
TP GIT-BLOC 3-JOBARD GUILLAUME- 2023/2024- MEWO



magatte seye - 20 février 2024



1 Introduction

Version 0.9

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. **(Extrait du tp)**

— <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

2.1

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. **(Extrait du tp)**

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 répertoire contient alors à la fois une version de travail (dans `monrepo`) et un dépôt Git (dans `monrepo/.git`). **(Extrait du tp)**

2.2 Add et Commit

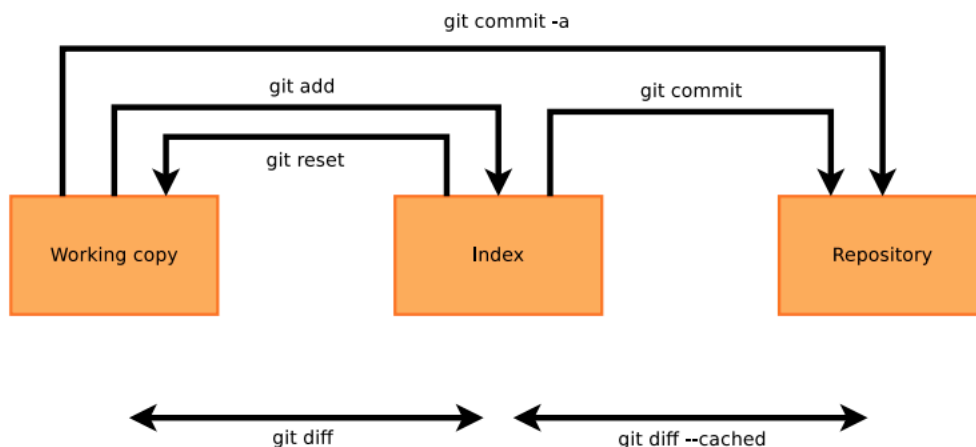


Figure 1 – git add commit workflow

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 commande `diff -u`. (Extrait du tp)

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 initialiser un nouveau depot git dans un repertoire j'utilise la commande
Git init sandwich

Sandwich est le nom de mon repertoire

Pour initialiser mon nouveau dépôt Git dans un répertoire appelé "sandwich" et créer le fichier "burger.txt" avec la liste des ingrédients d'un burger :

1) J'ouvre mon terminal ou bien un interface de ligne de commande et crée mon repertoire sandwich : **mkdir sandwich**

2) j'utilise la commande **git init** suivie du nom de mon repertoire (sandwich) pour initialiser mon nouveau dépôt Git: **git init sandwich**.

3)Après avoir initialisé mon dépôt de git ,je navigue dans mon repertoire "sandwich" en utilisant la commande **cd sandwich** et je tape la commande **ls** pour bien verifier que mon repertoire a été bien créer.

4) je crée mon fichier "burger.txt" avec la commande **touch** suivie du nom de mon fichier **touch burger.txt** . Et la commande **ls** pour bien verifier que mon fichier a été bien créer Je tape **nano burger.txt** dans mon terminal pour créer et ouvrir mon fichier "burger.txt".

```
sandwich — -zsh — 80x24
Last login: Tue Feb 20 10:49:27 on ttys000
[magatteseye@MacBook-Air-de-magatte-2 ~ % git init sandwich
Initialized empty Git repository in /Users/magatteseye/sandwich/.git/
[magatteseye@MacBook-Air-de-magatte-2 ~ % cd sandwich
[magatteseye@MacBook-Air-de-magatte-2 sandwich % touch burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % ls
burger.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

5)J'ouvre mon éditeur de texte avec la commande **nano** suivie du nom de mon fichier : **nano burger.txt** et j'ajoute la liste des ingrédients du burger dans ce fichier, un ingrédient par ligne.

6)Je sauvegarde et quitte mon éditeur de texte avec la commande **ctrl+x**

```
[magatteseye@MacBook-Air-de-magatte-2 ~ % cd sandwich
[magatteseye@MacBook-Air-de-magatte-2 sandwich % ls
burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % nano burger.txt
UW PICO 5.09 File: burger.txt
Deusx pains de formes rondes
un steak
salate
sauce tomate
oignon cornichon

```

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.

Pour vérifier l'état de mon dépôt Git, j'utilise la commande **git status**. Cette commande me montre l'état de mon copie de travail par rapport à l'index et par rapport à mon dernier commit.

