TP 2

Introduction à Git et GitLab

1/ Préambule

Comme vu dans le cours, un ingénieur DevOps, entre autres, doit être une personne hautement pointue sur de nombreuses technologies. Ce TP, nous permettra d'en apprendre une qui est incontournable.

Le développement d'une application/logiciel d'entreprise est en général un travail collaboratif faisant intervenir plusieurs contributeurs. La gestion technique de son code source est alors un problème, car ces contributeurs sont amenés à le modifier en même temps. Il naît donc le besoin d'un outil capable de versionner/historiser et de suivre l'évolution des changements effectués sur ce code source, mais surtout d'assurer la cohérence du travail collaboratif des différents acteurs qui y contribuent par l'ajout, la modification et la suppression des fichiers de code. *Git* est aujourd'hui le leader du marché parmi les outils existants dédiés à cette tâche. C'est un logiciel libre et donc gratuit d'utilisation. Au-delà d'être incontournable à la gestion de code source en contexte collaboratif comme dit, son intérêt reste intact même pour le cas des projets ayant un seul contributeur.

Comme illustré sur la Figure 1, Git propose une architecture décentralisée consistant à disposer d'un repository central, en général hébergé sur un système dédié du marché. Exemple : GitLab ou Github.

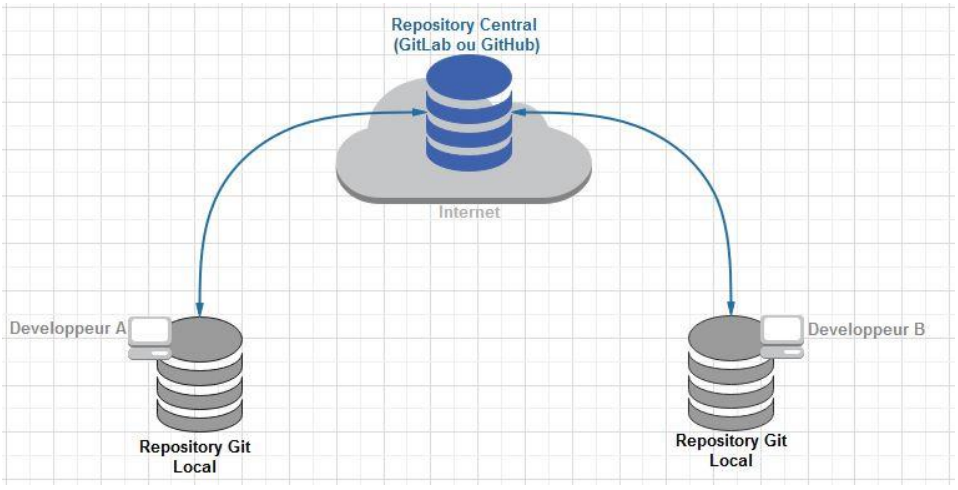


Figure 1 – Architecture décentralisée Git

Le repository central aura alors pour but de recueillir et de centraliser les différents fichiers de code qui lui seront transmis individuellement par chaque repository local (et donc par chaque développeur). De même tout repository local aura la possibilité de récupérer tous les fichiers situés sur le repository central de manière à lui être équivalent à un instant *t*.

Le but de ce TP est de s'initier à l'utilisation de Git et GitLab pour gérer un répertoire de code source par l'exécution de quelques-unes de ses commandes fondamentales.

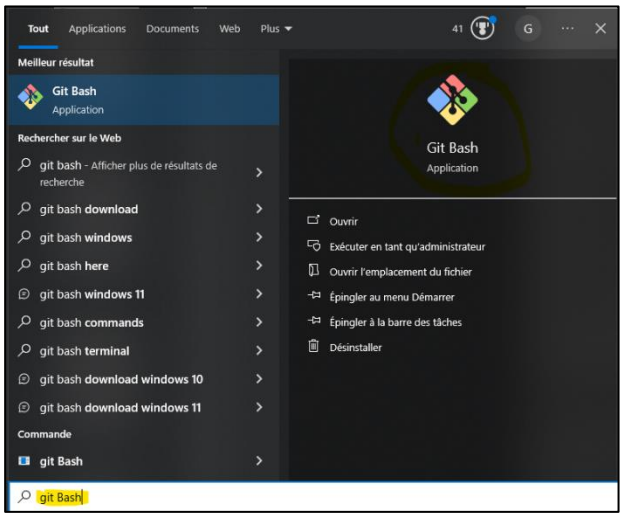
2/ Pratique

2.1/ Installation de Git

a) Téléchargez la version « 64-bit Git for Windows Setup » à partir du lien suivant : https://git-scm.com/download/win

