

Utilisation de l'outil Git

Introduction

Petite documentation pour apprendre à utiliser Git sous les systèmes Linux et Windows avec une mise en application sur GitHub. Git est un logiciel de gestion de version d'un projet informatique décentralisé. Il a été créé par Linus Torvald il est donc libre et gratuit. GitHub quant à lui est un service web d'hébergement et de gestion de développement de logiciel qui fonctionne avec l'outil Git.

1. Création de clés SSH

Avant tout, il va nous falloir une paire de clés SSH pour pouvoir dialoguer de manière sécurisée avec GitHub.

a. Sous Linux

Avant de créer une paire de clés SSH, nous allons déjà voir s'il n'y en a pas déjà. Pour ça :

- Ouvrir un terminal
- Se placer dans le répertoire d'entrée si ce n'est pas déjà le cas :

```
$ cd
```

- Normalement un ~ apparaît avant le prompt
- Se déplacer dans le répertoire .ssh

```
$ cd ~/.ssh
```

- Lister le contenu du répertoire

```
$ ls
```

Si les fichiers `id_rsa` et `id_rsa.pub` sont présents, c'est bon, nous avons nos clés SSH. Nous pouvons passer à l'étape suivante.

Dans le cas contraire, nous allons générer des clés ssh grâce à la commande :

```
$ ssh-keygen
```

Appuyer sur entrée pour sauvegarder les clés dans le répertoire .ssh.

Concernant la phrase de passe, il est conseillé d'en mettre une afin de protéger la clé privée mais il devra être taper à chaque utilisation de la clé.

Dans le cadre d'un projet étudiant sur GitHub il y a peu de chance que

quelqu'un essaye de voler notre clé, nous pouvons donc laisser vide en appuyant deux fois sur entrée.

Voilà nous avons nos clés ssh, nous pouvons le voir en faisant un ls.

```
ky002500@Lofi:~/.ssh$ ls
known_hosts
ky002500@Lofi:~/.ssh$
ky002500@Lofi:~/.ssh$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ky002500/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ky002500/.ssh/id_rsa
Your public key has been saved in /home/ky002500/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:LRAtEwVMhVrHjcGJKLVrppj2FQ9BDvxKWaxa7VxX/Fg ky002500@Lofi
The key's randomart image is:
+---[RSA 3072]-----+
|      ..*X%+= .      |
|      ooX+B.. o E|
|      . XB+o . + |
|      .oB+o=. . . .|
|      +=.+S...      |
|      .....o       |
|      .              |
+---[SHA256]-----+
ky002500@Lofi:~/.ssh$
ky002500@Lofi:~/.ssh$ ls
id_rsa id_rsa.pub known_hosts
ky002500@Lofi:~/.ssh$ _
```

Génération de clés SSH sous Linux

b. Sous Windows

Sous Windows, c'est relativement similaire à Linux à quelques détails près.

Commençons par ouvrir un terminal en utilisant le PowerShell. Pour ça :

Clic droit sur le menu démarrer en bas à gauche puis PowerShell en tant qu'administrateur ou bien le raccourci *Windows* + x puis appuyer sur *a*.

Une fois le terminal ouvert, faire :

```
> cd ~
```

Ceci pour être sûr d'être dans le répertoire utilisateur. Normalement

```
C:\Users\<nom_de_l'utilisateur>\.
```

Une fois dans ce répertoire, on vérifie s'il n'existe pas de clés ssh déjà existantes comme dans la partie sur Linux. S'il n'y en a pas on en crée exactement comme sous Linux.

2. Copie de la clé SSH publique.

Pour que GitHub puisse reconnaître notre machine, il faut lui donner notre clé SSH publique. Pour ce faire, on se replace dans le `.ssh` de l'étape précédente et on lit notre clé publique qui est `id_rsa.pub`.

