4- Outils de gestion de versions et de mesure de la qualité du code

4.1 - Les outils de gestion de versions (git/gitlab)

4.1.1 - Introduction

Outil de gestion de version est un outil (logiciel) permettant d'enregistrer, de suivre et de gérer plusieurs versions d'un fichier ou d'un code source. Il permet d'établir un historique de toutes les modifications effectuées sur un élément, pour ainsi avoir la possibilité de récupérer une version antérieure selon la date et l'heure de la sauvegarde, et ce en cas d'erreur ou de problème sur une version actuelle.

Le gestionnaire de version est :

- Un outil qui garde un historique des différentes mises à jour d'une application ou d'un logiciel.
- Un meilleur travail collaboratif : gestion de plusieurs versions du code source.

Exemples :

- **Git** est un logiciel de gestion de versions décentralisé. C'est un logiciel libre et gratuit, créé en 2005 par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2.
- **Mercurial** est un logiciel de gestion de versions décentralisé disponible sur la plupart des systèmes Unix et Windows.

4.1.2 - Présentation de git et gitlab

C'est quoi Git ?

Git est un outil de gestion de version ou VCS en anglais (version control system) qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.

Il fait parti de la famille des VCS dit décentralisés car dans son fonctionnement chaque développeur va avoir en local une copie complète de l'historique de son code source.

C'est quoi GitLab?

GitLab est un logiciel libre de forge basé sur git proposant les fonctionnalités de wiki, un système de suivi des bugs, l'intégration continue et la livraison continue.

Fonctionnalités

- Un **système de suivi des bugs** est un logiciel qui permet d'effectuer un suivi des bugs signalés dans le cadre d'un projet de développement de logiciel.
- **L'intégration continue (CI)** est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.
- **La livraison continue (CD)** est une approche d'ingénierie logicielle dans laquelle les équipes produisent des logiciels dans des cycles courts, ce qui permet de le mettre à disposition à n'importe quel moment. Le but est de construire, tester et diffuser un logiciel plus rapidement.
- Un **wiki** est une application web qui permet la création, la modification et l'illustration collaboratives de pages à l'intérieur d'un site web.

4.1.3 - Fonctionnement de base de Git

Git est un système de gestion de version décentralisé. Cela signifie que les données du dépôt Git ne se trouvent pas sur un serveur distant mais bel et bien sur votre machine.

Le mode décentralisé présente beaucoup d'avantages :

- Git est extrêmement rapide à mettre en place : en 3 secondes, on a créé un dépot sans avoir besoin de tripatouiller un serveur quelconque ;
- le dépôt n'étant pas dépendant du réseau, les opérations sont très rapides et vous pouvez travailler n'importe où, e.g dans le train ;
- il est possible de créer autant de dépôts que l'on veut sur une même machine.

Un dépôt Git (repository)

Un “dépôt” correspond à la copie et à l’importation de l’ensemble des fichiers d’un projet dans Git. Il existe deux façons de créer un dépôt Git :

- On peut importer un répertoire déjà existant dans Git ;
- On peut cloner un dépôt Git déjà existant.

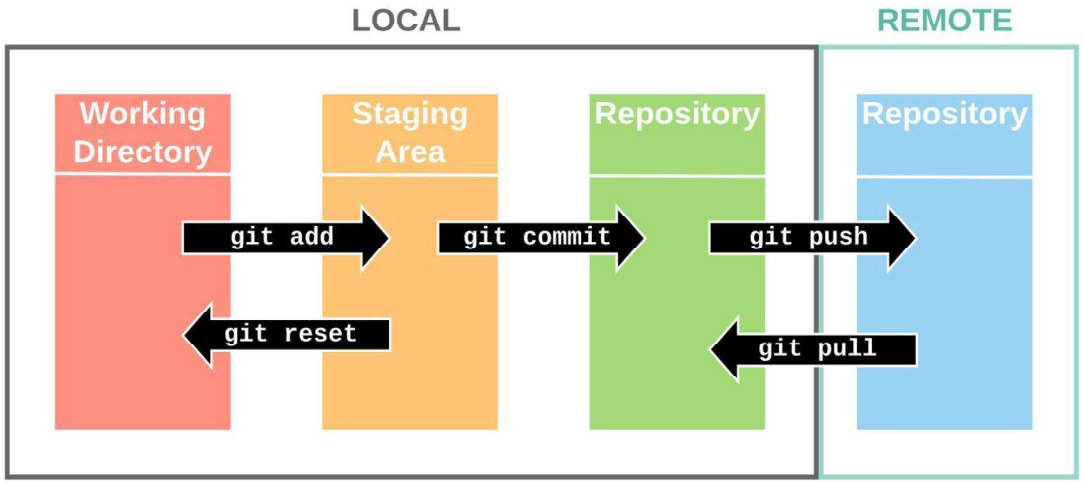
Les états des fichiers

Git gère trois états dans lesquels les fichiers peuvent résider : modifié, indexé et validé.

- **Modifié (“modified”)** signifie que vous avez modifié le fichier mais qu’il n’a pas encore été validé en base.
- **Indexé (“staged”)** signifie que vous avez marqué un fichier modifié dans sa version actuelle pour qu’il fasse partie du prochain instantané du projet.
- **Validé (“committed”)** signifie que les données sont stockées en sécurité dans votre base de données locale.