```
[magatteseye@MacBook-Air-de-magatte-2 sandwich % nano burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % nano burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main

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
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git add burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main

No commits yet

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

magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

Le résultat de la commande **git status** montre que je suis dans la branche main et qu'il n'y a pas encore eu de commit et que mon fichier **burger.txt** n'est pas suivi ce qui indique qu'il existe dans mon copie de travail mais n'a pas encore été ajouté à l'index.

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 et la dernière version présente dans l'historique de révision (qui est vide).*

Pour préparer mon fichier "burger.txt" pour le commit en l'ajoutant à l'index avec **git add burger.txt**, puis pour vérifier que les modifications ont bien été placées dans l'index avec **git status**, et enfin pour observer les différences entre l'index et la dernière version présente dans l'historique de révision (qui est vide), j'utilise:

1) **git add burger.txt** pour ajouter mon fichier "burger.txt" à l'index

- 2) j'utilise la commande **git status** pour vérifier que les modifications ont été placées dans l'index.
- 3) J'utilise la commande **git diff --cached** pour voir les différences entre l'index et la dernière version présente dans l'historique de révision (qui est vide).

```
[magatteseye@MacBook-Air-de-magatte-2 sandwich % nano burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % nano burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main

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
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git add burger.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main

No commits yet

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

magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

```
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git diff --cached
diff --git a/burger.txt b/burger.txt
new file mode 100644
index 0000000..fdb1292
--- /dev/null
+++ b/burger.txt
@@ -0,0 +1,6 @@
+Deusx pains de formes rondes
+un steak
+salate
+sauce tomate
+oignon cornichon
+
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

Après je constate que le fichier "burger.txt" est listé comme modifié dans l'index par **git status**. Et **git diff --cached** a affiché les différences entre l'index et la dernière version dans l'historique de révision, qui est vide parce que j'avait pas fait de commit.

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 caractères).*

Pour commiter les modifications avec un message descriptif, j'utilise la commande **git commit -m "<votre_message_de_commit>"**. Et je remplace **<votre_message_de_commit>** par le message que je souhaite mettre **<Le burger le plus classique>**

```
magatteseye@MacBook-Air-de-magatte-2 sandwich % git commit -m "<Le burger le pl
us classique>"
[main (root-commit) fee7554] <Le burger le plus classique>
 1 file changed, 6 insertions(+)
 create mode 100644 burger.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main
nothing to commit, working tree clean
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

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

Pour vérifier que mes dernières modifications ont été bien committées, j'utilise la commande **git status**

```
magatteseye@MacBook-Air-de-magatte-2 sandwich % git commit -m "<Le burger le plus classique>"
[main (root-commit) fee7554] <Le burger le plus classique>
1 file changed, 6 insertions(+)
create mode 100644 burger.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main
nothing to commit, working tree clean
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

Le résultat de la commande `git status` confirme que les modifications ont été bien committées et que j'ai une propre copie de travail. Ce qui signifie qu'il n'y a plus de changements en attente d'être ajoutés au dépôt.

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 ?