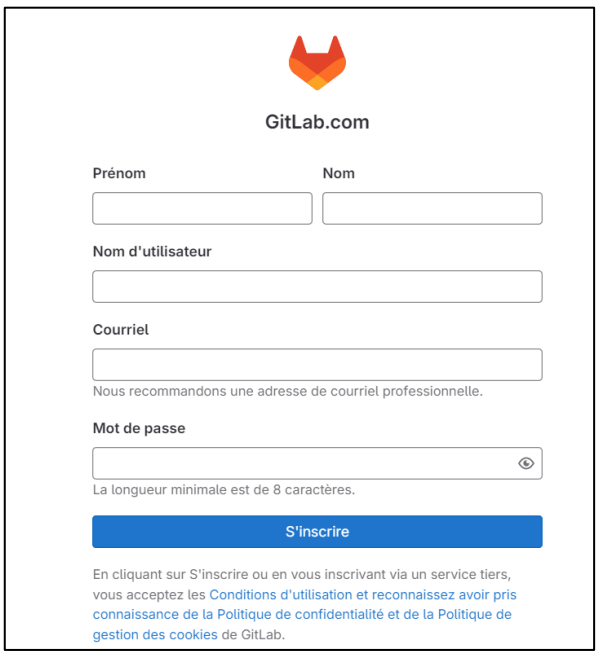
b) Double-cliquez sur le fichier exécutable téléchargé pour lancer l'installation et suivre les étapes

c) L'installation de Git est terminée. Pour afficher sa CLI (Command Line Interface – Interface de Ligne de Commande), allez dans la zone de recherche de la barre de tâche et saisir « Git Bash » :



2.2/ Création d'un compte GitLab

Sur le site https://about.gitlab.com , cliquez sur le bouton « Sign In », puis sur le lien « Incrivez-vous maintenant » pour vous créer un compte gratuit à l'aide de votre email ESIEE.



2.3/ Utilisation de Git

La Figure 2 illustre le workflow de fonctionnement des commandes git telles qu'elles s'appliquent sur tout répertoire **versionné** (vocabulaire indiquant qu'un répertoire est géré par Git).

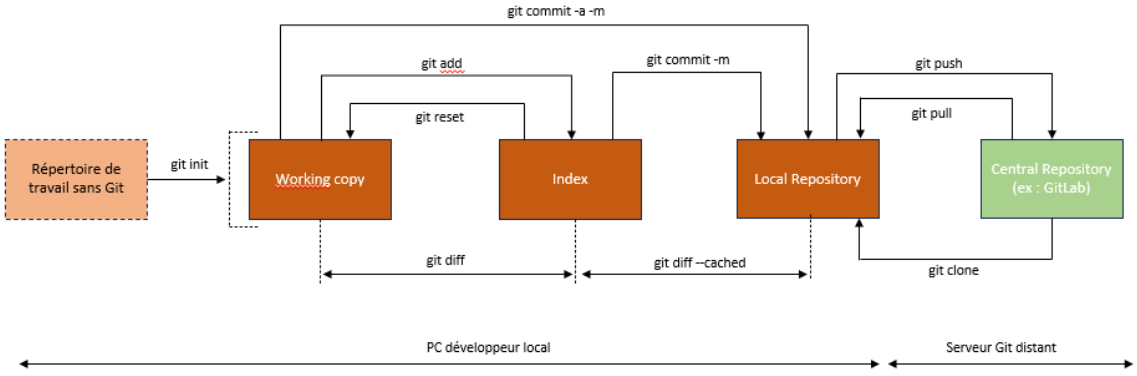


Figure 2 – Commandes Git et fonctionnement

2.3.1/ Création d'un nouveau repository/dépôt

Pour obtenir un repository Git sur votre machine sur lequel travailler, deux options sont possibles :

1. Méthode 1 : création de zéro à partir d'un répertoire de travail windows (typiquement utilisé pour commencer un nouveau projet de développement) ;

2. Méthode 2 : copie (*clone* dans le langage Git) d'un « central repository » sur sa machine afin de disposer d'un Working copy (typiquement utilisé pour contribuer et collaborer avec les développeurs sur un projet en cours).

