2025.01.21 - Bloc 3 - TP4 Git

**

1 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.

— https://fr.wikipedia.org/wiki/Git

Lors cette première séance du TPs, nous allons nous initier à l' utilisation du Git et puis à la maintenance

d'un répertoire de travail de manière structurée et ordonnée.

2 Premiers pas avec Git

Git, un logiciel de contrôle de versions

Git [4] est un logiciel de contrôle de versions, il permet de sauvegarder l'historique du contenu

d'un répertoire de travail. Pour ce faire l'utilisateur doit régulièrement enregistrer (en créant

une révision ou commit) les modifications apportées au répertoire, il pourra ensuite accéder à

l'historique de toutes les modifications et inspecter l'état du dossier à chaque révision.

Git a la particularité de permettre de créer une copie d'un répertoire de travail, working copy, et

de synchroniser entre eux plusieurs copies du même répertoire, permettant la décentralisation

du travail.

De plus, Git permet d'utiliser une ou plusieurs branches de développement et de fusionner

entre elles ces branches.

2.1 Création d'un nouveau dépôt

Nous allons d'abord nous intéresser à l'aspect gestionnaire de versions de Git : comment enregistrer l'historique des modifications apportées à un projet. Pour obtenir un dépôt Git sur lequel travailler, deux options sont possibles :

1. création d'un dépôt vide (typiquement utilisé pour commencer un nouveau projet de développement) ;
2. copie (clone dans le langage de Git) d'un dépôt existant pour travailler sur cette copie de travail (typiquement utilisé pour collaborer avec les développeurs d'un projet en cours).

Examinons la première option. Git a plusieurs interfaces utilisateur. La plus complète étant l'interface en ligne de commande (CLI), nous nous servirons de celle-ci.

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).

- J'ai créé un nouveau dépôt en utilisant la commande : git init monrepo

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~
$ git init monrepo
Initialized empty Git repository in C:/Users/Mewo/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.

steak

salade

tomate

cornichon

fromage

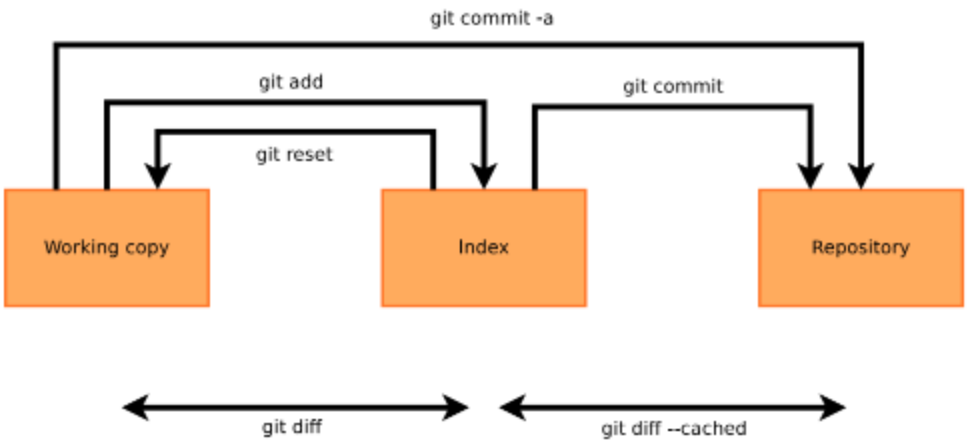
- J'ai maintenant initialisé un répertoire nommé "sandwich" et créé un fichier "burger.txt" qui contient la liste des ingrédients demandés.


```
Mewo@DESKTOP-D3JFAGO MINGW64 ~
$ git init sandwich
Initialized empty Git repository in C:/Users/Mewo/sandwich/.git/

Mewo@DESKTOP-D3JFAGO MINGW64 ~
$ cd sandwich

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ nano burger.txt
```

```
  GNU nano 8.2                                  burger.txt
steak
salade
tomate
cornichon
fromage
```

2.2 Add et Commit

Figure 1 – git add commit workflow



Pour être intégrée dans l'historique des révisions du dépôt (pour être "commitée") , chaque modification doit suivre le workflow montré en Figure 1 :

1. la modification est d'abord effectuée sur la copie de travail ;
2. elle est ensuite mémorisée dans une aire temporaire nommée index, avec la commande git add ;
3. Enfin, ce qui a été placé dans l'index peut être "commité" avec la commande git commit.

git diff, selon les paramètres d'appel peut être utilisé pour observer les différences entre les états en Figure 1 ;

le format d'affichage est le même de la command diff -u.

