
Pseudo-Git

LU2IN006

Contributions

Ramahatafandry Henintsoa (Maeva)

21104443

Table des matières

[Contributions](#)

[Table des matières](#)

[Concept](#)

[Fonctionnement](#)

[Structures manipulées](#)

[Listes:](#)

[WorkFile et WorkTree:](#)

[Commit:](#)

[Description du code](#)

[Fonctionnement de l'enregistrement d'instantanés](#)

[Fonctionnement de la réalisation d'un commit](#)

[Fonctionnement des branches](#)

[Fonctions principales](#)

[init:](#)

[list-refs:](#)

[create-ref <name> <hash>:](#)

[delete-ref <name>: Cette fonction supprime la référence <name>.](#)

[add <elem> \[<elem2> <elem3> ...\]:](#)

[list-add:](#)

[clear-add:](#)

[commit <branch name> -m message:](#)

[get-current-branch:](#)

[checkout-branch:](#)

[checkout-commit <pattern> :](#)

Concept

Git est un système de contrôle de version qui permet aux développeurs de travailler efficacement sur des projets d'équipes. Le but de ce projet est de développer une simulation du Git en utilisant le langage C et des structures principalement basées sur les listes chaînées et les tables de hachage. On a également utilisé les fonctions du bash ainsi que certaines fonctions issues de bibliothèques telles que stat ou dirent.

Le programme est composé de trois fonctionnalités principale:

- **l'enregistrement d'instantanés:**

On indique au programme les fichiers ou répertoire pour lesquels on souhaite créer un point de sauvegarde. Pour cela, on place les fichiers dans une zone appelée *zone de préparation*.

- **le versionnement des changements ("commits"):**

Pour créer le point de sauvegarde et pouvoir manipuler plusieurs versions, on organise les commits (qui sont des enregistrements instantanés associés à différentes étapes dans la chronologie d'un projet) de façon linéaire.

- **la séparation du travail avec plusieurs branches:**

Travailler avec plusieurs branches permet de garder plusieurs versions d'un même code. Cela permet de travailler simultanément sur différentes parties d'un projet sans affecter la branche principale.

Fonctionnement

Structures manipulées

Listes:

Une liste est conceptuellement une liste chaînée dont les éléments sont des chaînes de caractères.

Afin d'implémenter cette structure, nous avons un alias d'un pointeur sur une structure appelée Cell, qui est elle-même composée de deux champs:

- **data**, contenant une chaîne de caractères quelconque.
- **next**, contenant un pointeur vers une cellule, ce qui permet d'effectuer le chaînage.

WorkFile et WorkTree:

Un **WorkFile** est la représentation d'un fichier ou d'un répertoire. Cette structure nous permettra dans la suite de créer des instantanés d'un répertoire précis. Elle contient trois champs de données:

- **name**, correspondant au nom du fichier ou du répertoire.
- **hash**, correspondant au hash associé au contenu du fichier.
 - Cette donnée est initialisée à NULL.
- **mode**, constituant les autorisations associées au fichier.
 - Cette donnée est initialisée à 0.

Un WorkTree est un tableau dynamique WorkFiles, qui est composé de la taille du tableau maximale (**size**) et le nombre de cases qui ont déjà été initialisées (**n**).

Commit:

Un **Commit** est une table de hachage dont les clés et les valeurs sont des chaînes de caractères. Ces deux éléments forment ensemble les champs d'une structure appelée **key_value_pair**.

Les éléments de la table correspondent aux informations associées au point de sauvegarde.

Un Commit contiendra au moins une paire de la forme ("tree", hash) où hash est le fichier correspondant à l'instantané d'un WorkTree.

Afin de pouvoir insérer un élément dans cette table, nous avons utilisé la fonction **lose lose** issu du site: <http://www.cse.yorku.ca/~oz/hash.html>


Description du code

Fonctionnement de l'enregistrement d'instantanés

La zone de préparation est représentée par un fichier caché **.add**. Afin de créer un enregistrement instantané, on ajoute le nom de fichier ou du répertoire dans **./myGit add <file>**. Ce fichier va être ajouté à un Worktree qui va ensuite être importé vers **.add** avec les données suivantes: le nom du fichier, le hash du fichier et les informations sur les permissions

Fonctionnement de la réalisation d'un commit

On crée un répertoire **.refs** qui contiendra au moins deux éléments: le **master**, un fichier contenant le hash du commit le plus récent et le **head**, un fichier contenant le hash d'un commit quelconque. **Head** permet de simuler les déplacements vers la timeline, en utilisant la clé "predecessor" initialisée dans un commit.



Ensuite, en supposant qu'on a initialisé le `.refs` et qu'on a déjà effectué un `./myGit add`, on va charger le worktree représenté dans `.add` puis on va supprimer le fichier. Puis, on crée un enregistrement instantané de ce worktree, un Commit qui contiendra une paire ("`tree`",`hash`") et un *predecessor* s'il y'en a, ainsi qu'un message (optionnel).

Enfin, on crée un enregistrement instantané du Commit, qui sera affiché dans head et le fichier de la branche où l'on a réalisé le commit.

Fonctionnement des branches

pour naviguer entre les branches:

On utilise `./myGit checkout <branch>`, ce qui modifiera le fichier `.current_branch` pour contenir le nom de la branche donnée en paramètre, la référence HEAD pour contenir le hash du dernier commit de la branche. Ensuite, cela va restaurer le worktree correspondant au dernier commit de **branch**.

pour restaurer un Commit

On récupère la liste de tous les commits existants, on filtre pour ne garder que ceux qui nous intéressent puis on met à jour les fichiers dans **.refs**.

Fonctions principales

Pour exécuter les fonctionnalités suivantes du programme, il suffira de taper `./myGit` dans le terminal, accompagné du nom de la fonction.

init:

Cette fonction initialise le répertoire de références

list-refs:


Cette fonction affiche toutes les références existantes

create-ref <name> <hash>:

Cette fonction crée la référence `<name>` qui pointe vers le commit correspondant au hash donné

delete-ref <name>: Cette fonction supprime la référence `<name>`.

add <elem> [<elem2> <elem3> ...]:



Cette fonction ajoute un ou plusieurs fichiers ou répertoire à la zone de préparation (pour faire partie du prochain commit).

list-add:

Cette fonction affiche le contenu de la zone de préparation.

clear-add:

Cette fonction vide la zone de préparation.

commit <branch_name> -m message:

Cette fonction effectue un commit sur une branche <branch name> avec optionnellement un message <message>.

get-current-branch:

Cette fonction affiche la branche qui est actuellement en cours d'utilisation.

checkout-branch:

Cette fonction permet de naviguer entre les branches

checkout-commit <pattern> :

Cette fonction permet de restaurer les commits qui commencent par pattern.