Chapitre 3
Fondamentaux de Git pour GitLab

1. Introduction

Dans le chapitre Introduction à GitLab de cet ouvrage, nous avons défini GitLab comme une plateforme web collaborative construite autour du système de gestion de version Git. Nous avons mentionné aussi que GitLab rend ce dernier plus facile à utiliser tout en offrant un emplacement centralisé pour stocker les dépôts Git sur le Web.

Plus encore, GitLab offre plusieurs autres fonctionnalités non liées à Git qui peuvent être intégrées à ce dernier ou utilisées en parallèle. En fait, GitLab est, en quelque sorte, une enveloppe autour de Git qui le rend plus puissant et plus convivial.

Bien que cet ouvrage ne soit pas consacré à Git, il convient toutefois d'en avoir une bonne compréhension, car GitLab emprunte de nombreux concepts à ce gestionnaire de version.

Remarque

Si vous connaissez déjà Git, vous pouvez passer certaines sections, mais assurez-vous, d'une part, de bien maîtriser son fonctionnement et, d'autre part, d'être à l'aise avec les notions de branche et de fusion (merge).

Le présent chapitre permet, d'abord, de s'interroger sur le rôle des systèmes de gestion de version tels que Git en montrant comment ils sont essentiels à tout projet de développement. Nous proposons ensuite de faire une présentation de Git, de décrire son fonctionnement et de faire un rappel des principales commandes nécessaires pour travailler avec l'outil. Après ces sections sur les fondamentaux de Git, nous abordons la notion de branche qui permet de faire du développement en parallèle ainsi que les types de fusion (ou *merge*).

2. Qu'est-ce que Git ?

2.1 Contexte d'émergence de Git

Git est un système de gestion de version (*Version Control System*, VCS) gratuit et open source qui a été introduit en 2005 par Linus Torvalds dans le cadre du projet de développement du noyau Linux qu'il avait lui-même créé en 1991.

À partir de 2002, le code source de Linux était hébergé gratuitement sur la plateforme Bitkeeper sous une licence « community ». Cependant, le fait d'avoir recours à un gestionnaire de version propriétaire était controversé au sein de la communauté Linux, car plusieurs contributeurs estimaient que ce choix n'était pas conforme aux valeurs du logiciel libre qui permet notamment à quiconque de dupliquer et modifier le code source.

Logo de Git

Lorsque la compagnie BitMover qui offrait la solution Bitkeeper a décidé de mettre fin à la version gratuite en 2005, ce fut l'occasion de développer un nouvel outil qui, par le fait même, pourrait mieux répondre aux besoins d'un projet d'envergure comme celui du noyau Linux.

Il était donc essentiel que la conception d'un nouveau système de gestion de version, en l'occurrence Git, puisse remédier à certains inconvénients de son prédécesseur en insistant sur les points suivants :

– **Le mode de licence** : le nouveau VCS doit être gratuit et open source afin de pouvoir être distribué librement.
– **Le modèle centralisé** : le nouveau VCS ne doit plus dépendre d'un serveur centralisé qui contraint les développeurs à interagir constamment avec un dépôt distant.
– **La performance** : le nouveau VCS doit pouvoir prendre en charge des dépôts de code source volumineux dans le cadre de projets collaboratifs qui comprennent plusieurs développeurs sans que la performance en soit affectée.

Voyons maintenant en quoi consiste Git et ce qui le distingue des autres gestionnaires de version.

2.2 Définition générale de Git

Dans les pages du manuel Linux, Git est présenté comme « un stupide gestionnaire de contenu » (« *git - the stupid content tracker* ») et la section « description » le définit comme un « système distribué de contrôle de révisions » (« *distributed revision control system* ») qui répond aux trois exigences mentionnées plus haut puisqu'il est décrit comme étant rapide, extensible (*scalable*) et (libre accès aux composants internes).

NAME

git - the stupid content tracker

SYNOPSIS

```
git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
    [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
    [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]
    [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
    [--config-env=<name>=<envvar>] <command> [<args>]
```

DESCRIPTION

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.

Extrait de la page du manuel Linux consacrée à Git

Cette définition en apparence complexe est assez simple à comprendre lorsqu'elle est replacée dans son contexte. D'abord, Git est un « gestionnaire de contenu », ce qui signifie qu'il n'est pas exclusivement dédié au suivi de code source.

En effet, l'outil permet aussi de gérer tout type de projet individuel ou collaboratif (documentation, fichiers de présentation, rédaction d'articles ou de livres, etc.) pour lequel il est pertinent de conserver des versions dans le temps.

Notons au passage que la description de la page du manuel indique également que Git possède un « riche ensemble de commandes ». Cette particularité vient du fait que l'outil se présente comme une interface en ligne de commande (CLI), comme nous le verrons plus bas dans la section dédiée à son installation.

3. Les systèmes de gestion de version

Avant d'aborder les différents types de systèmes de gestion de version dits aussi « logiciels de gestion de version » ou « systèmes de contrôle de version » (*version control systems*), commençons par en donner une première définition.