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.

```
Mewo@DESKTOP-D3JFAGO 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

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, utiliser git diff --cached pour observer les différences entre l'index est la dernière version présente dans l'historique de révision (qui est vide).

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git add burger.txt
warning: in the working copy of 'burger.txt', LF will be replaced by CRLF the next time Git touches it

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ ^C

Mewo@DESKTOP-D3JFAGO 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


Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git diff --cached
diff --git a/burger.txt b/burger.txt
new file mode 100644
index 0000000..760f26f
--- /dev/null
+++ b/burger.txt
@@ -0,0 +1,6 @@
+steak
+salade
+tomate
+cornichon
+fromage
+
```

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).

- Je dois créer un compte et prouver mon identité

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git commit -m "<Recette pour burger.txt>"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Mewo@DESKTOP-D3JFAGO.(none)')
```


- J'ai donc créé mon nom d'utilisateur ainsi qu'une adresse mail fictive

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git config --global user.name "Bishop"

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git config --global user.email "Bishop@hotmail.fr"
```

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git commit -m "<La première recette pour burger.txt>"
[master (root-commit) abfc44c] <La première recette pour burger.txt>
 1 file changed, 6 insertions(+)
 create mode 100644 burger.txt
```

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

```
Mewo@DESKTOP-D3JFAGO 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
```

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 ?

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git log
commit abfc44cf27a12ca0a666209bbad49222423c5421 (HEAD -> master)
Author: Bishop <Bishop@hotmail.fr>
Date:   Tue Jan 21 11:28:58 2025 +0100

    <La première recette pour burger.txt>
```

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 a le même effet que git add suivie de git commit.


```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ nano hot_dog.txt

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ nano jambon_beurre.txt
```

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git add hot_dog.txt
warning: in the working copy of 'hot_dog.txt', LF will be replaced by CRLF the next time Git touches it

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git add jambon_beurre.txt
warning: in the working copy of 'jambon_beurre.txt', LF will be replaced by CRLF the next time Git touches it
```

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git diff --cached
diff --git a/hot_dog.txt b/hot_dog.txt
new file mode 100644
index 0000000..21f01a0
--- /dev/null
+++ b/hot_dog.txt
@@ -0,0 +1,2 @@
+Saucisse
+Pain
diff --git a/jambon_beurre.txt b/jambon_beurre.txt
new file mode 100644
index 0000000..2c38670
--- /dev/null
+++ b/jambon_beurre.txt
@@ -0,0 +1,4 @@
+Jambon blanc d'Iran
+Beurre demi-sel breton
+baguette de Lorraine
+
```

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git commit -m "<Recette pour hot_dog et Jambon_beurre>"
[master f318fbb] <Recette pour hot_dog et Jambon_beurre>
 2 files changed, 6 insertions(+)
 create mode 100644 hot_dog.txt
 create mode 100644 jambon_beurre.txt
```

- J'ai rajouté deux autres recettes de sandwich avec les ingrédients à l'intérieur

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ nano Kebab.txt

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ nano Beyrouth.txt

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git diff--cached
git: 'diff--cached' is not a git command. See 'git --help'.

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git add Kebab.txt
warning: in the working copy of 'Kebab.txt', LF will be replaced by CRLF the next time Git touches it

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git add Beyrouth.txt
warning: in the working copy of 'Beyrouth.txt', LF will be replaced by CRLF the next time Git touches it
```



```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git commit -m "<Recette d'Orient>"
[master 671b91a] <Recette d'Orient>
 2 files changed, 15 insertions(+)
 create mode 100644 Beyrouth.txt
 create mode 100644 Kebab.txt
```

En totalité nous nous retrouvons avec plusieurs fichier .txt :

Burger; hot_dog; jambon_beurre; Kebab et Beyrouth

Chacun avec leurs ingrédients

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) :

