POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

CONTRÔLE DE VERSION ET *GIT*

28 AOÛT 2023

CHARLES DE LAFONTAINE, JÉRÔME COLLIN, MERIAM BEN RABIA
GIGL | Polytechnique Montréal

N.B. Le masculin est utilisé pour alléger le texte.

TABLE DES MATIÈRES

CONTRÔLE DE VERSION ET *GIT*

1. CONTRÔLE DE VERSION

Le contrôle de version, également appelé gestion de version ou gestion de révision, est un système qui enregistre les modifications apportées à un fichier ou à un ensemble de fichiers au fil du temps. Ces systèmes permettent de retrouver des versions spécifiques ultérieurement. Ils sont essentiels dans le développement de logiciels où plusieurs personnes peuvent travailler sur **le même** code source.

Parmi les systèmes de contrôle de version, *Git* est le plus populaire. C'est un logiciel open source qui gère tout, des petits projets aux *superprojets*, avec efficacité et vitesse. Il a été créé par Linus Torvalds, le créateur de *Linux*, pour gérer le développement du noyau *Linux*.

2. *GIT*

Git est un système de contrôle de version distribué. Cela signifie que chaque développeur travaille avec une copie complète du référentiel. Les modifications sont importées sous forme de *commits* individuels, ce qui permet une flexibilité sans précédent en termes de collaboration et de gestion de version.

2.1. Termes importants

Termes	Définitions
Repository (référentiel)	C'est un endroit où *Git* a été initialisé pour suivre et gérer les versions du projet. Il contient tous les fichiers du projet ainsi que l'historique des modifications.
Branch (branche)	Il s'agit essentiellement d'une version unique du code. Les branches sont utilisées pour développer des fonctionnalités isolées les unes des autres. La branche par défaut de *Git* est la branche *master* ou *main*.
Commit	C'est un ensemble de modifications ou un instantané du code à un moment donné. Chaque *commit* a un identifiant unique (un hash *SHA*), qui permet à *Git* de retrouver ce *commit* spécifique quand c'est nécessaire.
Merge (fusion)	C'est le processus par lequel les modifications de deux branches sont combinées. *Git* tente de fusionner automatiquement les modifications, mais si deux branches ont modifié la même partie du même fichier, cela peut entraîner un conflit de fusion.

2.2. Commandes importantes

Commandes	Définitions
`git init`	Initialise un nouveau référentiel *Git*.
`git clone`	Clone un référentiel existant.
`git add`	Ajoute de nouveaux fichiers/dossiers ou modifie les fichiers existants dans l'index pour le prochain *commit*.
`git commit`	Enregistre les modifications dans le référentiel.
`git push`	Envoie les modifications vers le référentiel distant.
`git pull`	Récupère les modifications du référentiel distant et les fusionne avec la branche locale.
`git branch`	Permet de travailler avec des branches.
`git merge`	Fusionne une branche avec une autre branche.
`git checkout`	Permet de passer d'une branche à une autre.
Consulter toutes les commandes…	

3. *GITHUB* ET *GITLAB*

GitHub et *GitLab* sont des plateformes web basées sur *Git* qui fournissent une interface graphique pour gérer les référentiels Git, ainsi que d'autres fonctionnalités telles que les demandes de tirage, la gestion des problèmes, et l'intégration continue/déploiement continu.

GitHub est le plus populaire des deux et est utilisé par des millions de développeurs. *GitLab*, cependant, est une solution tout-en-un qui fournit également des fonctionnalités pour le cycle de vie complet du développement de logiciels. Il inclut des fonctionnalités de gestion de projet, de déploiement et de surveillance en plus du contrôle de version.

Les deux plateformes supportent le travail collaboratif et permettent aux utilisateurs de contribuer à des projets open source ou de gérer des projets privés. Ils offrent également des fonctionnalités pour le contrôle d'accès, la gestion des tâches, l'intégration continue et la documentation du projet.

En résumé, le contrôle de version est un aspect crucial du développement de logiciels moderne. Il permet non seulement un suivi précis des modifications, mais facilite également la collaboration entre les développeurs. *Git* est l'un des systèmes de contrôle de version les plus populaires et les plus puissants disponibles, et des plateformes comme *GitHub* et *GitLab* rendent son utilisation encore plus facile et plus intuitive.

Que vous soyez un développeur solo travaillant sur un projet personnel ou une grande équipe travaillant sur une application à grande échelle, comprendre et utiliser efficacement *Git* et les systèmes de contrôle de version est une compétence essentielle.

4. ENRICHISSEMENT : *GIT BISECT*

Git Bisect est un outil de débogage puissant inclus dans *Git*. Il utilise une approche de recherche dichotomique (« *binary search* ») pour trouver le *commit* qui a introduit un bogue dans votre projet. En utilisant *Git Bisect*, vous pouvez rapidement isoler le *commit* fautif sans avoir à vérifier chaque *commit* individuellement.

Pour utiliser *Git Bisect*, vous démarrez par indiquer un « bon » *commit* (un *commit* où le bogue n'était pas présent) et un « mauvais » *commit* (un *commit* où le bogue est présent). *Git Bisect* vérifiera alors le *commit* au milieu et vous demandera si le bogue est présent ou non. Il continuera à diviser la plage de *commits* jusqu'à ce qu'il trouve le *commit* spécifique qui a introduit le bogue.

C'est un excellent outil pour gérer des projets de grande envergure où retracer un bogue peut être une tâche ardue.