

Nom : Essabri
Prénom : Youness

Num étudiant : 28705780

Nom : Gao
Prénom : Rick

Num étudiant : 21110625

Rapport de projet : outil de suivi et de versionnage de code

Introduction :

Dans le cadre de ce projet, nous allons réaliser un outil de suivi et de versionnage de code similaire à Git. Il s'agit d'un logiciel permettant de stocker, suivre et gérer les différentes versions d'un projet ou de fichiers. Il permet de récupérer des versions antérieures en cas de problème et facilite le travail collaboratif.

Ce projet est divisé en plusieurs parties, chacune comprenant des exercices pour progressivement concevoir le programme final. L'objectif est de comprendre le fonctionnement d'un logiciel de gestion de versions et d'explorer différentes structures de données impliquées dans sa mise en œuvre.

Outils utilisés :

Nous avons utilisé le compilateur GCC pour le langage de programmation C, l'IDE Visual Studio Code pour l'écriture du code, et le système d'exploitation Linux pour le développement et l'exécution du programme.

Dans ce projet, nous aurons besoin de 6 bibliothèques C : **dirent.h**, **stdio.h**, **stdlib.h**, **string.h** et **unistd.h**.

Dirent.h est une bibliothèque permettant d'ouvrir, lire et fermer les répertoires, d'obtenir des informations sur les attributs de fichiers et de gérer les erreurs.

Stdio.h est une bibliothèque permettant de lire, écrire et manipuler des fichiers, ainsi que d'afficher des données formatées sur la console.

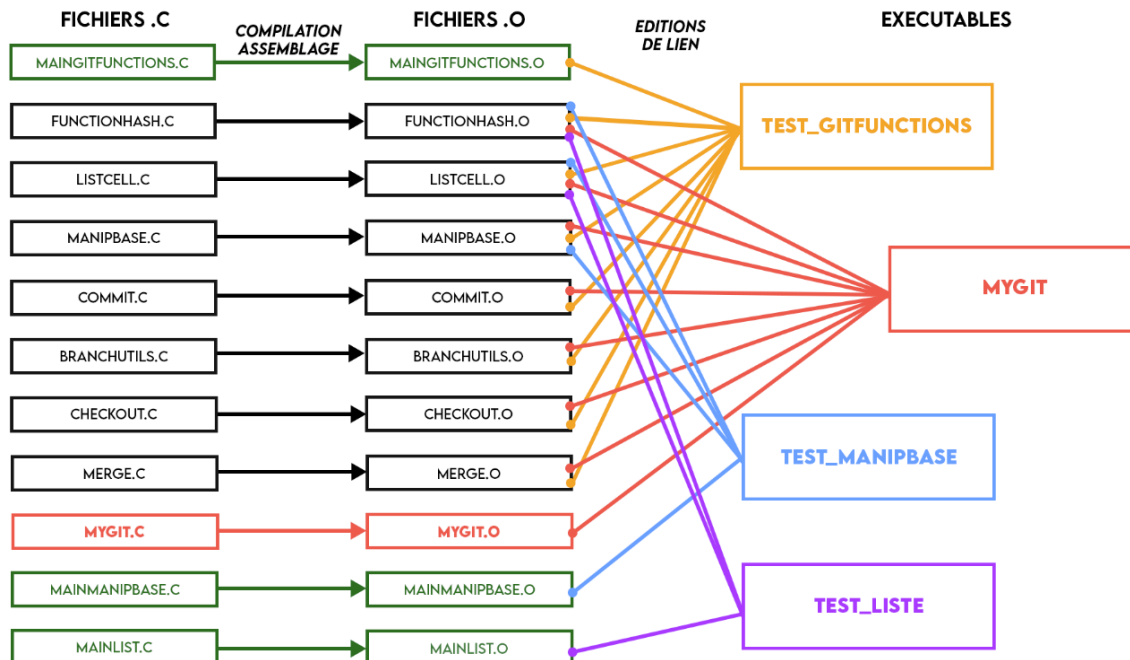
Stdlib.h est une bibliothèque pour l'allocation/désallocation de mémoire, la génération de nombres aléatoires, la manipulation de chaînes de caractères, etc...

String.h est une bibliothèque permettant de gérer les opérations sur les chaînes de caractères, y compris les opérations de copie, de recherche et de modification.