Pour afficher la liste des changements effectués dans le dépôt et trouver le numéro (hash cryptographique en format SHA-1) du dernier commit effectué, j'utilise la commande **git log**.

```
magatteseye@MacBook-Air-de-magatte-2 sandwich % git log
commit fee75547935381d5061855fe0b498849888d3385 (HEAD -> main)
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 11:54:51 2024 +0100

    <Le burger le plus classique>
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

Y'a eu un seul changement effectué dans ce dépôt parce que j'ai effectué une seule commit. Et le numéro du dernier commit effectué est `fee75547935381d5061855fe0b498849888d3385`.

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.*

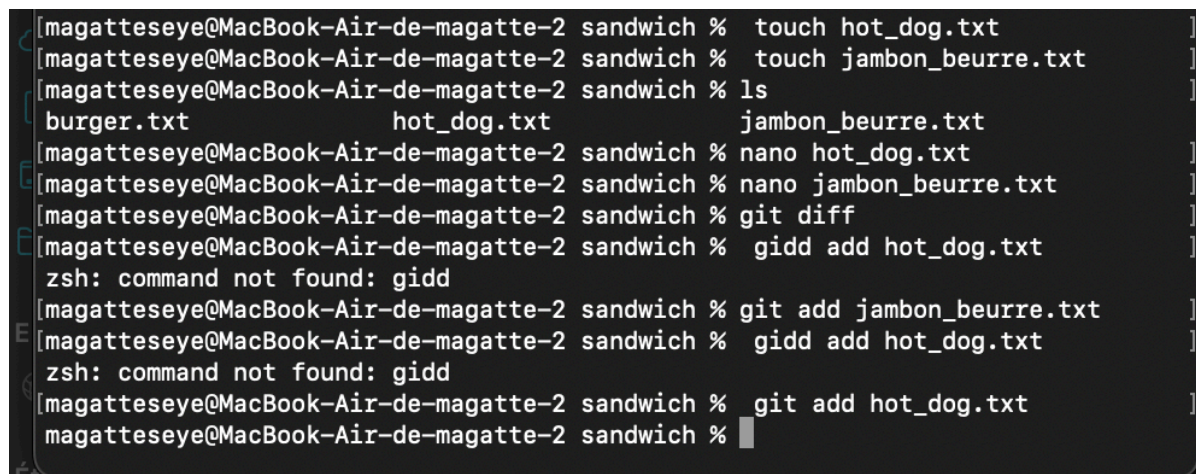
Note : la commande `git commit <file>` a le même effet que `git add <file>` suivie de `git commit`. 2

Pour effectuer plusieurs modifications et commits séparés dans mon dépôt Git, en ajoutant ou modifiant les fichiers `hot_dog.txt`, `jambon_beurre.txt`:

1) Je crée le fichier `hot_dog.txt` et `jambon_beurre.txt` (**`touch hot_dog.txt`**)

2) Je regarde les différences avant l'ajout à l'index (**`git diff`**).

3) J'ajoute le fichier `hot_dog.txt` à l'index (**`git add hot_dog.txt`**).



```
[magatteseye@MacBook-Air-de-magatte-2 sandwich % touch hot_dog.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % touch jambon_beurre.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % ls
burger.txt          hot_dog.txt          jambon_beurre.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % nano hot_dog.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % nano jambon_beurre.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git diff
[magatteseye@MacBook-Air-de-magatte-2 sandwich % gidd add hot_dog.txt
zsh: command not found: gidd
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git add jambon_beurre.txt
[magatteseye@MacBook-Air-de-magatte-2 sandwich % gidd add hot_dog.txt
zsh: command not found: gidd
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git add hot_dog.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

4) Je regarde à nouveau les différences après l'ajout à l'index (**`git diff --cached`**).

```

magatteseye@MacBook-Air-de-magatte-2 sandwich % git add jambon_beurre.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich % git diff --cached
diff --git a/hot_dog.txt b/hot_dog.txt
new file mode 100644
index 0000000..99d4a75
--- /dev/null
+++ b/hot_dog.txt
@@ -0,0 +1,4 @@
+pain
+saucisse
+fromage
+oignon frite
diff --git a/jambon_beurre.txt b/jambon_beurre.txt
new file mode 100644
index 0000000..1bfe78f
--- /dev/null
+++ b/jambon_beurre.txt
@@ -0,0 +1,5 @@
+jambon
+beurre
+salade
+sauce
+tomate
magatteseye@MacBook-Air-de-magatte-2 sandwich %