Les zones de travail

Les états de fichiers sont liés à des zones de travail dans Git. En fonction de son état, un fichier va pouvoir apparaitre dans telle ou telle zone de travail. Tout projet Git est composé de trois sections : le répertoire de travail, la zone d’index et le répertoire Git.



Le **répertoire de travail** (**working tree**) correspond à une extraction unique d'une version du projet. Les fichiers sont extraits de la base de données compressée située dans le répertoire Git et sont placés sur le disque afin qu'on puisse les utiliser ou les modifier.

La **zone d'index** (**staging area**) correspond à un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané ou du prochain "commit".

Le **répertoire Git** (**repository**) est l'endroit où Git stocke les méta-données et la base de données des objets de votre projet. C'est la partie principale ou le coeur de Git.

4.1.4 - Manipulation des commandes de base de Git

CONFIGURATION DES OUTILS

Configurer les informations de l'utilisateur pour tous les dépôts locaux

git config

Pour configurer les informations de l'utilisateur pour tous les dépôts locaux. Le flag -global rend ces changements persistants sur votre machine plutôt qu'uniquement pour le projet.

```
git config --global user.name <votre_nom_d'utilisateur>
git config --global user.email <votre_adresse_email>
```

CRÉER DES DÉPÔTS

Démarrer un nouveau dépôt ou en obtenir un depuis une URL existante

git init

Cette commande est utilisée pour créer un nouveau dépôt GIT :

```
git init
```

git clone

La commande git clone est utilisée pour la vérification des dépôts et le téléchargement d'un projet et tout son historique de versions. Si le dépôt se trouve sur un serveur distant, utilisez

```
git clone git@github.com:name/chemin/vers/dépôt
```

Inversement, si une copie de travail d'un dépôt local doit être créée, utilisez:

```
git clone /chemin/vers/dépôt
```

EFFECTUER DES CHANGEMENTS

Consulter les modifications et effectuer une opération de commit

git add

La **commande git add** peut être utilisée pour ajouter des fichiers à l'index. Par exemple, la commande suivante ajoutera un fichier nommé temp.txt dans le répertoire local de l'index :

```
git add temp.txt
```

git status

La **commande git status** affiche la liste des fichiers modifiés ainsi que les fichiers qui doivent encore être ajoutés ou validés. Usage :

```
git status
```

git commit

La **commande git commit** permet de **valider les modifications apportées** au HEAD. Notez que tout commit ne se fera pas dans le dépôt distant.

```
git commit –m “Description du commit”
```

git reset

Pour réinitialiser l’index et le répertoire de travail à l’état du dernier commit, la **commande git reset** est utilisée :

```
git reset --hard HEAD
```

git diff

La **commande git diff** permet de lister les conflits. Pour visualiser les conflits d’un fichier :

```
git diff --base <nom-fichier>
```

La commande suivante est utilisée pour afficher les conflits entre les branches à fusionner:

```
git diff <branche-source> <branche-cible>
```

Pour simplement énumérer tous les conflits actuels, utilisez :

```
git diff
```

GROUPER DES CHANGEMENTS

Nommer une série de commits et combiner les résultats de travaux terminés

git branch

La **commande git branch** peut être utilisée pour répertorier, créer ou supprimer des branches. Pour répertorier toutes les branches présentes dans le dépôt, utilisez :

```
git branch
```

Pour supprimer une branche :

```
git branch –d <nom-branche>
```

git checkout

La **commande git checkout** peut être utilisée pour créer des branches ou pour basculer entre elles. Par exemple nous allons créer une branche :

```
git checkout -b <nom-branche>
```

Pour passer simplement d’une branche à une autre, utilisez :

```
git checkout <nom-branche>
```

git merge

La **commande git merge** est utilisée pour fusionner une branche dans la branche active.

```
git merge <nom-branche>
```

SYNCHRONISER LES CHANGEMENTS

Référencer un dépôt distant et synchroniser l'historique de versions

git push

Git push est une autre commandes GIT de base. Un simple push envoie les modifications locales apportées à la branche principale associée :

```
git push origin master
```

git pull

Pour fusionner toutes les modifications présentes sur le dépôt distant dans le répertoire de travail local, la commande pull est utilisée.

```
git pull
```

VÉRIFIER L'HISTORIQUE DES VERSIONS

Suivre et inspecter l'évolution des fichiers du projet

git log

L'exécution de cette commande montre l'historique des versions pour la branche courante.

```
git log
```

git show

L'exécution de cette commande montre les modifications de métadonnées et de contenu inclues dans le commit spécifié.

```
git show [commit]
```

CHANGEMENTS AU NIVEAU DES NOMS DE FICHIERS

Déplacer et supprimer des fichiers sous suivi de version

git rm

Git rm peut être utilisé pour supprimer des fichiers de l'index et du répertoire de travail. Usage :

```
git rm nomfichier.txt
```

git mv

L'exécution de cette commande renomme le fichier et prépare le changement pour un commit :

```
git mv [fichier-nom] [fichier-nouveau-nom]
```