TP GIT

Introduction :

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.

Pré requis :

- installer GIT
- une vm a disposition (windows 10 pro ici)

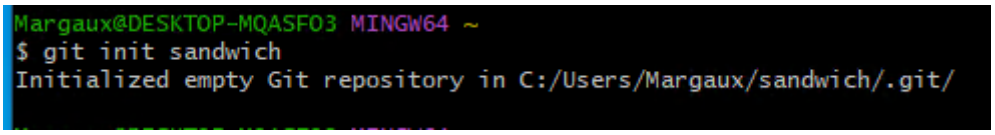
1 - Création d'un nouveau dépôt

Dans un premier temps, nous allons créer un nouveau dépôt vide grâce à l'interface en ligne de commande :

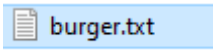
Pour créer un nouveau dépôt, on utilise la commande git init monrepo. Cette commande initialise un dépôt Git dans le répertoire monrepo (celui-ci est créé s'il n'existe pas). Ce repertoire contient alors à la fois une version de travail (dans monrepo) et un dépôt Git (dans monrepo/.git).

Question 2.1. Initialiser un nouveau dépôt Git dans un répertoire `sandwich`, et créez le fichier `burger.txt` qui contient la liste des ingrédients d'un burger, un ingrédient par ligne.

Pour créer in nouveau dépôt dans un répertoire sandwich faire : git init sandwich



Le fichier texte :



Question 2.2. Vérifiez avec `git status` l'état dans lequel se trouve votre dépôt. Vos modifications (l'ajout du fichier `burger.txt`) devraient être présentes seulement dans la copie de travail.

Avant de se faire, se rendre dans le bon répertoire actuel en faisant 'cd sandwich', réaliser ensuite git status. Nous pouvons observer que notre fichier texte n'a pas encore été ajouté pour réaliser le commit.

```
Margaux@DESKTOP-MQASFO3 MINGW64 ~
$ cd sandwich

Margaux@DESKTOP-MQASFO3 MINGW64 ~/sandwich (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        burger.txt.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Question 2.3. Préparez `burger.txt` pour le commit avec `git add burger.txt`. Utilisez `git status` à nouveau pour vérifier que les modifications ont bien été placées dans l'index. Puis, utilisez `git diff --cached` pour observer les différences entre l'index est la dernière version présent dans l'historique de révision (qui est vide).

Dans un premier temps, faire 'git add burger.txt (ici burger.txt.txt) ' puis git status. Nous voyons que le fichier a été ajouté et ne manque plus que de faire un commit.

```
Margaux@DESKTOP-MQASFO3 MINGW64 ~/sandwich (master)
$ git add burger.txt.txt

Margaux@DESKTOP-MQASFO3 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.txt
```

Nous pouvons visualiser les différences entre l'index (a/burger.txt.txt) et la dernière version (b/burger.txt.txt) présent dans l'historique de révision. Elle se trouve vide puisque nous n'avaons fait aucune modification :

```
Margaux@DESKTOP-MQASFO3 MINGW64 ~/sandwich (master)
$ git diff --cached
diff --git a/burger.txt.txt b/burger.txt.txt
new file mode 100644
index 0000000..7caa954
--- /dev/null
+++ b/burger.txt.txt
@@ -0,0 +1,5 @@
+steak
+salade
+tomate
+cornichon
+fromage
\ No newline at end of file
```

Question 2.4. Commitez votre modification avec `git commit -m "<votre_message_de_commit>"`. Le message entre guillemets doubles décrira la nature de votre modification (généralement ≤ 65 characteurs).

Avant de faire ce commit, il faut se donner les « droits » en faisant « git config –global user.name et user.email. Le commit peut être réalisé par la suite :

```
C:\Users\Margaux\sandwich>git config --global user.email "you@example.fr"

C:\Users\Margaux\sandwich>git config --global user.name "margaux"

C:\Users\Margaux\sandwich>git commit -m "premier commit"
[master (root-commit) c515d89] premier commit
 1 file changed, 5 insertions(+)
 create mode 100644 burger.txt.txt
```

Question 2.5. Exécutez à nouveau `git status`, pour vérifier que vos modifications ont bien été commités.

