



ATELIER GITLAB

Réalisé par: EL AYBOUDI Imane
ZARHOUNI El Mehdi



Plan :

- Définition
- Historique:
- Objectifs
- Fonctionnalités
- Git et Gitlab
- Systèmes d'exploitations supportés
- Gestion de projets avec Gitlab



Définition :

- GitLab est une plateforme de gestion du cycle de vie du développement logiciel (DevOps) qui offre une suite complète d'outils pour la gestion du code source, le suivi des problèmes, l'intégration continue, le déploiement continu, la gestion de la documentation, et bien plus encore.
- GitLab utilise le système de contrôle de version distribué Git comme base, fournissant une interface web conviviale et des fonctionnalités avancées pour faciliter la collaboration au sein des équipes de développement.



Historique de GitLab :

- GitLab a été créé en 2011 par Dmitriy Zaporozhets et Valery Sizov en tant que solution de gestion du code source basée sur Git. Depuis sa création, GitLab a connu plusieurs versions et améliorations, devenant progressivement une plateforme complète de gestion du cycle de vie du développement logiciel.



Ses principaux objectifs :

Collaboration Efficace:

GitLab vise à faciliter la collaboration au sein des équipes de développement en offrant un espace centralisé pour la gestion du code, des problèmes, et des déploiements.

Automatisation des Processus :

L'objectif est d'automatiser les processus de développement grâce à des fonctionnalités telles que l'intégration continue (CI) et le déploiement continu (CD).

Plateforme Complète :

Contrairement à Git, qui se concentre principalement sur le contrôle de version, GitLab offre une gamme étendue de fonctionnalités pour couvrir l'ensemble du cycle de vie du développement.

Ses fonctionnalités:

1. Gestion du Code Source



Création d'un Projet :

Pour démarrer un nouveau projet dans GitLab, vous pouvez le créer directement depuis l'interface web ou importer un projet existant depuis un dépôt distant. Cela initialise un espace de travail pour votre code.



Gestion des Répertoires :

GitLab permet d'ajouter, de supprimer et de déplacer des fichiers directement depuis l'interface web. L'utilisation de branches facilite le développement parallèle en isolant les fonctionnalités ou correctifs.

Ses fonctionnalités:

2. Suivi des Problèmes (Issues)



Création d'Issues :

Les issues dans GitLab sont utilisées pour signaler des bugs, proposer des fonctionnalités, ou discuter de tout élément lié au projet. Les issues peuvent être créées par les membres de l'équipe ou les contributeurs externes.



Attribution et Suivi :

Une fois une issue créée, elle peut être assignée à un membre de l'équipe. Les assignations facilitent le suivi des responsabilités, et les discussions dans les issues aident à suivre les progrès et les décisions prises.

Ses fonctionnalités:

3. Intégration Continue (CI/CD)



Mise en Place d'un Pipeline CI/CD:

GitLab utilise un fichier `.gitlab-ci.yml` pour définir les étapes du pipeline CI/CD. Chaque modification du code déclenche automatiquement le pipeline, exécutant des tests, des analyses de code, et déployant si configuré.



Utilisation des Runners :

Les runners GitLab sont utilisés pour exécuter les tâches définies dans le pipeline. Ils peuvent être exécutés localement ou sur des machines distantes, et ils permettent d'assurer l'automatisation des processus.

Ses fonctionnalités:

4. Wiki et Gestion de la Documentation



Création d'un Wiki :

GitLab propose un wiki intégré pour la documentation du projet. Les utilisateurs peuvent créer et éditer des pages directement depuis l'interface web. Cela offre un espace dédié pour stocker des informations essentielles.



Intégration de Documentation Externe :

En plus du wiki interne, GitLab peut être connecté à d'autres systèmes de documentation externes. La plupart des projets utilisent des formats tels que Markdown pour la documentation.

Ses fonctionnalités:

5. Gestion des Déploiements



Configuration des Environnements de Déploiement :

GitLab permet de définir des environnements tels que développement, staging, et production. Chaque environnement peut avoir ses propres variables d'environnement pour une configuration spécifique.