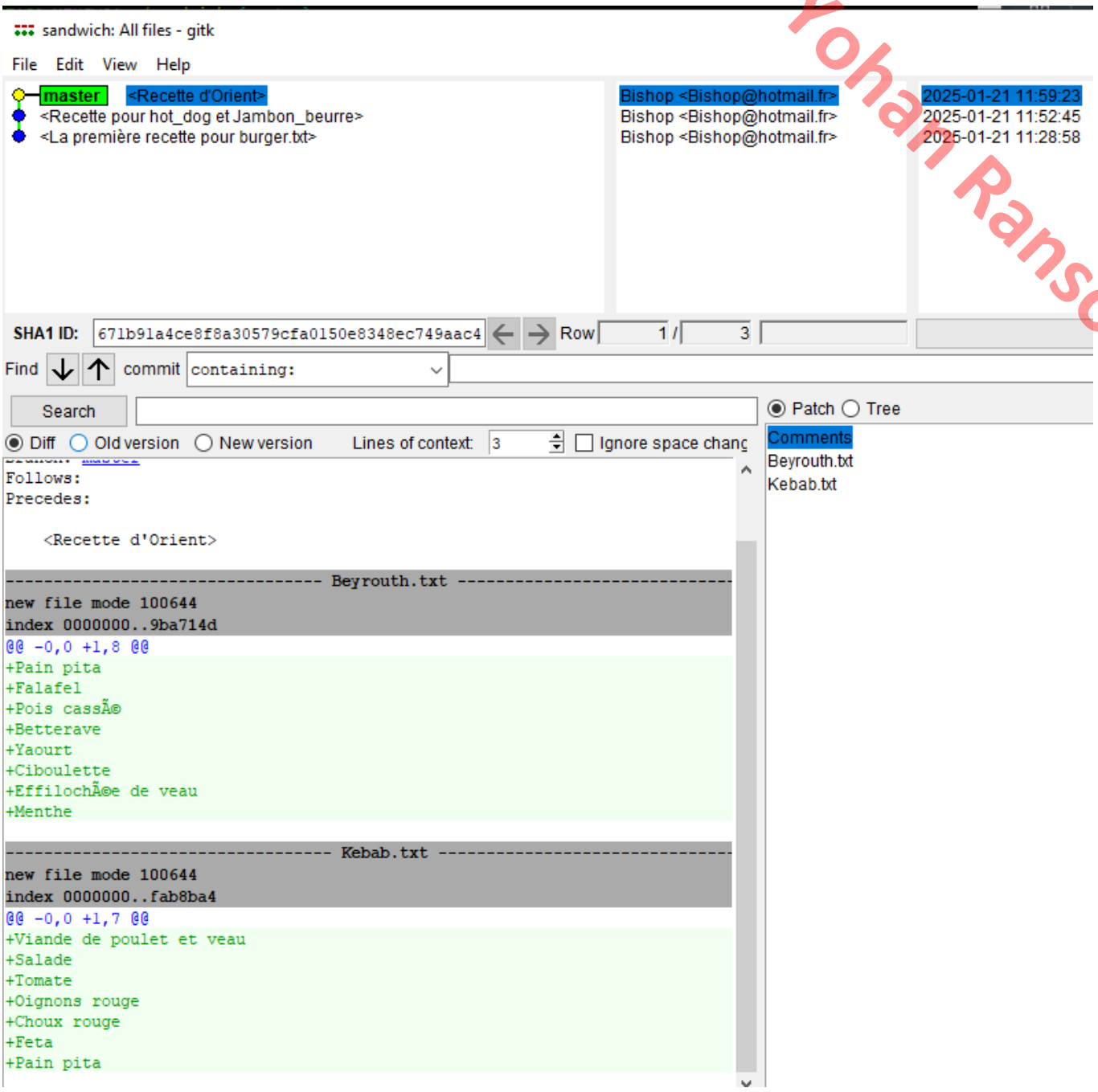
— git log

— git log --graph --pretty=short

— gitg

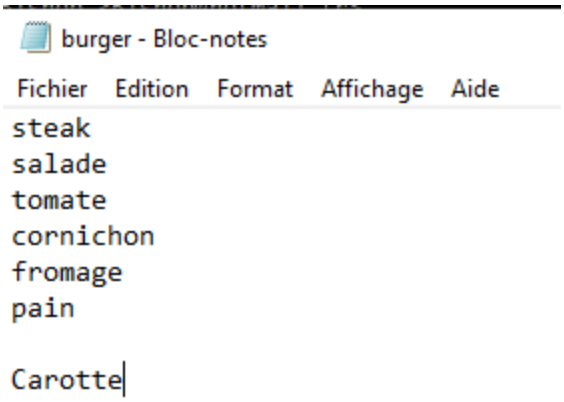
— gitk

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git log
commit 671b91a4ce8f8a30579cfa0150e8348ec749aac4 (HEAD -> master)
Author: Bishop <Bishop@hotmail.fr>
Date:   Tue Jan 21 11:59:23 2025 +0100

    <Recette d'Orient>

commit f318fbb8c04e4d2a6639df3905d637ff082bcc43
Author: Bishop <Bishop@hotmail.fr>
Date:   Tue Jan 21 11:52:45 2025 +0100

    <Recette pour hot_dog et Jambon_beurre>

commit abfc44cf27a12ca0a666209bbad49222423c5421
Author: Bishop <Bishop@hotmail.fr>
Date:   Tue Jan 21 11:28:58 2025 +0100

