

EXERCICE N°12

GESTIONNAIRE DE SOURCE GIT

Jean-Pierre Duchesneau, Cégep Sainte-Foy, DFC

Automne 2021

Évaluation : formative

Travail de préférence individuel.

Durée : 2 heures

Système d'exploitation : Ubuntu 20.04 LTS

Site de référence : <https://git-scm.com/> **Livre de référence :** Pro Git, disponible sur LÉA

1 Installation de Git

— Au sein de votre machine virtuelle, ouvrez un terminal et vérifiez l'installation de Git.

```
1 $apt-cache show git
```

— La commande show affiche les informations sur un paquet (depuis le cache).

— **Si Git n'est pas installé ce qui devrait être votre cas**, il faut utiliser la commande suivante pour installer Git :

```
2 $sudo apt-get install git
```

— Une fois que l'installation est terminée, pour tester cette installation et obtenir la version de Git qui a été installée il faut utiliser la commande suivante :

```
3 $git --version
```

— Placez-vous sur votre bureau et créer un répertoire ExerciceGit

```
4 $cd Bureau
```

```
5 $mkdir ExerciceGit
```

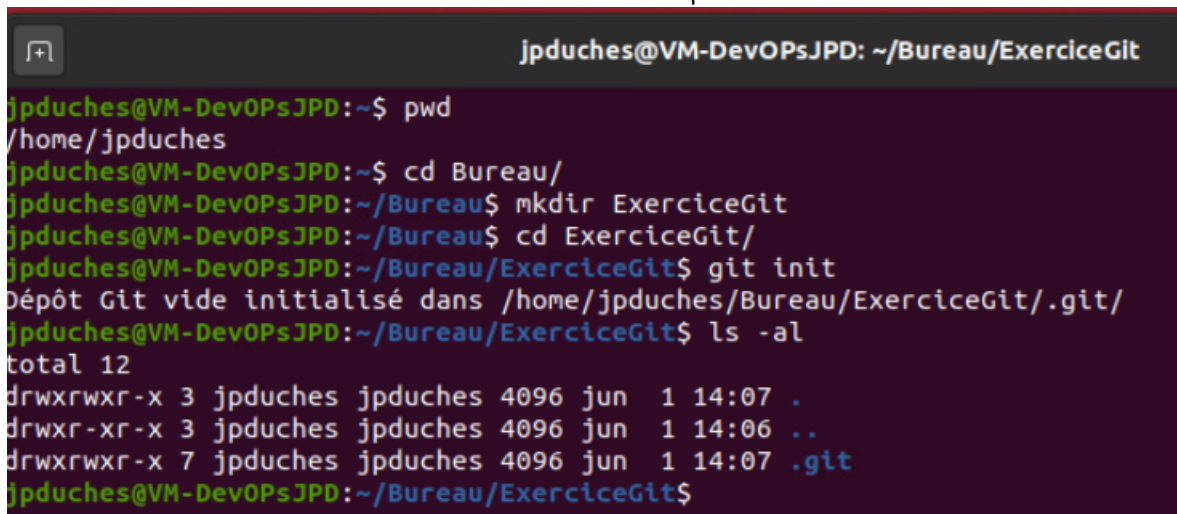
- Placez-vous dans le répertoire et initialisez ce répertoire en tant que dépôt Git en tapant la commande :

```
6      $cd ExerciceGit
7      $git init
```

- Cela crée un nouveau sous-répertoire nommé `.git` qui contient tous les fichiers nécessaires au dépôt. Pour l'instant, aucun fichier n'est encore versionné.
- Vérifiez la présence du nouveau répertoire `.git` dans votre arborescence (figure 1) :

```
8      $ls -al
```

FIGURE 1 – Dernières opérations



```
jpduches@VM-DevOPsJPD: ~/Bureau/ExerciceGit
jpduches@VM-DevOPsJPD:~$ pwd
/home/jpduches
jpduches@VM-DevOPsJPD:~$ cd Bureau/
jpduches@VM-DevOPsJPD:~/Bureau$ mkdir ExerciceGit
jpduches@VM-DevOPsJPD:~/Bureau$ cd ExerciceGit/
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git init
Dépôt Git vide initialisé dans /home/jpduches/Bureau/ExerciceGit/.git/
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ ls -al
total 12
drwxrwxr-x 3 jpduches jpduches 4096 jun  1 14:07 .
drwxr-xr-x 3 jpduches jpduches 4096 jun  1 14:06 ..
drwxrwxr-x 7 jpduches jpduches 4096 jun  1 14:07 .git
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$
```

1.1 l'aide de git

Pour afficher l'aide générale qui liste les commandes principales de Git, il faut utiliser la commande suivante :

```
9      $git --help
```

Cette commande permet d'avoir une liste et une explication brève (anglophone) des commandes principales de Git.

