Versionning avec Git - Projet Final

Rapport écrit sur les Systèmes de Gestion de Versions et Git

PRÉSENTÉ PAR :
UGO VILLERET
DUC PHUNG MAI

Sommaire

Concept des branches

Gestion et fusion des branches

3.3 Commits

Structure d’un commit

Importance des messages de commit

3.4 Fusion (Merge)

Types de merge

Gestion des conflits

4. Fonctionnalités Avancées de Git

4.1 Rebasing

Différences entre merge et rebase

Avantages et inconvénients du rebase

4.2 Cherry-Pick

Sélection et application de commits spécifiques

4.3 Stash

Sauvegarde temporaire des modifications

Gestion et récupération des stashes

4.4 Hooks

Automatisation avec les hooks Git

5. Présentation de GitHub et de ses Alternatives

Fonctionnalités principales de GitHub

Comparaison avec GitLab et Bitbucket

6. Cas d'Utilisation dans des Projets Collaboratifs

Développement open source

Gestion d’équipes de développement

Intégration et déploiement continus (CI/CD)

7. Conclusion

Impact des VCS sur le développement moderne

Importance de la maîtrise de Git

Perspectives d’évolution des systèmes de gestion de versions

Introduction

Les systèmes de gestion de versions (VCS) sont des outils essentiels dans le développement logiciel. Ils permettent de suivre les modifications apportées au code source, de collaborer efficacement et de gérer les différentes versions d'un projet. Ces outils jouent un rôle crucial dans la traçabilité des changements, la gestion des conflits et la coordination des équipes de développement.

Dans un contexte de développement agile, les VCS facilitent l'intégration continue et l'automatisation des déploiements, rendant le cycle de vie logiciel plus fluide et plus efficace. Ce rapport explore l'historique des VCS, les principes fondamentaux de Git, ses fonctionnalités avancées, ainsi que les plateformes populaires comme GitHub et ses alternatives. Nous examinerons également des cas d'utilisation concrets dans des projets collaboratifs.

1. Historique et Utilité des Systèmes de Gestion de Versions (VCS)

1.1 Historique

Les systèmes de gestion de versions (VCS) sont devenus un élément incontournable du développement logiciel moderne. Leur évolution a été marquée par plusieurs étapes clés, répondant aux besoins croissants des développeurs et des équipes de projet.

Les premières solutions centralisées Dans les années 1970, la gestion des versions était une tâche manuelle et fastidieuse. Les développeurs devaient conserver différentes copies de leurs fichiers sources en ajoutant des suffixes ou en utilisant des scripts rudimentaires pour enregistrer les modifications. Cette approche était sujette aux erreurs et compliquait la collaboration.

Pour répondre à ces difficultés, les premiers systèmes de gestion de versions ont vu le jour. Le SCCS (Source Code Control System), développé par Bell Labs en 1972, a été l'un des premiers outils formels permettant de suivre les modifications des fichiers individuels. Peu après, le RCS (Revision Control System), créé en 1982, a apporté des améliorations en stockant efficacement les différences entre les versions.

L'essor des systèmes centralisés collaboratifs Dans les années 1980 et 1990, avec la complexification des projets informatiques, il devenait essentiel de permettre à plusieurs développeurs de travailler simultanément sur un même projet. Le CVS (Concurrent Versions System) a été introduit pour répondre à ce besoin. Ce système permettait de centraliser le code sur un serveur et d'assurer une gestion efficace des modifications simultanées.

Dans les années 2000, Subversion (SVN) a remplacé CVS en offrant une meilleure gestion des répertoires et des métadonnées. SVN a également introduit la gestion des commits atomiques, garantissant que chaque modification est enregistrée de manière cohérente.

L'avènement des systèmes distribués Les limites des systèmes centralisés, notamment les problèmes de performance et de dépendance à un serveur unique, ont conduit à l'émergence des systèmes de gestion de versions distribués (DVCS). Git, créé par Linus Torvalds en 2005 pour le développement du noyau Linux, est devenu la référence dans ce domaine. D'autres solutions comme Mercurial ont également vu le jour, mettant l'accent sur la simplicité d'utilisation et la robustesse.

Aujourd'hui, Git est le système de gestion de versions le plus utilisé, notamment grâce à son intégration avec des plateformes comme GitHub, GitLab et Bitbucket, facilitant ainsi la collaboration et le déploiement automatisé.

1.2 Utilité

Les VCS offrent de nombreux avantages qui améliorent la qualité du développement logiciel et la collaboration entre les équipes. Voici les principaux bénéfices qu'ils apportent :

1. Suivi des modifications L'un des principaux atouts des VCS est la capacité de suivre chaque modification apportée au code source. Chaque commit enregistre un instantané du projet, permettant de revenir à une version précédente en cas de problème. Cela permet une meilleure traçabilité et une gestion plus efficace des erreurs.

2. Collaboration efficace Les systèmes de gestion de versions permettent à plusieurs développeurs de travailler simultanément sur un même projet sans

interférence. Les branches facilitent le travail sur de nouvelles fonctionnalités ou corrections de bugs en toute indépendance avant d'être fusionnées avec la version principale.

3. Gestion avancée des versions Avec les VCS, il est possible de créer et de gérer des versions parallèles d'un projet. Cela permet aux équipes de travailler sur plusieurs versions en même temps, par exemple en maintenant une version stable pendant que de nouvelles fonctionnalités sont développées.

