Cybersécurité des services informatiques

PROFESSEURS : MR JOBARD

TP N°4 : GIT

Introduction

Dans ce TP, nous allons découvrir l'utilisation de Git, crée par Linus Torvald. Git est un outil open source très populaire parmi les développeurs, les personnes DevOps et les adeptes de scripting. Git permet de suivre l'évolution de tous les fichiers de votre projet pour gérer efficacement les différentes versions. C'est un système de contrôle de version décentralisé, aussi appelé DVCS. On peut l'utiliser localement en installant l'outil sur sa propre machine, ou bien en ligne via des plateformes comme GitHub ou GitLab. Pour illustrer cela, je vais utiliser mon ordinateur sous Windows, mais les étapes sont similaires sur Linux et MacOs.

A . Configuration de notre profil utilisateur

Avant de débuter un nouveau projet, je vais d'abord définir deux éléments dans la configuration de Git : le ***nom d'utilisateur et l'adresse e-mail***. Cela permettra d'associer les différentes actions, en particulier les "commit" sur les fichiers, à cet utilisateur, ce qui est indispensable pour le suivi.

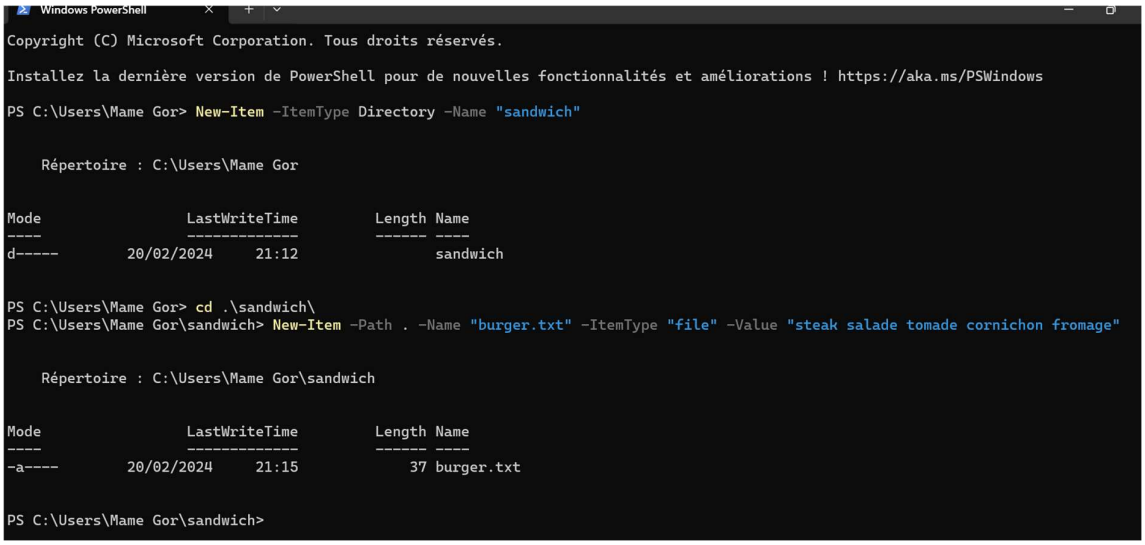
Commandes :

- ***git config --global*** user.name "Mame Gor"
- ***git config --global*** user.emai "mamegorciss5@gmail.com"

B. Initialisation d'un nouveau dépôt Git

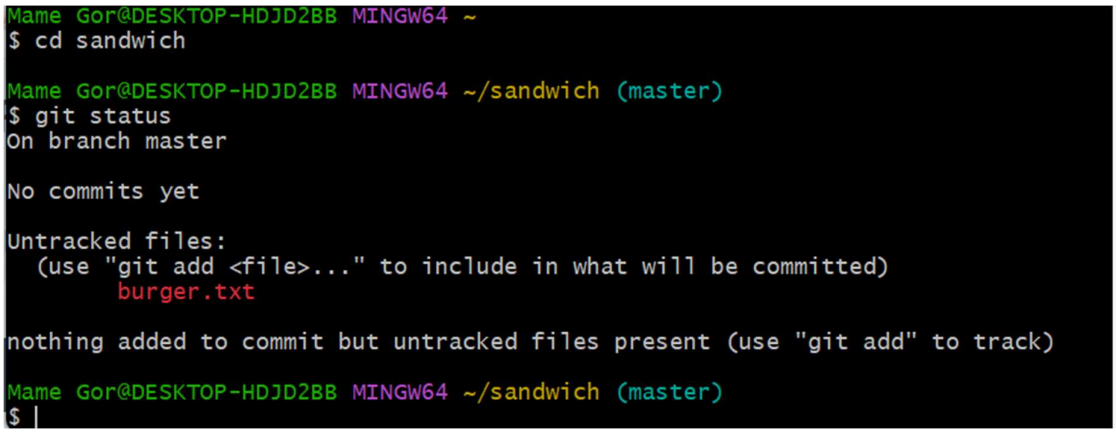
- Je crée d'abord un répertoire nommé sandwiche dans C:/users/Mame gor/ , ensuite un fichier burger dans sandwich qui contient la liste des ingrédients d'un burger 🍔 (steak , salade , tomate cornichon et fromage)