Par exemple, pour obtenir de l'aide sur la commande `git add`, il faut utiliser la commande suivante :

```
10     $git add --help
```

Le résultat de cette commande représente un important volume de texte, mais est découpé en plusieurs parties présentées ci-dessous :

- **NAME** : présente le nom de la commande et la brève description présente dans la commande `git -help`.
- **SYNOPSIS** : présente la syntaxe détaillée de la commande et de ses options.
- **DESCRIPTIONS** : présente une explication complète des fonctionnalités de la commande.
- **OPTIONS** : présente les options disponibles et décrit précisément pour chacune d'elles, sa fonctionnalité.
- **CONFIGURATION** : présente les interactions avec la configuration de Git. Par exemple, la variable de configuration `core.excludesfiles` empêche `git add` d'ajouter certains fichiers.
- **EXAMPLES** : présente des exemples d'utilisation de la commande avec une explication de l'exemple.
- **SEE ALSO** : présente des commandes en corrélation avec la commande dont l'aide est affichée.

L'aide est très complète et permet d'explorer les fonctionnalités évoluées de chaque commande.

1.2 Configuration requise de Git

Maintenant que vous avez installé Git sur votre système, vous voudrez personnaliser votre environnement Git. Vous ne devriez avoir à réaliser ces réglages qu'une seule fois ; ils persisteront lors des mises à jour. Vous pouvez aussi les changer à tout instant en relançant les mêmes commandes. Git contient un outil appelé `git config` pour vous permettre de voir et modifier les variables de configuration qui contrôlent tous les aspects de l'apparence et du comportement de Git. Ces variables peuvent être stockées dans trois endroits différents sous **Linux** :

- Fichier `/etc/gitconfig` : Contient les valeurs pour tous les utilisateurs et tous les dépôts du système. Si vous passez l'option `--system` à `git config`, il lit et écrit ce fichier spécifiquement.
- Fichier `~/.gitconfig` : spécifique à votre utilisateur. Vous pouvez forcer Git à lire et écrire ce fichier en passant l'option `--global`.
- Fichier `config` dans le répertoire Git (c'est-à-dire `.git/config`) du dépôt en cours d'utilisation : spécifique au seul dépôt en cours.

Chaque niveau surcharge le niveau précédent, donc les valeurs dans `.git/config` surchargent celles de `/etc/gitconfig`.

Dans Windows ils sont stockés dans :

- Fichier `C:\Program Files\Git\etc\gitconfig` : Contient les valeurs pour tous les utilisateurs et tous les dépôts du système. Si vous passez l'option `--system` à `git config`, il lit et écrit ce fichier spécifiquement.
- Fichier `~/.gitconfig` : spécifique à votre utilisateur. Vous pouvez forcer Git à lire et écrire ce fichier en passant l'option `--global`.
- Fichier config dans le répertoire Git (c'est-à-dire `.git/config`) du dépôt en cours d'utilisation : spécifique au seul dépôt en cours.

Il existe deux paramètres de configuration qui sont nécessaires avant toute utilisation de Git. Ces paramètres de configuration sont l'**adresse e-mail** de l'utilisateur du dépôt et son **nom**. Pour qu'à la fin de cette partie Git soit fonctionnelle, il est nécessaire d'effectuer cette configuration.

Pour **configurer le nom de l'utilisateur**, il faut utiliser la commande suivante¹ :

```
11 $git config --global user.name "Prenom Nom"
```

Pour **configurer l'e-mail de l'utilisateur** il faut utiliser la commande suivante :

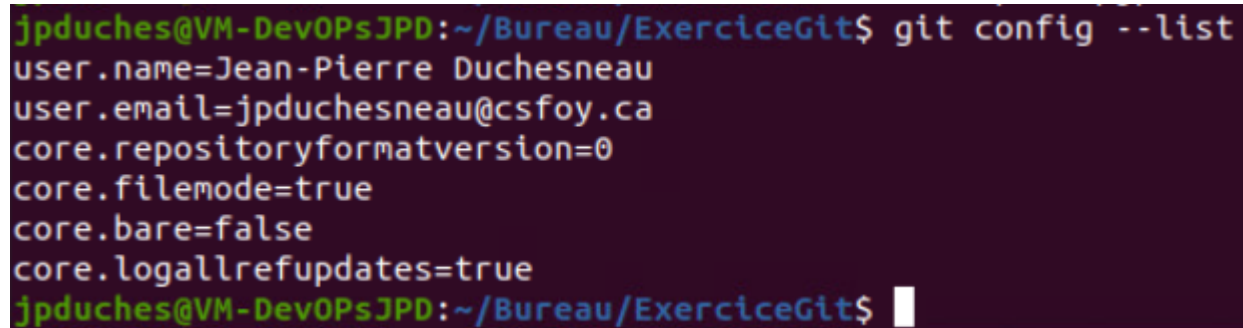
```
12 $git config --global user.email "email@domaine.extension"
```