Un VCS est un outil qui permet de suivre et d'enregistrer l'évolution d'un ou plusieurs fichiers dans le temps tout en donnant la possibilité de restaurer une version antérieure ou de consulter l'historique des modifications.

Remarque

Pour le dire de façon imagée, un gestionnaire de versions est, en quelque sorte, une machine à voyager dans le temps : elle offre la possibilité de retourner rapidement à une date précise du passé et de revenir dans le présent par la suite.

À propos de Git, la page du manuel que nous venons de consulter précise aussi qu'il s'agit d'un système de gestion de version « distribué » (parfois dit aussi « décentralisé »). Pour comprendre cette distinction, il faut faire une petite incursion dans l'histoire des VCS.

3.1 Les systèmes de gestion de version centralisés

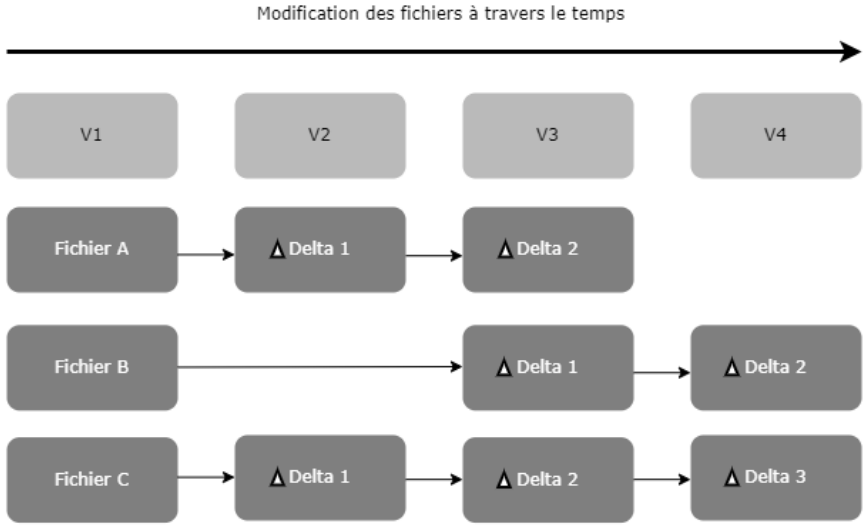
Les gestionnaires de version introduits dans les années 1990 comme CVS (*Concurrent Version System*) ou Subversion permettent de stocker les fichiers (et leurs différentes versions) sur un serveur distant qui joue un rôle de point de dépôt centralisé.

En plus de dépendre d'un équipement sur le réseau, ces VCS conservent deux versions des fichiers :

– la version initiale (lors de la création) ;
– des versions contentant les différences avec l'état qui précède chaque modification.

3.1.1 Une approche basée sur le delta

Cette approche axée sur le delta ou la différence (*delta-based version control*) peut être illustrée de la manière suivante :



Système de gestion de version qui stocke les changements en tant que différences à partir d'un fichier initial

Modification des fichiers d'un VCS centralisé à travers le temps

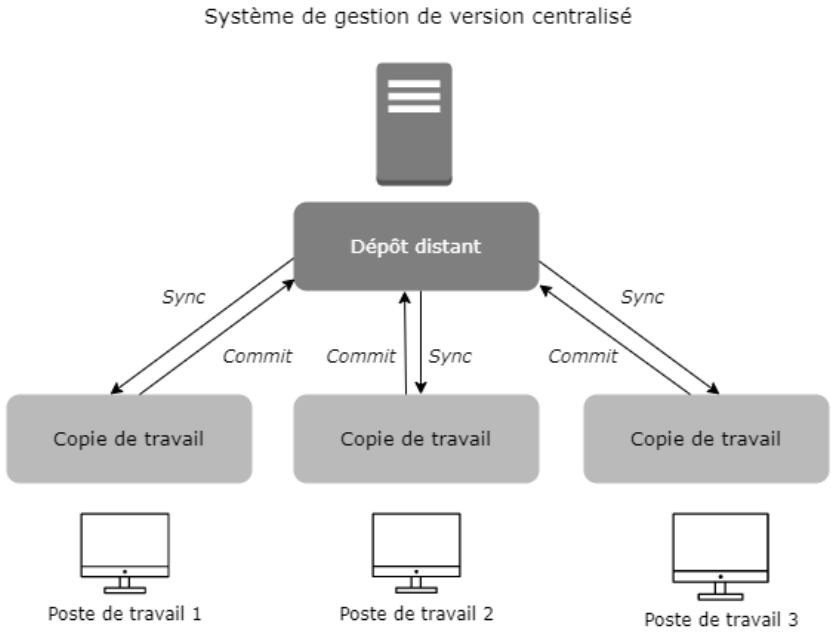
Dans les VCS centralisés, l'information est stockée comme un ensemble de fichiers initiaux (version 1 : V1) ainsi que des fichiers contenant les différences par rapport à ceux-ci (versions subséquentes : V2, V3, V4).