```

5) Je fais le commit avec un message descriptif (**git commit -m "Message de mon commit"**). Comme je les bien expliqué sur la question 2.4)

```

magatteseye@MacBook-Air-de-magatte-2 sandwich % git commit -m "<hot dog le plus classique>"
[main 5e39f09] <hot dog le plus classique>
2 files changed, 9 insertions(+)
create mode 100644 hot_dog.txt
create mode 100644 jambon_beurre.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main
nothing to commit, working tree clean
magatteseye@MacBook-Air-de-magatte-2 sandwich % git log
commit 5e39f093ef91e9e31345b56a4890dc8e8257ec24 (HEAD -> main)
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 12:23:14 2024 +0100

    <hot dog le plus classique>

commit fee75547935381d5061855fe0b498849888d3385
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 11:54:51 2024 +0100

    <Le burger le plus classique>
magatteseye@MacBook-Air-de-magatte-2 sandwich %

```

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

Je regarde à niveau l'historique des modifications en utilisant la commande git log et vérifie avec git status que j'ai tout commité.

```
magatteseye@MacBook-Air-de-magatte-2 sandwich % git commit -m "<hot dog le plus classique>"
[main 5e39f09] <hot dog le plus classique>
 2 files changed, 9 insertions(+)
 create mode 100644 hot_dog.txt
 create mode 100644 jambon_beurre.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
On branch main
nothing to commit, working tree clean
magatteseye@MacBook-Air-de-magatte-2 sandwich % git log
commit 5e39f093ef91e9e31345b56a4890dc8e8257ec24 (HEAD -> main)
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 12:23:14 2024 +0100

    <hot dog le plus classique>

commit fee75547935381d5061855fe0b498849888d3385
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 11:54:51 2024 +0100

    <Le burger le plus classique>
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

Et vu que ce sont les deux fichiers que j'ai commité, il l'est a bien affiché.

— Pour visualiser l'historique des modifications dans mon dépôt Git et vérifier que j'ai bien commité toutes les modifications j'utilise :

J'utilise les lignes de commandes:

1) git log pour afficher l'historique des commits.

2) git log --graph --pretty=short pour afficher l'historique des commits avec un graphique montrant les relations entre les branches.


```

[magatteseye@MacBook-Air-de-magatte-2 sandwich % git log
commit 5e39f093ef91e9e31345b56a4890dc8e8257ec24 (HEAD -> main)
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 12:23:14 2024 +0100

    <hot dog le plus classique>

commit fee75547935381d5061855fe0b498849888d3385
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 11:54:51 2024 +0100