Exemple :

```
13 $git config --global user.email "jpduchesneau@csfoy.ca"
```

```
14 $git config --list
```

1. Pour avoir l'ensemble des informations sur l'identité référez-vous au site Web de Git : <https://git-scm.com/book/fr/v2/Démarrage-rapide-Paramétrage-à-la-première-utilisation-de-Git>



```
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git config --list
user.name=Jean-Pierre Duchesneau
user.email=jpduchesneau@csfof.ca
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$
```

FIGURE 2 – Vérification de la configuration de base

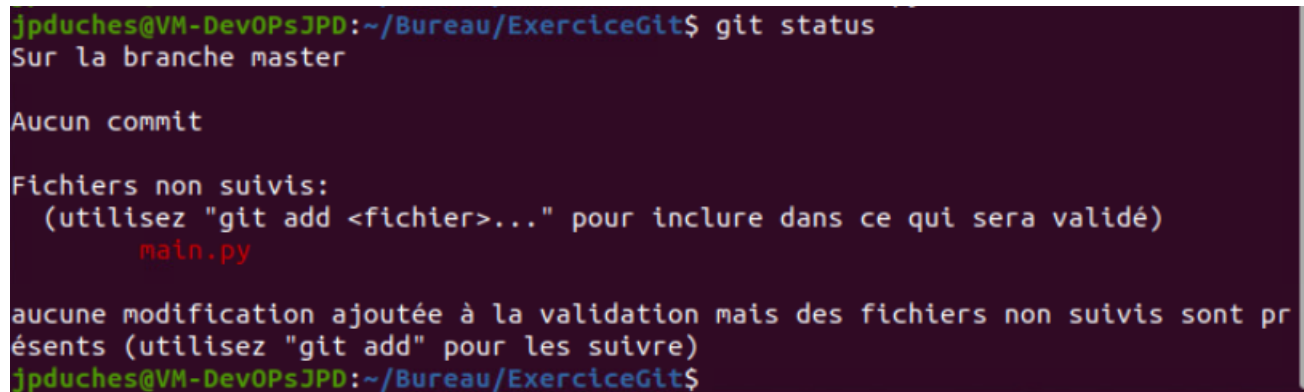
2 Création d'un dépôt

Précédemment, nous avons utilisé la commande qui permet de créer un dépôt local : [git init](#).¹

- Pour suivre les modifications sur un fichier, nous allons procéder quelques tests :
- Créer fichier `main.py`² à l'aide de la commande `touch main.py` dans le répertoire `ExerciceGit` :

```
15      $touch main.py
```

- Taper maintenant la commande `git status` (Figure 3) :



```
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master

Aucun commit

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
       main.py

aucune modification ajoutée à la validation mais des fichiers non suivis sont pr
ésents (utilisez "git add" pour les suivre)
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$
```

FIGURE 3 – Git status

Vous avez donc un fichier qui est dans l'état modifié, ce qui signifie que vous avez modifié un fichier, mais qu'il n'est pas encore validé.

- Nous allons utiliser la commande `git add main.py` pour l'indexer ou si vous préférez le suivre.

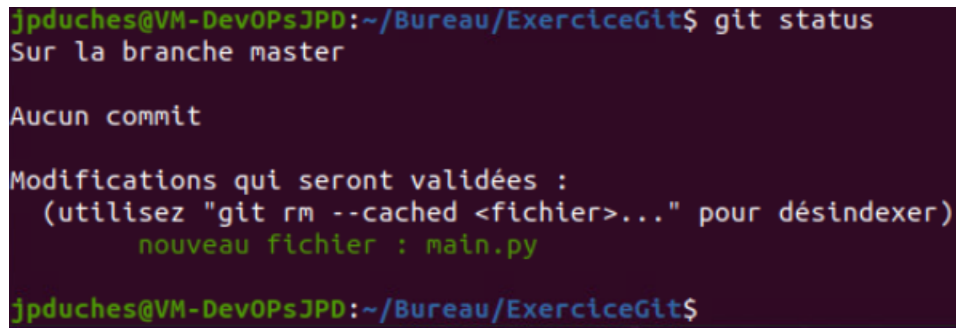
```
16      $git add main.py
```

2. Pour nos tests git, nous allons utiliser le langage de programmation Python. Pour en savoir plus sur ce langage, vous pouvez lire [https://fr.wikipedia.org/wiki/Python/_\(langage\)](https://fr.wikipedia.org/wiki/Python/_(langage))

