Annexe 4 : Gestion des versions avec GIT

Sommaire :

I - Introduction

I.1. Introduction

Git est un système libre et open source de **gestion de version** (**VCS** : **V**ersion **C**ontrol **S**ystem) collaboratif.

Un **gestionnaire de versions** est un programme qui permet aux développeurs de conserver un historique des modifications et des versions de tous leurs fichiers. L'action de contrôler les versions est aussi appelée "**versioning**" en anglais.

Le gestionnaire de versions permet de garder en mémoire :
- chaque modification de chaque fichier ;
- pourquoi elle a eu lieu ;
- et par qui !

Dans ce qui suit, nous allons utiliser **Git** et **GitLab**.

I.2. Git

Git est un **gestionnaire de versions**, qui permet de stocker de manière optimisée et sécurisée des fichiers et toutes leurs modifications dans le temps. **GIT** est décentralisé, et permet à de multiples personnes de travailler ensemble sur le même projet, puis d'intégrer harmonieusement les travaux de chacun.

On l'utilisera pour héberger notre **dépôt**. Dans ce cas, on parle de **dépôt local** puisqu'il est stocké sur notre poste de travail.

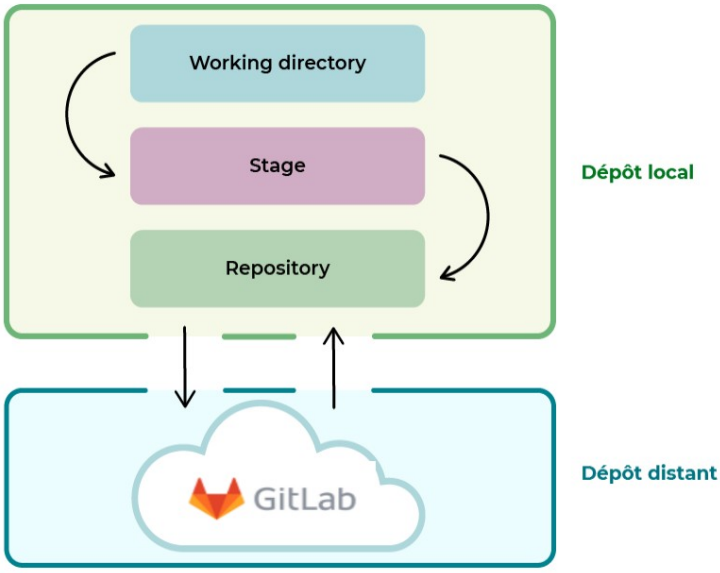
I.3. GitLab

GitLab (similaire à **GitHub**) est une plateforme de développement collaborative open source éditée par la société américaine du même nom. Elle couvre l'ensemble des étapes du **DevOps**. Se basant sur les fonctionnalités du logiciel **Git**, elle permet de piloter des dépôts de code source et de gérer leurs différentes versions. Son usage est particulièrement indiqué pour les développeurs qui souhaitent disposer d'un outil réactif et accessible.

On l'utilisera pour créer un **dépôt (repository) distant** et pour gérer les versions de nos fichiers.

I.4. Git et GitLab

Le schéma ci-dessous représente le fonctionnement de **Git**. On y distingue le **dépôt local** (composé de 3 zones) et le **dépôt distant GitLab**.

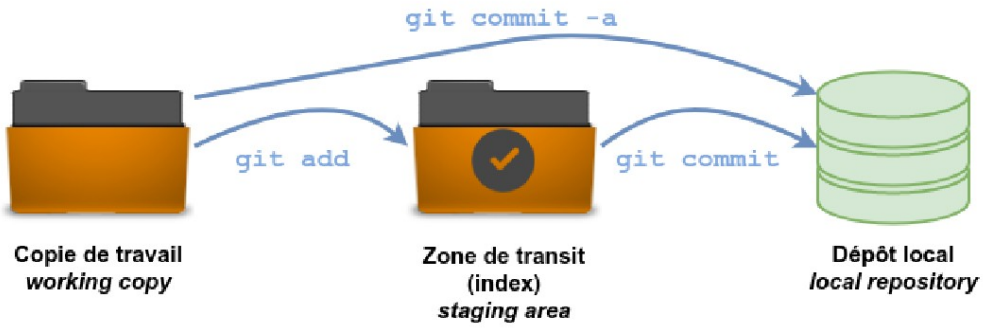


Le **dépôt local** est composé de 3 zones présentes dans votre poste de travail, en local :