    <Le burger le plus classique>
[magatteseye@MacBook-Air-de-magatte-2 sandwich % git log --graph --pretty=short
* commit 5e39f093ef91e9e31345b56a4890dc8e8257ec24 (HEAD -> main)
| Author: Magatte-Dev <magatteseye2023@icloud.com>
|
|    <hot dog le plus classique>
|
| * commit fee75547935381d5061855fe0b498849888d3385
|   Author: Magatte-Dev <magatteseye2023@icloud.com>
|
|    <Le burger le plus classique>
magatteseye@MacBook-Air-de-magatte-2 sandwich %

```

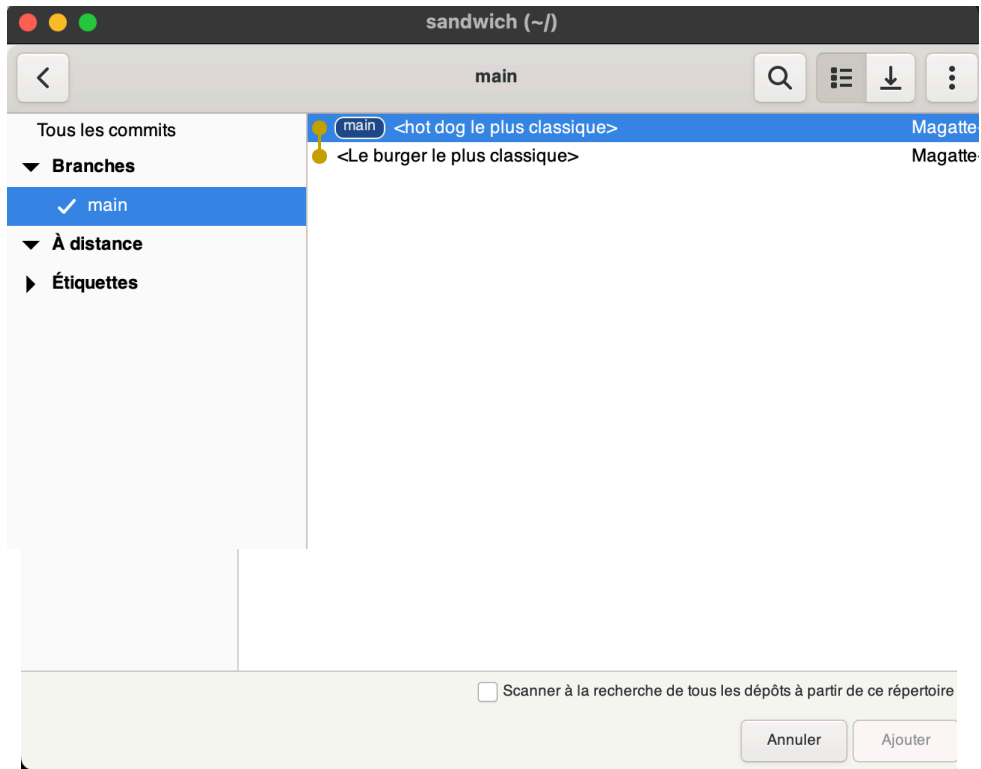
J'utilise les outils graphiques:

- 1)Gitg : Gitg est un visualiseur de dépôt pour Git, mais il n'est toujours pas installé par défaut, je l'est installé sur mon terminal.

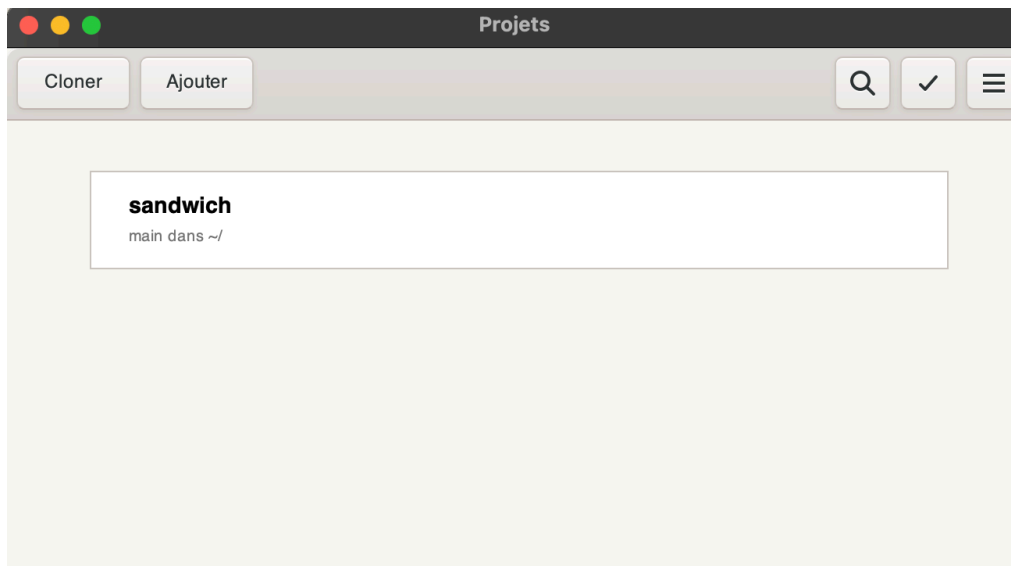
```