— et a nouveau :

```
17      $git status
```

— Voici la sortie qui vas suivre votre commande (figure 4) :



```
jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master

Aucun commit

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)
    nouveau fichier : main.py

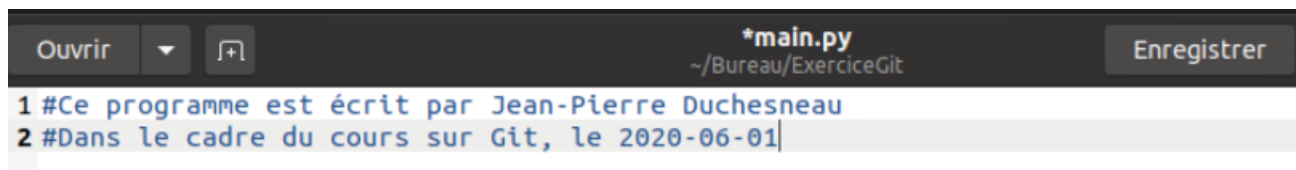
jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$
```

FIGURE 4 – git status

— Modifions le contenu du fichier de la façon suivante :

```
18      $gedit main.py
```

— Ajouter le commentaire suivant dans le fichier(modifier pour avoir vos informations) figure 2 :



```
Ouvrir  [icon]  *main.py  ~/Bureau/ExerciceGit  Enregistrer

1 #Ce programme est écrit par Jean-Pierre Duchesneau
2 #Dans le cadre du cours sur Git, le 2020-06-01
```

FIGURE 5 – Entête

— Enregistrez et quittez.

— Taper à nouveau la commande [git status](#), voici le résultat (figure 6) :

— Tapez à la commande `git checkout main.py` et vérifier le résultat et l'état du fichier. Il est vide.

— [checkout](#) a annulé les modifications que nous avons faites au fichier.

— Modifiez à nouveau votre fichier comme à la figure 2 avec gedit, mais cette fois, faite plutôt un [commit](#) après l'avoir modifié au lieu d'un checkout. Voici l' étape du commit (figure 7) :

Nous avons utilisé la commande [git commit](#) pour valider, ce qui signifie que les données sont stockées en sécurité dans votre base de données locale.

L'option `-m` permet d'ajouter un commentaire au commit pour l'identifier plus tard.

```

jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master

Aucun commit

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)
    nouveau fichier : main.py

Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git restore <fichier>..." pour annuler les modifications dans le ré
  pertoire de travail)
    modifié :      main.py

jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$

```

FIGURE 6 – git status

```

jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git commit -m "Ajout de l'entête"
[master (commit racine) a6704a8] Ajout de l'entête
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 main.py
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$

```

FIGURE 7 – git commit

— Tapez la commande `git log` (figure 8) :

```

jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git log
commit a6704a821ff5b79a46fd6de026ef81fb4f8fa298 (HEAD -> master)
Author: Jean-Pierre Duchesneau <jpduchesneau@csfoyc.ca>
Date:   Mon Jun 1 16:00:03 2020 -0400

    Ajout de l'entête
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$

```

FIGURE 8 – git log

— Modifiez le contenu du fichier de la façon suivante gedit main.py et ajoutez le code suivant (figure 9) dans le fichier, enregistrez et quittez le mode édit.

```

Ouvrir ▼ + main.py ~/Bureau/ExerciceGit Enregistrer
1 #Ce programme est écrit par Jean-Pierre Duchesneau
2 #Dans le cadre du cours sur Git, le 2020-06-01
3
4 print("Hello World!")

```

FIGURE 9 – Hello World !

- Taper à nouveau la commande `git status`, vous pouvez voir le résultat à la figure 10 :

```

jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master
Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce qui sera validé)
  (utilisez "git restore <fichier>..." pour annuler les modifications dans le ré
pertoire de travail)
    modifié :      main.py

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git
commit -a")
jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$

```

FIGURE 10 – git status

- Tapez la commande `git commit -am "Ajout de print Hello World"` (figure 11)

```

jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$ git commit -am "Ajout de print Hello
World"
[master f1db4b6] Ajout de print HelloWorld
1 file changed, 4 insertions(+)
jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$

```

FIGURE 11 – commit

- Tapez à nouveau la commande `git status` (figure 12)

```

jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master
rien à valider, la copie de travail est propre
jpduches@VM-Dev0PsJPD:~/Bureau/ExerciceGit$

```

FIGURE 12 – git status

- Et maintenant, pourquoi pas la commande `git log` (figure 13)

Constat, les deux modifications sont indexées.

Au besoin, relisez l'ensemble depuis la création du dépôt (section 2) pour être certain d'avoir bien suivi.

