# Historique + contexte

## Historique du contrôle des versions

Le contrôle de version est un système qui enregistre les modifications apportées à un fichier ou à un ensemble de fichiers au fil du temps afin que vous puissiez accéder ultérieurement à des versions spécifiques. Dans le domaine du développement de logiciels, les **systèmes de contrôle des versions (VCS)** permettent aux développeurs de gérer les modifications apportées à leur code et de savoir qui a effectué chaque changement.

Le contrôle des versions a une longue histoire qui remonte aux années 1980.

Il a été populaire pendant de nombreuses années, mais sa conception présente d'importantes lacunes. Le **CVS** n'inclut pas de contrôles d'intégrité, ce qui signifie que vos données peuvent être corrompues.

En outre, le système a été conçu principalement pour les fichiers texte, et non pour les fichiers binaires tels que les images ou les vidéos.

Le principal successeur de **CVS** a été **Subversion (SVN)**.

CollabNet a développé Subversion en 2000 et a résolu de nombreux problèmes présents dans **CVS**. Pour garantir l'intégrité des données, il a inclus des contrôles d'intégrité dans sa conception. Il supporte également mieux que **CVS** le versionnage des fichiers binaires. Grâce à ces améliorations, SVN est devenu populaire dans la communauté des logiciels libres et Google et SourceForge ont offert un hébergement gratuit pour les projets de logiciels libres.

Cependant, Subversion utilise un modèle **VCS** centralisé. Cela signifie que toutes les opérations doivent être effectuées à l'aide d'un serveur centralisé. Si le serveur était indisponible ou lent, le développement s'en trouverait entravé.

En 2005, deux nouveaux projets ont été lancés pour développer des systèmes de contrôle de version distribués : Mercurial et Git. Ces deux projets ont été créés en réponse à un événement concernant le développement du noyau Linux.

Auparavant, le noyau Linux utilisait un **système de contrôle** des versions propriétaire connu sous le nom de BitKeeper. BitKeeper a été l'un des premiers systèmes de contrôle de version distribués, lancé en 2000. À l'origine, BitKeeper avait fourni une licence gratuite à Linus Torvalds pour soutenir le développement de Linux. Cependant, en 2005, la licence a été révoquée. Cette controverse a conduit à la création des projets Mercurial et Git.

Mercurial a été développé par Olivia Mackall. Il s'agit d'un **VCS** distribué de haute performance. De nombreuses plateformes proposant l'hébergement de Subversion ont commencé à proposer également l'hébergement de Mercurial. Ce projet est devenu populaire car les utilisateurs de Subversion ont trouvé qu'il était facile de passer à un dépôt Mercurial, grâce aux fournisseurs d'hébergement et à une courbe d'apprentissage réduite.

Git a été développé par Linus Torvalds pour héberger le code source du noyau Linux. Comme Mercurial, il s'agit d'un **VCS** distribué. Sa première version publique date de 2007.

Git est devenu populaire dans la communauté open-source en raison de sa conception de **VCS** distribué et du fait que Github offre un hébergement Git gratuit pour les projets open-source. Depuis, Git est devenu le système de contrôle de version choisi pour de nombreux projets de logiciels libres et propriétaires.

## Le contrôle des versions dans le développement de logiciels professionnels

Le contrôle des versions joue un rôle crucial dans le développement des logiciels. En tant que développeur, vous travaillerez avec d'autres développeurs sur des projets visant à fournir des logiciels aux clients. Selon votre rôle, vous pourrez travailler avec une petite équipe de 2 ou 3 développeurs sur un seul projet ou avec une grande équipe couvrant plusieurs projets. Dans les deux cas, le contrôle de version sera un outil essentiel pour aider votre équipe à réussir.

Cependant, le contrôle de version doit être complété par d'autres outils et procédures afin de garantir la qualité et l'efficacité tout au long du processus de développement du logiciel. Dans cette leçon, nous allons explorer quelques-uns des outils et stratégies couramment utilisés par les développeurs en conjonction avec le contrôle de version.

**Flux de travail**

Utiliser le contrôle des versions sans un flux de travail approprié revient à construire une ville sans feux de circulation.

Par exemple, disons que vous travaillez sur un gros projet et que vous modifiez un fichier. Un autre développeur commence également à modifier un fichier. Vous soumettez tous deux le fichier au VCS en même temps. Il y a maintenant un conflit ! Comment le conflit doit-il être résolu ? Un bon flux de travail comporte une procédure de résolution des conflits.

Un autre exemple est celui d'un nouveau développeur junior qui rejoint votre équipe. Si le code du projet est utilisé pour un système critique, il est risqué de l'autoriser à soumettre des modifications de code directement. Pour résoudre ce problème, de nombreux développeurs utilisent un système de révision par les pairs, dans lequel un autre développeur doit réviser le code avant qu'il puisse être intégré.

Les flux de travail sont essentiels pour garantir une gestion correcte du code et éviter les erreurs. Les flux de travail varient d'un projet à l'autre. Dans ce cours, vous apprendrez quelques flux de travail courants en utilisant le système de contrôle de version Git.

**Intégration continue**

L'intégration continue, ou CI, est utilisée pour automatiser l'intégration des modifications de code provenant de plusieurs développeurs dans un flux principal unique. L'utilisation d'un flux de travail dans lequel les petites modifications sont fusionnées fréquemment, souvent plusieurs fois par jour, permet de réduire le nombre de conflits de fusion.

Ce processus est très répandu dans les stratégies de développement de logiciels axées sur les tests. L'intégration continue est souvent utilisée pour compiler automatiquement le projet et exécuter des tests sur chaque modification du code afin de s'assurer que la compilation reste stable et d'éviter les régressions de fonctionnalité.

**Livraison continue**

La livraison continue est une extension de l'intégration continue. Une fois que les modifications ont été intégrées dans le flux principal, un système de livraison continue conditionne automatiquement l'application et la prépare au déploiement.

**Déploiement continu**

Le déploiement continu est une extension de la livraison continue. L'objectif du déploiement continu est de déployer et de mettre à disposition des clients des logiciels fréquemment et en toute sécurité. La stratégie consiste généralement à déployer automatiquement dans un environnement de test (également connu sous le nom de "staging") afin de valider le paquet de déploiement et les modifications apportées au logiciel. Une fois validés, ils peuvent être automatiquement déployés dans l'environnement réel (également appelé production) pour les clients.

**Conclusion**

Grâce à ces outils et procédures, il est possible de comprendre comment un logiciel passe du stade de l'écriture du code par un développeur à celui du déploiement dans un environnement réel pour que les clients puissent l'utiliser. Bien entendu, l'exploitation d'un service logiciel en direct comporte bien d'autres aspects.