```
C:\Users\Margaux\sandwich>git status
On branch master
nothing to commit, working tree clean
```

Question 2.6. Essayez à présent la commande `git log` pour afficher la liste des changements effectués dans ce dépôt ; combien y en a-t-il ? Quel est le numéro (un hash cryptographique en format SHA1) du dernier commit effectué ?

Il y a un seul changement : premier depot

Le dernier commit se trouve en jaune.

```
C:\Users\Margaux\sandwich>git log
commit c515d892a5080044b8284fbb4d8c1e9928cb7ae7 (HEAD -> master)
Author: margaux <you@example.fr>
Date:   Tue Apr 11 10:21:21 2023 +0200

    premier commit
```

Question 2.7. Créez quelques autres sandwiches `hot_dog.txt`, `jambon_beurre.txt`. . .et/ou modifiez les compositions de sandwiches déjà créés, en commitant chaque modification séparément. Chaque commit doit contenir une et une seule création ou modification de fichier. Effectuez au moins 5 modifications différentes (et donc 5 commits différents). À chaque étape essayez les commandes suivantes :
— `git diff` avant `git add` pour observer ce que vous allez ajouter à l'index ;
— `git diff --cached` après `git add` pour observer ce que vous allez committer.
Note : la commande `git commit <file>` a le même effet que `git add <file>` suivie de `git commit`

Voici la création des fichiers hot_dog.txt et jambon_beurre.txt avec à chaque fois quelques commits pour chaque fichier texte:

Commandes utilisées : git add, git status, git commit -m « »

```
C:\Users\Margaux\sandwich>git add jambon-beurre
fatal: pathspec 'jambon-beurre' did not match any files

C:\Users\Margaux\sandwich>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   hot_dog.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        jambon-beurre.txt


C:\Users\Margaux\sandwich>git add jambon-beurre.txt

C:\Users\Margaux\sandwich>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   hot_dog.txt
        new file:   jambon-beurre.txt


C:\Users\Margaux\sandwich>git commit hot_dog.txt -m "modif hot_dog"
[master cb3c494] modif hot_dog
 1 file changed, 2 insertions(+)
 create mode 100644 hot_dog.txt
```

Question 2.8. ***Regardez à nouveau l'historique des modifications avec*** **git log** ***et vérifiez avec*** **git status** ***que vous avez tout commité.*** **Git** ***offre plusieurs interfaces, graphiques ou non, pour afficher l'historique. Essayez les commandes suivantes (*****gitg*** ***et*** **gitk** ***ne sont pas forcément installés) :***

— git log
— git log --graph --pretty=short
— gitg
— gitk

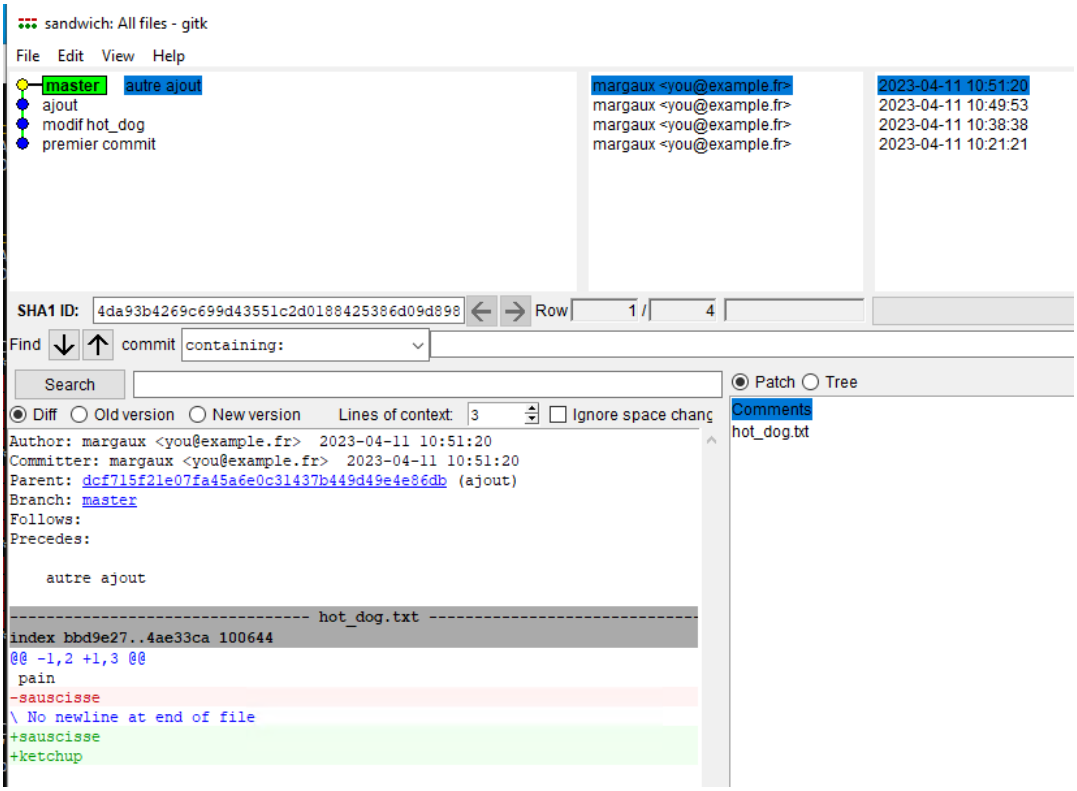
La commande git log permet de voir les différents commits qui ont été réalisés.