Continuons

- Créez un nouveau fichier appelez majuscule.py à l'aide de la commande `gedit` (figure 14).

```
19      $gedit majuscule.py
```

- Enregistrer et quittez l'éditeur.
- Taper la commande `git status` (Figure 15)


```

jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git log
commit f1db4b67d92c04e9d8cf49326f2a74f4e2bb5575 (HEAD -> master)
Author: Jean-Pierre Duchesneau <jpduchesneau@csfoyc.ca>
Date:   Mon Jun 1 20:03:22 2020 -0400

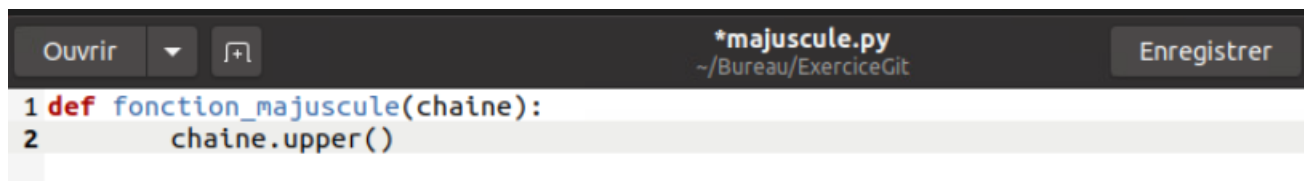
    Ajout de print HelloWorld

commit a6704a821ff5b79a46fd6de026ef81fb4f8fa298
Author: Jean-Pierre Duchesneau <jpduchesneau@csfoyc.ca>
Date:   Mon Jun 1 16:00:03 2020 -0400

    Ajout de l'entête
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$

```

FIGURE 13 – git log



```

Ouvrir  [v]  [f]  *majuscule.py  Enregistrer
~/Bureau/ExerciceGit
1 def fonction_majuscule(chaine):
2     chaine.upper()

```

FIGURE 14 – Édition d'un second fichier python

```

jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)
    majuscule.py

aucune modification ajoutée à la validation mais des fichiers non suivis sont pr
ésents (utilisez "git add" pour les suivre)
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$

```

FIGURE 15 – Édition d'un second fichier python

- Faites la commande `git add` suivi du nom du fichier à indexer (figure 16)

```

jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git add majuscule.py
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$

```

FIGURE 16 – git add

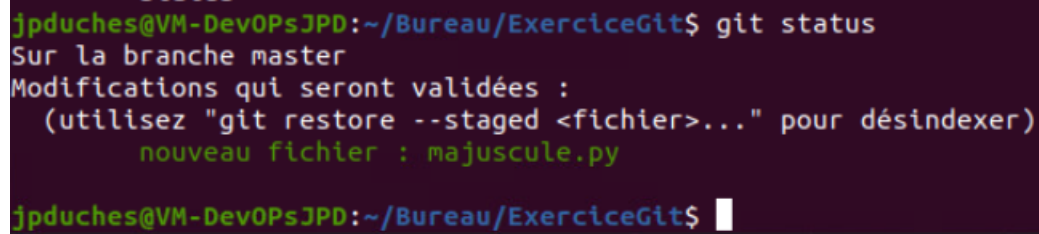
- À nouveau faite la commande `git status`(figure 17).
- Faisons maintenant un `git commit -m` pour indexer le fichier :

```

20 $git commit -m "Ajout de la fonction minuscule"

```

- **Petite question :** Qu'est-ce qui se passe si vous ne mettez pas l'option -m lors des commit ?



```
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git restore --staged <fichier>..." pour désindexer)
    nouveau fichier : majuscule.py
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$
```

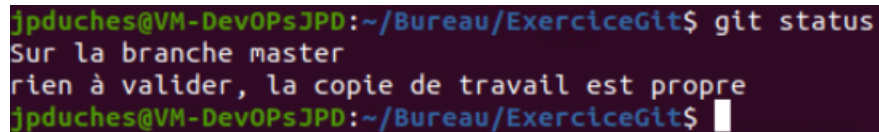
FIGURE 17 – git status

Utilisez l'aide pour répondre.

```
21 $git commit --help
```

La commande commit est l'une des plus importante de GIT. C'est elle qui indexe les fichiers. Parfois, un petit message est suffisant pour expliquer le commit. Dans ce cas vous pouvez utiliser l'option -m. Par contre, si vous devez écrire un message plus important, n'utiliser pas l'option -m et entrez dans l'éditeur de texte par défaut pour rédiger votre texte de commit.

— Finalisons avec un `git status` (figure 18).



```
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master
rien à valider, la copie de travail est propre
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$
```

FIGURE 18 – git status

— Et un `git log` (figure 19).

```
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git log
commit c2504be09b2016ebdb72d4e769b79e66b117ba3f (HEAD -> master)
Author: Jean-Pierre Duchesneau <jpduchesneau@csfooy.ca>
Date: Tue Jun 2 09:05:26 2020 -0400

    Ajout de la fonction majuscule

commit f1db4b67d92c04e9d8cf49326f2a74f4e2bb5575
Author: Jean-Pierre Duchesneau <jpduchesneau@csfooy.ca>
Date: Mon Jun 1 20:03:22 2020 -0400

    Ajout de print HelloWorld

commit a6704a821ff5b79a46fd6de026ef81fb4f8fa298
Author: Jean-Pierre Duchesneau <jpduchesneau@csfooy.ca>
Date: Mon Jun 1 16:00:03 2020 -0400

    Ajout de l'entête
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$
```

FIGURE 19 – git status

3 Cloner un dépôt Git distant

Vous pouvez démarrer un dépôt Git de trois manières. La première, vous initialisez un dépôt vide, comme vous avez fait dans la partie précédente ; la seconde consiste à prendre un projet ou un répertoire existant et à l'importer dans Git ; la troisième consiste à cloner un dépôt Git existant sur un autre serveur.