- Sous Windows :
> notepad .\id_rsa.pub

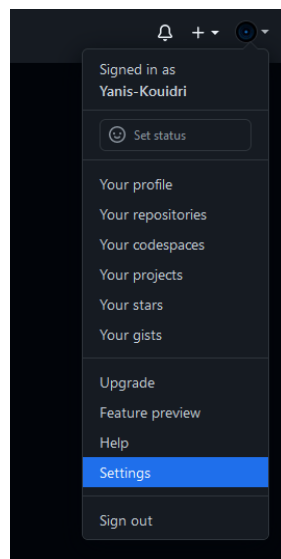
Ce qui va ouvrir la clé dans un bloc note pour pouvoir la copier facilement.

- Sous Linux :
\$ gedit ./id_rsa.pub

Ou bien tout simplement, s'il n'y a pas d'interface graphique :

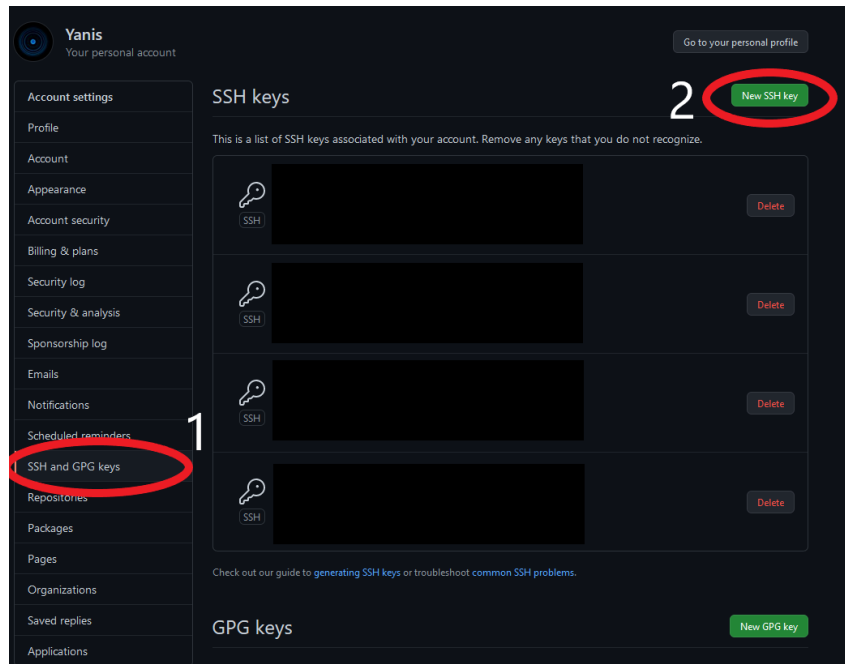
```
$ cat ./id_rsa.pub
```

Une fois la clé copiée, aller sur [GitHub](https://github.com), se connecter, cliquer en haut à droite pour aller dans setting.



Menu déroulant en haut à droite sur GitHub

Aller dans le menu SSH and GPG keys, puis cliquer sur New SSH key



Partie SSH et GPG keys du menu Settings de GitHub

Enfin coller votre clé dans la partie "key" et donner lui un nom (logiquement le nom de la machine sur laquelle vous avez cette clé).

Faire add SSH key et c'est bon, GitHub est prêt à être utilisé.

The image shows the 'SSH keys / Add new' form in GitHub. The 'Title' field is labeled 'Title' and contains the text 'PC fixe Asus'. The 'Key' field is labeled 'Key' and contains a long SSH key string starting with 'ssh-rsa'. The 'Add SSH key' button is at the bottom of the form.

Formulaire pour l'ajout d'une clé SSH publique sur GitHub

3. Comment installer Git

a. Sous Linux

Pour installer Git sous Linux il faut taper :

```
#sudo apt update  
#sudo apt install git
```

b. Sous Windows

Pour installer Git sous Windows il faut télécharger l'exécutable [ici](#)
L'ouvrir et faire « suivant » pour toutes les fenêtres de l'installation.

