GIT AND COMMAND LINE BASICS

I. Command Line

https://explainshell.com/

https://www.learnenough.com/command-line-tutorial

- pwd

Affiche le répertoire courant dans lequel on se trouve

- ls

Abbréviation de listing. Elle affiche le contenu du répertoire courant.

Variation :

- ls -F

classe la sortie en ajoutant un marqueur aux noms de fichiers et de répertoires pour indiquer ce qu'ils sont :

/ indique un répertoire

@ indique un lien

* indique un exécutable

- ls --help ou man ls

en ajoutant "--help" à la commande ou "man" va permettre d'afficher les différentes façon d'utiliser la commande.

- ls -R

liste les dossiers, sous dossiers et fichiers imbriqués dans le répertoire.

- ls -l

liste détaillé du répertoire

- ls -s / ls -s -h

Affiche la taille de chaque répertoire. -h permet de rendre plus lisible la taille.

- ls -t

Répertorie les éléments par date de dernière modification

- ls -r

Répertorie en ordre inverse

- ls -F *directory*

applique la commande dans le chemin précisé.

- ls -a

Fais apparaître les fichiers et dossiers cachés.

Les options peuvent s'accoler les unes aux autres après le "-".

- cd

Commande qui nous amène dans notre répertoire personnel. Utile pour repartir d'une bonne base.

Variation :

- cd ..

permet d'accéder au répertoire parent.ls

- cd ~

Ce caractère signie répertoire personnel de l'utilisateur actuel et il nous y amène

- cd ~/*directory*

permet d'accéder à un répertoire à partir du répertoire personnel : /users/name/

- cd –

- cd *directory*

Commande qui permet d'accéder à un répertoire à partir du répertoire courant.

- cd */directory*

Permet d'accéder à un répertoire à partir du répertoire racine

- cd /

Nous amène dans le répertoire racine du disque.

permet d'accéder au répertoire précédent.	

- Clear

Permet de nettoyer le terminal

- mkdir *name*

permet de créer un nouveau répertoire

Variation

- mkdir -p dir/fold1 dir/fold2 Pemret de créer dossier et sous dossier en même temps.nano draft.txt	

- nano draft.txt

ouvre un fichier dans le terminal et permet l'édition de texte.

- touch file.txt

Permet de créer le fichier file.txt dans le répertoire courant.

- mv fold1/file.txt fold2/file.txt

Permet de déplacer un fichier du fold1 au fold2. On peut renommer le fichier au passage en changeant le nom dans la 2e partie de la commande.

- cp

Fonctionne de la même manière que mv mais fait une copie du fichier.

Commande $ nano fichier,txt
permet de créer un fichier dans le répertoire et d'en éditer le texte

Commande $ touch my_file.txt

ne fait que créer le document

commande $ mv dossier/file.txt dossier/index.txt
commande $ mv fichier1.txt fichier2.txt
permet de renommer le fichier situer dans le dossier,

commande $ mv dossier1/file.txt dossier2/file.txt
déplace le fichier du dossier1 au dossier2. Tout deux sous dossier du répertoire courant

commande $ mv -t *home*/destination source1 source 2 etc...
déplacer plusieurs sources vers la même destination

commande $ cp fichier.txt dossier/fichier.txt
copie le document dans un autre dossier

Si on veut copier tout un répertoire, on utiliser la commande $ cp -r chemin1 chemin2

La commande cp fonctionne avec plusieurs non de fichier mais elle doit terminer par un répertoire de destination qui doit être le dernier argument,

commande $ rm chemin/file.txt
permet de supprimer le fichier
Attention suppression définitive pas de corbeille dans le shell UNIX

commande $ rm -i chemin/file.txt
permet d'avoir un message de confirmation avant la suppression

par défaut cette commande ne supprime que les fichiers, pas les dossiers,
Pour un dossier et son contenu
Commande $ rm -r dossier
Commande $ rm -rf dossier
le f pour dire supprimer en force en cas de problème,

Commande $ wc fichier,txt
commande qui compte le nombre de lignes, de mots et de caractères (dans cet ordre) dans le fichier,

Pour le faire dans tous les fichiers
commande $ wc *,txt

Ouvrir ou Créer un fichier :

Commande $ nano fichier.txt : la console permet alors d'écrire du texte. Si le fichier existe déjà il est ouvert et on peut le compléter. S'il n'existe pas un éditeur vide prend la place du terminal et le fichier sera créé à la sauvegarde,

Commande $ echo "n'importe quel texte" > vieuxtacos.txt
echo permet d'afficher du texte dans le terminal.

