# **GITLAB CI/CD : TP & Correction**

# **TP1**

* Installation de git
* Création d’un projet git
* Créer et ajouter des fichiers dans le staging du commit
* Commit en local sur une branch “dev”
* Afficher l’historique des commits (CLI)
* Effectuer des modifications
* Afficher les différences avec le HEAD
* Commit les modifications
* Créer des modifications sans commit
* Les annuler sans garder celle-ci
* Pour les plus rapides merge dev à master (git merge)

# Correction TP1 : découverte de Git

Réponse basée sur un système Debian.

## Installation de Git

* apt update : on met à jour les dépôts de package
* apt install -y git : on installe le package git
* vim ~/.gitconfig : on paramètre notre fichier de configuration Git

## Création d'un projet Git

* mkdir monprojet && cd monprojet : on créer le dossier qui va contenir le projet (niveau système de fichier)
* git init : on initialise un répertoire git local au dossier (le fameux .git)

## Ajout de fichier sur le staging du commit

* git add votrefichier : on ajoute le fichier donnée en paramètre dans l'espace de "stage" qui va servir à construire votre commit.
* git status : vous pouvez voir votre fichier dans l'espace de staging à ce moment

## Créer un commit sur la branch "dev"

* git checkout -b dev : permets de passer sur la branche de dev (et va la créer au passage)
* git commit -m "mon premier commit" : on créer notre premier commit

## Afficher l'historique des commits

* git log : vous permets de consulter l'historique des commits

## Effectuer des modifications & afficher la différence

* On effectue des modifications dans un fichier précédemment commit
* git diff HEAD : va nous afficher les différences entre la dernière version commit et la version actuellement sur le FS

## Commit les modifications

* git add votrefichier : vous ajoutez le fichier dans lequel vous faites les modifications dans l'espace de stage.
* git commit -m "mes modifs" : vous commitez les modifications précédemment ajoutées dans le stage.

## Annuler les modificaitons

* git reset --hard : va supprimer les dernières actions en supprimant aussi les modifications faites (--hard)

## Bonus : merge dev à master

* git checkout -b master : on se rend sur master
* git merge dev : on merge la branche dev sur master

# **TP2 – Découverte de Gitlab.com**

* Création d’un compte sur gitlab.com
* Configuration du compte (clé SSH : ssh-keygen -t ed25519)
* Création d’un projet
* Créer un dashboard et ajoutez des “issues”
* Commit et push le fichier (TP N°1) sur une branche “dev”
* Merge Request pour mettre sur Master lié à une issue
* Afficher l’historique des commits (GUI + CLI)
* Accepter la MR sur gitlab.com

## Création de compte sur Gitlab.com

* Compléter le formulaire sur la page d'inscription de [Gitlab.com](https://gitlab.com/users/sign_in?redirect_to_referer=yes)

## Ajouter une clé SSH à son compte

* ssh-keygen -t ed25519 : pour générer une paire de clé SSH (une publique et une privé). La privé doit rester sur votre poste, il ne faut jamais la communiquer pour des raisons de sécurité.
* Sur l'interface web l'ajouter dans la partie profil SSH keys

## Créer un projet

* Dans projet : Mes projets > New project

## Créer un ticket dans un board

* Créer un board : Dans son projet > menu > issues > boards
* Créer une issue : Dans son projet > menu > issues > boards > news

## Push le projet TP1

* cd projet\_TP1 : On se rend dans le projet
* git status : On vérifie que rien ne reste à commit. Sinon on commit comme vu dans le TP1.
* git remote add origin <SSH URL de votre dépot> : On ajoute le remote du projet créé.
* git push -u origin dev : On pousse le code commit localement vers le repos distant

## Créer une Merge Request

* Dans le projet > Menu > Merge Request > Nouvelle > dev vers master
* Donner les informations sur la MR, titre information etc...
* Tag l'issue dans la merge request en écrivant simplement #ID\_issue
* Une fois la MR créé vous pouvez consulter l'ensemble des modifications qu'elle contient dans l'onglet "commits" ou "modifications"

## Accepter la MR

* Accepter la MR
* Gitlab va automatiquement merge le code sur master

# **TP3 - Installation de Gitlab CE Omnibus**

* Créer une machine virtuelle (2 CPUs / 4Go RAM min)
* Installer GitLab CE
* Démarrer GitLab CE
* Créer le compte Admin

## Créer une machine virtuelle

* Créer une machine virtuelle avec au minimum :
  + 4 Go de RAM
  + 2 CPUs
  + 25 Go de disque

## Installer Gitlab CE

* apt update && apt install -y curl openssh-server ca-certificates : On installes les packets nécessaire à l'installation de Gitlab
* curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash : On lance le script d'installation qui va ajouter les dépôts Gitlab pour pouvoir l'installer.
* sudo EXTERNAL\_URL="https://<VOTRE ENDPOINT>" apt install gitlab-ce : On lance l'installation de Gitlab en définissant avant la variable de votre endpoint (IP ou Host).

