à la liste des fichiers qui seront envoyés au serveur avec la commande suivante



* Initialiser un nouveau dépôt Git dans un projet
* permet de récupérer les dernières modifications du dépôt distant (origin) et de les fusionner avec la branche master en local.
* Validez les changements et ajoutez un petit message explicatif
* Envoyez les changements sur le serveur :



* Ajouter un dépôt distant à un projet Git local.



* Gérer les branches dans un dépôt Git. Les branches permettent de travailler sur différentes versions du projet sans affecter directement la branche principale (main ou master).



* Changer de branche ou de restaurer une version précédente d’un fichier ou d’un commit dans un dépôt Git.



* Fusionner une branche dans une autre. Elle est utilisée pour intégrer les modifications d’une branche secondaire dans la branche principale



* Afficher l'état actuel du dépôt Git.

permet de voir les fichiers modifiés, ajoutés, en attente de commit

### II.3 Fonctionnalités avancées de Git

Voici un tableau bien structuré que vous pouvez insérer dans Word :

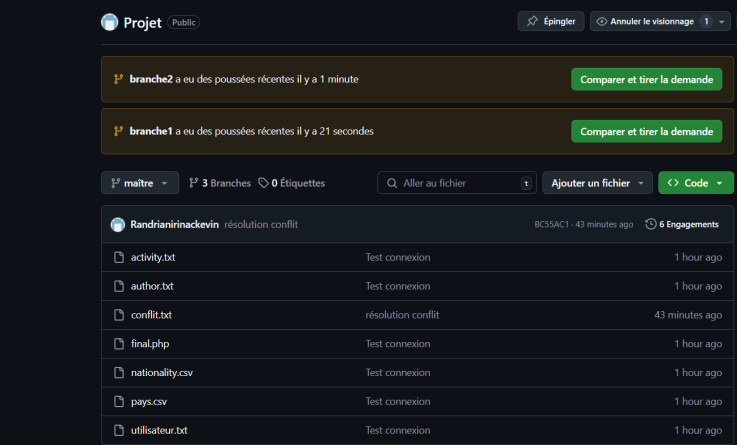
|  |  |  |
| --- | --- | --- |
| **Fonctionnalité** | **Description** | **Commande(s) associée(s)** |
| **Rebasing** | Intègre les commits d'une branche dans une autre sans commit de fusion, maintenant un historique linéaire. | git rebase,  git rebase -i |
| **Reverting, Resetting et Unstaging** | Permet d'annuler ou de modifier des commits et des changements dans le dépôt. | git revert, git reset, git restore |
| **Cherry-picking** | Applique un commit spécifique d'une branche à une autre, utile pour récupérer des modifications ciblées. | git cherry-pick <commit\_id> |
| **Stashing** | Sauvegarde temporairement les modifications en cours sans les committer, puis les restaure plus tard. | git stash, git stash apply, git stash pop |
| **Bisecting** | Identifie un commit précis introduisant un bug via une recherche binaire dans l'historique. | git bisect start, git bisect good, git bisect bad |
| **Blaming** | Affiche quel commit et quel auteur ont modifié chaque ligne d’un fichier. | git blame <fichier> |
| **Tagging** | Marque des points spécifiques dans l’historique du dépôt, souvent pour indiquer une version stable. | git tag <nom\_du\_tag> |
| **Submodules** | Intègre et gère des dépôts Git externes dans un dépôt principal. | git submodule add <URL>git submodule update --init |
| **Hooks** | Exécute automatiquement des scripts à des moments précis du cycle de vie Git (pré-commit, post-merge, etc.). | Localisés dans .git/hooks |
| **Différences avancées** | Compare les différences entre branches, commits ou fichiers pour analyser les modifications. | git diff, git diff --staged |
| **Archivage** | Crée une archive compressée de l'état actuel du projet, utile pour les sauvegardes ou déploiements. | git archive --format=tar HEAD > archive.tar |
| **Récupération de commits perdus** | Restaure des commits ou des branches supprimés accidentellement via le journal des références. | git reflog, git reset --hard HEAD@{n} |

Haut du formulaire

# III - **Présentation de GitHub et ses alternatives**

GitHub est une plateforme d’hébergement de référentiels Git basée sur le cloud, facilitant la gestion de versions et la collaboration en équipe. Son interface conviviale permet aux débutants d’utiliser Git sans maîtriser la ligne de commande. Très populaire pour les projets open-source, GitHub offre l’hébergement gratuit de dépôts publics. L’entreprise se finance en proposant des dépôts privés payants et des solutions avancées pour les entreprises, incluant la gestion des équipes et la sécurité.

GitHub héberge plus de 100 millions de dépôts, dont la majorité sont des projets open-source. Cette statistique montre que GitHub fait partie des clients GUI Git les plus populaires et est utilisé par divers professionnels et grandes entreprises, comme Hostinger.



Voici un aperçu de l'interface GitHub : Ici, vous pouvez voir les différentes branches sur lesquelles on travaille, ainsi que quand quelqu’un a fait un commit (c’est un peu comme « enregistrer » un fichier). Selon la façon dont un référentiel est configuré, vous pouvez également créer votre propre branche et y effectuer vos propres commits. Et une fois que vous avez fait quelques changements, vous pouvez soumettre ce code à une branche en faisant une pull request. Cela consiste essentiellement à demander à la personne responsable d’inclure votre code. Et cela aide aussi cette personne à voir exactement ce que vous avez changé dans le code.

