GIT : Approfondissement partie 1

Objectif de l'exercice Introduire les participants aux concepts avancés de Git
Pré-requis : Chaque participant doit avoir :
1. Un ordinateur avec Git installé.
2. Un compte sur GitHub.com.
3. Une clef *token* (c.f. https://www.geeksforgeeks.org/git-how-to-solve-remote-invalid-username-or-password-fatal-authentication-failed/)

Travail à rendre
Pour chaque étape, donnez la commande et répondez aux questions

I. Premiers commits

1. Première partie :

a) Connaissances abordées :

- Visualiser le .git
- Git help config
- Git commit -a
- Lors du commit, mettre un message plus long
- Commande linux touch
- Git add *
- Git add -all
- Git status

b) Instructions :

i. Initialisation et Visualisation du .git :

Étape N°1 : Créez un nouveau dossier pour votre projet et naviguez-y à l'aide de la commande cd.

Étape N°2 : Initialiser un nouveau dépôt Git avec git init.

Étape N°3 : Utilisez la commande ls -la pour afficher les fichiers cachés et visualisez le dossier .git. Que voyez-vous à l'intérieur ? A quoi sert ce dossier ?

ii. Aide et Configuration :

Étape N°4 : Tapez git help config pour visualiser la documentation associée à la configuration de Git. Si vous êtes en ligne de commande, entrez la lettre 'q' pour quitter

Étape N°5 : Configurez votre nom et votre e-mail (ceci n'est nécessaire que la première fois que vous utilisez Git sur un ordinateur) :

iii. Création et Ajout de Fichiers :

Étape N°6 : Utilisez la commande touch pour créer trois fichiers : file1.txt, file2.txt, et README.md.

Étape N°7 : Ajoutez quelques lignes de texte dans chacun de ces fichiers à l'aide de votre éditeur de texte préféré.

Étape N°8 : Utilisez git add * pour ajouter tous les fichiers au suivi de version. Vérifiez avec git status

iv. Commit avec Message Long :

Étape N°9 : Effectuez une première validation avec la commande git commit -a. Cela ouvrira un éditeur où vous pourrez écrire un message de commit. Assurez-vous que le message est détaillé, expliquant les fichiers que vous avez ajoutés et leur contenu.

v. Modification et Nouvel Ajout :

Modifiez README.md pour ajouter plus de détails.

Étape N°10 : Créez deux nouveaux fichiers à l'aide de la commande touch: file3.txt et file4.txt. Ajoutez du contenu dans ces fichiers.

Étape N°11 : Utilisez git add --all pour ajouter toutes les modifications et les nouveaux fichiers.

vi. Dernier Commit :

Étape N°12 : Effectuez une autre validation avec git commit -a, et cette fois, utilisez un message encore plus long pour décrire toutes les modifications que vous avez effectuées. (consultez les commandes de *vi* un vieux éditeur de texte qu'on trouve sur linux : https://www.linuxtricks.fr/wiki/guide-de-sur-vi-utilisation-de-vi)

vii. Vérification :

Étape N°13 : Vérifiez avec git status

viii. Différence entre git add * et git add –all :

La différence entre git add * et git add --all réside dans la manière dont ils ajoutent des fichiers au suivi de version et dans les fichiers auxquels ils s'appliquent.

git add * :

C'est une combinaison de Git avec une expansion de shell. Le caractère * est interprété par le shell, et non par Git.
Il ajoutera tous les fichiers du répertoire actuel (en excluant les fichiers commençant par un point, c'est-à-dire les fichiers cachés) au suivi de version.
Il ne prendra pas en compte les suppressions de fichiers.
Si vous êtes dans un sous-répertoire et exécutez cette commande, seuls les fichiers de ce sous-répertoire seront ajoutés, et non l'ensemble du dépôt.

git add --all ou git add -A:

C'est une commande native de Git qui n'a pas d'interaction avec l'expansion du shell.
Elle ajoutera tous les nouveaux fichiers, toutes les modifications et toutes les suppressions du dépôt entier à l'index, quel que soit le répertoire dans lequel vous vous trouvez.
Contrairement à git add *, cette commande prend en compte les fichiers supprimés.

Exemple :
Supposons que vous ayez un fichier caché .secret.txt, un fichier normal normal.txt, et un fichier old.txt que vous avez supprimé depuis votre dernier commit.

Si vous exécutez git add *:
- Seul normal.txt sera ajouté à l'index.
- .secret.txt ne sera pas ajouté car il est caché.
- La suppression de old.txt ne sera pas prise en compte.

Si vous exécutez git add --all:
- normal.txt sera ajouté.
- .secret.txt sera également ajouté.
- La suppression de old.txt sera également prise en compte dans l'index.