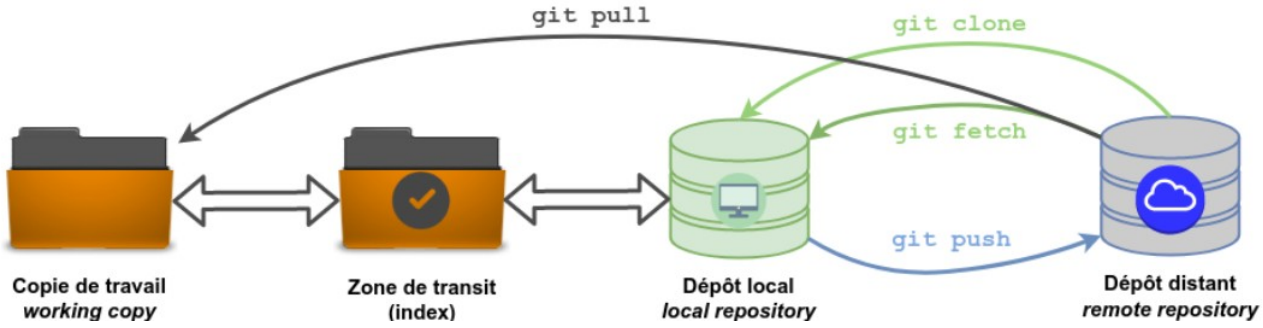
- le **Working directory** : Cette zone correspond au dossier du projet sur votre ordinateur ;
- le **Stage** ou **index** : Cette zone est un intermédiaire entre le **working directory** et le **repository**. Elle représente tous les fichiers modifiés que vous souhaitez voir apparaître dans votre prochaine version de code ;
- le **Repository** ou **dépôt local** : Lorsque l'on crée de nouvelles versions d'un projet, c'est dans cette zone qu'elles sont stockées.

En-dessous, on trouve le **repository GitLab**, c'est-à-dire le **dépôt distant**.

On peut visualiser aussi sur le schéma ci-dessous le passage d'un fichier dans les différentes zones du **dépôt local** :

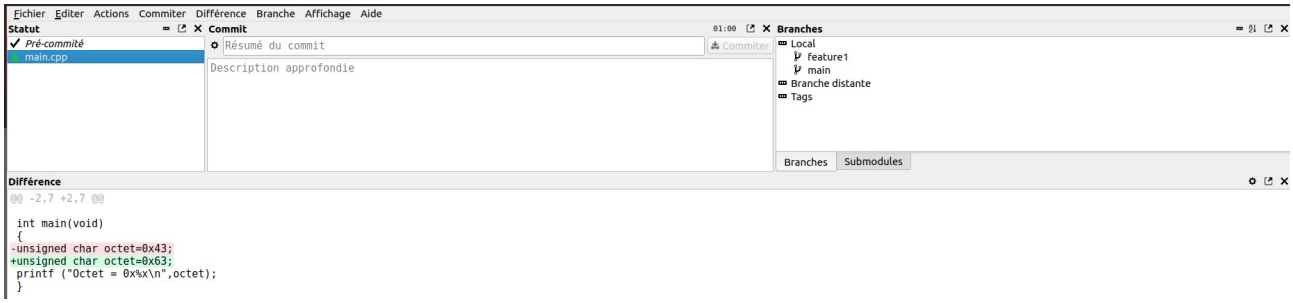


Et aussi avec le **dépôt distant** :



Il existe de nombreuses **interfaces graphiques** permettant de gérer des projets **Git** : **GitWeb**, **gitk**, **qgit**, **Giggle**, **Source Tree**, **Git Cola**, ….

On pourra utiliser simplement **Git Cola** qui s'installe sous **Ubuntu** à l'aide de la commande **apt install git-cola**.