    <La première recette pour burger.txt>

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git status
On branch master
nothing to commit, working tree clean
```




```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git log --graph --pretty=short
* commit 671b91a4ce8f8a30579cfa0150e8348ec749aac4 (HEAD -> master)
| Author: Bishop <Bishop@hotmail.fr>
|
|     <Recette d'Orient>
|
* commit f318fbb8c04e4d2a6639df3905d637ff082bcc43
| Author: Bishop <Bishop@hotmail.fr>
|
|     <Recette pour hot_dog et Jambon_beurre>
|
* commit abfc44cf27a12ca0a666209bbad49222423c5421
  Author: Bishop <Bishop@hotmail.fr>

      <La première recette pour burger.txt>
```

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.




```
Mewo@DESKTOP-D3JFAGO 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

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

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.

git checkout est une commande très puissante. Elle vous permet de voyager entre différentes branches (voir plus loin) et aussi de revenir temporairement à une version précédente de votre copie de travail.

```
Mewo@DESKTOP-D3JFAGO 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

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

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 ?

git log n'affiche plus les commits postérieurs à l'état actuel, sauf si vous ajoutez l'option --all.

Attention, avec git checkout les fichiers de votre copie de travail sont modifiés directement par Git pour les remettre dans l'état que vous avez demandé. Si les fichiers modifiés sont ouverts par d'autres programmes (e.g. un éditeur de texte comme Emacs), il faudra les réouvrir pour observer les modifications.


```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich ((671b91a...))
$ git status
HEAD detached at 671b91a
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   burger.txt
```

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich ((671b91a...))
$ git log
commit 671b91a4ce8f8a30579cfa0150e8348ec749aac4 (HEAD, master)
Author: Bishop <Bishop@hotmail.fr>
Date:   Tue Jan 21 11:59:23 2025 +0100

    <Recette d'Orient>

commit f318fbb8c04e4d2a6639df3905d637ff082bcc43
Author: Bishop <Bishop@hotmail.fr>
Date:   Tue Jan 21 11:52:45 2025 +0100

    <Recette pour hot_dog et Jambon_beurre>

commit abfc44cf27a12ca0a666209bbad49222423c5421
Author: Bishop <Bishop@hotmail.fr>
Date:   Tue Jan 21 11:28:58 2025 +0100

    <La première recette pour burger.txt>

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich ((671b91a...))
$ git checkout abfc44cf27a12ca0a666209bbad49222423c5421
M       burger.txt
Previous HEAD position was 671b91a <Recette d'Orient>
HEAD is now at abfc44c <La première recette pour burger.txt>
```

Question 2.12. Vous pouvez retourner à la version plus récente de votre dépôt avec git checkout master. Vérifiez que cela est bien le cas. Que dit maintenant git status ?

```
Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich ((abfc44c...))
$ git checkout master
M       burger.txt
Previous HEAD position was abfc44c <La première recette pour burger.txt>
Switched to branch 'master'

Mewo@DESKTOP-D3JFAGO MINGW64 ~/sandwich (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   burger.txt
```

4 Git- Cheat sheet [1], [2]

Creation
Cloner localement un dépôt distant `git clone https://github.com/<remote>`
Créer un nouveau dépôt local `git init`
Changements locaux
Fichiers modifiés dans le répertoire de travail. À utiliser fréquemment ! `git status`
Intégrer les changements d'un fichier au prochain commit `git add <fichier>`
Retirer un fichier du prochain commit `git rm <fichier>`
Envoyer les changements vers le dépôt `git commit`

Historique des commits
Afficher tous les commits commençant par les plus récents `git log`
Afficher les changements dans le temps pour un fichier spécifique `git log -p <fichier>`
Responsable du dernier changement d'un fichier `git blame <fichier>`
Mettre à jour et publier
Télécharger toutes les modifications depuis REMOTE et les fusionner / les intégrer à HEAD `git pull <remote> <branch>`
Télécharger toutes les modifications depuis REMOTE, mais ne pas les intégrer à HEAD `git fetch <remote>`
Fusionner et Versionnage
Fusionner une branche dans le HEAD actuelle `git merge <branch>`

Références

[1] git cheat sheet. https://services.github.com/on-demand/downloads/github-git-cheat-sheet.

pdf.

[2] git cheat sheet interactif. http://ndpsoftware.com/git-cheatsheet.html.

[3] git livre. https://git-scm.com/book/en/v2.

[4] git page d'accueil. https://git-scm.com/.

[5] git tutoriel. https://git-scm.com/docs/gittutorial.

**