2. Fichier .gitignore

Objectif :
Comprendre et maîtriser l'utilisation du fichier .gitignore pour éviter de suivre certains fichiers ou dossiers indésirables dans un dépôt Git.

Création de fichiers et dossiers :

Étape N°14 : Utilisez la commande touch pour créer les fichiers suivants : file2.log, temp.txt, notes.txt.

Étape N°15 : Créez également un dossier nommé *logs* et à l'intérieur, ajoutez quelques fichiers .log en utilisant la commande touch.

Étape N°16 : Créez également un dossier nommé *src* et à l'intérieur, ajoutez quelques fichiers .log en utilisant la commande touch. Donnez l'arborescence avec les dossiers et les fichiers

Configuration de .gitignore :

Étape N°17 : Créez un fichier .gitignore dans votre dépôt, à la racine

Étape N°18 : À l'intérieur du fichier .gitignore, avec nano ajoutez les lignes suivantes :

```
*.log
temp.txt
```

Étape N°19 : Sauvegardez et fermez le fichier.

Vérification du statut de Git :

Étape N°20 : Exécutez la commande git status Quels fichiers voyez-vous prêts à être suivis ?

Ajout de tous les fichiers :

Étape N°21 : Utilisez la commande git add . pour ajouter tous les fichiers de votre répertoire.

Étape N°22 : Exécutez à nouveau git status. Quels fichiers ont été ajoutés à l'index ?

Validation des changements :

Étape N°23 : Faites un commit de vos changements avec la commande git commit -m "Initial commit with .gitignore".

Étape N°24 : Utilisez git log pour voir l'historique de vos commits.

3. Exploration de l'historique avec git log

Objectif :
Apprendre à utiliser la commande git log pour explorer l'historique des commits d'un dépôt Git, ainsi que comprendre diverses options de filtrage et de formatage de cette commande.

Plusieurs commits :

Étape N°25 : Créez au moins trois commit avec le fichier Readme.md, en y ajoutant une ligne à chaque fois. Appelez les commits, Update 1, update 2 ….

Exploration de l'historique des commits :

Étape N°26 : Testez ces commandes, et donnez leurs significations :

- git log
- git log -n 2
- git help log
- git log –oneline
- git log -p Readme.md
- git log -n 1 -p Readme.md

Comparaison des modifications :

Étape N°27 : Modifiez à nouveau le fichier Readme.md sans commiter.

Étape N°28 : Utilisez la commande git diff pour voir les modifications non commitées de votre fichier.

II. Retour en arrière

Objectif :
Comprendre et pratiquer différentes méthodes pour revenir à un commit précédent dans Git.

1. Préparation de l'environnement :

Étape N°29 : Créez un nouveau dossier nommé git-practice pour votre projet et naviguez-y à l'aide de la commande cd.

Étape N°30 : Initialisez un nouveau dépôt Git avec git init.

Étape N°31 : Créez un fichier notes.txt avec quelques lignes de texte.

Faites un commit de votre fichier avec le message "Initial commit".

2. Ajout de modifications :

Étape N°32 : Ajoutez quelques lignes supplémentaires à votre fichier notes.txt.

Étape N°33 : Faites un nouveau commit avec le message "Second commit".

Étape N°34 : Répétez l'opération pour un troisième commit intitulé "Third commit".

3. Utilisation de git checkout :

Étape N°35 : Utilisez la commande git log pour afficher l'historique des commits et notez les SHA des commits.

Étape N°36 : Utilisez la commande git checkout [sha-du-second-commit] pour revenir au second commit. Qu'observez-vous dans le fichier notes.txt ?

4. Création d'une branche à partir d'un commit précédent :

Étape N°37 : Depuis votre état actuel (tête détachée), créez une nouvelle branche nommée feature-branch avec git checkout -b feature-branch.

5. Utilisation de git reset :

Étape N°38 : Retournez sur la branche principale avec git checkout main.

Étape N°39 : Utilisez la commande git reset --hard [sha-du-second-commit]. Que se passe-t-il dans l'historique et dans le fichier notes.txt ?

6. Annulation des effets d'un commit avec git revert :

Étape N°40 : Ajoutez quelques lignes au fichier notes.txt et faites un nouveau commit.

Utilisez la commande git revert HEAD pour annuler les modifications du dernier commit.

7. Résumé :

Pour revenir à un commit précédent dans Git, vous avez plusieurs méthodes à votre disposition, en fonction de vos besoins. Voici quelques méthodes courantes :

1. **Checkout à un commit spécifique** (ce qui vous met en mode "detached HEAD") :
   `git checkout [sha-du-commit]`
   Où **[sha-du-commit]** est le hash du commit auquel vous voulez revenir. Après avoir exécuté cette commande, vous vous retrouverez en mode "detached HEAD", ce qui signifie que vous n'êtes plus sur une branche, mais sur un commit spécifique. Si vous souhaitez créer une nouvelle branche à partir de cet état, vous pouvez le faire avec **git checkout -b [nouveau-nom-de-branche]**.