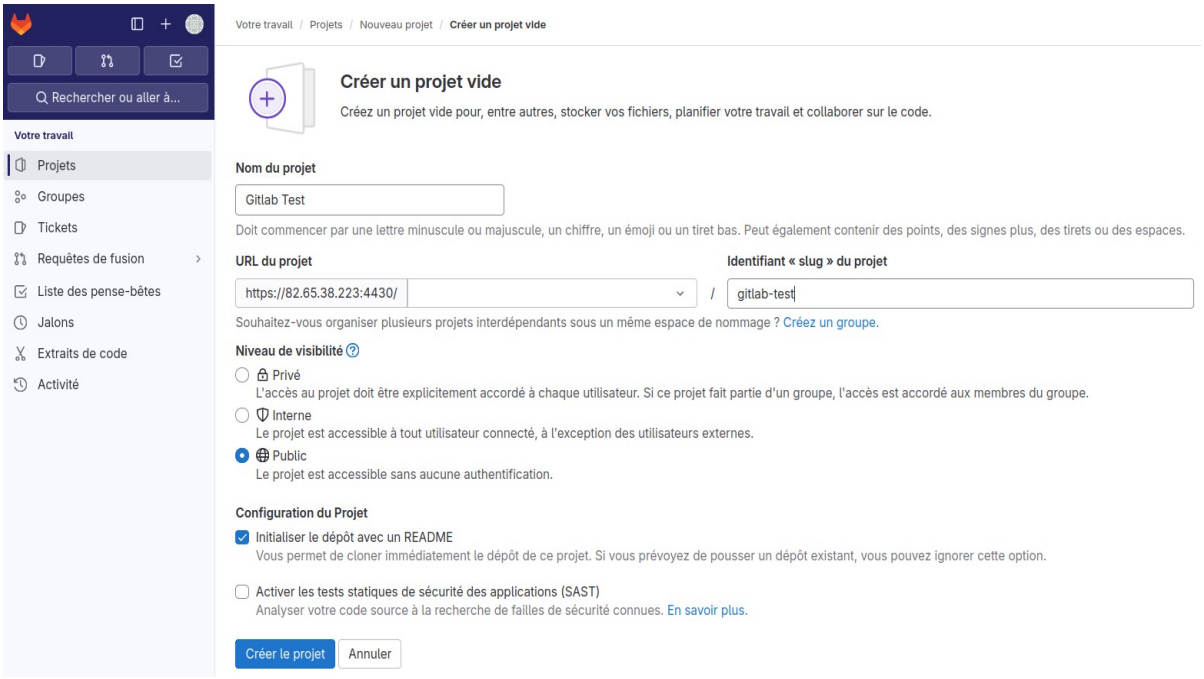


II - Utilisation de Git et de GitLab

II.1. GitLab

Nous disposons d'un serveur **GitLab** à la section du **BTS CIEL-IR**. Pour s'y connecter, il faut ouvrir un navigateur avec l'URL **https://82.65.38.223:4430** et se loger avec son compte sur le royaume **LDAP Scribe**.

Vous arrivez sur l'espace « Votre travail » à partir duquel vous pouvez créer vos projets :



Remarque *:* Un projet **GitLab** est **public** (accessible à tout le monde), **interne** (accessible aux utilisateurs déclarés sur le serveur) ou **privé** (accessible uniquement par le propriétaire ou le groupe).

II.2. Installation et Initialisation de Git

Suite à son installation (**apt install git**), **Git** n'est pas configuré sur son poste de travail. On peut le configurer à l'aide des commandes suivantes :

```
git config --global user.name "votre_login"
git config --global user.email "votre_login@i-gcrampe.snir.lan"
git config --global http.sslverify false
git config --global core.editor mcedit
```

II.3. Cloner un dépôt distant en local

On peut cloner un dépôt distant en local à l'aide des commandes suivantes :

```
mkdir save
cd save
git clone https://82.65.38.223:4430/jcabianca/tp2-iot-makefile-sources.git
cd  tp2-iot-makefile-sources
git pull origin main                    // Pour mettre à jour le dépôt local
```

II.4. Création d'un dépôt local

On peut créer un dépôt local à l'aide des commandes suivantes :

```
mkdir premierProjet
cd premierProjet
git init                                // Pour initialiser le nouveau dépôt local
git add .                               // Pour ajouter tous les fichiers à l'index (stage)
git rm toto.txt                         // Pour supprimer le fichier de l'index
git commit -m "Fichiers d'origine"      // Pour créer la version dans le Repository
```

II.5. Envoyer un dépôt local sur GitLab

On peut envoyer un dépôt local sur GitLab à l'aide des commandes suivantes :

```
cd premierProjet
git remote add origin https://82.65.38.223:4430/votre_login/premierProjet.git
git branch -M main
git push -u origin main                    // Pour envoyer la branche main
```

II.6. Enregistrer une nouvelle version d'un fichier sur le dépôt

On peut enregistrer une nouvelle version d'un fichier sur le dépôt local et sur le dépôt distant à l'aide des commandes suivantes :

```
git add main.cpp                           // Pour ajouter le fichier main.cpp à l'index
git commit -m "Modification initiale"      // Pour créer la version dans le Repository
git push origin main                       // Pour envoyer la branche main
```

II.7. Créer une nouvelle branche nommée feature1 sur le dépôt

On peut créer une nouvelle branche nommée **feature1** sur le dépôt local et sur le dépôt distant à l'aide des commandes suivantes :

```
git branch feature1                        // Pour l'effacer, git branch -d feature1
git checkout feature1
git commit -m "Réalisation de la partie feature1"
git push origin feature1                   // Pour envoyer la branche feature1
git checkout main                          // Pour revenir à la branche principale
```

II.8. Fusionner l'évolution feature1 à main

On peut intégrer l'évolution réalisée dans la branche **feature1** à la branche **main** à l'aide des commandes suivantes :

```
git checkout main
git merge feature1                         // Pour fusionner feature1 à main
```