Unistd.h est une bibliothèque offrant des fonctionnalités spécifiques à l'environnement d'exécution du programme, telles que la gestion des fichiers, des répertoires et des processus.

Diagramme de compilation et d'édition de lien du fichier Makefile :

Le schéma ci-dessous illustre le processus de compilation, d'assemblage et d'édition de lien de notre programme à partir du fichier Makefile, montrant comment les différents fichiers sources et les dépendances sont traités pour générer les fichiers exécutables finaux.



Description des structures manipulées et la description globale du code :

| CELL | WORKFILE | WORKTREE | KVP | HASHTABLE |
|--------------|------------|---------------|-------------|-----------|
| CHAR* DATA | CHAR* NAME | WORKFILE* TAB | CHAR* KEY | KVP** T |
| CELL* NEXT | CHAR* HASH | INT SIZE | CHAR* VALUE | INT N |
| ----- | INT MODE | INT N | | INT SIZE |
| LIST = CELL* | | | | ----- |
| | | | | COMMIT = |
| | | | | HASHTABLE |


Pour réaliser ce projet, nous allons utiliser différents fichiers .h et .c pour implémenter les algorithmes et les structures de données nécessaires. Parmi celles-ci, nous retrouvons des structures de données intitulées WorkTree pour stocker et manipuler efficacement l'ensemble des fichiers et répertoires stockés chacun dans un WorkFile, ainsi que des structures de données intitulées Commit pour enregistrer les changements.

Avec notre implémentation, un commit est associé à l'enregistrement instantané d'un WorkTree, accompagné d'autres informations relatives au point de sauvegarde, comme par exemple l'auteur du commit, un message décrivant le commit, des informations permettant d'ordonner chronologiquement ce commit par rapport aux autres, avec notamment les champs predecessors et merge_predecessor (dans le cas d'un merge).

En termes d'organisation du code, nous avons choisi de séparer les différentes parties du programme en fichiers distincts, chacun d'entre eux correspondant à une fonctionnalité spécifique. Nous avons ainsi créé les fichiers suivants :