4. Prise en main de Git et application à GitHub

Une fois Git installé, trois exécutables apparaissent sur Windows. Celui qui nous intéresse ici est le Git bash. En l'ouvrant, un terminal apparaît. Avant tout il convient de configurer quelques informations afin de dire à git qui l'on est :

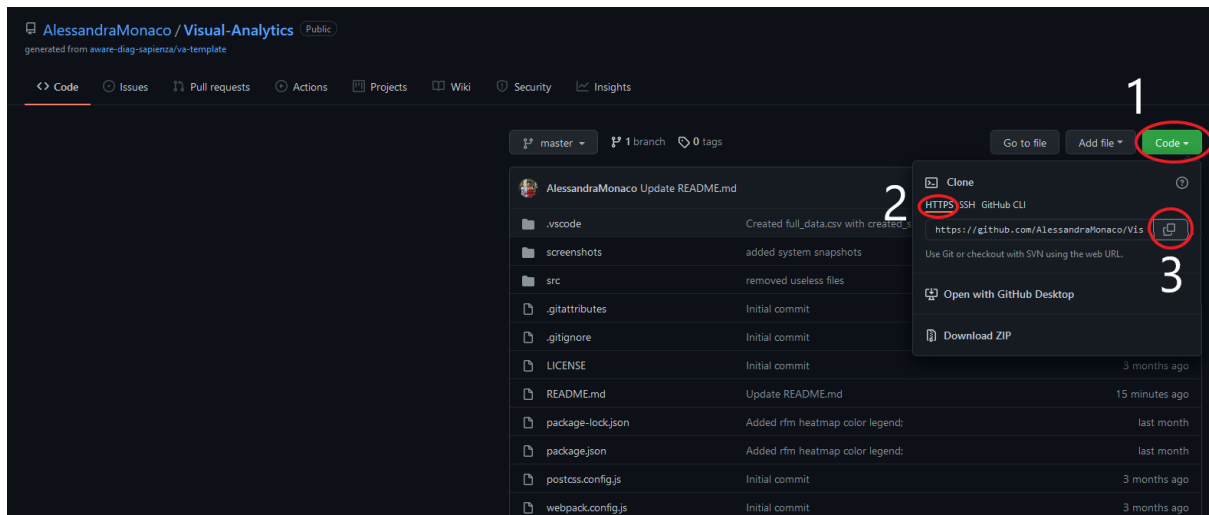
```
$ git config --global user.email "yanis.kouidri@etu.univ-  
codedazur.fr"  
$ git config --global user.name "Yanis Kouidri"
```

Cela permet lorsqu'on fera le push de mettre dans le paquet « commit » notre nom et notre courriel. Ces informations ne seront jamais interprétées par Git ou GitHub, vous pouvez donc mettre l'adresse mail que vous souhaitez, pas nécessairement celle utilisée pour l'inscription à GitHub.

Dans ce terminal, on se déplace comme dans le bash de Linux (cd, ls, pwd). Une fois dans le répertoire où l'on souhaite voir le dossier avec les fichiers de développement, il suffit de taper :

```
$ git clone <adresse_du_GitHub>
```

Le champ <adresse_du_GitHub> se trouve sur la page GitHub du projet dans la partie code. Voir la capture ci-après.



Capture d'écran de la partie code d'un projet quelconque

Une fois la commande git clone exécutée, un dossier contenant tous les fichiers de la branche main sera créé dans le répertoire courant. Voir la capture ci-après.

```
MINGW64:/F/IUT/S3/Projet-Python
Yanis@Lofi MINGW64 /F/IUT/S3
$ git clone https://github.com/Yanis-Kouidri/Projet-Python.git
Cloning into 'Projet-Python'...
remote: Enumerating objects: 101, done.
remote: Counting objects: 100% (101/101), done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 101 (delta 55), reused 66 (delta 23), pack-reused 0
Receiving objects: 100% (101/101), 265.21 KiB | 3.58 MiB/s, done.
Resolving deltas: 100% (55/55), done.

Yanis@Lofi MINGW64 /F/IUT/S3
$ cd Projet-Python/

Yanis@Lofi MINGW64 /F/IUT/S3/Projet-Python (main)
$ |
```