Je le fais en ligne de commande. En PowerShell, cela donne :



Après avoir créé ce dossier, je m'y rends pour démarrer ce nouveau projet et le lier à Git en lançant la commande : ***git init.***

- Maintenant vérifions avec ***git status*** l'état dans lequel se trouve notre dépôt



On peut voir que l'on est actuellement sur la branche "master", et que notre fichier burger.txt est non suivis par la phrase « Untracked files ». On peut voir également qu'il n'y a pas encore eu de commit via la phrase "No commits yet".

C. Action de commit

- Je prépare notre fichier burger.txt pour le commit avec la commande ***git add burger.txt***
- En utilisant de nouveau ***git status*** on peut voir que les modifications ont bien été placée dans l'index

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git add burger.txt

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   burger.txt


Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

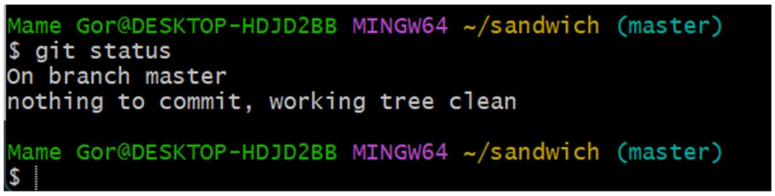
- Maintenant utilisons ***git diff –cached*** pour observer les différences entre l'index est la dernière version présente dans l'historique de révision. Cette commande affiche les changements entre l'index et le HEAD (qui est le dernier commit de cette branche). Cela montre ce qui a été ajouté à l'index et préparé pour un commit.

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git diff --cached
diff --git a/burger.txt b/burger.txt
new file mode 100644
index 0000000..aca4173
--- /dev/null
+++ b/burger.txt
@@ -0,0 +1 @@
+steak salade tomade cornichon fromage
\ No newline at end of file

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

- Le fichier de notre projet étant ajouté à Git, il est temps de réaliser un premier commit. En effet lorsqu'un commit est réalisé, une capture instantanée de notre code est effectuée. En cas de besoin de revenir en arrière suite à des modifications effectuées, il sera possible de restaurer le fichier tel qu'il était au moment du commit. Avec ***l'option "-m",*** on peut ajouter un commentaire qui sera associé à ce commit.

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git commit -m "nouvelle version"
[master (root-commit) 12c871a] nouvelle version
 1 file changed, 10 insertions(+)
 create mode 100644 burger.txt

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

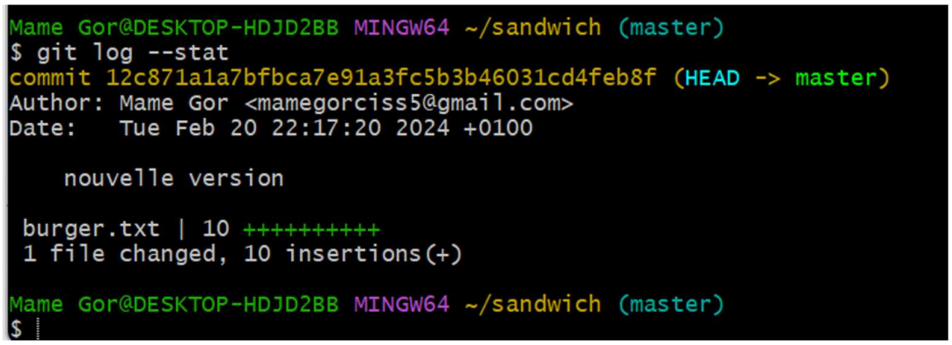
- En relançant la commande ***git status***, il est possible d'observer que nos modifications ont été correctement enregistrés.