Déploiement Automatisé :

Les scripts de déploiement peuvent être intégrés au pipeline CI/CD pour automatiser le déploiement des applications. Les informations sur les déploiements réussis ou échoués sont enregistrées pour une traçabilité complète.

Les Systèmes d'exploitation supportés

	Supporté	Non supporté
OS X		Yes
Ubuntu	Yes	
Fedora		Yes
Debian	Yes	
CentOS	Yes	
Red Hat Enterprise Linux	Yes	
Oracle Linux	Yes	
Windows		Yes



Différences entre GitLab et Git :

- **Interface Utilisateur Web** : GitLab fournit une interface web conviviale, ce qui simplifie l'accès aux fonctionnalités sans avoir à utiliser uniquement la ligne de commande Git.
- **Gestion du Cycle de Vie** : GitLab étend les fonctionnalités de Git pour inclure la gestion des projets, l'intégration continue, le suivi des problèmes, la gestion des déploiements, et plus encore.



Installation et Configuration:

Installation de GitLab:

- **Options d'installation (cloud, serveur local):** GitLab peut être installé de différentes manières, notamment sur le cloud (GitLab.com) ou sur un serveur local. L'option choisie dépend des besoins spécifiques de l'équipe ou de l'organisation.
- **Configuration minimale requise:** Avant l'installation, assurez-vous que le serveur répond aux exigences minimales en termes de ressources matérielles et logicielles pour garantir des performances optimales de GitLab.




Installation et Configuration:

Configuration initiale :

- **Création d'un compte:** Une fois GitLab installé, la première étape consiste à créer un compte utilisateur. Cela peut être réalisé via l'interface web de GitLab.
- **Configuration du profil utilisateur:** Personnalisez votre profil utilisateur en ajoutant des informations pertinentes telles que l'adresse e-mail, le nom d'utilisateur, et les préférences personnelles.
- **Création d'un nouveau projet:** Initiez un nouveau projet dans GitLab en définissant les paramètres de base, tels que le nom du projet, la visibilité (public, privé), et les options de suivi du code source.



Création d'un compte:





Start your 30-day free trial

No credit card required

- ✓ Invite unlimited colleagues
- ✓ Free guest users
- ✓ Ensure compliance
- ✓ Built-in security

Register with:

 Google

 GitHub

or

First name

test

Last name

atelier

Username

testatelier123

Username is available.

Email

elmehdizarhouni@gmail.com

Password

.....

Minimum length is 8 characters.

Continue

By clicking Continue or registering through a third party you accept the GitLab [Terms of Use](#) and acknowledge the [Privacy Policy](#) and [Cookie Policy](#)

Already have an account? [Sign in](#)



Création d'un compte:

Welcome to GitLab, test!

To personalize your GitLab experience, we'd like to know a bit more about you. We won't share this information with anyone.

Role

Other

I'm signing up for GitLab because:

A different reason

Why are you signing up? (optional)

Who will be using this GitLab trial?

☐ My company or team ☒ Just me

Email updates (optional)

☐ I'd like to receive updates about GitLab via email

Continue



Création d'un projet:

+

🔍 Search or go to...

Your work

Projects

Groups

Issues

Merge requests

To-Do List

Milestones

Snippets

Activity

Workspaces

Environments

Operations

Security

Help

Your work / Projects / New project / Create blank project

Project name

test_atelier

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

Project URL

https://gitlab.com/ test8963750

Project slug

test_atelier

Want to organize several dependent projects under the same namespace? [Create a group.](#)

Project deployment target (optional)

Select the deployment target

Visibility Level ?

☐ Private

Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

☒ Public

The project can be accessed without any authentication.

Project Configuration

☒ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

☐ Enable Static Application Security Testing (SAST)

Analyze your source code for known security vulnerabilities. [Learn more.](#)

Create project

Cancel



Création d'un projet:

Ici on crée le SSH key et on utilise la commande suivante :

```
ssh-keygen -t rsa -b 2048 -C "cadresse_electronique"
```

test / test_atelier

⚠ You can't push or pull repositories using SSH until you add an SSH key to your profile.