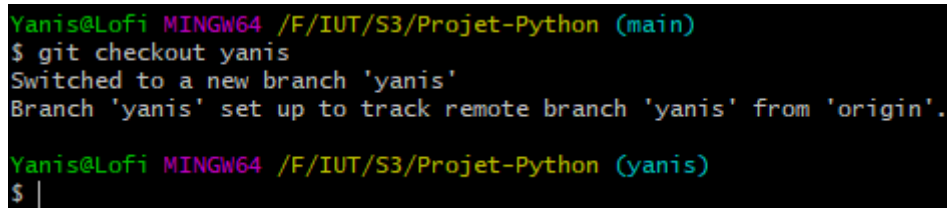
Capture d'écran de la fenêtre Git Bash

J'ai donc ici fait un git clone avec le lien de mon GitHub. Git a alors créé un dossier 'Projet-Python' (c'est le nom de mon projet). Je me déplace dedans. Une fois à l'intérieur je vois que je suis dans la branche main grâce au champ (main) en bleu.

La branche main est la branche par défaut, chaque branche contient ses propres fichiers. Si je fais :

```
$git checkout yanis
```

Je me déplace dans la branche « yanis ». « yanis » étant le nom d'une branche créée au préalable.

A terminal window with a black background and green text. The prompt is 'Yanis@Lofi MINGW64 /F/IUT/S3/Projet-Python (main)'. The user enters '\$ git checkout yanis'. The output is 'Switched to a new branch 'yanis'' and 'Branch 'yanis' set up to track remote branch 'yanis' from 'origin''. The prompt then changes to 'Yanis@Lofi MINGW64 /F/IUT/S3/Projet-Python (yanis)' and the user enters '\$ |'.

Changement de branche

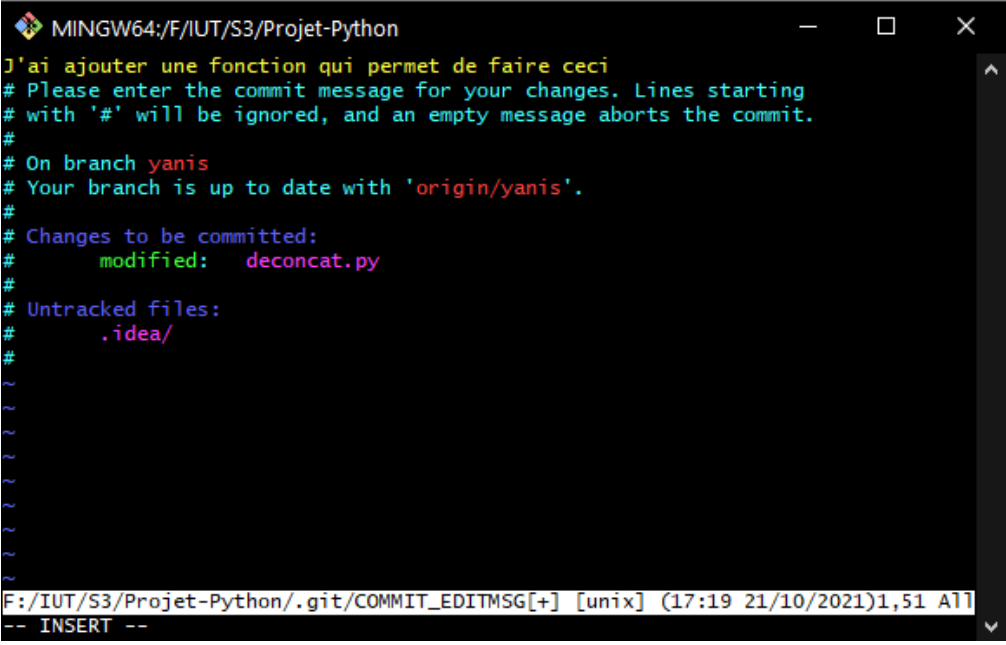
Maintenant que les fichiers sont là et que je suis me suis déplacé dans la branche souhaité (si besoin), je peux coder comme je le fais habituellement. Donc typiquement j'ouvre PyChram (dans le cas d'un projet en Python) et dans PyChram j'ouvre un fichier .py présent dans /F/IUT/S3/Projet-Python/. Je peux aussi créer un nouveau fichier dans ce même dossier. Une fois toutes les modifications souhaitées effectuées, je reviens dans le Git Bash, je me place dans le dossier Projet-Python et pour chaque fichier que j'ai modifié je fais :

```
$git add <nom_fichier>
```