D. Historique des commits

En utilisant la commande git log : on peut visualiser un historique des commits, avec notamment :

- La liste des changements effectués dans ce dépôt/ « 1 file change, 10 insertions » d'où une seul commit
- Le numéro (un hash cryptographique en format SHA1) du dernier commit effectuer : 12c871a1a7bfbca7e91a3fc5b3b46031cd4feb8f
- Le nom de l'auteur : Mame Gor
- La date et l'heure, et le commentaire : « Tue Feb 20 22 :17 :20 2024 »



- Maintenant je vais créer d'autres sandwiches hot_dog.txt, jambon.txt, salade_seasar.txt, king_fritte.txt , en commitant chaque modification séparément.

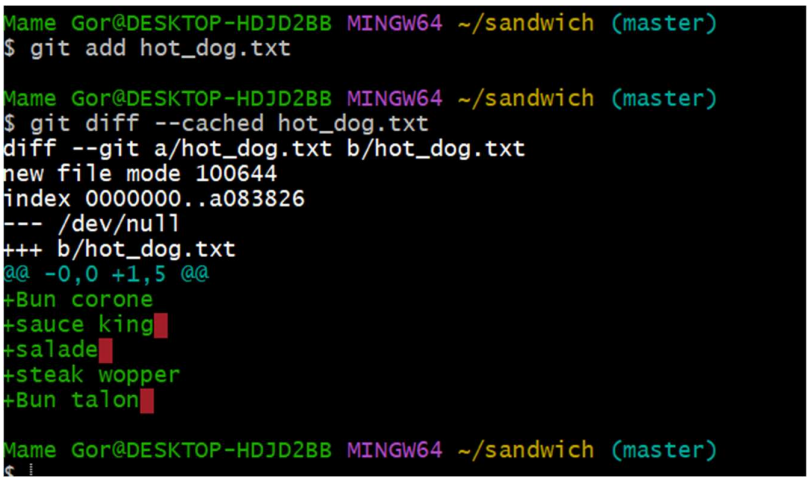
À chaque étape je vais essayer les commandes suivantes :

- ***git diff*** avant ***git add*** pour observer ce que J'ajoute à l'index ;
- ***git diff --cached*** après ***git add*** pour observer ce que je committe.

En suivant ces étapes, nous pourrons observer les modifications que j'ai apporté à mes fichiers à chaque étape du processus de commit dans Git. Cela nous aidera à comprendre ce qui est ajouté à l'index « **-ketchup,- tomate** » et ce qui est prêt à être commis. « **+Bun coron,+sauce king , +salade, +steak wopper, + Bun talon** »

- ***git diff*** avant ***git add***

- ***git diff --cached*** après ***git add***



- Afin de retrouver l'historique des modifications, je consulte ***git log*** et je vérifie que tout a bien été commité en utilisant ***git status***.

Historique des modifications avec ***git log***

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git log
commit e84c3de9b29785277f300914cf379bce74dd040a (HEAD -> master)
Author: Mame Gor <mamegorciss5@gmail.com>
Date:   Wed Feb 21 11:52:32 2024 +0100

    nouvelle version v1.1

commit 12c871a1a7bfbca7e91a3fc5b3b46031cd4feb8f
Author: Mame Gor <mamegorciss5@gmail.com>
Date:   Tue Feb 20 22:17:20 2024 +0100

    nouvelle version

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

git status : « nothing to commit »