4. Résolution des conflits Les VCS intègrent des outils de fusion et de résolution des conflits, facilitant l'intégration des modifications effectuées par différents développeurs. Cela assure une gestion fluide des contributions multiples.

5. Automatisation des tests et du déploiement L'utilisation des VCS s'intègre parfaitement avec les processus d'intégration et de déploiement continus (CI/CD). Des outils comme Jenkins, GitHub Actions ou GitLab CI permettent d'automatiser les tests, la vérification de la qualité du code et le déploiement sur les environnements de production.

6. Sécurité et sauvegarde Les systèmes distribués comme Git garantissent que chaque développeur possède une copie complète de l'historique du projet, réduisant ainsi les risques de perte de données en cas de panne d'un serveur central.

7. Amélioration de la qualité du code Grâce à la relecture de code (code review) et aux outils de suivi des modifications, les VCS favorisent l'amélioration continue du code en encourageant les bonnes pratiques de développement.

2. Principes Fondamentaux de Git

Git est un système de gestion de versions distribué (DVCS) qui offre un contrôle total sur l'historique des modifications d'un projet.

2.1 Dépôts

Un dépôt Git est une base de données contenant l'historique des modifications d'un projet. Il peut être local (sur la machine du développeur) ou distant (sur un

serveur comme GitHub, GitLab ou Bitbucket). Chaque développeur possède une copie complète du dépôt.

2.2 Branches

Les branches permettent de travailler sur des fonctionnalités ou des corrections indépendamment de la branche principale (main ou master). Cela facilite le développement parallèle et évite les conflits inutiles.

2.3 Commits

Un commit est un instantané du projet à un moment donné. Chaque commit possède un identifiant unique (hash), un message descriptif et une référence à son commit parent.

2.4 Fusion (Merge)

La fusion (merge) permet de combiner les modifications de plusieurs branches. Cela est couramment utilisé lorsque le développement d'une fonctionnalité est terminé et qu'elle doit être intégrée dans la branche principale.

2.5 Rebase

Le rebase est une alternative à la fusion qui permet de réappliquer une branche sur une autre, en modifiant l’historique des commits pour une intégration plus propre.

2.6 Gestion des conflits

Lorsqu'un même fichier est modifié dans deux branches différentes, un conflit peut survenir. Git permet de résoudre ces conflits manuellement ou à l'aide d'outils spécialisés comme les merge tools.

2.7 Historique et logs

Git permet d'examiner l'historique des modifications à l'aide de commandes comme `git log`, facilitant la compréhension de l'évolution du projet et la résolution de problèmes.

3. Fonctionnalités Avancées de Git

3.1 Rebasing

Le rebasing permet de réappliquer les commits d'une branche sur une autre, ce qui permet de maintenir un historique propre et linéaire. Contrairement à un merge classique, le rebase réécrit l'historique en appliquant chaque commit individuellement, facilitant la lecture des évolutions du projet.

3.2 Cherry-Pick

Le cherry-pick permet de sélectionner un commit spécifique d'une branche et de l'appliquer à une autre sans fusionner toutes les modifications. Cette fonctionnalité est particulièrement utile pour corriger un bug sur plusieurs branches sans impacter l'ensemble du projet.

3.3 Stash

Le stash permet de sauvegarder temporairement les modifications en cours sans les valider (commit). Cela est utile lorsque l'on doit changer de branche ou travailler sur une autre tâche sans perdre les modifications non terminées.

3.4 Hooks

Les hooks sont des scripts exécutés automatiquement lors d'événements Git, comme avant un commit ou après un push. Ils permettent d'automatiser certaines tâches comme la vérification de code, l'exécution de tests unitaires ou l'envoi de notifications.

3.5 Bisect

La commande `git bisect` est un outil permettant de trouver plus efficacement l'origine d'un bug en effectuant une recherche binaire sur l'historique des commits.

3.6 Submodules

Les submodules permettent d'intégrer des dépôts Git externes dans un projet, facilitant ainsi la gestion des dépendances et la modularité du code.

---

4. Présentation de GitHub et de ses Alternatives

GitHub, **GitLab** et **Bitbucket** sont des plateformes populaires permettant l'hébergement et la gestion des dépôts Git. Elles offrent des outils collaboratifs avancés, comme les pull requests, la gestion des tickets et l'intégration continue (CI/CD).

- **GitHub** : Plateforme la plus populaire, utilisée principalement pour le développement open-source et les projets collaboratifs.
- **GitLab** : Offre une solution complète avec CI/CD intégré et un modèle open-source.
- **Bitbucket** : Très utilisé dans les entreprises, notamment pour son intégration avec Jira et d'autres outils Atlassian.

---

5. Cas d'Utilisation dans des Projets Collaboratifs

Les VCS sont utilisés dans de nombreux contextes :

- **Développement open source** : Permet la collaboration à l'échelle mondiale, avec des contributions de nombreux développeurs.
- **Développement en équipe** : Facilite la gestion des fonctionnalités et des correctifs en assurant une meilleure coordination.
- **CI/CD** : Automatise les tests et déploiements pour assurer une intégration continue et une livraison rapide des nouvelles fonctionnalités.
-

Conclusion

Git et les systèmes de gestion de versions ont transformé le développement logiciel en offrant des outils puissants pour collaborer, suivre les changements et automatiser les processus. Maîtriser Git est un atout majeur pour tout développeur moderne.