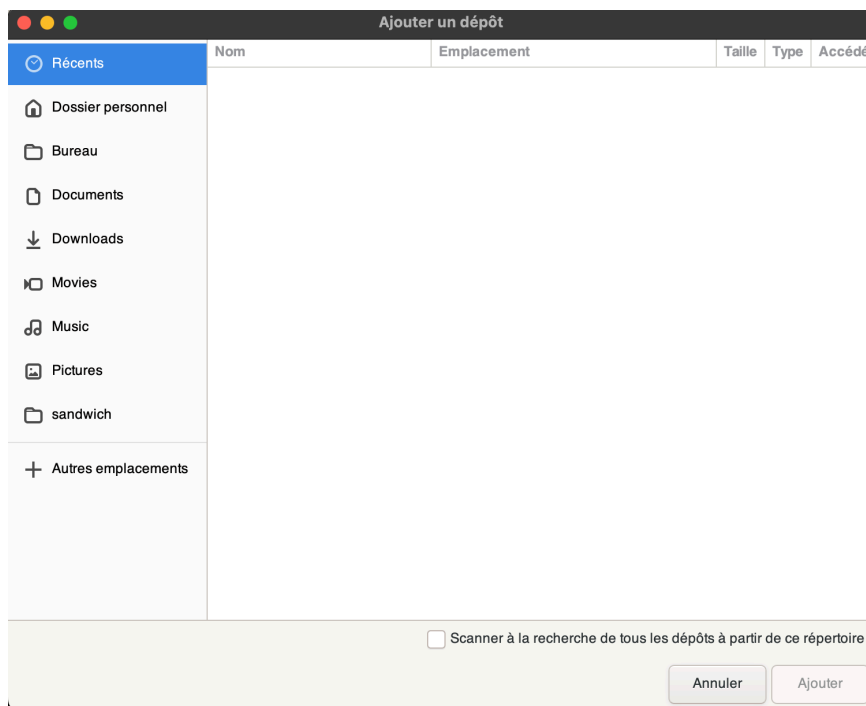
sandwich — gitg — 80x24
Last login: Tue Feb 20 12:33:39 on ttys000
/Users/magatteseye/.zshrc:export:1: not valid in this context: Fusion.app/Contents/Public
[magatteseye@MacBook-Air-de-magatte-2 ~ % cd sandwich
[magatteseye@MacBook-Air-de-magatte-2 sandwich % gitg

```



- Gitk : Gitk est un outil de visualisation graphique de l'historique des commits inclus dans Git.





Je constate que Gitk et Gitg sont similaires mais avec une interface utilisateur différente. Ils m'ont permis de visualiser l'historique des commits de manière graphique et de vérifier que toutes mes modifications ont été commitées.

2.3 Voyage dans le temps

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 commitiez 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.*

1) Je fais une modification sur un ou plusieurs sandwiches.

2) J'ajoute ces modifications à l'index avec **git add**.

3) Je vérifie l'ajout avec **git status**.

4) J'utilise **git reset nom_du_fichier_a_modifier** cette commande me permet d'annuler la modification d'un sandwich après l'avoir ajouté à l'index.

Git reset burger.txt

5) Je vérifie le résultat avec **git status**.