## Démarrer Gitlab CE

* systemctl start gitlab-runsvdir.service : On lance le démon Gitlab avec systemd

## Création du compte administrateur

* Dans votre navigateur, vous rendre sur votre IP ou votre hostname.
* Définissez un mot de passe pour l'administrateur
* Vous pouvez vous connecter ensuite avec l'utilisateur "root" avec le mot de passe défini à l'étape précédente

# **TP4 - Administration de Gitlab CE Omnibus**

* Ajouter un message général pour avertir d’une maintenance
* Créer 2 utilisateurs (un admin, un standard)
* Vérifiez avec la partie monitoring que tout fonctionne (EN CLI)
* Redémarrer le composant Nginx
* Vérifiez le status (EN CLI)
* Redémarrer tout le GitLab (avec systemd ou gitlab-ctl)
* Exécutez 2 appels APIs

## Ajouter un message général d'alterte

* Avec l'API :
  + curl --data "message=Deploy in progress&color=#cecece" --header "PRIVATE-TOKEN: <your\_access\_token>" "https://gitlab.example.com/api/v4/broadcast\_messages"
* Sur l'interface web :
  + Aller dans administration > message > nouveau message

## Créer deux utilisateurs

* Aller dans le panel d'administration
* Dans la partie "utilisateurs", faire créer utilisateurs

## Vérifier le fonctionnement de Gitlab

* systemctl status gitlab-runsvdir.service : On vérifie le fonctionnement du démon Gitlab avec systemd

## Redémarrer le composant Nginx

* gitlab-ctl restart nginx : On redémarre uniquement le composant Nginx, sans couper les autres

## Executer deux appels API

* Aller dans votre profil utilisateur et générez un token en choissisant le niveau de droit que vous souhaitez avoir

Deux exemple, voir <https://docs.gitlab.com/ee/api/api_resources.html> pour plus d'explications.

* curl --header "PRIVATE-TOKEN: <your\_access\_token>" "https://gitlab.example.com/api/v4/projects/4/repository/commits" : On list les commits du projet avec l'id 4
* curl --header "PRIVATE-TOKEN: <your\_access\_token>" "https://gitlab.example.com/api/v4/runners" : List les runners

# **TP\_Docker**

* Installer Docker
* Créer un conteneur
* Créer un Dockerfile
* Lister les images locals

## Installer Docker

* apt update : On met à jour la liste des packets
* apt install apt-transport-https ca-certificates curl gnupg-agent software-properties-common : On installe les prérequis
* curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add - : On récupère la clé GPG des repos Docker
* add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian$(lsb\_release -cs) stable" : On ajoute le repository de Docker dans ceux de votre Debian
* apt update && apt install docker-ce docker-ce-cli containerd.io : On installe Docker et le Runtime de celui-ci.

## Créer un conteneur

* docker run -d -p 1234:2368 ghost : On créer un conteneur Ghost en mappant le port 2368 sur le host

## Créer un dockerfile et construire l'image

* Créer le dockerfile ci dessous :

FROM alpine

CMD ["echo", "Hello Sup De Vinci"]

* docker build -t hello : On construit l'image à partir du fichier dockerfile créé précédament
* docker run hello : On démarre l'image créé précadament

## Lister les images

* docker images : On list les images stocké en local

# **TP5**

* Installer un runner “shell”
* Donner un tag “cli” au runner
* Créer un .gitlab-ci.yml qui va juste faire un :
  + echo “Mon premier job !” dans un premier job
  + echo “Mon second job !” dans un second job
* Créer un second runner “Docker”
* Lui donner le tag “conteneur”
* Ajouter dans le .gitlab-ci.yml
  + Un troisième job qui est dépendant des deux premiers et qui s'exécute uniquement sur Master
  + echo “Mon job final !”

**Correction fichier .coonfig-ci.yml**

stages:

- demoshell

- democontainer

hello1:

stage: demoshell

tags:

- toto

script:

- echo "Hello job1 !"

- exit 3

hello2:

stage: demoshell

tags:

- toto

script:

- echo "Hello job2 !"

helloformdocker:

stage: democontainer

image: bash

tags:

- container

script:

- echo "Hello from Docker !"

only:

- master

dependencies:

- hello1

- hello2

# **TP6 - Ma première chaine CD**

1. Push votre projet « shell » du TP précédent
2. Créer un CD qui va :
   1. Tester les scripts que vous avez push
      1. Files=(git diff-tree --no-commit-id --name-only -t $CI\_COMMIT\_SHA)
   2. Vérifier la présence des scripts ajoutés dans le README
      1. Utiliser grep
   3. Créer une archive de vos scripts
      1. Tar -cvf archive.tar script/
      2. Uniquement sur Master
      3. L’artefact doit être disponible 1 semaine

**Correction fichier .coonfig-ci.yml**

stages:

- QA

- Documentation

- Delivery

shellexectest:

stage: QA

tags:

- shell

script:

- files=`git diff-tree --no-commit-id --name-only -r $CI\_COMMIT\_SHA scripts/`

- for i in $files; do echo $i; done

- for i in $files; do bash $i; done

Readme:

stage: Documentation

tags:

- shell

script:

- files=`git diff-tree --no-commit-id --name-only -r $CI\_COMMIT\_SHA scripts/`

# - for i in $files; grep $i README.md; done

Archive:

stage: Delivery

tags:

- shell

script:

- tar -cvf archive.tar scripts/

only:

- master

artifacts:

name: "$CI\_JOB\_NAME-$CI\_COMMIT\_REF\_NAME"

paths:

- archive.tar

expire\_in: 1 week