Dans cette exercice, nous allons explorer la troisième méthode. Nous allons aller chercher un dépôt git qui est situé sur le serveur GitHub. Vous allez faire une copie de mon projet [introGit](#) dans votre poste de travail.

3.1 Clone avec HTTPS

— Dans votre terminal, placez-vous sur le bureau (figure 20).

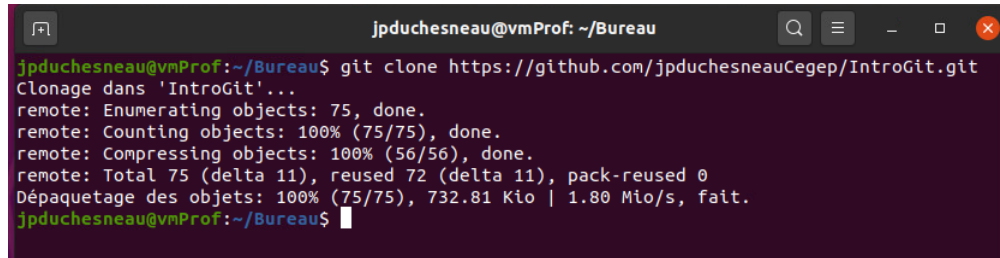
```
jpduches@VM-SEJPD:~/Bureau/ExerciceGit$ cd ..
jpduches@VM-SEJPD:~/Bureau$ pwd
/home/jpduches/Bureau
jpduches@VM-SEJPD:~/Bureau$
```

FIGURE 20 – Déplacement sur le bureau

— Tapez la commande suivante :

```
22 $git clone https://github.com/jpduchesneauCegep/IntroGit.git
```

- Vous devriez avoir les informations suivantes et un nouveau dossier **introGit** sur votre bureau (figure 21).



```
jpduchesneau@vmProf: ~/Bureau
jpduchesneau@vmProf:~/Bureau$ git clone https://github.com/jpduchesneauCegep/IntroGit.git
Clonage dans 'IntroGit'...
remote: Enumerating objects: 75, done.
remote: Counting objects: 100% (75/75), done.
remote: Compressing objects: 100% (56/56), done.
remote: Total 75 (delta 11), reused 72 (delta 11), pack-reused 0
Dépaquetage des objets: 100% (75/75), 732.81 Kio | 1.80 Mio/s, fait.
jpduchesneau@vmProf:~/Bureau$
```

FIGURE 21 – git clone sur le serveur GitHub

- Utilisez la commande **cd** pour vous y rendre :

```
23      $cd IntroGit
```

- Nous allons regarder les informations que nous avons concernant ce dépôt git.
- D'abord, vérifions son contenu avec la commande **ls** et le paramètre **-l** pour la version longue et **-a** pour voir les fichiers et les dossiers cachés. N'oubliez pas, un paramètre passé à une commande demande de saisir - devant les paramètres (sauf exception).

```
24      $ls -al
```

- Vous y trouverez un seul fichier : README.md que vous pouvez lire en utilisant la commande **cat** :

```
25      $cat README.md
```

- Maintenant, continuons avec les commandes **git** : d'abord un git status (figure 22)

```
26      $git status
```

- Ici, ça vaut la peine de faire une comparaison avec le dépôt git local fait à l'étape précédente. Est-ce que le message était le même quand il n'y avait rien à valider ? **Vérifiez**.
- Au besoin créer un nouveau dépôt local et vérifier.
- OK, je vous aide un peu, ici à la figure 22 on parle que la branche est à jour avec '**origin/main**'.

```
jpduchesneau@vmProf:~/Bureau/IntroGit$ git status
Sur la branche main
Votre branche est à jour avec 'origin/main'.

rien à valider, la copie de travail est propre
jpduchesneau@vmProf:~/Bureau/IntroGit$
```

FIGURE 22 – git status sur dépôt distant

- Dans l'étape précédente, quand nous avons créé un dépôt local ça disait : Sur la branche '[master](#)'.(Figure 23)

```
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$ git status
Sur la branche master

Aucun commit

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)
    nouveau fichier : main.py
jpduches@VM-DevOPsJPD:~/Bureau/ExerciceGit$
```

FIGURE 23 – git status sur le dépôt local

- Pour bien comprendre la différence, tapez la commande suivante :

```
27      $git remote -v
```