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git status
On branch master
nothing to commit, working tree clean
```

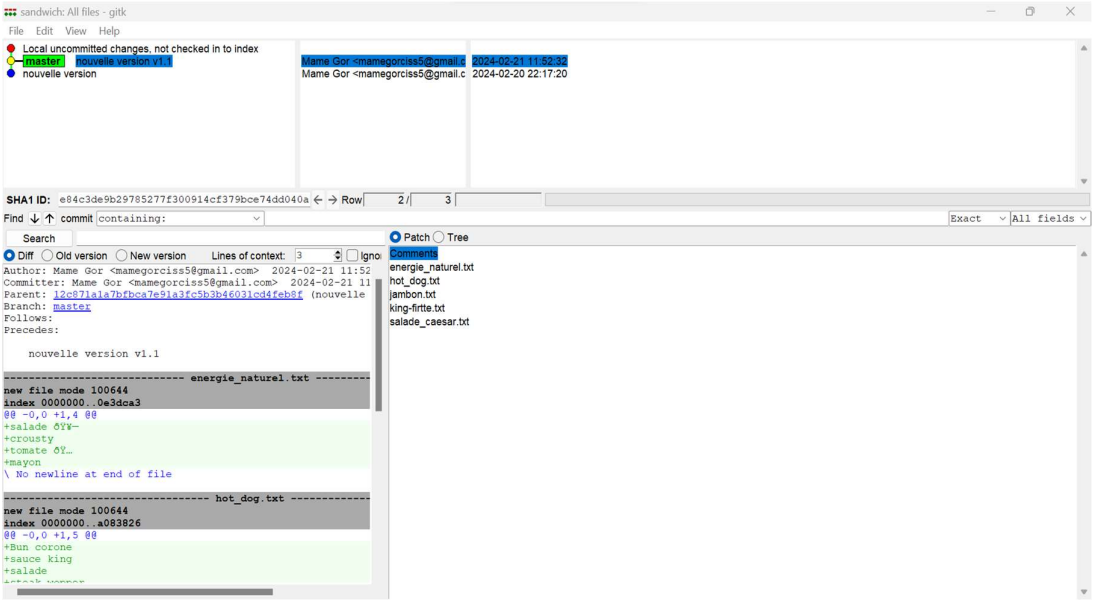
Quelques commandes

- **git log --graph --pretty=short**: la commande git log --graph --pretty=short affichera l'historique des commits avec un graphique représentant les relations entre les commits, et chaque commit sera affiché de manière succincte avec les informations essentielles
    - **graph**: Cette option permet d'afficher un graphique ASCII représentant la structure de l'historique des commits
    - **pretty=short**: Cette option spécifie le format de sortie pour chaque commit.

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git log --graph --pretty=short
* commit 1b3ea71aef29d2535897a724bbaf10dc23b55fa2 (HEAD -> master)
| Author: Mame Gor <mamegorciss5@gmail.com>
|
|     nouvelle version V1.1.2
|
* commit e84c3de9b29785277f300914cf379bce74dd040a
| Author: Mame Gor <mamegorciss5@gmail.com>
|
|     nouvelle version v1.1
|
* commit 12c871a1a7bfbca7e91a3fc5b3b46031cd4feb8f
  Author: Mame Gor <mamegorciss5@gmail.com>

      nouvelle version

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

- **gitg :** Il fournit une interface graphique pour explorer l'historique des commits, les branches, les tags et les différences entre les versions.

- **gitk :** est un autre outil graphique pour Git, qui fournit une interface graphique pour visualiser et naviguer dans l'historique des commits .il est écrit en Tcl/Tk, ce qui signifie qu'il est plus portable et peut être utilisé sur une variété de systèmes d'exploitation



E. Restaurer un fichier

- Je vais effectuer des modifications sur les sandwiches (burger.txt, king_fritte.txt et salade_caesar.text), les ajouter à l'index avec ***git add *txt*** sans effectuer de commit.

Avec **git status** on voit bien nos modifications qui sont effectués

- J’exécuter la commande ***git reset *txt*** sur les noms des fichiers.
- Maintenant le résultat de ***git staus*** dit que : aucune modification ajoutée est comité et nous propose ses deux commandes

git add :pour mettre en jour
git restore :pour ignorer les modifications dans le répertoire de travail

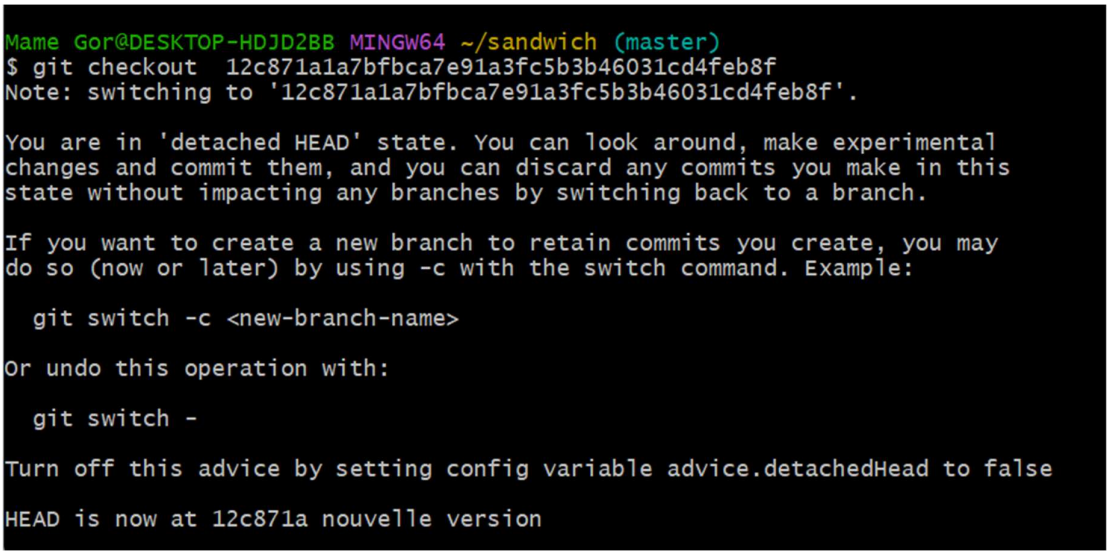
```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git reset *txt
Unstaged changes after reset:
M       burger.txt
M       king-firtte.txt
M       salade_caesar.txt