Cette commande va stocker le texte dans un fichier vieuxtacos.txt
Si le fichier n'existe pas il est créé. S'il existe il est écrasé,

Un chevron écrit dans le fichier, deux chevrons écrits à la suite,

dans le répertoire :
$ touch hello_world.txt permet de créer le fichier dans le répertoire

$ code .
Permet d'ouvrir visual studio code dans le répertoire en cours.

II. Git Basics

1. Configuration

Installer GIT :

- sudo apt update
- sudo apt upgrade
- sudo apt install git
- contrôle version
- git --version

Base :

- git config --global user.name "Your Name"
- git config --global user.email "blabla@ygmail.com"

Il faut laisser les guillemets !

- git config --global init.defaultBranch main

Cela sert à modifier la branche par défaut lui donner le nom de main

- git config --global color.ui auto

active la sortie colorée avec git

- git config --get user.name
- git config --get user.email

permet de vérifier que tout a bien fonctionner.

Création de la clé SSH :

- ls ~/.ssh/id_ed25519.pub

si un message du type "aucun fichier ou répertoire de ce type" on doit donc créer une clé. Sinon étape suivante.

- ssh-keygen -t ed25519 -C <yourmail>

Il faut mettre le mail entre les crochets sans écrire les crochets dans la commande. On enregistre la clé. Le mot de passe demandé n'est pas obligatoire.

Lier la clé à GitHub

Se connecter à GitHub, aller sur *settings* puis chercher *SSH and GPG keys/* Cliquer sur le bouton vert dans le coin supérieur droit qui indique *New SSH Key*. Donner un nom descriptif à la clé pour se rappeler d'où elle vient. Puis :

- cat ~/.ssh/id_ed25519.pub

Copier alors la sortie qui commence par ssh-ed25519 et se termine par l'adresse mail.

Revenir à GitHub dans la fenêtre précédente et coller la clé dans le champ puis *Add SSH key*.

Pour tester la clé :

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/testing-your-ssh-connection

2. Travailler dans un dépôt git local

- $ git init

pour initialiser le système de contrôle de version GIT

- $ git status

A tout moment, permet de voir les fichiers qui sont nouveaux, modifié etc...

- $ git add <nom_du_fichier>

va ajouter le fichier à la liste des choses à synchro. Comme une salle d'attente. Cela se fait en deux étapes. Cela veut dire que le fichier sera surveillé, on pourra savoir si le fichier a été modifié avec la commande $ git status.

- $ git add .

Permet d'ajouter tous les fichiers créés ou modifés à la zone de staging, A utiliser avec précaution si tous les fichiers ne sont pas à suivre. Surtout dans l'optique d'un push sur GitHub (voir après)

- $ git commit -m "Ecrire ici son message entre les guillemets"

Fait le commit pour sauvegarder l'état des modifications qui fichier qui ont été marqué par la commande $ git add précédente. Le message est important on y indique la raison de notre commit, donc ce que l'on a fait,

- $ git log

montre les log du git

3. Travailler dans un dépôt Git & Gitub

1ère méthode :

- Il faut d'abord créer un repo sur le compte en ligne GitHub,
- Cocher la case *"Add a README file"* et valider.
- Dans le repo, cliquer sur le bouton *Code*, sélectionner l'option *SSH* et copier la ligne.
- Ouvrir le terminal puis aller dans le dossier voulu (commande mkdir, cd au besoin)
- Taper la commande :
    - git clone *ligneCopiéePrécédemment* (git clone git@github.com:Florent-V/nom_du_repo.git
    - cd nom_du_repo pour aller dans le repo.
- git remote -v

cela affichera l'URL du référentiel que l'on a créé sur GitHub.

- git status

Permet de voir l'état actuel du git, les fichiers qui ont été créés et/ou modifiés.

- git add file

Permet d'ajouter le fichier à la zone de préparation pour effectuer un commit dans git. Comme une salle d'attente.

- git add .

Permet d'ajouter tous les fichiers en même temps sans avoir à taper leur nom.

- git commit -m "message du commit"

Cela permet d'entrer le message du commit pour détailler ce que l'on a fait dans le commit en cours.