```
C:\Users\Margaux\sandwich>git log
commit 4da93b4269c699d43551c2d0188425386d09d898 (HEAD -> master)
Author: margaux <you@example.fr>
Date:   Tue Apr 11 10:51:20 2023 +0200

    autre ajout

commit dcf715f21e07fa45a6e0c31437b449d49e4e86db
Author: margaux <you@example.fr>
Date:   Tue Apr 11 10:49:53 2023 +0200

    ajout

commit cb3c494a5ea2d89b4799654c3cf49eb0664d1185
Author: margaux <you@example.fr>
Date:   Tue Apr 11 10:38:38 2023 +0200

    modif hot_dog

commit c515d892a5080044b8284fbb4d8c1e9928cb7ae7
Author: margaux <you@example.fr>
Date:   Tue Apr 11 10:21:21 2023 +0200

    premier commit
```

Ce qu'on obtient avec git log --graph --pretty=short :

```
C:\Users\Margaux\sandwich>git log --graph --pretty=short
* commit 4da93b4269c699d43551c2d0188425386d09d898 (HEAD -> master)
  Author: margaux <you@example.fr>

      autre ajout

* commit dcf715f21e07fa45a6e0c31437b449d49e4e86db
  Author: margaux <you@example.fr>

      ajout

* commit cb3c494a5ea2d89b4799654c3cf49eb0664d1185
  Author: margaux <you@example.fr>

      modif hot_dog

* commit c515d892a5080044b8284fbb4d8c1e9928cb7ae7
  Author: margaux <you@example.fr>