```
magatteseye@MacBook-Air-de-magatte-2 sandwich % git add burger.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
Sur la branche main
rien à valider, la copie de travail est propre
magatteseye@MacBook-Air-de-magatte-2 sandwich % git reset burger.txt
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
Sur la branche main
rien à valider, la copie de travail est propre
```

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.*

Pour "jeter à la poubelle" les modifications d'un ou plusieurs fichiers qui ont été retirées de l'index et revenir à la version correspondant au tout dernier commit, j'utilise la commande **git checkout** **nom_du_fichier_a_modifier**:

git checkout burger.txt

```
magatteseye@MacBook-Air-de-magatte-2 sandwich % git checkout burger.txt
0 chemin mis à jour depuis l'index
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
Sur la branche main
rien à valider, la copie de travail est propre
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

Après avoir exécuté la commande **git checkout**, je constate qu'il a 0 chemin mis à jour depuis l'index ce qui signifie que les

modifications du fichier sont annulées et qu'il a pas pas de modification en attente d'être commit.

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. Je regardez l'historique de votre dépôt avec **git log**.

1) Je choisisse un commit dans la liste (autre que le dernier) et notez son ID (hash).

2) J'exécute **git checkout COMMITID**, où COMMITID est l'ID du commit que j'ai choisi.

```
magatteseye@MacBook-Air-de-magatte-2 sandwich % git checkout 5e39f093ef91e9e31345b56a4890dc8e8257ec24
Note : basculement sur '5e39f093ef91e9e31345b56a4890dc8e8257ec24'.
[
Vous êtes dans l'état « HEAD détachée ». Vous pouvez visiter, faire des modifications
expérimentales et les valider. Il vous suffit de faire un autre basculement pour
abandonner les commits que vous faites dans cet état sans impacter les autres branches
]

Si vous voulez créer une nouvelle branche pour conserver les commits que vous créez,
il vous suffit d'utiliser l'option -c de la commande switch comme ceci :

    git switch -c <nom-de-la-nouvelle-branche>

Ou annuler cette opération avec :

    git switch -

Désactivez ce conseil en renseignant la variable de configuration advice.detachedHead à false

HEAD est maintenant sur 5e39f09 <hot dog le plus classique>
magatteseye@MacBook-Air-de-magatte-2 sandwich %
```

```

magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
HEAD détachée sur 5e39f09
rien à valider, la copie de travail est propre
magatteseye@MacBook-Air-de-magatte-2 sandwich % git log
commit 5e39f093ef91e9e31345b56a4890dc8e8257ec24 (HEAD, main)
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 12:23:14 2024 +0100

    <hot dog le plus classique>

commit fee75547935381d5061855fe0b498849888d3385
Author: Magatte-Dev <magatteseye2023@icloud.com>
Date: Tue Feb 20 11:54:51 2024 +0100

    <Le burger le plus classique>
magatteseye@MacBook-Air-de-magatte-2 sandwich % git checkout master
erreur : le spécificateur de chemin 'master' ne correspond à aucun fichier connu de git
magatteseye@MacBook-Air-de-magatte-2 sandwich % git status
HEAD détachée sur 5e39f09
rien à valider, la copie de travail est propre
magatteseye@MacBook-Air-de-magatte-2 sandwich %
magatteseye@MacBook-Air-de-magatte-2 sandwich %

```

Après avoir effectué **git checkout**, **git status** indique que je suis dans un état "HEAD détachée sur 5e39f09" , ce qui signifie que je suis pas sur une branche mais je peux passer à une branche ou crée une nouvelle branche pour sauvegarder mes modifications.

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 ?*

Pour retourner à la version plus récente de mon dépôt j'utilise la commande **Git checkout main** , **main** est le nom de ma branche principale.

```
[Airdemagatte2:sandwich magatteseye$ git checkout main
Basculement sur la branche 'main'
[Airdemagatte2:sandwich magatteseye$ pwd
/Users/magatteseye/sandwich
[Airdemagatte2:sandwich magatteseye$ git checkout main
Déjà sur 'main'
[Airdemagatte2:sandwich magatteseye$ git status
Sur la branche main
rien à valider, la copie de travail est propre
Airdemagatte2:sandwich magatteseye$
```

Git checkout montre que je suis déjà sur la branche main. Et la sortie de git status indique que mon dépôt est à jour avec la dernière version de la branche main et qu'il n'y a pas de modifications en attente de commit.

