Rapport de projet MyGit

SHEN DU
GROUPE 1
NO.28706448

Introduction

==========

Le projet MyGit consiste en une implémentation basique du système de gestion de version Git en langage C. Le but du projet est de permettre aux utilisateurs de suivre les modifications apportées à un ensemble de fichiers au fil du temps, et de faciliter la collaboration entre plusieurs contributeurs.

Dans ce rapport, nous allons décrire les structures de données, les fonctions principales et les choix d'implémentation de notre projet MyGit.

Structure de données

==========

Le projet MyGit utilise plusieurs structures de données pour stocker les informations nécessaires pour la gestion de version. Les principales structures de données sont les suivantes :

1. **ListC**: Liste chaînée sous forme:

```
typedef struct cell{
    char *data;
    struct cell *next;
}Cell;

typedef Cell* List;
```

2. **WorkFile et WorkTree** pour stocker les fichiers et les répertoires du répertoire de travail.

```
typedef struct {
    char * name ;
    char * hash ;
    int mode ;
} WorkFile ;

typedef struct {
    WorkFile * tab ;
    int size ;
    int n ;
} WorkTree ;
```

3. **Commit**: Hashmap pour stocker les key-value pairs des commits.

```
typedef struct key_value_pair {
    char* key;
    char* value;
} kvp;

typedef struct hash_table {
    kvp** T;
    int n;
    int size;
} HashTable;

typedef HashTable Commit;
```

4. **Branch**: Une série de commits gestionnés dans les fichiers (hash de latest commits)
   message : commit message
   tree : 9023f0fdb3dba53e570ba7395eae4138decd22369d9767b3679e55cdc6456bfd
   predecessor :
26993e6809bdd7758559b78d881f7f4ba6f99ed541b7b98fcc6e498c18df55be
Utiliser predecesseur pour parcourir les commits

Fonctions principales

==========

---

1. mygitAdd

La fonction myGitAdd permet d'ajouter un fichier ou un dossier spécifié au système de versionnage Git. Elle commence par vérifier si le fichier d'indexation (.add) existe, et le crée s'il n'existe pas encore. Ensuite, elle initialise un objet WorkTree et le remplit avec les informations actuelles de l'arbre de travail. Si le fichier ou le dossier spécifié existe, elle l'ajoute à l'objet WorkTree et écrit les informations mises à jour dans le fichier d'indexation. Si le fichier ou le dossier spécifié n'existe pas, elle renvoie un message d'erreur.

---

2. mygitCommit

Cette fonction permet de créer un commit pour une branche spécifiée avec un message de commit donné. Il vérifie que le projet a été initialisé, que la branche existe et que HEAD pointe sur le dernier commit de la branche. Ensuite, il crée un nouveau commit avec les informations fournies et met à jour les références de la branche et de HEAD. Enfin, il supprime le fichier d'index .add.

---

3. myGitCheckoutBranch

La fonction myGitCheckoutBranch permet de changer de branche. Elle modifie le fichier .currentbranch pour indiquer la branche courante, met à jour la référence HEAD et restaure l'arborescence de travail correspondante au dernier commit de la branche donnée en argument.

---

4. myGitCheckoutCommit

La fonction myGitCheckoutCommit prend en entrée un motif pattern et permet de changer de commit. Tout d'abord, la fonction récupère une liste de tous les commits grâce à la fonction getAllCommits(). Ensuite, elle filtre la liste pour ne garder que les commits qui correspondent au motif pattern. Si un seul commit correspond à ce motif, la fonction change la référence HEAD et restaure le worktree correspondant à ce commit. Si aucun commit ne correspond à pattern, la fonction affiche un message d'erreur. Si plusieurs commits correspondent à pattern, la fonction affiche la liste de ces commits et laisse l'utilisateur choisir lequel utiliser pour changer de commit.

---

5. merge

La fonction "merge" prend en entrée le nom de la branche distante et un message facultatif. Elle récupère ensuite les commits correspondant à la branche courante et à la branche distante, puis fusionne les arbres de travail correspondants et retourne la liste des conflits éventuels. Si cette liste est vide, elle crée un nouveau commit de fusion contenant les modifications fusionnées et met à jour les références de la branche courante et de la HEAD, supprimant ainsi la référence à la branche distante. Si elle n'est pas vide, elle retourne la liste des conflits pour permettre une résolution manuelle.

---

6. createMergeCommit

La fonction "createDeletionCommit" crée un commit de suppression pour une branche donnée, en prenant en entrée le nom de la branche, la liste des conflits et un message facultatif. Si la liste des conflits est vide, cela signifie qu'il n'y a pas de conflits à résoudre, ce qui est une erreur, donc la fonction quitte. Si la branche n'existe pas, elle affiche également une erreur et quitte. Elle récupère ensuite l'arbre de travail correspondant au commit de la branche donnée, ajoute les fichiers qui ne sont pas en conflit et qui ont été modifiés, crée un nouveau commit de suppression, le référence à la branche donnée et met à jour la HEAD pour pointer sur ce commit.

Manuel d'utilisation

==========

- **init** : Initialise le répertoire de références .refs et crée la première branche.
- **refs-list** : Liste toutes les références (branches, tags, etc.) enregistrées dans le répertoire .refs.

- **create-ref** : Crée ou met à jour une référence donnée en entrée.
- **delete-ref** : Supprime une référence donnée en entrée.
- **add** : Ajoute un ou plusieurs fichiers à la zone de préparation.
- **clear-add** : Efface tous les fichiers dans la zone de préparation.
- **add-list** : Liste les fichiers présents dans la zone de préparation.
- **commit** : Enregistre les changements de la zone de préparation dans un nouveau commit.
- **get-current-branch** : Récupère le nom de la branche courante.
- **branch** : Crée une nouvelle branche avec un nom donné.
- **branch-print** : Affiche les informations de la branche correspondant au nom donné.
- **checkout-branch** : Change de branche pour la branche correspondant au nom donné.
- **checkout-commit** : Change l'état des fichiers pour celui d'un commit spécifique.
- **merge** : Fusionne la branche courante avec la branche donnée en entrée. Si des conflits sont détectés, le programme propose des options pour les résoudre manuellement.

Les options sont passées en tant qu'arguments de ligne de commande. Par exemple, pour ajouter un fichier nommé file.txt à la zone de préparation, on pourrait utiliser la commande ./myGit add file.txt.