A/ Examinons la première méthode : création d'un repository de zéro

Ouvrir la CLI git, tel que présenté au 2.1

Pour créer un nouveau « repository git » nommé « monrepo », on utilise la commande « git init monrepo ». Cette commande crée un sous-répertoire « monrepo/.git ». C'est ce sous répertoire « .git » qui gère et maintient les différentes phases (en langage git, *stage*) illustrées à la Figure 2 ci-dessus : Working copy, index, local repository.

Note : toutes les commandes git, se saisissent à la racine du répertoire de travail. C'est-à-dire que le prompt de la CLI doit pointer sur ce répertoire.

Prérequis :

Appliquez les commandes suivantes pour vous identifier sur le système git :

git config --global user.name « Votre prénom nom tel que renseigner dans Gitlab »

git config --global user.email « Votre email tel que renseigner dans Gitlab »

Question 1 : Dans un endroit de votre choix sur votre machine, créer un répertoire windows vide appelé « monprojetjava ». Y initialiser un nouveau repository Git. Et enfin, dans ce repository, créez ou copiez y un fichier de code Java (i.e, utilisez par exemple un des code source que vous avez écrit lors de votre cours sur l'initiation à Java) : *test.java*
Question 2 : Vérifiez avec « git status » l'état dans lequel se trouve votre repository. Vos modifications (l'ajout du fichier test.java) devraient être présentes seulement dans le Working copy.

Question 3 : Ajoutez test.java dans l'index avec « git add test.java ». Cela s'appelle la préparation au commit.
Utilisez « git status » à nouveau pour vérifier que les modifications ont bien été placées dans l'index. Puis, utilisez « git diff --cached » pour observer les différences entre l'index est la dernière version présente dans le local repository (qui est normalement vide).

Question 4 : Commitez votre modification avec « git commit -m "<votre_message_de_commit>" ». Le message entre guillemets décrira la nature de votre modification (généralement ≤ 65 caractères).

Question 5 : Exécutez à nouveau « git status », pour vérifier que vos modifications ont bien été commités. Donc, ne sont plus affichées dans l'index.

Question 6 : Essayez à présent la commande « git log » pour afficher la liste des changements effectués dans votre repository ; combien y en a-t-il ? Quel est le numéro (un hash cryptographique en format SHA1) du dernier commit effectué ?

Question 7 : Créez quelques autres fichiers java toto.java, titi.java... ou alors modifier le fichier test.java plusieurs fois, en commitant chaque modification séparément. Chaque commit doit contenir une et une seule création ou modification de fichier. Effectuez au moins 2 modifications différentes (et donc 2 commits différents). À chaque étape, essayez les commandes suivantes :

- « git diff » avant « git add » pour observer ce que vous allez ajouter à l'index ;
- « git diff --cached » après git add pour observer ce que vous allez committer.

Note : la commande « git commit -a -m "<votre_message_de_commit>" <fichier> » a le même effet que l'application consécutive de « git add <fichier> » suivie de « git commit -m <fichier> ».

Question 8 : Regardez à nouveau l'historique des modifications avec « git log » et vérifiez avec « git status » que vous avez tout commité. Git offre plusieurs interfaces, graphiques ou non, pour afficher l'historique. Essayez les commandes suivantes (Attention : gitg et/ou gitk ne sont pas forcément installés par défaut, donc non susceptibles de fonctionner) :

- git log
- git log --graph --pretty=short
- gitg
- gitk

Question 9 : Allez sur GitLab et créer un projet nommé « monprojetjava ». Récupérez le lien https de ce projet (format : https://gitlab.com/<votre_espace>/monprojetjava.git)

Question 10 : Configurez le repository git local pour lui indiquer qui est son repository central :
« git remote add origin https://gitlab.com/<votre_espace>/monprojetjava.git »

Question 11 : Poussez sur le repository central (GitLab), toutes les modifications précédentes commitées dans le local repository : « git push -u origin master ».

Cette commande est appliquée une et une seule fois pour un nouveau repository git local envoyé pour la première fois sur un repository central **vide**. Lors des prochains envois de modifications, on utilise simplement « git push »

Question 12 : Allez sur GitLab vérifier l'apparition de votre répertoire de travail et la présence des fichiers de code. Naviguez sur l'interface GitLab et dans ce nouveau repository central pour relever quelques constats et remarques. Notamment le traçage de tous les évènements que vous avez réalisés en local.