- git log

Permet de voir les détails des commit : auteur, heure etc….

- git push ou git push origin main

Tant que l'on à affaire qu'à une seule branche git push suffit.

2ème méthode :

- On crée un dossier de travail sur notre ordinateur
- On crée un repos sans le fichier README.md

Repo local inexistant	Repo local existant
- $ echo "# Init test" ” README.md - $ git init - $ git add README.md - $ git commit -m "first commit" - $ git branch -M main - $ git remote add origin git@github.com:Florent-V/nom_du_repo.git - $ git push -u origin main	- $ git remote add origin git@github.com:Florent-V/nom_du_repo.git - $ git branch -M main - $ git push -u origin main

A partir de là on pourra faire simplement $ git push par la suite.

$ git push origin main pour préciser où se fait le push. Important s'il y a plusieurs branches.

Pour qu'un fichier ne soit jamais suivi :
- On crée un fichier nommé : .gitignore
- On écrit dans ce fichier le nom des fichier à ignorer

4. Créer une nouvelle branche pour travailler dessus

- $ git branch <branch_name>
- $ git checkout <branch_name>

ou alors en une seule commande
- $ git checkout -b <branch_name>

- $ git branch

Permet de voir les différentes branches qui existent. Il y aura une étoile devant la branche de travail actuelle.

- $ git status / $ git add / $ git commit (commandes habituelles lors du travail)

Pour mettre en ligne la branche :
- $ git push origin <branch_name>

Pour fusionner la branche créée précédemment avec main.
On se place d'abord dans main :
- $ git checkout main

Pour fusionner :
- $ git merge <branch_name>

- $ git log

Pour voir ce que l'on a fait sur la branche.

- $ git push origin main

Pour pousser sur notre branche main la fusion effectuée à l'instant,

D'une manière générale pour aller sur <branch_name> on fait
$ git checkout <branch_name>

Pour supprimer une branche du référentiel local :
$ git branch -d <branch_name>

$ git push --delete origin <branch_name>
pour supprimer la branche sur gitHub,

Sinon dans le terminal vous pouvez avoir une vue graphique avec la commande :
git log --graph --oneline --decorate

5. Quand on travailler à plusieurs sur un repo

- $ git push origin main (ou une autre branche) peut renvoyer une erreur

Car le dépôt sera en avance donc on commence par un pull :

- $ git pull

Un message arrive sur la console car le pull fait un commit en même temps
Le message par défaut peut être gardé donc à la fin :
:q ! pour quitter le message.
Ensuite on peut faire :

- $ git push

Pas besoin de préciser origin main si on l'a paramétré au début.

Si on travaille sur une autre branche d'un repo qui ne nous appartient pas
Il faut ensuite faire un pull request : se fait sur GitHub : attention au sens de la flèche (elle est de la droite vers la gauche).
Permet au propriétaire du repo de contrôler la demande de merge. Si on bosser sur notre propre repo : pas besoin.

S'il y a des conflits : cliquer sur command line
Si peu de conflits, on peut le faire sur le web editeur
si trop de conflits, plus pratique de le faire dans notre éditeur de code.

Donc on rapatrie le repo pour travailler en local avant de push une fois les conflits réglés et le merge fait.

$ git fetch all
permet de récupérer l'intégralité du repo
$ git pull
ne récupère que les informations de la branche en cours

Aide-mémoire

Ceci est une liste de référence des commandes Git les plus couramment utilisées. (Vous pourriez envisager de mettre cette page pratique en signet.) Essayez de vous familiariser avec les commandes afin de pouvoir éventuellement toutes vous en souvenir :

- Commandes liées à un dépôt distant :
    - git clone git@github.com:USER-NAME/REPOSITORY-NAME.git
    - git pushou git push origin main(Les deux atteignent le même objectif dans ce contexte)
- Commandes liées au workflow :
    - git add .
    - git commit -m "A message describing what you have done to make this snapshot different"
- Commandes liées à la vérification de l'état ou de l'historique des journaux
    - git status

- git log

La syntaxe de base de Git est program | action | destination.

Par exemple,

- git add .est lu comme git | add | ., où le point représente tout dans le répertoire courant ;
- git commit -m "message"est lu comme git | commit -m | "message"; et
- git statusest lu comme git | status | (no destination).