HUANG Kevin …
SPORTES-MILOT Nino 21105127

Rapport Projet de Structure des Données

INTRODUCTION :

Le projet consiste à recréer l'environnement git qui permet la gestion des fichiers, en créant des points de sauvegarde et en permettant d'y revenir, d'avoir différentes versions du projet, de les fusionner et de traiter les conflits entre les deux versions. Notre projet a pour but de permettre toutes ces commandes, en omettant la partie qui permet de travailler en groupe. Le programme que l'on a créé ne permettra que de travailler en local sur son ordinateur.

Dans un premier temps, nous allons regarder les différentes étapes et avancés de notre travail, puis rentrer davantage dans les détails des difficultés que l'on a pu rencontrer et ainsi expliciter nos choix d'implémentation pour résoudre ces problèmes. Enfin, nous verrons quelques exemples illustrant l'usage de notre code.

L'évolution du projet :

Dans un premier temps, nous avons coder des fonctions permettant de résumer le contenu d'un fichier en un « hash », et d'être capable de mettre ce hash dans un fichier.

Ensuite, nous avons créé une structure de liste chainée, et des fonctions de manipulations ces listes. Notamment de convertir une liste en une chaine de caractère, et de la mettre dans un fichier, et inversement.

Puis quelques fonctions de manipulations des hash, des path, des modes d'accès à un fichier, et de copier un fichier. Une des fonctions notables est celle de blobFile, qui met le contenu d'un fichier dans un autre qui est à un emplacement particulier : on hash le contenu, on crée un dossier avec les deux premiers caractères et un fichier nommé par le reste du hash. La fonction retourne le hash du fichier.

On crée deux nouvelle structure -> les WorkFile, qui possède trois champs : le nom, le hash et le mode. Ils représentent des fichiers. Contre-intuitivement, les dossiers peuvent être des WorkFile, et leur hash serait alors le hash d'un fichier représentant son WorkTree qui est la deuxième structure. Un WorkTree est simplement une structure avec trois champs également : un tableau de WorkFile, le nombre d'éléments dans le tableau, ainsi que la taille maximale du tableau. Avec ces structures vient des fonctions de manipulations, similaires à celles de gestion des listes chainées. De même que blobFile, il existe une fonction blobWorkTree qui représente la représentation du WorkTree sous forme de chaines de caractères, la met dans un fichier, hash son contenu et le retourne.

Vient alors un des points culminant du projet, où l'aspect concret commence à apparaitre. Il s'agit des fonctions de saveWorkTree et restoreWorkTree. Le principe est simple, l'application moins. La fonction saveWorkTree crée une sauvegarde d'un WorkTree, en copiant tous les contenu des fichiers dans le WorkTree et des fichiers dans les dossiers, et des fichiers dans les dossier dans les dossier … En plus de cela, la fonction crée aussi des fichiers représentant chacun des WorkTree. Tous ces fichiers créés sont créés à un endroit à partir du hash des fichiers et des dossiers, ce qui va permettre de récupérer la sauvegarde avec restoreWorkTree. La fonction restoreWorkTree, récupère à partir du WorkTree, l'ensembles des fichiers et les restaure comme ils étaient lors de la sauvegarde du WorkTree.

On crée alors deux nouvelles structures, une qui associe juste une clé et une valeur, et une autre structure qui est un tableau de ces clés-valeurs, sa taille et son nombre d'éléments. Cette deuxième structure s'appelle Commit, elle permet dans git de faire une sauvegarde du programme dans une de ces versions. Tout comme les liste et les WorkFile & WorkTree, nous avons fait des fonctions de manipulations des commit.

La suite gère ce qu'on appelle les branches (une version de nos fichiers) :

- ➔ Tout d'abord, on apprend à gérer une branche. Une branche est un fichier stocké dans un dossier .refs qui contient toutes les branches. Une branche stockera le hash du dernier commit sur cette branche (comme les WorkTree, le hash d'un commit correspond au hash du fichier représentant ce commit). On intègre également deux fonctions, myGitAdd et myGitCommit qui simule les commandes add et commit de git.
- ➔ Ensuite on fait des fonctions pour gérer plusieurs branches, les créer, passer d'une à une autre en restaurant les fichiers en l'état de celle dans laquelle on arrive, récupérer la liste des commit de toutes les branches

Les dernières fonctionnalités consistent à restaurer le code à partir d'une branche ou bien d'un commit (en entrant les premiers caractères du commit que l'on souhaite restaurer), puis de merge deux branches ensembles.