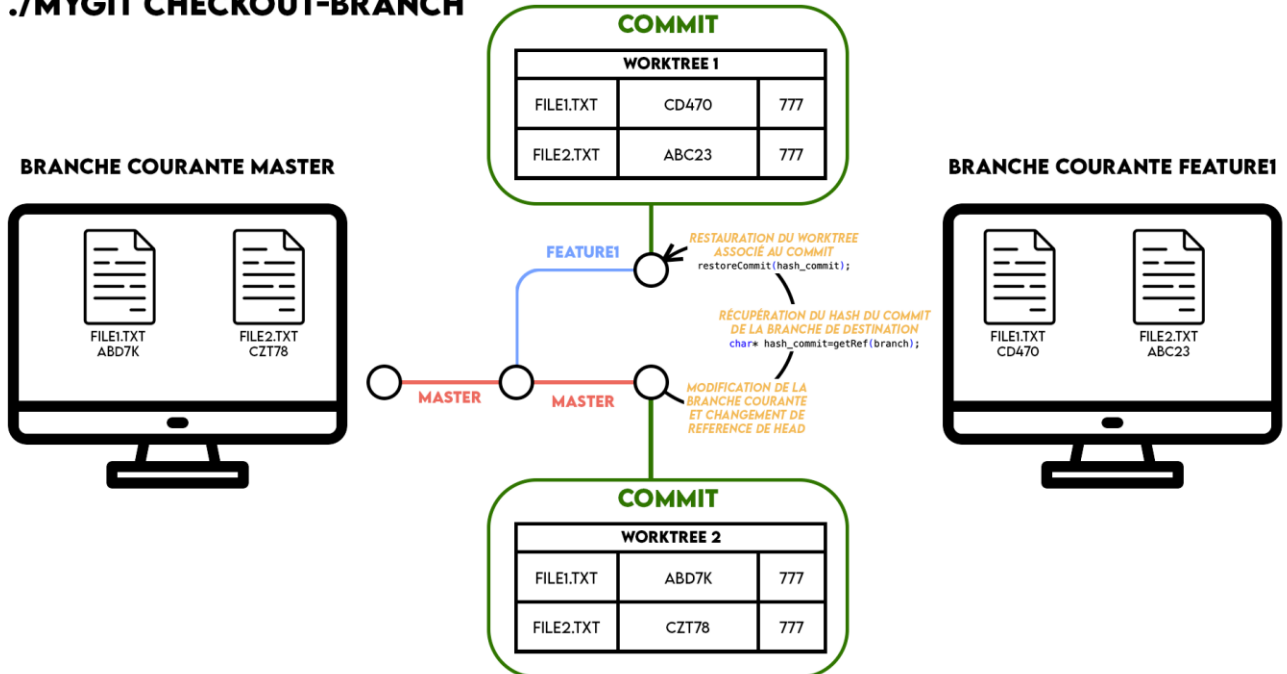
- `FonctionHash.c` et `FonctionHash.h` : qui implémentent les fonctions de hachage nécessaires à la création et à la manipulation des `WorkTrees`, des `commits` et des `blobs`. Ces fonctions sont utilisées pour calculer le hash de chaque `WorkTree`, `Commit` et `Blob` manipulé par le programme, qui sera ensuite stocké dans la base de données de notre implémentation git.
- `listCell.c` et `listCell.h`: qui contiennent la structure de données de liste chaînée utilisée pour stocker et manipuler les différents fichiers et répertoires.
- `ManipBase.c` et `ManipBase.h` : qui contiennent les fonctions nécessaires à la manipulation d'un ensemble de fichiers structuré en arborescence, avec la possibilité de créer instantanément plusieurs fichiers. Ces fonctions incluent la création, la suppression, la copie et la détection de conflits, ainsi que la gestion des répertoires. Leur utilisation est essentielle pour la création et la manipulation de `WorkFile` et `WorkTree` dans notre implémentation git.
- `commit.c` et `commit.h`: qui implémentent la structure et les fonctions nécessaires à la création et à la manipulation des `commits`, qui représentent les points de sauvegarde dans l'historique de version.
- `branchutils.c` et `branchutils.h`: qui contiennent les fonctions nécessaires pour gérer les branches, notamment pour les créer, les afficher et afficher leurs `commits`.
- `checkout.c` et `checkout.h`: qui implémentent les fonctions nécessaires pour passer d'une branche à une autre, pour se positionner sur un `commit` spécifique ou pour revenir à un état antérieur de l'arbre de travail.
- `merge.c` et `merge.h` : qui implémentent les fonctions nécessaires pour fusionner deux branches et gérer les conflits en cas de modifications simultanées d'un même fichier dans deux branches différentes
- Les fichiers `mainManipBase.c`, `mainGitFunctions.c`, `mainList.c`: qui implémentent les tests pour chaque fonctionnalité en cours de développement liée à l'exercice en question.

L'objectif de cette organisation est de faciliter la maintenance et l'évolution du code en permettant une meilleure isolation des différentes parties du programme. De plus, chaque fichier a été conçu pour être facilement compréhensible et modifiable de manière autonome.

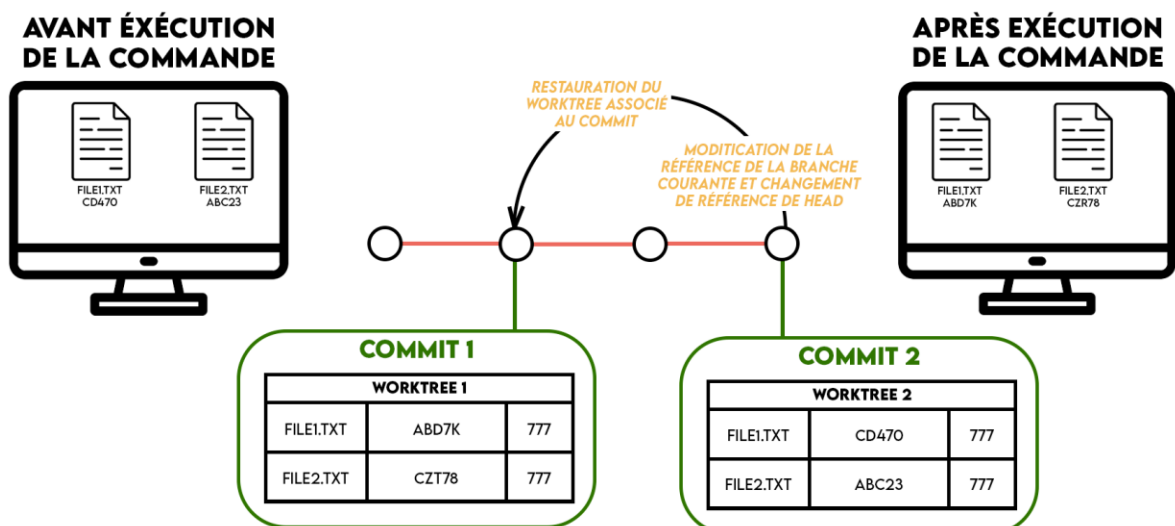
 **myGit.c**: regroupe toutes les fonctions principales nécessaires pour simuler le fonctionnement de Git. Le fichier est conçu pour être appelé avec des paramètres spécifiques qui définissent les actions à effectuer, telles que l'initialisation d'un nouveau dépôt Git, la création d'une nouvelle branche, la fusion de deux branches, la récupération de la liste des `commits`, et bien d'autres encore.

