|B2| TP N° 3: GIT

Belerge Jérémy BTS SIO 2

Généralités

Qu'est-ce que Git ?

Git est un système de gestion de versions décentralisé qui permet de suivre les modifications apportées à des fichiers et des projets informatiques. Il est largement utilisé par les développeurs de logiciels pour collaborer sur des projets et gérer l'historique des modifications mais également pour l'IaC.

Pourquoi utiliser Git ?

Suivi des modifications : Git permet de suivre les modifications apportées aux fichiers, ce qui est essentiel pour la collaboration et la gestion de versions.

Collaboration : Plusieurs personnes peuvent travailler sur le même projet simultanément.

Historique : Git maintient un historique des modifications, ce qui facilite le suivi des évolutions d'un projet.

Branches : Vous pouvez travailler sur différentes fonctionnalités ou corrections de bogues dans des branches distinctes.

Gestion de conflits : Git permet de gérer les conflits lors de la fusion (merge) de modifications concurrentes.

Sommaire

Consignes:

-GIT c'est quoi?

Qu'est-ce que Git ?

Git est un système de gestion de versions décentralisé qui permet de suivre les modifications apportées à des fichiers et des projets informatiques. Il est largement utilisé
par les développeurs de logiciels pour collaborer sur des projets et gérer l'historique des
modifications mais également pour l'IaC.

Pourquoi utiliser Git ?

Suivi des modifications : Git permet de suivre les modifications apportées aux fichiers, ce qui
est essentiel pour la collaboration et la gestion de versions.
Collaboration : Plusieurs personnes peuvent travailler sur le même projet simultanément.
Historique : Git maintient un historique des modifications, ce qui facilite le suivi des évolutions d'un projet.
Branches : Vous pouvez travailler sur différentes fonctionnalités ou corrections de bogues
dans des branches distinctes.
Gestion de conflits : Git permet de gérer les conflits lors de la fusion (merge) de modifications concurrentes

Consignes:

-LAB0

Lab0

- Installation de Git et vérification de la bonne installation
- création du répertoire siosisr
- Copier les fichiers du dossier appli (Moodle)dans siosisr
- Initialiser le repository
- Vérifier que votre branche est bien main ,sinon changer
- Créer le fichier README.md
- Ajouter votre fichier dans le staging (1)
- Faites un commit (2)
- Créer le fichier `.gitignore` afin de ne pas tenir compte des fichiers log
- Créer un fichier control.log puis ajouter tous les fichiers au staging ,vérifier qu'il n'est pas traquer
- Créer la release 1.0.0 à l'aide de tags

Consignes:

-LAB0

->Installation de GIT :
apt install git –y

->Vérifier la version de Git :
git --version

->Création du répertoire de travail :
mkdir siosisr
cd siosir (on se déplace dans le répertoire)

->Création d'un fichier test.txt
nano test.txt

->Initialisation du dépôt GIT :
git init

->Configuration de la branche principale
git branch -M main (cela renomme la branche principale en main)
git branch (Vérifie que la branche active est bien main)

->Création du fichier README
echo "# Projet SIOSISR" > README.md (à l'intérieur il y sera écrit Projet SIOSISR)

Consignes:

-LAB0

->Ajout du fichier au suivi Git:
git add README.md

->Création du premier commit :
git commit -m "Ajout du fichier README.md" (enregistrée les modifications avec un message de commit)

->Ajout du .gitignore
echo "*.log" > .gitignore # (ignore tous les fichiers avec l'extension .log)
git add .gitignore
git commit -m "ignorer les fichiers log" #(commit pour le fichier .gitignore)

->Ajout du fichier de control.log (test)
echo "Log de test" > control.log #(Créer un fichier log pour tester le gitignore)

->Ajout d'un tag
git tag -a v1.0.0 -m "Release 1.0.0" #(créer un tag versionné, ici 1.0.0)
git tag (permet de lister les tags existants)

Consignes:

-LAB0

->Configuration du dépôt distant et push:
git remote add origin https://github.com/tonpseudo/nomdurepertoire # Lier au dépôt GitHub distant

Exemple : https://github.com/Jrm-blg/TP-git

git push -u origin main (Envoyer la branche principale)
git push origin v1.0.0 (Envoyer le tag vers GitHub)

Consignes:

-LAB1

Lab1

- Rendez-vous dans votre répertoire siosisr
- Créer une branche feature1 dans laquelle vous modifierez index.html en y ajoutant votre nom comme firstname et prénom comme lastname, commiter
- Merger la branche feature1 (à supprimer après)avec votre branche main.
- Créer une branche fix5 dans laquelle vous modifierez `script.sh` puis commiter.
- Revenir dans la branche main et modifier le fichier script.sh (même ligne avec contenu différent),commiter.
- Merger la branche main avec la branche fix5,que constatez-vous ?
- Régler le conflit ,commiter la modification finale puis supprimer la branche fix5.