Add SSH key

Don't show again

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations ! https://aka.ms/PSWindows

PS C:\Users\User0\Desktop> ssh-keygen -t rsa -b 2048 -C "<elmehdizarhouni@gmail.com>"
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\User0\.ssh\id_rsa): test_atelier
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in test_atelier
Your public key has been saved in test_atelier.pub
The key fingerprint is:
SHA256:cp+9LjjTWL7r7WrvbX7gLygP4Pq5zI7KdBFwwSvBoT8 <elmehdizarhouni@gmail.com>
The key's randomart image is:
+----[RSA 2048]-----+
|  .ooo.                |
|  .oo.                 |
|   ...                 |
|  .. ..               |
|   E.o S               |
|    . = o o .          |
|   . . .o= oo .       |
|  o . =++oo+o.o       |
| o.o o0=.BB=====    |
+----[SHA256]-----+
PS C:\Users\User0\Desktop>
```

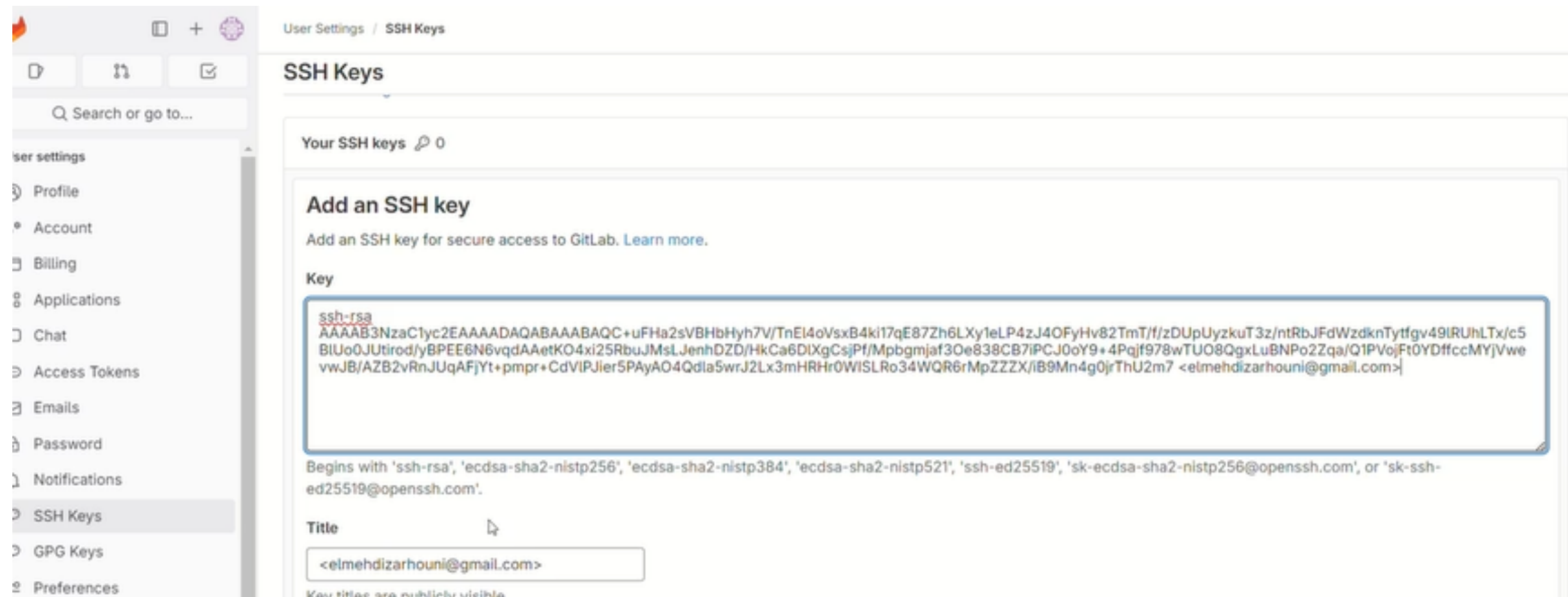


Création d'un projet:

puis on passe à la commande suivante : `cat nom_projet.pub`