Pour illustrer quelques fonctions clés du programme, voici des schémas explicatifs pour les fonctionnalités complexes de fusion (merge), de passage d'une branche à une autre (checkout-branch) et de passage à un commit spécifique (checkout-commit). Ces schémas décrivent les étapes clés de ces fonctionnalités ainsi que les structures de données impliquées.

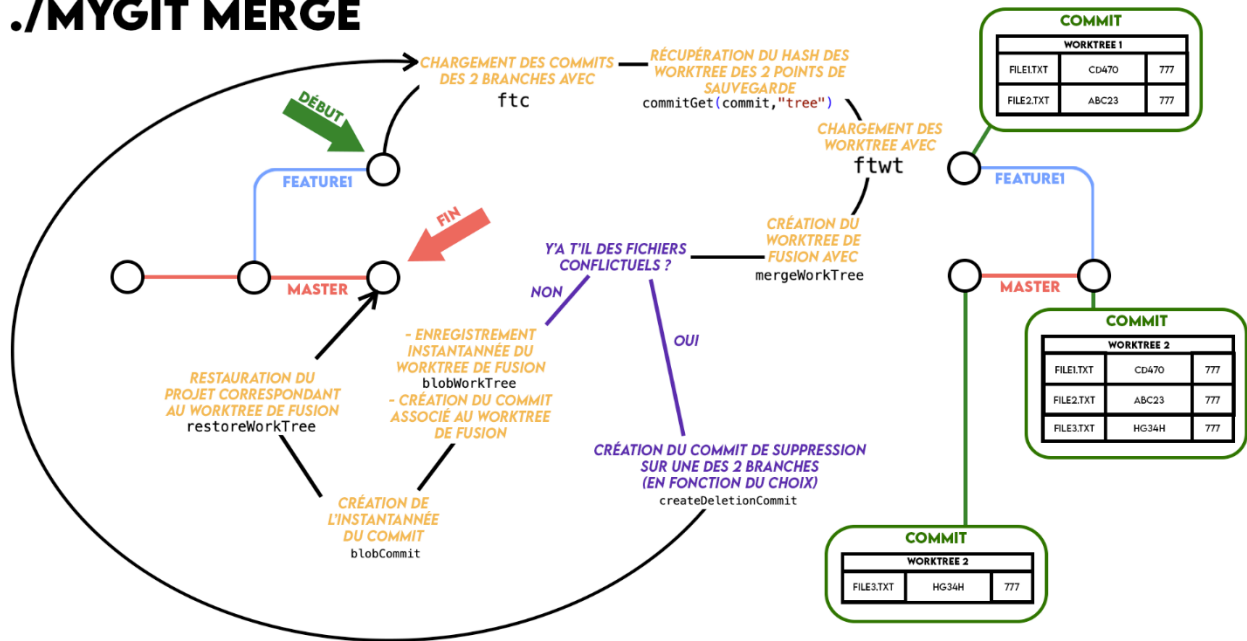
./MYGIT CHECKOUT-BRANCH



./MYGIT CHECKOUT-COMMIT



./MYGIT MERGE



Explications de nos choix de tests et résultats :

Nous allons réaliser un test complet de notre programme en simulant les actions suivantes :

1. Création du répertoire «.refs » et de ses branches principales.
2. Création de deux fichiers, file.txt et file1.txt, dans la branche master avec comme contenu "master".
3. Modification de file.txt et file1.txt, et création de file2.txt dans la branche Feature1 avec comme contenu "Feature1".
4. Merge de la branche Feature1 avec la branche master en choisissant manuellement les fichiers conflictuels à conserver (option 3).
5. Conservation de file1.txt de la branche master et de file.txt de la branche Feature1.
6. Vérification du résultat.