⇨ **Gestion des retours arrière**

Question 13 : En local, faites une modification d'un ou plusieurs de vos fichiers java et ajoutez-les à l'index avec « git add » (vérifiez cet ajout avec « git status »). Maintenant, supposons que vous avez changé d'avis et ne souhaitez plus commiter ce/ces fichier(s). Exécutez « git reset <fichier> » sur le nom de fichier (ou les noms de fichiers) que vous avez préparés pour le commit ; vérifiez avec « git status » le résultat.

Question 14 : Votre modification a été « retirée » de l'index. Vous pouvez maintenant la jeter à la poubelle avec la commande *git checkout* sur le ou les noms des fichiers modifiés (« git checkout <fichier> »), qui récupère dans l'historique, le contenu de ce fichier correspondant au tout dernier commit avant la modification actuelle. Essayez cette commande, et vérifiez avec « git status » qu'il n'y a maintenant plus aucune modification à commiter et que le contenu du fichier est revenu à son ancien état.

Question 15 : Regardez l'historique de votre repository avec « git log » ; choisissez dans la liste un commit (autre que le premier de la liste). Exécutez « git checkout COMMITID » où COMMITID est le numéro de commit que vous avez choisi. Vérifiez que l'état de vos fichiers java est maintenant revenu en arrière, au moment du commit choisi. Que dit maintenant « git status » ?

Attention, avec « git checkout », le contenu des fichiers de votre Working copy sont automatiquement modifiés par Git pour les remettre dans l'état que vous avez demandé. Si les fichiers modifiés sont ouverts par d'autres éditeurs de texte, il faudra les fermer et réouvrir pour observer les modifications.

Question 16 : Vous pouvez retourner à la version la plus récente de votre dépôt avec « git checkout master ». Vérifiez que cela est bien le cas. Que dit maintenant « git status » ?

B/ Examinons la deuxième méthode : clone d'un repository existant

L'objectif ici est de partir d'un repository central existant sur GitLab pour récupérer une copie afin d'y apporter sa/ses modifications/ajouts personnelles.

Ce repository central est : https://gitlab.com/gkemayo/projetcommun.git. Il contient un fichier nommé tp.java

Question 17 : Clonez ce projet à un endroit de votre choix sur votre machine à l'aide de la commande « git clone https://gitlab.com/gkemayo/projetcommun.git »

Question 18 : En local chacun, faites une modification quelconque sur le fichier tp.java, puis le *commiter* et le *pusher* sur GitLab : « git add .. », « git commit -m … » et « git push »

Si la commande git push échoue, faites « git pull » pour récupérer les modifications de vos collègues qui ont été plus rapide dans l'opération que vous.
Ainsi, vous remarquerez : que le contenu de votre fichier à changer et que vous y retrouver des choses que vous n'avez pas écrites vous-même ; des potentiels conflits de modification dans votre fichier tp.java sont signalés par Git ; l'impossibilité de pusher à nouveau votre fichier sur GitLab en l'état si conflit il y a.

⇨ Gestion des branches

L'utilisation des branches permet d'éviter les problèmes de conflits de modification du même fichier par plusieurs personnes, rencontrés précédemment.

Question 19 : Clonez à nouveau, sur GitLab, le projet « tpcommun » dans un endroit différent du précédent sur votre machine.

Question 20 : Créez une branche personnalisée à l'aide de la commande « git branch <nomdelabranche> » et placez-vous sur cette branche via « git checkout <nomdelabranche> ».
Une commande raccourcie qui fait les deux est : « git checkout -b <nomdelabranche> ».

Ensuite, modifiez le contenu du fichier tp.java, commitez (« git add » et « git commit ») puis « pushez » sur GitLab via « git push -u origin <nomdelabranche> »

NB : Lorsqu'on pousse une branche pour la toute première fois, on utilise la commande ci-dessus, sinon, on fait simplement « git push ».

Que remarquez-vous ? Avez-vous eu des conflits à nouveau ?
Allez-voir sur GitLab la présence de toutes les branches créées par chacun de vous. Switchez entre les branches pour voir les modifications des uns et des autres.

Note de fin : Nous arrivons au terme de notre TP. Mais sachez que Git ne s'arrête pas là. Il existe de nombreuses opérations relativement complexes réalisables sur les branches et au-delà : merge, fusion, suppression, modification, cherry-pick, etc. Mais cela représente des notions avancées que nous n'aborderons pas dans ce TP. Vous pouvez consulter le lien suivant pour aller plus loin : https://gkemayo.developpez.com/tutoriels/git/memo-commandes-git/