- Nous pouvons voir que le dépôt distant s'appelle [origin](#) et nous avons l'adresse HTTP du dépôt (figure 24).

```
jpduchesneau@vmProf:~/Bureau/IntroGit$ git remote -v
origin https://github.com/jpduchesneauCegep/IntroGit.git (fetch)
origin https://github.com/jpduchesneauCegep/IntroGit.git (push)
jpduchesneau@vmProf:~/Bureau/IntroGit$
```

FIGURE 24 – git remote -v

- Profitez-en pour regarder l'aide sur la commande [git remote](#), ainsi que sur les commandes [git fetch](#) et [git push](#) :

```
28      $git remote -h
29      $git remote --help
30      $git fetch --help
31      $git push --help
```

Essayer les commandes suivantes :

```
32
33      $git fetch
34      $git push
```

Pour l'instant, ce dépôt distant ne vous permet que de récupérer des fichiers par la commande fetch. Pour pouvoir placer des fichiers sur le dépôt, avec la commande push, il vous faudra un compte sur GitHub et des droit sur le projet.

Faite un git log pour voir l'historique du projet.

```
35      $git log
```

Vous avez en main, l'ensemble du code source de site Web basique fait avec C, HTML, CSS et JavaScript. Vous pourriez travailler dessus. Mais pas publier votre code.

4 Création d'un compte GitHub

A l'aide du Livre ProGit, présent sur Léa, créez votre compte GitHub (Chapitre GitHub, page 163)

- Prenez soin de bien choisir votre adresse courriel, ce compte pourra vous suivre pendant plusieurs années.
- La connexion pour SSH n'est pas essentielle pour le moment. Vous pouvez sauter cette section.
- Faites la partie Duplication des projets, mais utiliser mon dépôt : <https://github.com/jpduchesneauCegep/420-W12-SF-4393>, il s'agit du contenu du cours système d'exploitation.
- Arrêtez-vous avant la Création d'une requête de tirage qui est un niveau trop avancé pour le moment.
- Faites les étapes déjà faites au point 3 cloner un dépôt distant, mais cette fois, mais changer l'adresse du dépôt pour votre adresse de dépôt dupliqué dans votre compte.
- Apporter une modification au fichier README.md. Faites les commandes nécessaires (git add, git commit -m "Message", git status)
- Essayer de faire un push, il devrait fonctionner.

5 Défis : Git dans Windows

Refaites l'exercice au sein de votre machine virtuelle Windows ou de votre propre ordinateur.

- Vous devrez probablement installer git.<https://git-scm.com/download/win>
- Cochez l'option **Git Bash Here** lors de l'installation, dans la fenêtre **Select Components** (figure 26)

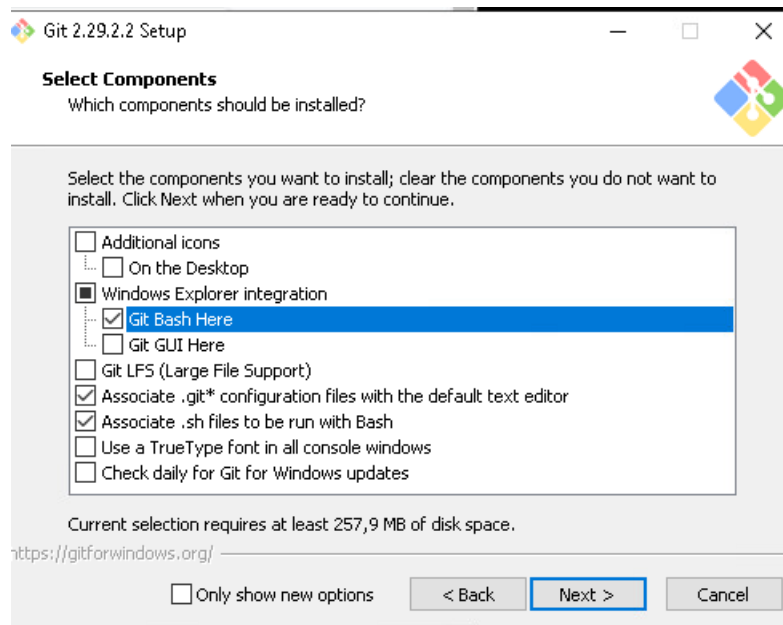


FIGURE 25 – Cocher Git Bash here

- Garder toutes les autres options par défaut.
- Après l'installation, vous devriez avoir accès sur tous les dossier et fichier à un raccourci gitbash sur le bouton droit de la souris (figure 27).