Afin de s'assurer du bon fonctionnement du programme et de détecter toute fuite de mémoire, chaque commande sera précédée de l'instruction "valgrind --leak-check=full".

Résultat théorique :

- Bon fonctionnement des différentes fonctionnalités du programme, aucune fuites mémoires et aucune erreur
- Le fichier file.txt contiendra le texte "Feature1"
- Le fichier file1.txt contiendra le texte "master"
- Le fichier file2.txt contiendra le texte "Feature1"



• Étape : Initialisation du dossier .refs

Commande exécutée : `./myGit init`

```
yns75@LAPTOP-ONI46KCL: /mnt/c/Users/essab/Desktop/Projet_RDD/Projet_ESSABRIYOUNESS_GAORICK$ valgrind --leak-check=full ./myGit init
==289== Memcheck, a memory error detector
==289== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==289== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==289== Command: ./myGit init
==289== Dossier .refs initialisé
==289==
==289== HEAP SUMMARY:
==289==   in use at exit: 0 bytes in 0 blocks
==289==   total heap usage: 3 allocs, 3 frees, 2,008 bytes allocated
==289==
==289== All heap blocks were freed -- no leaks are possible
==289==
==289== For lists of detected and suppressed errors, rerun with: -s
==289== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

On exécute les commandes suivantes dans la console :

```
echo "master" > file.txt
```

```
echo "master" > file1.txt
```

- Étape : Ajout de file.txt et file1.txt dans le fichier .add

Commande exécutée : `./myGit add file.txt file1.txt`

```
yns75@LAPTOP-0N146KCL:/mnt/c/Users/essab/Desktop/Projet_SDD/Projet_ESSABRIYOUNESS_GAORICK$ valgrind --leak-check=full ./myGit add file.txt file1.txt
==318== Memcheck, a memory error detector
==318== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==318== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==318== Command: ./myGit add file.txt file1.txt
==318==
Ajout du fichier : file.txt
Ajout du fichier : file1.txt
==318==
==318== HEAP SUMMARY:
==318==   in use at exit: 0 bytes in 0 blocks
==318==   total heap usage: 22 allocs, 22 frees, 4,688 bytes allocated
==318==
==318== All heap blocks were freed -- no leaks are possible
==318==
==318== For lists of detected and suppressed errors, rerun with: -s
==318== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

- Étape : Premier commit sur la branche courante master

Commande exécutée : `./myGit commit master -m firstcommitmaster`

```
yns75@LAPTOP-0N146KCL:/mnt/c/Users/essab/Desktop/Projet_SDD/Projet_ESSABRIYOUNESS_GAORICK$ valgrind --leak-check=full ./myGit commit master -m first
commitmaster
==331== Memcheck, a memory error detector
==331== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==331== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==331== Command: ./myGit commit master -m firstcommitmaster
==331==
Suppression de .add réussie
-----
Localisation de l'instantané du wt: ce/c03b79550a4fbef8b35caadd5fc6adb8d5a962bafd95235f7e349256e6aa.t
-----
Localisation de l'instantané du commit: 1acdbc85cc577adfb8d774b85a8a716c32949cf58421fb9df699f9a04c5b7856.c
-----
commit sur la branch master realise avec succes
==331==
==331== HEAP SUMMARY:
==331==   in use at exit: 0 bytes in 0 blocks
==331==   total heap usage: 74 allocs, 74 frees, 55,528 bytes allocated
==331==
==331== All heap blocks were freed -- no leaks are possible
==331==
==331== For lists of detected and suppressed errors, rerun with: -s
==331== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

- Étape : Affichage de la liste des références pour vérifier que master contient le hash de son dernier commit et HEAD contient le hash du dernier commit de la branche courante.

Commande exécutée : `./myGit list-refs`