```
PS C:\Users\User0\Desktop> cat test_atelier.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC+uFHa2sVBHbHyh7V/TnEl4oVsx84ki17qE87Zh6LXy1eLP4zJ40FyHv82TmT/f/zDUpUyzkuT3z/ntRbJ
FdWzdknTytf9v49lRUhLTx/c5BlUo8JUtirod/yBP6E6N6vqdAAetKO4xi25RbuJMsLJenhDZD/HkCa6DlXgCsJpF/Mpbgnjaf30e838CB7iPCJ8oY9+4Pqj
f978wTU08QgxLuBNPo2Zqa/Q1PVojFt0YDffccMYjVwevwJB/AZB2vRnJUqAFjYt+pmpr+CdVLPJier5PAyAO4Qdla5wrJ2Lx3mHRHr0WISLro34WQR6rMpZ
ZZX/iB9Mn4g0jrThU2m7 <elmehdizarhouni@gmail.com>
PS C:\Users\User0\Desktop> |
```

on copie le output et on le colle dans l'espace de key au niveau de l'interface de gitlab





Gestion de Projet avec GitLab:

Gestion du Code Source:

- **Clonage d'un dépôt:** Utilisez la commande Git clone pour copier un dépôt GitLab sur votre machine locale, permettant ainsi de travailler sur le code en mode déconnecté.
- **Ajout, modification et suppression de fichiers:** Apprenez à effectuer des modifications sur le code source en ajoutant de nouveaux fichiers, en modifiant les existants, et en supprimant ceux qui ne sont plus nécessaires.
- **Utilisation des branches:** Comprenez l'utilisation des branches pour le développement parallèle, la gestion des fonctionnalités, et la résolution de conflits.

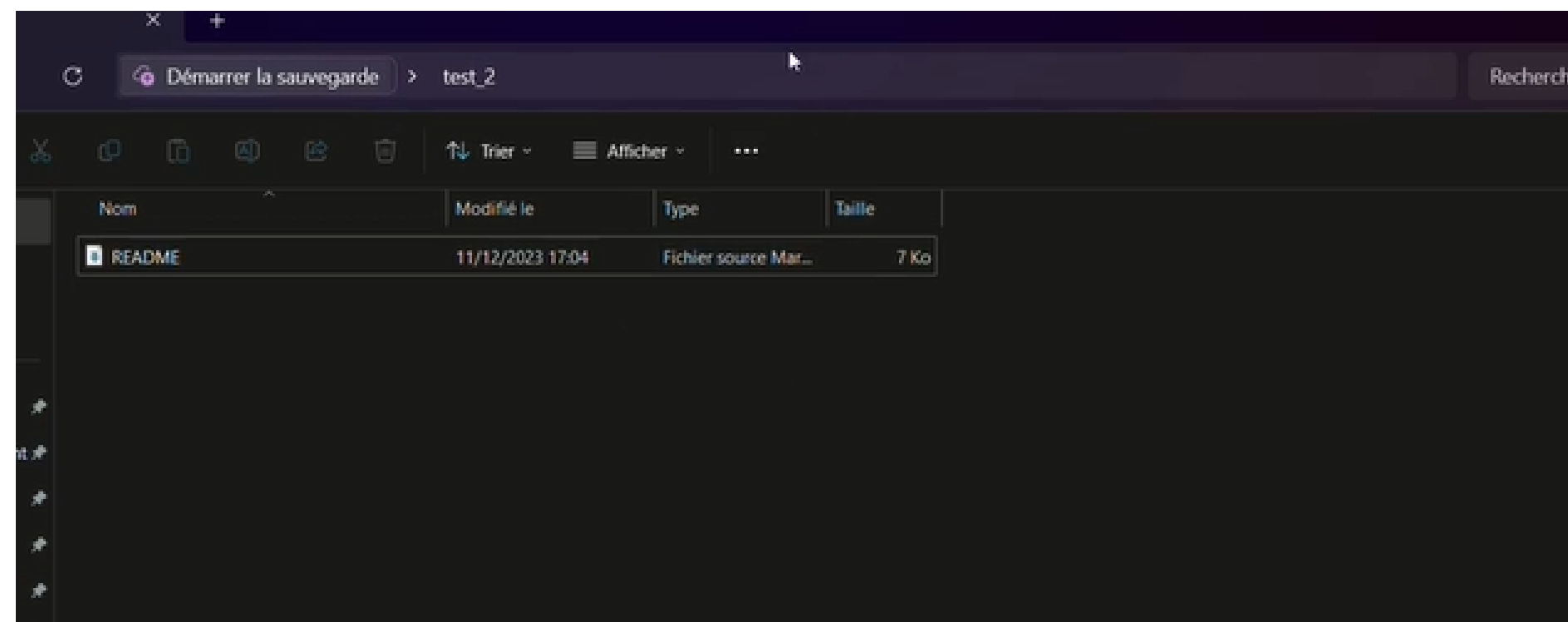


Clonnage d'un projet:

en utilisant : **git clone**