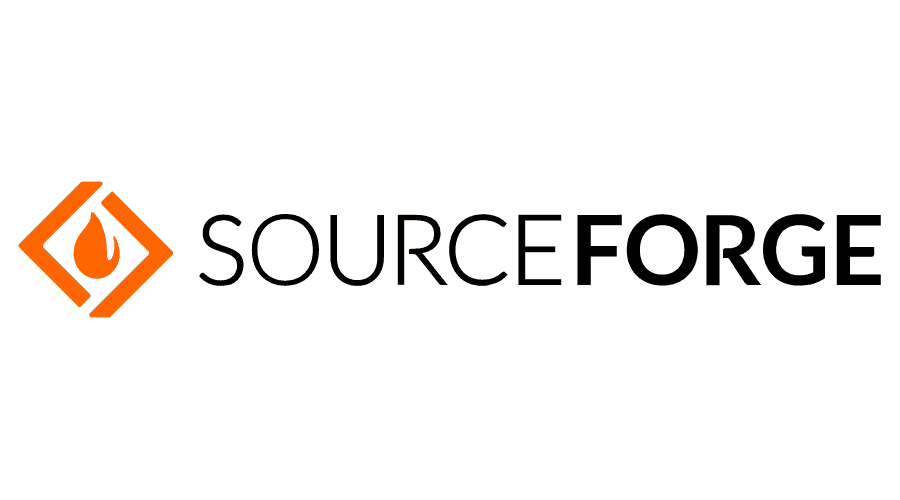


* 1. **GitLab**

Si vous cherchez à remplacer GitHub, GitLab est sans doute la meilleure option. Cet outil open source est utilisé par des entreprises mondialement connues telles que la NASA, Bayer, ING ou encore Sony. En termes d’utilisation, GitLab est très proche de GitHub et vous permet de migrer facilement votre code de cette dernière plateforme. Parmi les nombreuses fonctionnalités proposées, retenez que vous pouvez collaborer avec d’autres membres et vérifier la qualité de votre code. De nombreuses données vous sont fournies pour vous aider à mieux gérer votre projet et sécuriser votre code.

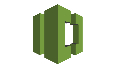
* 1. **BitBucket**

Sur BitBucket, vous apprécierez l’interface intuitive et le dépôt de référentiels privés illimités et gratuits. La possibilité de donner des permissions aux utilisateurs s’ajoute aux demandes d’extraction et de révision du code. Cette application de gestion de code source peut être une bonne alternative à GitHub pour les grandes entreprises : hébergée par Atlassian, BitBucket peut être intégré à d’autres outils de l’éditeur, notamment Jira, Bamboo et Confluence.

* 1. **SourceForge**

Avec ses 33 millions d’utilisateurs, SourceForge est l’outil open source de gestion de code le plus apprécié dans la communauté Linux. Le point fort de cet outil ? La possibilité de créer en illimité des projets open source, et ce gratuitement. D’autres fonctionnalités sont à découvrir. Vous pouvez fusionner des requêtes d’extraction et profiter d’un répertoire de code intégré. Des données statistiques vous sont communiquées sur les systèmes d’exploitation et les emplacements des personnes ayant téléchargé votre code.

* 1. **Cloud Source**

****Hé oui, Google a lui aussi son propre gestionnaire de code. Compatible avec GitHub et BitBucket, vous pouvez dupliquer votre code sur Cloud Source pour bénéficier des fonctionnalités proposées. L’avantage de cet outil est la possibilité de profiter des autres outils de GCP (Google Cloud Platform) comme Google App Engine. Grâce à Cloud Source, vous pouvez aussi effectuer une recherche rapide de code via le navigateur source.

* 1. **AWS CodeCommit**

Comme Google, Amazon possède sa propre solution pour héberger des référentiels Git privés. AWS CodeCommit s’avère être une bonne alternative à GitHub, surtout si vous utilisez les autres services d’Amazon Web Services. Avec cet outil votre code est entre de bonnes mains puisqu’il est stocké dans le AWS Cloud. Grâce à AWS Identity and Access Management, vous pouvez aussi connaître qui accède à votre dépôt, et pourquoi. AWS CodeCommit facilite la collaboration entre les utilisateurs et détient toutes les fonctionnalités que vous pouvez attendre d’une alternative à GitHub. Vous pouvez par exemple créer et fusionner des demandes d’extraction, créer des révisions de code et suivre les différents problèmes.

# Conclusion

Git est bien plus qu’un simple outil de gestion de versions : il est devenu un pilier essentiel du développement collaboratif. Que ce soit en entreprise ou dans l’open-source, il permet d’organiser le code, de suivre son évolution et de travailler efficacement en équipe.

Grâce à des workflows comme GitFlow, les développeurs peuvent structurer leur travail, gérer les branches et assurer la stabilité du projet. Des plateformes comme GitHub facilitent encore davantage cette collaboration en intégrant des fonctionnalités de revue de code, de pull requests et de gestion de projets.

En maîtrisant ses fonctionnalités avancées (rebase, cherry-pick, stash, bisect...), un développeur gagne en productivité, évite les erreurs et garantit la pérennité de son code. Aujourd’hui, Git est indispensable pour quiconque souhaite évoluer dans le monde du développement logiciel avec rigueur et professionnalisme.