```
yns75@LAPTOP-0N146KCL:/mnt/c/Users/essab/Desktop/Projet_SDD/Projet_ESSABRIYOUNESS_GAORICK$ valgrind --leak-check=full ./myGit list-refs
==390== Memcheck, a memory error detector
==390== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==390== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==390== Command: ./myGit list-refs
==390==
Les références :
- master      1acdbc85cc577adfb8d774b85a8a716c32949cf58421fb9df699f9a04c5b7856
- HEAD       1acdbc85cc577adfb8d774b85a8a716c32949cf58421fb9df699f9a04c5b7856
==390==
==390== HEAP SUMMARY:
==390==   in use at exit: 0 bytes in 0 blocks
==390==   total heap usage: 13 allocs, 13 frees, 35,990 bytes allocated
==390==
==390== All heap blocks were freed -- no leaks are possible
==390==
==390== For lists of detected and suppressed errors, rerun with: -s
==390== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```


- Étape : Création de la branche Feature1

Commande exécutée : `./myGit branch Feature1`

```
yns75@LAPTOP-ONI46KCL:/mnt/c/Users/essab/Desktop/Projet_5DD/Projet_ESSABRIYOUNESS_GAORIC$ valgrind --leak-check=full ./myGit branch Feature1
==406== Memcheck, a memory error detector
==406== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==406== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==406== Command: ./myGit branch Feature1
==406==
Branche Feature1 creer avec succes !
==406==
==406== HEAP SUMMARY:
==406==    in use at exit: 0 bytes in 0 blocks
==406==   total heap usage: 4 allocs, 4 frees, 2,073 bytes allocated
==406==
==406== All heap blocks were freed -- no leaks are possible
==406==
==406== For lists of detected and suppressed errors, rerun with: -s
==406== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

- Étape : Déplacement sur la branche Feature1 afin de pouvoir réaliser les manipulations nécessaires dessus

Commande exécutée : `./myGit checkout-branch Feature1`

```
yns75@LAPTOP-ONI46KCL:/mnt/c/Users/essab/Desktop/Projet_5DD/Projet_ESSABRIYOUNESS_GAORIC$ valgrind --leak-check=full ./myGit checkout-branch Feature1
==433== Memcheck, a memory error detector
==433== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==433== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==433== Command: ./myGit checkout-branch Feature1
==433==
Changement de branche réussi !
Vous etes actuellement sur Feature1
==433==
==433== HEAP SUMMARY:
==433==    in use at exit: 0 bytes in 0 blocks
==433==   total heap usage: 56 allocs, 56 frees, 10,542 bytes allocated
==433==
==433== All heap blocks were freed -- no leaks are possible
==433==
==433== For lists of detected and suppressed errors, rerun with: -s
==433== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

- Étape : On modifie les fichier file.txt et file1.txt, on crée le fichier file2.txt

On exécute les commandes suivantes dans la console :

`echo "feature1" >file.txt`

`echo "feature1" >file1.txt`

`echo "feature1" >file2.txt`

- Étape : On ajoute les fichiers dans le .add

Commande exécutée : `./myGit add file.txt file1.txt file2.txt`

```
yns75@LAPTOP-ONI46KCL:/mnt/c/Users/essab/Desktop/Projet_5DD/Projet_ESSABRIYOUNESS_GAORIC$ valgrind --leak-check=full ./myGit add file.txt file1.txt file2.txt
==496== Memcheck, a memory error detector
==496== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==496== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==496== Command: ./myGit add file.txt file1.txt file2.txt
==496==
Ajout du fichier : file.txt
Ajout du fichier : file1.txt
Ajout du fichier : file2.txt
==496==
==496== HEAP SUMMARY:
==496==    in use at exit: 0 bytes in 0 blocks
==496==   total heap usage: 38 allocs, 38 frees, 7,114 bytes allocated
==496==
==496== All heap blocks were freed -- no leaks are possible
==496==
==496== For lists of detected and suppressed errors, rerun with: -s
==496== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```


- Étape : On effectue un premier commit sur la branche Feature1

Commande exécutée : `./myGit commit Feature1 -m firstcommitFeature1`

