Département Informatique
R5.07 – TP Git 1 (1ère partie)
Responsable : X.Roirand
Durée : 90mn machine (pour la 1ère partie)

Le but de ce TP va être de travailler sur un gestionnaire de version appelé Git et de traiter les cas courant d'utilisation. La bonne maitrise d'un gestionnaire de version est la base dans la CI/CD.

1) Connection sur la machine distante

Pour se connecter sur la machine distante, il faut utiliser une connections ssh. C'est une connexion sécurisée qui permet de ne pas faire passer le mot de passe en clair sur le réseau. Vous pouvez utiliser le programme « putty » sur windows ou « ssh » sur linux.

La connexion se fait en 2 étapes, dans le cadre de nos TP. il faut d'abord se connecter sur une sandbox dont l'adresse IP est 149.202.85.73. Le login est student et le mot de passe ?Student_56. Une fois sur cette sandbox, connectez-vous sur votre machine distante, pour cela il faut connaitre le numéro que l'enseignant vouxs a donné, et ajouter ce numéro au nom : machine pour obtenir le nom de la machine sur laquelle vous devez vous connecter.

Exemple : si votre numero est le 13 alors il faut faire un ssh sur la machine machine13

Ensuite le login et mot de passe sur votre machine distante sont :

login : student
mot de passe : ?<votre numero>Student_56

donc si votre numero est le 13 alors le mot de passe est ?13Student 56, comme suit :

login : student
mot de passe : ?13Student_56

Une fois sur votre machine distante, vous pouvez l'administrer totalement.

Pour pouvoir noter le TP, il faut que vous mettiez dans un fichier nommé /root/info.txt
Votre nom et prénom, ou vos noms et prénoms si vous êtes plusieurs. Attention, pas de fichier = zéro !

Toute le TP doit être fait en tant que student.

Récupérer un projet Git hébérgé sur Gitlab :

Pour récupérer un projet Git hébergé sur Gitlab nous avons besoin de 3 choses :

- Un projet hébergé sur GitLab
- Un espace ou cloner le projet en local sur votre machine (ici ca sera la machine qui est mis à votre disposition pour le TP).
- Une paire de clé privée/publique pour pouvoir communiquer avec Gitlab (voir détail ci-dessous).

Création du projet :

Pour créér le projet sous Gitlab, il faut vous enregistrer d'abord et ensuite créér votre projet,

allez sur gitlab.iu-vannes.org, enregistrez vous et créer votre projet (a partir de blank, pas à partir d'un template). Assurez vous que le projet porte dans son nom le numéro de votre qui a été donné par votre enseignant sinon vous risquez d'avoir 0 comme note. Le projet doit être en niveau de visibilité « private » et la case « Initialize repository with a README" doit être cochée.

Lorsque la création du projet est finie, il doit vous afficher un certain nombre d'informations, et entre autre la commande pour créer un nouveau repository en local qui sera une copie du projet que vous venez de créer.

On passe maintenant à la creation de la paire de clé:

En règle générale, la paire de clé est le moyen privilégié pour communiquer avec le repository (ici c'est Gitlab le repository) mais il arrive parfois qu'on utilise un user/password.

La paire de clé permet faire la communication entre votre client Git et le repository. Pour que cela fonctionne, il faut que vous ayez la clé privée coté client Git et la clé publique coté repository. Cela voudra donc dire dans notre cas, qu'il faut mettre votre clé publique dans Gitlab pour que Gitlab « connaisse » votre clé publique. Ca tombe bien, dans Gitlab, il existe un endroit dans chaque projet pour déposer toutes les clés publiques qu'on veut. On peut en mettre plusieurs clés publiques, et heureusement car on est en général plusieurs à travailler sur un même projet, chacun avec sa clé publique.

1) Créez votre paire de clé privée/publique

   Sur la machine mise à disposition pour ce TP, créez votre paire de clé privée/publique de type RSA avec les valeurs par défaut. Pensez bien à définir votre passphrase qui protègera cette paire de clé si vous vous faites volez votre clé privée:

   pas de passphrase = impossible d'utiliser la clé privée volée.

2) Importation de la clé publique dans Gitlab

   Vous devez ajouter votre clé publique dans votre profil Gitlab (cherchez sur internet comment faire si vous ne trouvez pas)

3) Clone du repository

   Vous allez maintenant cloner le repository de votre projet avec la commande git clone:

   Normalement la commande serait:

   git clone git@gitlab.iu-vannes.org:<votre_utilisateur>/<votre_projet> Mais

   comme la configuration ici est un peu particulière, la commande sera:

   git clone ssh://git@192.168.0.143/<votre_username>/<votre_projet>.git

   Il doit vous demander votre passphrase, sinon c'est qu'il y a un souci au niveau de l'enregistrement de votre clé côté Gitlab.