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   burger.txt
        modified:   king-firtte.txt
        modified:   salade_caesar.txt

no changes added to commit (use "git add" and/or "git commit -a")

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

- J’utilise maintenant la commande ***git checkout*** sur les noms des fichiers modifiés, qui récupère dans l’historique leurs versions correspondant au tout dernier commit. Ensuite je vérifie avec **git status** qu’il n’y a maintenant plus aucune modification à commiter

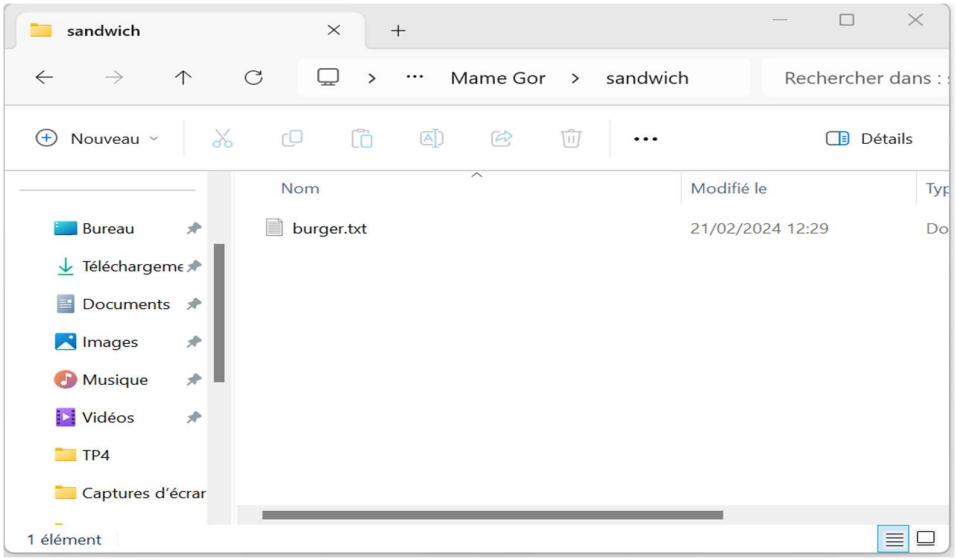
```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git checkout *txt
Updated 3 paths from the index

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git status
On branch master
nothing to commit, working tree clean

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

- Je vais choisir dans la liste de mon historique de dépôt ce commit : `12c871a1a7bfbca7e91a3fc5b3b46031cd4feb8f,` pour exécuté la commande ***chekout*** **.**

NB dans cette étape c'est pour la restauration des fichiers



- On est bien revenu en arrière au moment de notre commit choisi



- Le résultat de ***git satus*** après :

— On est plus sur la branche **master** : `HEAD detached at 12c871a`

— Pas de commit: `nothing to commit, working tree clean`

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich ((12c871a...))
$ git status
HEAD detached at 12c871a
nothing to commit, working tree clean

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich ((12c871a...))
$
```