Le principal inconvénient que présente cette approche apparaît lorsqu'il faut restaurer une version antérieure d'un fichier. Pour effectuer cette tâche, le VCS doit reconstruire le fichier désiré en appliquant tous les changements accumulés depuis la version initiale.

Ce processus peut être très long et il suppose, bien entendu, que la seule version centralisée soit intègre. Ce dernier point met en lumière deux autres inconvénients des VCS centralisés :

– Les utilisateurs sont dépendants d'une connexion réseau pour mettre à jour leur dépôt (*repository*) ou accéder à l'historique des modifications : ils ne disposent pas l'entièreté du dépôt sur leur poste de travail.
– Le serveur qui est désigné comme point de dépôt centralisé constitue un point de défaillance unique (*single point of failure*).

Ces derniers aspects permettent de mieux comprendre pourquoi Linus Torvalds souhaitait créer un outil performant qui aurait la particularité de ne pas dépendre d'un modèle centralisé. Mais ce qui était sans doute l'idée la plus ingénieuse tient au fait que Git est un VCS distribué, c'est-à-dire qu'il permet d'avoir l'entièreté d'un même dépôt de code source localement sur un ou plusieurs ordinateurs. Nous reviendrons sur cette particularité dans la section suivante.



Système de gestion de version centralisé

3.1.2 Les contraintes du modèle centralisé

De manière générale, les utilisateurs d'un VCS centralisé sont limités à deux opérations principales :

– Mettre à jour (*synchroniser*) leur copie de travail locale à partir du dépôt centralisé.
– Enregistrer (faire un commit) des modifications apportées à leur copie de travail locale pour les envoyer sur le dépôt centralisé.

Remarque

Nous reviendrons plus bas sur la notion de « commit » qui réfère à l'enregistrement de l'état d'un fichier ou d'un ensemble de fichiers dans un système de gestion de versions.

Git tente de remédier aux inconvénients des gestionnaires de version centralisés en proposant :

– un VCS qui permet à la fois de disposer d'un dépôt complet en local sur un poste de travail ;
– un mode d'enregistrement des fichiers qui ne s'appuie pas sur la notion de delta ou de différences entre chaque version d'un fichier.

3.1.3 La notion de dépôt

Comme nous venons d'évoquer à quelques reprises la notion de « dépôt » (*repository* ou *repo*), prenons un instant pour en donner une définition plus précise avant de poursuivre avec les systèmes de gestion de version distribués.

Dans le contexte qui nous occupe, un dépôt est un répertoire dans lequel sont stockées les modifications apportées aux fichiers d'un projet (de développement, de documentation, etc.).

Un dépôt permet de faire le suivi (*tracking*) des modifications apportées par ses différents contributeurs en les consignant dans un historique. De manière générale, un dépôt offre également des options de contrôle d'accès et, dans certains cas, des fonctionnalités de sauvegarde permettant la récupération de versions antérieures du code source si nécessaire.

Il existe généralement deux types de dépôts :

– **Le dépôt local (*local repository*)** : il est situé sur le poste de travail d'un utilisateur. Il peut s'agir, par exemple, d'un répertoire personnel non partagé ou d'une copie d'un dépôt distant.
– **Le dépôt distant (*remote repository*)** : il est situé sur une plateforme web d'hébergement de code source comme GitLab ou sur un serveur interne créé à cet effet. Ils permettent de centraliser les fichiers d'une équipe tout en favorisant la collaboration.

Remarque

Les termes « dépôt » et « projet » sont souvent utilisés de manière interchangeable parce que, dans GitLab et d'autres solutions similaires comme GitHub, un projet contient toujours un dépôt, en l'occurrence Git. Un projet sur une plateforme web bénéficie de diverses fonctionnalités spécifiques à la plateforme que Git n'offre pas localement. Nous reviendrons sur ces aspects dans le chapitre Travailler avec un dépôt distant sur GitLab.

Maintenant que nous avons un meilleur aperçu de ce qu'est un dépôt, voyons en quoi consistent les systèmes de gestion de version distribués.

3.2 Les systèmes de gestion de version distribués

Comme nous l'avons évoqué plus haut, Git tente de remédier aux inconvénients des gestionnaires de version centralisés en proposant un VCS qui permet, d'une part, de disposer d'un dépôt complet sur un poste de travail et, d'autre part, de ne plus stocker le contenu d'un projet en s'appuyant sur la notion de delta ou de différences entre chaque version d'un fichier.

Ce qui différencie Git des VCS centralisés est qu'il fonctionne avec la notion de *snapshot* (« instantané »). À chaque commit ou nouvel enregistrement de l'état d'un projet, Git prend une image complète (snapshot) du nouvel état des fichiers. Ce qui n'a pas changé n'est pas stocké à nouveau : des pointeurs vers le contenu préexistant sont créés automatiquement afin d'optimiser le stockage.