```

yns75@LAPTOP-ONI46KCL:/mnt/c/Users/essab/Desktop/Projet_SDD/Projet_ESSABRIYOUNESS_GAORICH$ valgrind --leak-check=full ./myGit commit Feature1 -m
firstcommitFeature1
==509== Memcheck, a memory error detector
==509== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==509== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==509== Command: ./myGit commit Feature1 -m firstcommitFeature1
==509==
Suppression de .add réussie
-----
Localisation de l'instantané du wt: 20/933125270f7fe1865a2178c919c2aee3af9e6a5b5d70f8dc44d19336f097c7.t
-----
Localisation de l'instantané du commit: d0033a20413ad59cdc8279b57516bc975b3957ab1c32b05918ba7dcf205fc078.c
-----
commit sur la branch Feature1 realise avec succes
==509==
==509== HEAP SUMMARY:
==509==   in use at exit: 0 bytes in 0 blocks
==509==   total heap usage: 97 allocs, 97 frees, 69,437 bytes allocated
==509==
==509== All heap blocks were freed -- no leaks are possible
==509==
==509== For lists of detected and suppressed errors, rerun with: -s
==509== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

- Étape : On retourne sur la branche master (donc les fichiers file.txt et file1.txt vont retourner sur la version de la branche master)

Commande exécutée : `./myGit checkout-branch master`

```

yns75@LAPTOP-ONI46KCL:/mnt/c/Users/essab/Desktop/Projet_SDD/Projet_ESSABRIYOUNESS_GAORICH$ valgrind --leak-check=full ./myGit checkout-branch master
==579== Memcheck, a memory error detector
==579== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==579== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==579== Command: ./myGit checkout-branch master
==579==
Changement de branche réussi !
Vous etes actuellement sur master
==579==
==579== HEAP SUMMARY:
==579==   in use at exit: 0 bytes in 0 blocks
==579==   total heap usage: 56 allocs, 56 frees, 10,542 bytes allocated
==579==
==579== All heap blocks were freed -- no leaks are possible
==579==
==579== For lists of detected and suppressed errors, rerun with: -s
==579== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

- Étape : Fusion des branches master & Feature1

Commande exécutée : `./myGit merge Feature1 firstcommitmerge`

- On choisit l'option 3

```

yns75@LAPTOP-ONI46KCL:/mnt/c/Users/essab/Desktop/Projet_SDD/Projet_ESSABRIYOUNESS_GAORICH$ valgrind --leak-check=full ./myGit merge Feature1
firstcommitmerge
==599== Memcheck, a memory error detector
==599== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==599== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==599== Command: ./myGit merge Feature1 firstcommitmerge
==599==
Erreur merge: il y'a des conflits
Il y'a des collision, choisissez une des options suivantes:
1- Garder les fichiers de master (branche courante)
2- Garder les fichiers de Feature1 (remote branch)
3- Choix manuel des fichiers/repertoires en conflits à garder sur chaque branche

```

- On choisit les fichiers conflictuels à garder. Dans notre cas, on choisit de garder file1.txt de la branche master et file.txt de la branche Feature1.

```
yns75@LAPTOP-ONI46KCL:/mnt/c/Users/essab/Desktop/Projet_SDD/Projet_ESSABRIYOUNESS_GAORICK$ valgrind --leak-check=full ./myGit merge Feature1
firstcommitmerge
==599== Memcheck, a memory error detector
==599== Copyright (c) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==599== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==599== Command: ./myGit merge Feature1 firstcommitmerge
==599==
Erreur merge: il y'a des conflits
Il y'a des collision, choisissez une des options suivantes:
1- Garder les fichiers de master (branche courante)
2- Garder les fichiers de Feature1 (remote branch)
3- Choix manuel des fichiers/repertoires en conflits à garder sur chaque branche
3
Voici les fichier/repertoires en conflits:
-> file1.txt
-> file.txt
-----
Choisir 1 pour garder le fichier/repertoire dans master (branche courante) et 2 pour le garder dans Feature1 (branche distante)
-> file1.txt
1
```

- fusion entre la branche master et Feature1 réalisée avec succès