```
PS C:\Users\User0\Desktop> git clone https://gitlab.com/test8963750/test_1.git
Cloning into 'test_1'...
info: please complete authentication in your browser...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
PS C:\Users\User0\Desktop> |
```

le projet s'enregistre au niveau des fichiers





Pour ajouter des fichiers à GitLab :

On utilise les commande suivante:

`git add .`

`git commit -m "commentaire_du_commit"`

`git push origin nom_branche`

```
PS C:\Users\User0\Desktop\test_2> git add .
PS C:\Users\User0\Desktop\test_2> git commit -m "testing page"
[main edd560b] testing page
 1 file changed, 13 insertions(+)
 create mode 100644 test.html
PS C:\Users\User0\Desktop\test_2> git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 471 bytes | 471.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/test8963750/test_2.git
 576ccdd..edd560b  main -> main
PS C:\Users\User0\Desktop\test_2>
```



Pour récupérer des modifications de GitLab :

On utilise les commandes suivantes:
`git pull origin nom_de_la_branche`

```
PS C:\Users\User0\Desktop\test_2> git pull origin main
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 340 bytes | 37.00 KiB/s, done.
From https://gitlab.com/test8963750/test_2
 * branch                main                -> FETCH_HEAD
    edd560b..a203b02      main                -> origin/main
Updating edd560b..a203b02
Fast-forward
 code.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 code.py
PS C:\Users\User0\Desktop\test_2> |
```



Comment changer de branches:

On utilise les commande suivante:

`git init`

`git switch nom_de_la_branche_actuelle`

`git checkout -b nom_de_la_nouvelle_branche`

```
Windows PowerShell
PS C:\Users\User0\Desktop\test_2> git init
Reinitialized existing Git repository in C:/Users/User0/Desktop/test_2/.git/
PS C:\Users\User0\Desktop\test_2> git switch main
Already on 'main'
Your branch is up to date with 'origin/main'.
PS C:\Users\User0\Desktop\test_2> git checkout -b dev_branch
Switched to a new branch 'dev_branch'
PS C:\Users\User0\Desktop\test_2> |
```




Suivi des Problèmes (Issues):

- **Création d'une issue:** Découvrez comment créer une issue pour signaler des bugs, demander des fonctionnalités, ou discuter de questions spécifiques liées au projet.
- **Attribution et suivi des problèmes:** Assignez des issues à des membres de l'équipe pour clarifier les responsabilités et suivez l'évolution des problèmes au fil du temps.
- **Utilisation des labels et des milestones:** Utilisez des labels pour catégoriser et organiser les issues, et définissez des milestones pour regrouper les problèmes liés à des versions spécifiques du logiciel.



Cas des merges:

- Si deux branches différentes ont modifier au sein du même fichier après le push on obtient le message suivant:

