Rapport de projet

Le but de notre projet est de réaliser un outil de gestion de version, c'est-à-dire un outil qui permet de stocker, de suivre et de gérer plusieurs versions d'un projet. Il permet un accès facile aux anciennes versions effectuées et de récupérer une ancienne version en cas de perte ou de problème. De plus, ces outils sont très utiles pour un travail de groupe car ils donnent la possibilité de fusionner différentes versions du projet.

Dans le cadre de notre projet nous avons utilisé plusieurs structures, qui sont :

-La "List" est une simple liste chaînée avec un champ "data".
-Le "WorkFile" représente un fichier/répertoire avec son nom de fichier, son hash et son mode d'autorisation sur le fichier.
-Le "WorkTree" est un tableau de "WorkFile", avec une taille maximale et le nombre d'éléments qu'il contient actuellement.
-Le "kvp" est une paire de clé-valeur.
-Le "Commit" est un tableau de "kvp" avec une taille maximale et le nombre d'éléments qu'il contient actuellement.

Dans les fichiers .h nous avons défini 2 "define" : "NBWF" pour le nombre maximum de workfile que le tableau de workfile doit contenir, et "NBC" pour le nombre maximum de commit que le tableau de kvp doit contenir.

Le fichier List.c gère les listes et le hash en proposant des fonctions de gestion pour les listes ainsi que le calcul du hash d'un fichier ou répertoire, la copie d'un fichier vers un autre et enfin la création d'un enregistrement instantané d'un fichier.

Le fichier WorkTree.c gère les WorkFile et WorkTree en proposant des fonctions de gestion pour les WorkFile et Worktree. Il permet également d'enregistrer et de restaurer un instantané d'un WorkTree.

Le fichier Commit.c gère la majeure partie de notre projet en proposant des fonctions de gestion pour les Commit, les branches et les références. On rappelle qu'une branche est une pseudo liste chaînée représentant des Commits. Chacun de ces Commits a une clé "predecessor" qui a pour valeur le hash du Commit précédent, sauf pour le premier. Afin de manipuler facilement les branches, nous utilisons des références qui sont des pointeurs vers des Commit spécifiques.

Le fichier Makefile compile avec les options -g -Wall -Wextra -pedantic pour avoir des symboles de débogage générés par le compilateur, pour afficher tous les avertissements de compilation ainsi que des avertissements supplémentaires que le -Wall n'affiche pas, et pour respecter la norme du langage C.Cela facilite également la compilation.

Et enfin, le fichier myGit.c est notre main principale contenant toutes les fonctions précédentes. Nous avons ainsi réalisé notre propre outil de gestion de version en simulant certaines commandes Git telles que init, add, commit, checkout et merge. Nous pouvons presque les utiliser de la même manière que les commande de Git, sauf que la nôtre sera appelée avec ./myGit.

Utilisation de nos commandes myGit :

./myGit init

➔ Initialise le répertoire de références. refs avec les fichiers HEAD et master ainsi qu'un fichier .current_branch dans le répertoire courant qui pointe vers master.

./myGit list-refs

➔ Affiche toutes les références existantes en parcourant les fichiers du répertoire .refs et les affiche une par une.

./myGit create-ref <name> <hash>

➔ Créer la référence qui pointe vers le commit correspondant en écrivant dans le fichier name le hash donné.

./myGit delete-ref <name>

➔ Supprime la référence name en vérifiant si elle existe.

./myGit add <elem> [<elem2>….]

➔ Ajoute un ou plusieurs fichiers/répertoire dans le fichier caché .add en créant un WorkTree qui lira le fichier et ajoute les nouveaux éléments dans ce WorkTree et enfin écrase le fichier .add avec le WorkTree.

./myGit clear-add

➔ Supprime le fichier .add.

./myGit list-add

➔ Affiche le contenu du fichier .add en utilisant un WorkTree.

./myGit commit <branch_name> [-m <message>]

➔ Créer un commit avec les noms de fichiers contenus dans le fichier .add sur une branche avec ou sans message en sauvegardant les fichiers et ajouter le « predecessor » avec son message si besoin.

./myGit get-current-branch

➔ Affiche le nom de la branche courante en lisant dans le fichier .current_branch.

./myGit branch <branch-name>

➔ Créer une branche <branch-name> en cas d'existence on affiche un message d'erreur.

./myGit branch-print <branch-name>

➔ Affiche le hash tous les commits de branch-name avec si besoin leur message en cas de branche inexistante on affiche un message d'erreur.

./myGit checkout-branch <branch-name>

➔ Se déplace sur branch-name, un message d'erreur s'affiche si elle n'existe pas.

./myGit checkout-commit <pattern>

➔ Se déplace sur le commit commençant par <pattern>, un message d'erreur s'affiche si le <pattern> correspond à rien.

./myGit merge <branch> <message>

➔ Fusionne la branch et la branche courante s'il y a des collisions alors on demande à l'utilisateur de garder le fichier à partir de quelle branche ou de choisir un par un.