> **Pensez à effectuer régulièrement les cmdes (1) et (2) du Lab0**

Consignes:

-LAB1

->Création d'une branche de développement (feature1):
cd siosisr (se rendre dans le répertoire du projet)
git checkout -b feature1 (Créer la branche feature1 et basculer dessus)
git add test.txt (Ajouter test.txt à l'index)
git commit -m "Ajout prénom et nom dans test.txt" (Commit du fichier avec message descriptif, c'est simplement le commentaire du fichier)

->Fusion de la branche feature1 dans main :
git checkout main (Bascule sur la branche principale)
git merge feature1 (Fusionne la branche feature1 dans main)
git branch -d feature1 (Supprime la branche feature1 après la fusion)

->Création d'une branche de correction (fix5) :
git checkout -b fix5 (Créer la branche fix5 et basculer dessus)
git add script.sh (Ajout du fichier modifié script.sh)
git commit -m "Modification de script.sh dans fix5" (Commit des changements apportés)

->Modification du fichier script.sh dans la 'main':
git checkout main (Revenir dans la branche principale)
nano script.sh (Modifier la même ligne que dans fix5 , différemment)
git add script.sh (Ajout du fichier modifié)
git commit -m "Modification différente de script.sh dans main" (Commit)

Consignes:

-LAB1

->Test de fusion entre les deux branches:
git merge fix5 (Fusion de fix5 dans la main)

Résultat :
On constate alors qu'un conflit apparait a cause du fichier script.sh , il y a une ligne de différente.
Il est donc impossible de 'merge / 'fusionner' nos de branche.

Solution :
Il suffit simplement de se rendre soit dans la branche fix5, soit dans la branche main, puis de modifier la ligne en conflit afin de rendre les deux fichiers identiques.

->Envoie du fichier au GIT)
git add script.sh (Ajout du fichier au git)
git commit -m "Résolution du conflit entre main et fix5" (Commit de la version corrigée)
git branch -d fix5 (Supprimer la branche fix5)

Consignes:

-LAB2

Lab2

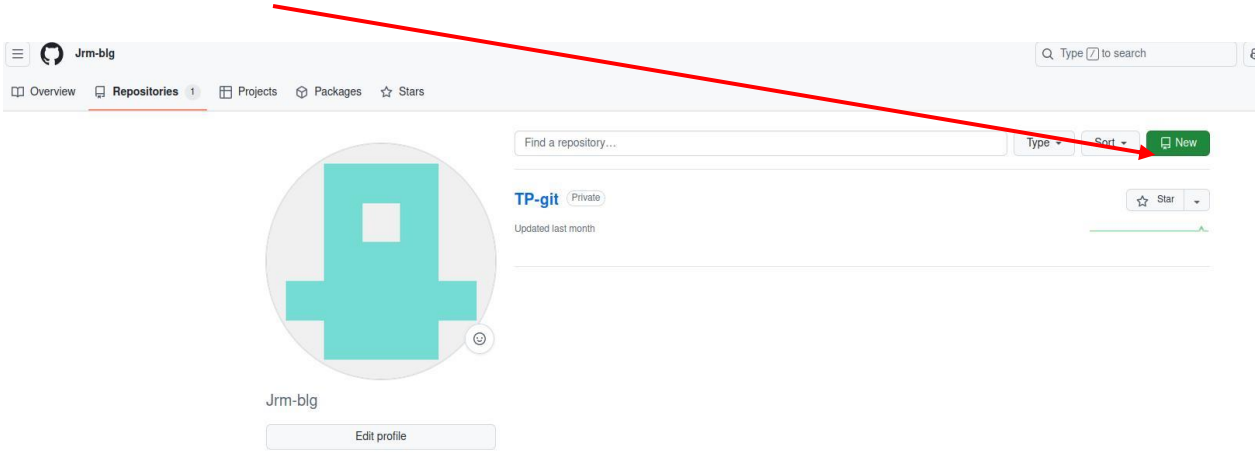
- Créer un compte sur github
- Créer le projet siosisr
- Créer une clé SSH et mettre la clé publique sur Github
- Configurer votre repo local afin de pouvoir communiquer avec votre repo remote. (A faire en ssh)
- Pousser votre projet sur github