- Maintenant je décide de retourner à la version plus récente de notre dépôt. Pour se faire exécutons la commande ***git checkout master***

NB : dans ce cas ***git checkout*** permet principalement de naviguer entre différentes branches

```
Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich ((12c871a...))
$ git checkout master
Previous HEAD position was 12c871a nouvelle version
Switched to branch 'master'

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$ git status
On branch master
nothing to commit, working tree clean

Mame Gor@DESKTOP-HDJD2BB MINGW64 ~/sandwich (master)
$
```

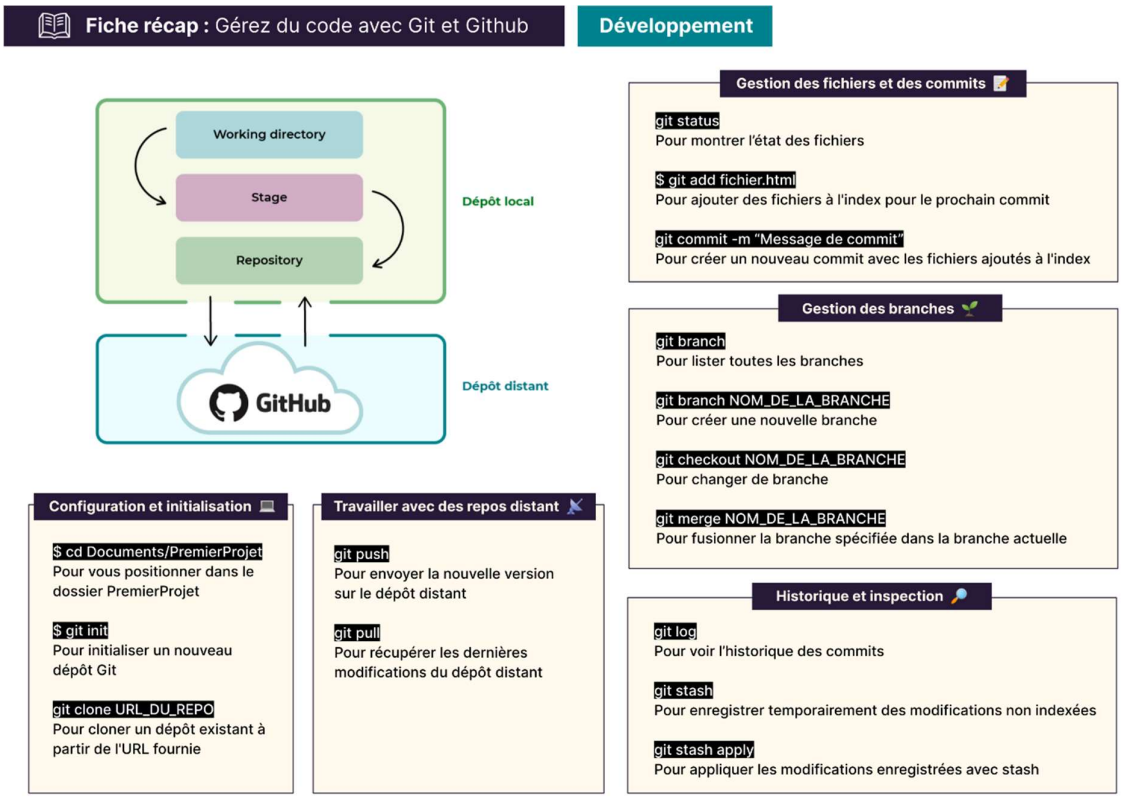
- Maintenant git status nous dit :

— Qu'on est dans la branche master : `On branch master`

— Rien à comité : `nothing to commit, working tree clean`

Conclusion

Git est un système de contrôle de version décentralisé qui révolutionne la façon dont les équipes de développement gèrent leur code source. Avec sa flexibilité, sa robustesse et sa richesse en fonctionnalités, Git s'est imposé comme l'outil de choix pour le suivi des modifications de code, la collaboration et la gestion des versions dans le développement logiciel moderne.



Source: https://openclassrooms.com/fr/courses/7162856-gerez-du-code-avec-git-et-github/7166041-tirez-le-maximum-de-ce-cours