2. **Réinitialiser votre branche à un commit précédent** : Si vous souhaitez effacer tous les commits après le commit cible sur votre branche actuelle, vous pouvez utiliser la commande **reset**. Soyez prudent avec cette méthode car elle modifie l'historique !
   - Réinitialisation douce (**soft**): `git reset --soft [sha-du-commit]`
     Cela déplace le pointeur de votre branche actuelle au commit cible, mais conserve les modifications dans votre zone de travail et votre index.
   - Réinitialisation mixte (par défaut): `git reset [sha-du-commit]`
     Cela déplace également le pointeur de votre branche actuelle vers le commit cible, mais cela réinitialise également votre index. Votre zone de travail reste inchangée.
   - Réinitialisation dure (**hard**): `git reset --hard [sha-du-commit]`
     Cela déplace le pointeur de votre branche actuelle vers le commit cible et supprime toutes les modifications de votre index et de votre zone de travail. Soyez très prudent avec cette option car vous perdrez toutes les modifications non commitées.

3. **Revenir à un commit précédent avec un nouveau commit** : Si vous ne souhaitez pas modifier l'historique, mais simplement annuler les effets d'un ou de plusieurs commits précédents, vous pouvez utiliser **git revert**.
   `git revert [sha-du-commit]`
   Cette commande crée un nouveau commit qui annule les modifications du commit spécifié.

N'oubliez pas de toujours faire une copie de sauvegarde ou une branche de sauvegarde avant de faire des modifications importantes ou destructrices à l'historique, surtout si vous travaillez sur des dépôts partagés.

III. Travailler avec les branches

Objectifs : À la fin de ce TP, l'étudiant sera capable de manipuler efficacement les branches dans Git, d'effectuer des fusions, de visualiser l'historique des commits et d'utiliser des commandes avancées pour gérer l'état de son dépôt.

Outils nécessaires : Git, ungit pour la visualisation.

Installation de ungit

ungit est une interface graphique open source pour Git. Il vise à simplifier la gestion et la visualisation de vos dépôts Git en offrant une interface utilisateur plus conviviale que la ligne de commande.

Étape N°41 : Installez node.js https://nodejs.org/fr

Étape N°42 : Ensuite entrez dans un terminal : *npm install -g ungi*t ou *sudo -H npm install -g ungit* (si vous avez besoin de droit d'administrateur)

Étape N°43 : Dans un terminal lancer la commande ungit& (tâche de fond)

1. Mise en place

Étape N°44 : Initialisez un nouveau dépôt Git.

Étape N°45 : Créez un fichier README.md avec du contenu basique à l'aide de la comme linux *echo*.

Étape N°46 : Quelle commande permet de voir le statut actuel de votre dépôt ?

Étape N°47 : Visualisez votre dépôt dans ungit.

2. Branching

Étape N°48 : Créez trois nouvelles branches: feature-1, feature-2, et bugfix-1.

Étape N°49 : Comment lister toutes les branches présentes dans le dépôt ?

Étape N°50 : Basculez sur la branche feature-1 et ajoutez quelques modifications au fichier README.md.

Étape N°51 : Après avoir effectué des modifications, comment ajouter en même temps, tous les changements en attente à la zone de staging ?

Étape N°52 : Visualisez vos branches et vos commits dans ungit.

3. Merging et Fast Forward

Étape N°53 : Basculez sur la branche master et fusionnez feature-1 avec la branche master.

Étape N°54 : Que signifie le terme "fast forward" lors d'une fusion ?

Étape N°55 : Sur la branche bugfix-1, faites quelques modifications et créez un commit. Ensuite, sur feature-2, faites d'autres modifications et créez un autre commit.

Étape N°56 : Après avoir commité sur bugfix-1, comment annuler le dernier commit ?

Étape N°57 : Visualisez l'état actuel des branches dans ungit.

Étape N°58 : Tentez de fusionner feature-2 dans master.

Étape N°59 : Si une fusion ne peut pas être effectuée en "fast forward", quel type de commit Git crée-t-il ?

4. Partie 4 : Nettoyage

Étape N°60 : Supprimez les branches feature-1 et bugfix-1.

Étape N°61 : Question 7 : Comment visualiser l'historique des commits de manière condensée ?

Étape N°62 : Vous avez accidentellement supprimé la branche feature-1. Comment revenir à l'avant-dernier commit ?

Étape N°63 : Visualisez le dépôt final dans ungit.