Consignes:

-LAB2

->Créer un compte sous GitHub depuis ce lien :
SignUp GIThub

->Créer un projet et le renommer projet SISR:



->Générer une clé SSH et l'ajouter à GitHub

ssh-keygen -t ed25519 -C nom.prénom@mail.com

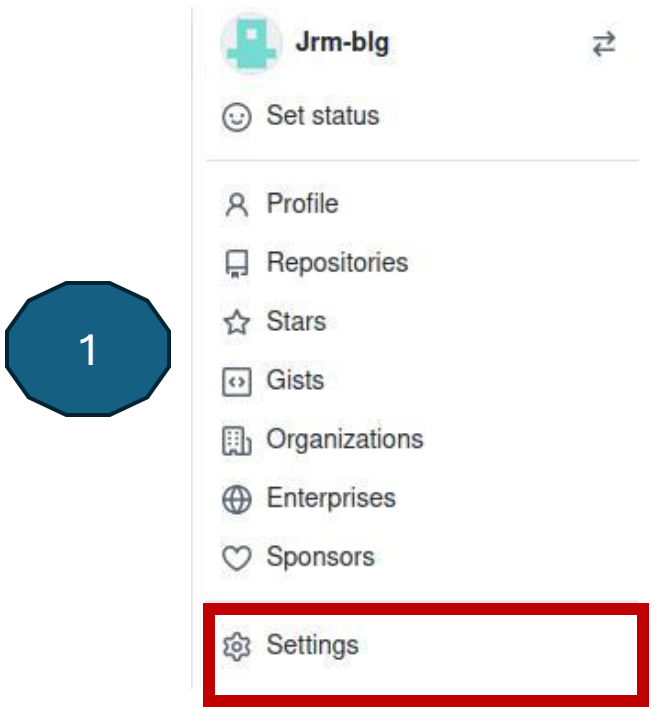
->Afficher la clé publique:
cat ~/.ssh/id_ed25519.pub

La copier puis la rentrer dans le GIT, voir page suivante

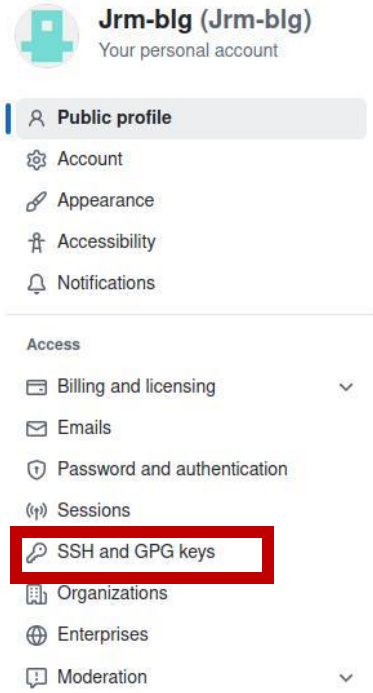
Consignes:

-LAB2

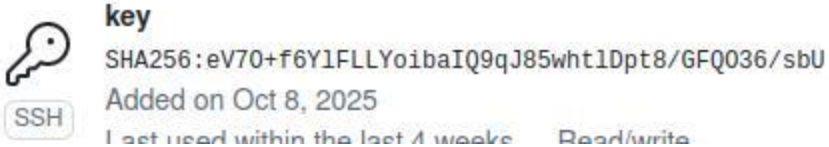
->Trouver / ajouter la clé publique sur le GIT :



1



2



3

Consignes:

-LAB2

->Configurer le dépôt local pour communiquer en SSH avec GitHub :

git remote set-url origin git@github.com:pseudo/siosisr.git

Exemple : git@github.com:Jrm-blg/siosir.git

Attention il faut s'assurer de bien avoir rentrer les commandes:

```
echo "# siosir" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:Jrm-blg/siosir.git
git push -u origin main
```

Il nous restera plus que a push le tout :

```
git remote add origin git@github.com:Jrm-blg/siosir.git
git branch -M main
git push -u origin main
```