Une fois fait, je fais :

```
$git commit
```

Un fichier va alors s'ouvrir, il faut la remplir avec les modifications que l'on a effectuées depuis le précédent commit.



```

MINGW64:/F/IUT/S3/Projet-Python
J'ai ajouter une fonction qui permet de faire ceci
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch yanis
# Your branch is up to date with 'origin/yanis'.
#
# Changes to be committed:
#   modified:   deconcat.py
#
# Untracked files:
#   .idea/
#
F:/IUT/S3/Projet-Python/.git/COMMIT_EDITMSG[+] [unix] (17:19 21/10/2021)1,51 All
-- INSERT --

```

Exemple de commit

Rappel : pour écrire du texte sur Vim il faut appuyer sur i, taper le texte puis appuyer sur Echap. Une fois le texte tapé on fait :x pour quitter (si l'on est sur Vim). Pour plus d'aide sur Vi et Vim voir [Vi](#).

Une fois fait, je tape

```
$git push
```

Normalement à ce moment-là, les clés SSH vont permettre de nous authentifier et le push se fera sans que l'on nous demande de nous connecter à GitHub. Et normalement les fichiers ont bien été déposés sur le projet GitHub.

Donc au prochain git clone ce sont les nouveaux fichiers qui seront clonés.


```

Yanis@Lofi MINGW64 /F/IUT/S3/Projet-Python (yanis)
$git commit
[yanis 98ae2aa] J'ai ajouter une fonction qui permet de faire ceci
1 file changed, 1 insertion(+)

Yanis@Lofi MINGW64 /F/IUT/S3/Projet-Python (yanis)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 356 bytes | 356.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Yanis-Kouidri/Projet-Python.git
 94b6929..98ae2aa  yanis -> yanis

Yanis@Lofi MINGW64 /F/IUT/S3/Projet-Python (yanis)

```

Commit et push

Pour copier un fichier d'une branche à une autre il faut faire :

```
$git checkout Evan exec.c
```

Je viens de copier le fichier exec.c depuis la branche nommée Evan vers ma branche courante.

Sur linux c'est la même chose mais toutes les commandes tapées dans le git Bash peuvent être tapées dans le bash linux classique (le terminal).

5. Commandes Git essentielles

Nous allons faire un petit récapitulatif des commandes Git qu'il faut connaître.

Pour copier un dépôt Git sur sa machine :

```
$git clone <adresse_du_dépôt>
```

Pour savoir dans quelle branche nous nous trouvons ainsi que quelles modifications ont été effectuées par rapport au dépôt :

```
$git status
```

Pour créer une branche :

```
$git checkout -b <nom_de_la_nouvelle_branche>
```

Pour changer de branche :

```
$git checkout <nom_branche>
```

Pour supprimer une branche :

```
$git branch -d <nom_branche>
```

Tout comme il est impossible de scier une branche d'un arbre sur laquelle on est assis, il est impossible de supprimer une branche Git sur laquelle on est. Pour lister toutes les branches d'un dépôt :

```
$git branch -a
```

Sans le -a, uniquement les branches présentes localement seront affichées. Pour mettre à jour le contenu de notre répertoire local à partir du dépôt git :

```
$git pull
```

Pour copier notre travail fait sur le répertoire local vers le dépôt Git (en 3 étapes) :

```
$git add <le_nom_du/des_fichiers_modifiés>
```

```
$git commit -m "Qu'est-ce que j'ai fait (comme modifications) ?"
```

```
$git push
```

Pour supprimer un/des fichiers (et que la suppression soit synchronisée avec le dépôt git) :

```
$git rm <le_nom_du/des_fichiers_à_supprimer>
```

```
$git commit -m "Pourquoi je les ai supprimés ?"
```

```
$git push
```

Pour copier un fichier de la branche alpha à la branche beta.

```
$git checkout apha #on se place dans alpha
```

```
$git checkout beta <nom_fichier_dans_la_branche_beta>
```