   Vérifiez visuellement et rapidement que vous avez bien sous le repertoire qui vient d'être créé sur la machine de TP, un contenu similaire à celui present visuellement sous:

   http://gitlab.iu-vannes.org/<votre_username>/<votre_projet>

   Vous devez avoir un repertoire nommé comme votre nom de projet et dedans le fichier README.md

Faites un “git status” dans votre projet pour vous assurez que vous tout est ok et ca devrait donner:

```
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

4) Modification d’un fichier

On va maintenant essayer d’ajouter un fichier dans le repository Git, cela pourrait se faire en 2 phases:

Création du fichier local
“Envoi” du fichier créé sur Gitlab

Sauf que ca ne se passe pas tout à fait comme ca, sinon ca pourrait se révéler laborieux si par exemple on devait envoyer 100 fichiers alors il faudrait faire 100 envois. Pour remédier à ca, mais surtout pour pouvoir gérer le fonctionnement géneral du versioning de fichier avec Git, on fait ce qu’on appelle un COMMIT.

A noter que par expérience, je vous conseille de faire des commit lorsqu’une fonctionnalité a été ajouté, ou un bug corrigé ou…toute étape qui permet d’avancer dans le projet. Evitez de mettre plusieurs fonctionnalités ou corrections de bug dans un seul commit, cela permet deux choses :

- Faciliter les code review afin qu’elles soient plus petites et donc faciles à revoir.
- Faciliter le retour en arrière en cas de problème sans chercher dans un commit fourre-tout d’un peu venir le problème.

Ce commit donne donc des operations en 3 phases:

Création du ou des fichier(s) local(aux)
Ajout du ou des fichiers dans le commit
“Envoi” du commit sur Gitlab

Ce commit va donc contenir un ensemble de modifications apportées au projet, cela peut-être:

Un fichier ajouté

ou

Un fichier modifié

ou

Un fichier supprimé

ou

Deux fichiers ajoutés et un supprimé

ou

Un fichier supprimé
Trois repertoires créés
Deux fichiers ajoutés
Dix fichiers modifies

…

Vous voyez que ce commit va contenir un ensemble de modifications au projet et du coup c'est cet ensemble qui va être envoyé en une fois à distance sur Gitlab. Vu que lorsqu'on va developer un programme, on va parfois créér des fichiers, en virer, en modifier mais qu'on ne va pas tout vouloir envoyer vers Gitlab, ca va être à nous à chaque fois de "dire" ce qu'on veut mettre dans le commit qu'on voudra envoyer de ce qu'on voudra garder en local, par ex je veux envoyer vers Gitlab la dernière mise à jour du fichier conf.ini mais je ne veux pas envoyer vers Gitlab le fichier conf.ini.perso qui est un fichier que j'ai créé mais juste pour moi en local.

Du coup on doit rajouter dans le commit toutes les modifications qu'on enverra vers Gitlab, avec les commandes:

git add <nom_du_ou_des_fichiers_modifiés_ou_ajoutés>

cette commande rajoute dans le commit courant (celui qu'on est en train de créer et qui sera envoyé) un fichier que j'ai créé et que je veux ajouter dans le repository Gitlab ou un fichier déjà existant que j'ai modifié.

git rm <nom_du_ou_des_fichiers_a_supprimer>

cette commande rajoute dans le commit courant la suppression d'un fichier déjà existant dans le repository Gitlab, c'est à dire qu'une fois le commit envoyé sur Gitlab, le fichier n'existera plus dans le repository.

Vous allez donc créer votre premier commit. Editez le fichier README.md qui doit contenir l'information suivante:

Projet: …
Nom: …
Prénom: …

Ensuite, préparez le commit en ajoutant dedans le fichier README.md. Si tout s'est bien passé, alors vous devriez voir le fichier README.md present dans la section "Changes to be committed:" lorsque vous faites "git status" sinon, si le fichier README.md est present dans la section "Untracked files:" c'est que vous n'avez pas fait ou bien fait le "git add".

Une fois que ce commit est préparé, il va falloir le créér, on n'a plus aucun fichier modifié ou nouveau ou effacé à mettre dans ce commit donc on peut le créer. Pour ca, on fait un git commit (je vous conseille "git commit -v" qui permet de mieux voir les modifications présentes dans le commit).

Attention, cette action lance parfois l'éditeur par défaut qui est "vi". Si jamais celui -ci ne vous convient pas, vous pouvez changer la configuration de votre projet git avec cette commande (avant de lancer git commit):

git config –global core.editor "nano"

Si jamais vous ne voulez pas utiliser vi et que vous êtes dedans lors de l'édition du commit, pour en sortir tapez :

Esc puis : puis q!

Lors du premier commit dans ce TP, il va demande de renseigner un ou deux info indispensables, avant d'arriver à éditer le commit, mais pas de panique, il vous indique exactement les commandes à taper. Faites ce qu'il vous dit et vous devriez ensuite à pouvoir éditer votre commit.

Lors de cette action git commit, vous allez pouvoir remplir une premiere ligne générale, puis une ligne sans rien, puis plusieurs lignes qui vont préciser le contenu des modifications. Ces lignes sont très importantes et doivent toujours être remplies afin de mieux comprendre à quoi à servi le commit, ex:

Creation des infos du README.md

Le fichier README.md contient maintenant les informations de l'élève et le nom du projet en cours.

Il faut ensuite quitter l'éditeur en sauvegardant et le commit est créé. Il est possible de verifier que le commit a bien été créé avec la commande "git status" qui doit afficher

[…]
Your branch is ahead of 'origin/preprod' by 1 commit.
[…]

Dans Git on travaille toujours sur des branches, on verra plus tard à quoi ca sert. Ici on voit que la branche courante sur laquelle on taf est en avance d'un commit par rapport à la branche dans Gitlab, ce qui est normal car on a créé un commit sur notre ordi mais on ne l'a pas encore envoyé sur Gitlab. On peut aussi faire "git log" pour voir les derniers commit présents dans notre répertoire git en local sur notre branche actuelle, ce qui devrait donner un truc du genre:

commit 7e8c496ea98e7202fcd37b7d64a8f4124e192780 (HEAD -> master)
Author: Xavier Roirand <Xavier.roirand@univ-ubs.fr>
Date: Fri Jan 17 10:33:11 2020 +0100

Creation des infos du README.md

Il faut maintenant envoyer le commit sur Gitlab et verifier qu'il a bien été pris en compte. Pour ca on devrait faire un "git push", sauf que dans la vraie vie, quand on veut pousser un commit, il faut toujours verifier qu'il n'entre pas en conflit avec un commit que quelqu'un d'autre aurait fait et qui contiendrait lui aussi des choses qui porte sur un élément aussi present dans notre commit, ce qui créérait un conflit, ex un collègue qui bosse sur le même projet modifie lui aussi le fichier README.md et il a déjà envoyé sa modification sur Gitlab donc on ne pourra pas pousser notre commit sur Gitlab, il faudra résoudre le problème de conflit avant.

Du coup on ne fait pas un commit mais on fait un "git pull –rebase"

Faites cette commande, et constatez que l'output doit être:

Current branch master is up to date.

car personne d'autre n'a modifié quoique ce soit dans le repository à part vous.

Du coup c'est bon on peut envoyer le commit vers Gitlab avec la commande:

git push

L'output devrait être du genre:

Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 273 bytes | 136.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://gitlab.iu-vannes.org:12220/<votre_user>/<votre_projet>.git
b7ed52f..24eee27 master -> master

La commande “git status” doit donner:

On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

Tout est ok et vous devez voir dans l’interface Gitlab de votre projet, en allant sur

Repository à gauche et ensuite Commits, le dernier commit que vous venez de faire.

Cliquez sur le dernier commit et verifiez visuellement que vous voyez bien les modifications que vous venez de mettre dans le commit.

5) Ajout d’un fichier

Vous allez maintenant rajouter deux fichiers dans votre repository git, un fichier à la racine nommé “info.txt” et deux fichiers dans le dossier “system”, nommé “config1.txt” et “config2.txt”. Remplissez ces 3 fichiers avec 2 ou 3 lignes contenant ce que vous voulez.

Vous allez bientôt créez un commit contenant ces 3 fichiers.

Vérifiez d’abord avec la command “git status” que les 3 fichiers sont bien dans la partie “Untracked”.

Créez ensuite le commit (n’oubliez pas de rajouter les 3 fichiers) avec le commentaire que vous voulez, vérifiez que le commit contient bien les 3 fichiers et les modifications (donc ici il n’y a pas de modification, juste le contenu initial des fichiers) avec la commande “git log” qui affiche la liste des commits du repository, et celui que vous venez de faire se trouve en premier. Puis utilisez la commande “git show xxxxxx” ou
xxxxx est le “commit id” de votre commit qui a été affiché par la commande précedente, ex:

ATTENTION: la commande “git log” affiche dans l’ordre inverse, le dernier est en haut et le plus vieux en bas.

myhost> git log

commit 34b0a206a4feac8779cfc775cee9a58de5ebc5c9 (HEAD -> master, origin/master, origin/HEAD)
Author: Xavier Roirand <xavier@toto.com>
Date: Thu Feb 6 17:22:41 2020 +0100

Re Re Bla bla

commit
24eee27ac4aa61cdad8c3d4d86a95ea978500903
Author: Xavier Roirand <xavier@toto.com> Date: Thu
Feb 6 17:04:14 2020 +0100

[…]

Ici le “commit id” du dernier commit (le 1er affiché) est fera 34b0a206a4feac8779cfc775cee9a58de5ebc5c9 donc on un “git show le contenu 34b0a206a4feac8779cfc775cee9a58de5ebc5c9” pour afficher du commit qu’on vient de créer

Maintenant “poussez” (on dit ce terme pour dire envoyer) vers Gitlab. ATTENTION n’oubliez pas de faire un “git pull –rebase” avant

Vérifiez ensuite dans l’interface Gitlab que vous voyez bien vos 3 nouveaux fichiers dans “Repository” puis “Files”.

6) Effacement d’un fichier

Vous allez maintenant effacer le fichier system/config2.txt, faire un commit avec cet effacement et pousser ce commit et verifier que ce fichier à bien disparu dans Gitlab.