```
Voici les fichier/repertoires en conflits:
-> file1.txt
-> file.txt
-----
Choisir 1 pour garder le fichier/repertoire dans master (branche courante) et 2 pour le garder dans Feature1 (branche distante)
-> file1.txt
1
-> file.txt
2
Suppression de .add réussie
-----
Localisation de l'instantané du wt: be/804d4daad8dd45e8bb68aa6eff0e0daa86afaa88c0f3171879c8391c331e8.t
-----
Localisation de l'instantané du commit: de54ab2ea35edc44dac2adab07eeef2ae2116323b3cdfba11c3a7b03aa99682d.c
-----
commit de suppression sur la branch master realise avec succes
Suppression de .add réussie
-----
Localisation de l'instantané du wt: 3e/c1f089d2c1bb88fce6db440b11174d3c71ddcec8c9328f7be7713063a6d8bf.t
-----
Localisation de l'instantané du commit: a964cae67b8faa0407049dea272092da2f91941fcc75cbaf46e89d8b50925eac.c
-----
commit de suppression sur la branch Feature1 realise avec succes
Fusion de master avec Feature1 reussie
Suppression de Feature1 réussie
==599==
==599== HEAP SUMMARY:
==599==   in use at exit: 0 bytes in 0 blocks
==599== total heap usage: 777 allocs, 777 frees, 216,556 bytes allocated
==599==
==599== All heap blocks were freed -- no leaks are possible
==599==
==599== For lists of detected and suppressed errors, rerun with: -s
==599== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Résultats pratique :

Après avoir effectué le test complet décrit plus haut, nous avons obtenu les résultats suivants :

- La création du répertoire .refs et de ses branches principales s'est déroulée sans erreur.
- La création des fichiers file.txt et file1.txt dans la branche master a également été effectuée avec succès. Les contenus de ces fichiers ont bien été définis comme "master".
- La modification des fichiers file.txt et file1.txt, ainsi que la création du fichier file2.txt dans la branche Feature1, ont été réalisées sans problème. Les contenus de ces fichiers ont bien été définis comme "Feature1".
- Le merge entre la branche master et la branche Feature1 s'est déroulé comme prévu, avec une demande de choix de fichiers conflictuels à garder. Nous avons choisi de garder le fichier file1.txt de la branche master et le fichier file.txt de la branche Feature1, comme indiqué dans la section précédente.
- Après le merge, nous avons vérifié que les fichiers file.txt, file1.txt et file2.txt contenaient bien les contenus attendus : "Feature1" pour file.txt, "master" pour file1.txt et "Feature1" pour file2.txt.

```
yns75@LAPTOP-ONI46KCL: /mnt/c/Users/yns75/AppData/Local/Programs/Python/Python38-32/Scripts $ cat file.txt
Feature1
yns75@LAPTOP-ONI46KCL: /mnt/c/Users/yns75/AppData/Local/Programs/Python/Python38-32/Scripts $ cat file1.txt
master
yns75@LAPTOP-ONI46KCL: /mnt/c/Users/yns75/AppData/Local/Programs/Python/Python38-32/Scripts $ cat file2.txt
Feature1
```

De plus, aucune fuite de mémoires et aucune erreur n'a été détectée lors de l'exécution du programme, comme vérifié grâce à l'utilisation de l'instruction « valgrind --leak-check=full » avant chaque commande

Nous pouvons donc conclure que le programme fonctionne correctement et est capable de gérer les merges entre différentes branches, en permettant à l'utilisateur de choisir les fichiers à conserver en cas de conflit. De plus, il est également capable de gérer les commits et les changements de branche.

Conclusion :

En conclusion, nous avons développé un outil de suivi et de versionnage de code similaire à Git, qui permet de gérer des versions différentes d'un projet et de fusionner des branches. Nous avons implémenté des fonctionnalités telles que la création de commit, la gestion des branches, le changement de branche, le retour à une ancienne version et la fusion (merge) entre branches avec plusieurs options dont celle donnant la possibilité de choisir les fichiers à conserver en cas de conflit.

De plus, nous avons ajouté d'autres fonctionnalités, notamment celles d'historique des commits pour une meilleure visualisation de l'historique des modifications du projet. Cependant, ce programme peut encore être amélioré pour répondre à des besoins spécifiques. Par exemple, une méthode de fusion basée sur l'algorithme de Needleman-Wunsch pourrait être implémentée pour améliorer la fusion entre les branches, en identifiant les zones de différences du code et en les fusionnant de manière plus intelligente. Ainsi, cela permettrait d'obtenir des fusions de fichiers plus précises et plus cohérentes, en minimisant les conflits et les modifications inutiles.