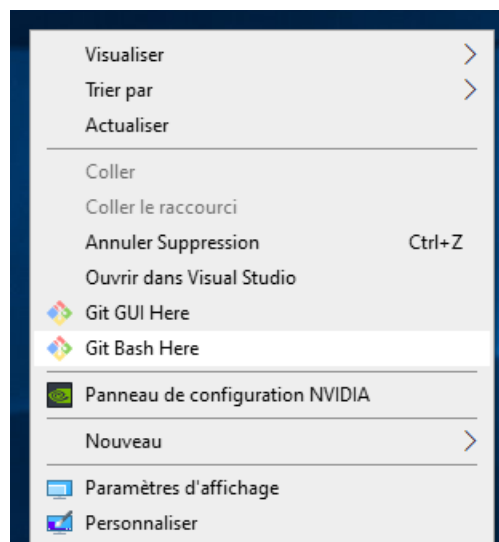
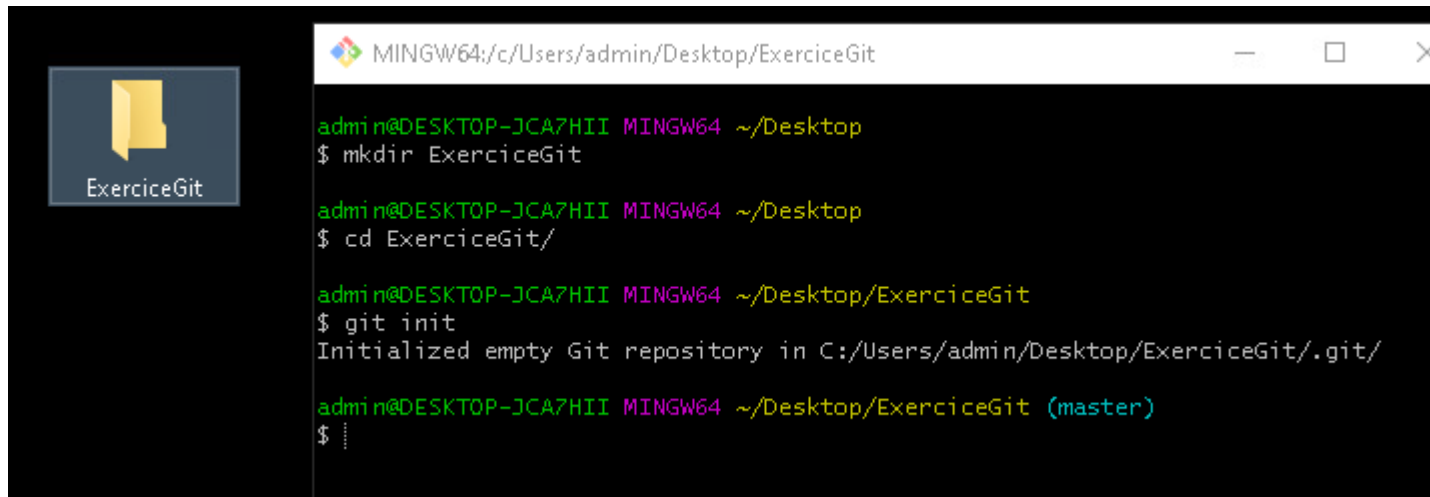


FIGURE 26 – gitbash en Windows

- En vous plaçant sur votre bureau de Windows, ouvrez git en cliquant Git Bash Here.
- Toutes les autres commandes seront identiques à celles pratiquées plus haut.



The screenshot shows a Windows desktop environment. On the left, there is a folder icon labeled 'ExerciceGit'. To its right, a terminal window titled 'MINGW64:/c/Users/admin/Desktop/ExerciceGit' is open. The terminal displays the following commands and output:

```
admin@DESKTOP-JCA7HII MINGW64 ~/Desktop
$ mkdir ExerciceGit

admin@DESKTOP-JCA7HII MINGW64 ~/Desktop
$ cd ExerciceGit/

admin@DESKTOP-JCA7HII MINGW64 ~/Desktop/ExerciceGit
$ git init
Initialized empty Git repository in C:/Users/admin/Desktop/ExerciceGit/.git/

admin@DESKTOP-JCA7HII MINGW64 ~/Desktop/ExerciceGit (master)
$
```

FIGURE 27 – Les débuts en Windows

Fin de l'exercice 12.

Sommaire

1	Installation de Git	1
1.1	l'aide de git	2
1.2	Configuration requise de Git	3
2	Création d'un dépôt	5
3	Cloner un dépôt Git distant	11
3.1	Clone avec HTTPS	11
4	Création d'un compte GitHub	14
5	Défis : Git dans Windows	14
	Sommaire	16
	Références	18

Ce document a été écrit avec LaTeX.

Cette oeuvre, création, site ou texte est sous licence Creative Commons Attribution - Pas d'Utilisation commerciale - Partage dans les Mêmes Conditions 4.0 International. Pour accéder à une copie de cette licence, merci de vous rendre à l'adresse suivante <http://creativecommons.org/licenses/by-nc-sa/4.0/>

ou envoyez un courrier à Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.