```
add
PS C:\Users\User0\Desktop\test_2> git add .
PS C:\Users\User0\Desktop\test_2> git commit "fourth commit"
error: pathspec 'fourth commit' did not match any file(s) known to git
PS C:\Users\User0\Desktop\test_2> git commit -m "fourth commit"
[main 1c2f439] fourth commit
1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\User0\Desktop\test_2> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 277 bytes | 138.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/test8963750/test_2.git
   bf60f17..1c2f439  main -> main
PS C:\Users\User0\Desktop\test_2> git pull origin dev_branch
From https://gitlab.com/test8963750/test_2
 * branch                dev_branch -> FETCH_HEAD
Auto-merging code.py
CONFLICT (content): Merge conflict in code.py
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\User0\Desktop\test_2>
```

```
merge branch 'dev_branch' of https://gitlab.com/test8963750/test_2

# Conflicts:
#   code.py
#
# It looks like you may be committing a merge.
# If this is not correct, please run
#   git update-ref -d MERGE_HEAD
# and try again.
#
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Your branch is up to date with 'origin/main'.
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#   modified:   code.py
#
```



Cas des merges:

- Supprimez les marqueurs de conflit <<<<<<<, =====, et >>>>>>> et ajustez le code selon vos besoins.
- Après on exécute git add. et git merge

Et puis on ajoute ces modifications à l'aide des commandes de push qu'on a vu

```
Accepter la modification actuelle | Accepter la modification entrante | Rejeter les deux modifications | Comparer les modifications
<<<<<<< HEAD (Modification actuelle)
print("Hello, alo ")
=====
print("Hello, tous le monde")
print("testing test")
>>>>>> 5517f1a0392b35af25e9f5647ecaa7110e08eb42 (Modification entrante)
```

```
Automatic merge failed; fix conflicts and then commit the result.
PS C:\Users\User0\Desktop\test_2> git add .
PS C:\Users\User0\Desktop\test_2> git merge --continue
```



Intégration Continue (CI/CD) avec GitLab:

Configuration de l'intégration continue:

- **Mise en place d'un pipeline CI/CD:** Créez un fichier `.gitlab-ci.yml` pour définir les étapes du pipeline CI/CD. Chaque modification déclenche automatiquement le pipeline pour construire, tester, et déployer le code.
- **Utilisation des runners:** Configurez et utilisez des runners GitLab pour exécuter les jobs du pipeline, assurant ainsi l'automatisation des processus.



Intégration Continue (CI/CD) avec GitLab:

Tests Automatisés:

- **Intégration de tests automatisés dans le pipeline:** Intégrez des tests automatisés dans le pipeline CI/CD pour garantir la qualité du code avant le déploiement.
- **Analyse des résultats:** Explorez les résultats des tests pour identifier et résoudre les problèmes potentiels.



Intégration Continue (CI/CD) avec GitLab:

Déploiement Automatisé:

- **Configuration des étapes de déploiement:** Définissez les étapes de déploiement dans le pipeline pour automatiser le déploiement du code sur des environnements spécifiques.
- **Stratégies de déploiement:** Explorez les différentes stratégies de déploiement, telles que le déploiement progressif, le déploiement complet, etc.



Collaboration et Communication:

Collaboration sur GitLab:

- **Gestion des merge requests:** Apprenez à créer des merge requests pour proposer des modifications au code, permettant ainsi aux autres membres de l'équipe de les examiner avant la fusion.
- **Revue de code:** Pratiquez la revue de code en examinant les modifications apportées par d'autres membres de l'équipe et en fournissant des commentaires constructifs.



Sécurité et Bonnes Pratiques:

Gestion des Accès et des Permissions:

- **Configuration des rôles et des permissions:** Définissez des rôles et attribuez des permissions spécifiques aux membres de l'équipe pour garantir un accès approprié aux ressources du projet.
- **Authentification à deux facteurs:** Renforcez la sécurité en activant l'authentification à deux facteurs (2FA) pour les comptes utilisateur.



Sécurité et Bonnes Pratiques:

Sécurité du Code:

- **Analyse statique du code:** Intégrez des outils d'analyse statique du code dans le pipeline CI/CD pour identifier les vulnérabilités potentielles.
- **Gestion des secrets:** Protégez les informations sensibles en utilisant la gestion des secrets de GitLab pour stocker et partager des informations telles que des clés API et des jetons.



Merci !!