      premier commit
```

Ce que l'on obtient avec la commande Gitk :



2.3 Voyage dans le temps

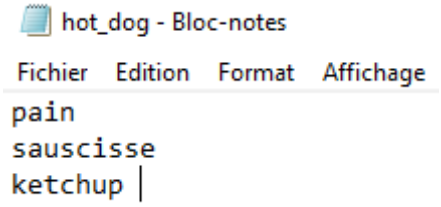
Questions faites toute d'un coup :

Question 2.9. ***Vous voulez changer d'avis entre les différents états de la Figure 1 ? Faites une modification d'un ou plusieurs sandwiches, ajoutez-la à l'index avec*** **git add** ***(vérifiez cet ajout avec*** **git status***), mais ne la commitez pas. Exécutez*** **git reset** ***sur le nom de fichier (ou les noms de fichiers) que vous avez préparés pour le commit ; vérifiez avec*** **git status** ***le résultat.***

Question 2.10. ***Votre modification a été « retirée » de l'index. Vous pouvez maintenant la jeter à la poubelle avec la commande*** **git checkout** ***sur le ou les noms des fichiers modifiés, qui récupère dans l'historique leurs versions correspondant au tout dernier commit. Essayez cette commande, et vérifiez avec*** **git status** ***qu'il n'y a maintenant plus aucune modification à commiter.***

Question 2.11. ***Regardez l'historique de votre dépôt avec*** **git log** ***; choisissez dans la liste un commit (autre que le dernier). Exécutez*** **git checkout COMMITID** ***où*** **COMMITID** ***est le numéro de commit que vous avez choisi. Vérifiez que l'état de vos sandwiches est maintenant revenu en arrière, au moment du commit choisi. Que dit maintenant*** **git status** ***?***

Voici ce que comporte hot_dog.txt avant la récupération du checkout : (rajouté ketchup)



Dans un premier temps avec git status nous pouvons voir que le fichier passe bien en modifier.

Nous allons donc l'ajouter en faisant git add puis créer un commit qui sera notre dernière version avant le checkout. Nous allons réalisé le git log pour savoir quels commit a été fait pour savoir quelle version précédente pouvons nous récupérer. Celle s'appelant « modif hot dog » sera celle que nous prendrons pour récupérer le plus récente version par rapport au commit que nous avons réalisé juste avant. Nous faisons donc git commit et le nom du commit.

```
C:\Users\Margaux\sandwich>git status
HEAD detached at cb3c494
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:   hot_dog.txt

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

C:\Users\Margaux\sandwich>git add hot_dog.txt

C:\Users\Margaux\sandwich>git commit -m "dernière versiona avant checkout"
[detached HEAD 3d81df5] dernière versiona avant checkout
 1 file changed, 2 insertions(+), 1 deletion(-)

C:\Users\Margaux\sandwich>git log
commit 3d81df5197273b759fe007deba000dedffaaad32 (HEAD)
Author: margaux <you@example.fr>
Date:   Tue Apr 11 11:33:02 2023 +0200

    dernière versiona avant checkout

commit cb3c494a5ea2d89b4799654c3cf49eb0664d1185
Author: margaux <you@example.fr>
Date:   Tue Apr 11 10:38:38 2023 +0200

    modif hot_dog

commit c515d892a5080044b8284fbb4d8c1e9928cb7ae7
Author: margaux <you@example.fr>
Date:   Tue Apr 11 10:21:21 2023 +0200

    premier commit

C:\Users\Margaux\sandwich>git checkout cb3c494a5ea2d89b4799654c3cf49eb0664d1185
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

  3d81df5 dernière versiona avant checkout
```

Ce dernier nous indique alors que la version précédente a bien été récupéré et qu'elle se prénomme maintenant comme la version que nous venons de récupérer (voir la dernier ligne cb3c494).

```
C:\Users\Margaux\sandwich>git checkout cb3c494a5ea2d89b4799654c3cf49eb0664d1185
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

  3d81df5 dernière versiona avant checkout

If you want to keep it by creating a new branch, this may be a good time
to do so with:

 git branch <new-branch-name> 3d81df5

HEAD is now at cb3c494 modif hot_dog
```

Conclusion :

Git permet donc de pouvoir travailler sur des projets à plusieurs tout en gardant la possibilité de récupérer les versions précédente du travail effectué. Cependant, il pet être difficile à être utiliser au début, en effet certaines enchainement de commande sont dur a assimiler au début si on n'en a jamais fait avant. I peut être très facil de créer des conflits entre les différentes versions, il est donc très important de rester rigoureux et prudent lors de la gestion de celles-ci.