Projet: Réalisation d'un logiciel de gestion de versions

Nous considérons dans ce projet la réalisation d'un outil de suivi et de versionnage de code (type git). Ce projet est découpée en plusieurs parties, qui seront ajoutées progressivement au sujet pendant le semestre. Il faudra donc télécharger le sujet du projet à chaque début de séance de TME (pour avoir accès à la suite du projet). Chaque partie est divisée en exercices, qui vont vous permettre de concevoir progressivement le programme final. Il est impératif de travailler régulièrement pendant le semestre, afin de ne pas prendre de retard et de pouvoir profiter des séances de TME associées à chacune des parties du projet.

Cadre du projet

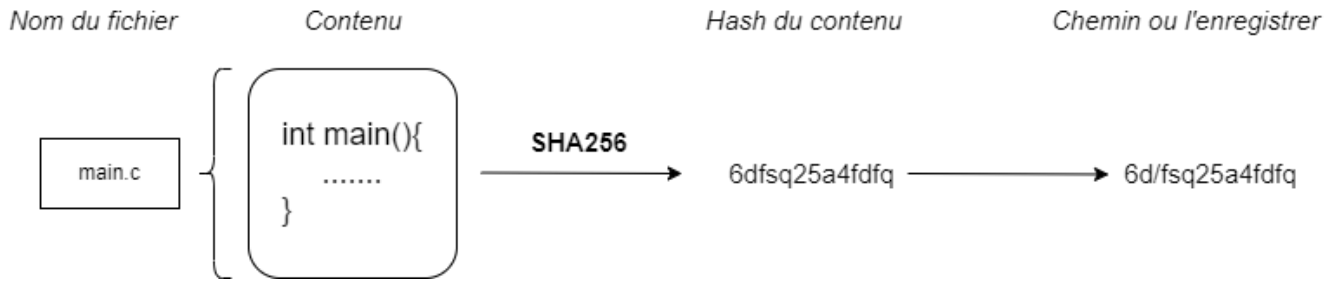
Un logiciel de gestion de versions est un outil permettant le stockage, le suivi et la gestion de plusieurs versions d'un projet (ou d'un ensemble de fichiers). En particulier, ces outils offrent un accès aisé à l'historique de toutes les modifications effectuées sur les fichiers, permettant notamment de récupérer une version antérieure en cas de problème. Par ailleurs, ces outils sont très utiles dans le cadre de travail collabaratif, permettant de fusionner de manière intelligente différentes versions d'un même projet. Par exemple, ces outils sont couramment utilisés en développement logiciel, pour faciliter le travail en équipe et conserver le code source relatifs à différentes versions d'un même logiciel.

L'objectif de ce projet est d'étudier le fonctionnement d'un logiciel de gestion de versions, en détaillant différentes structures de données impliquées dans sa mise en oeuvre. En particulier, nous allons nous intéresser aux fonctionnalités suivantes :

- Comment permettre à un utilisateur de créer des enregistrements instantanés de son projet ?
- Comment lui permettre de naviguer librement à travers les différents instantanés ?
- Comment construire et maintenir une arborescence des différentes versions de son projet ?
- Comment vérifier l'identité des utilisateurs ?
- Comment sauvegarder des changements qui ne sont pas dans un instantané ?

Vers la création d'enregistrements instantanés

Sous git, tout les objets, qu'ils soient relatifs aux fichiers versionnés ou à leurs méta-données, sont enregistrés sous forme de fichiers. Ces fichiers ont pour particularité de pouvoir dériver le chemin où ils sont stockés à partir de leur contenu, par exemple comme décrit dans la figure suivante :



Ici la fonction de hachage SHA256 est appliquée sur le contenu du fichier, puis le chemin où doit être stocké le fichier est obtenu en insérant un "/" entre le deuxième et le troisième caractères du hash. Faire dépendre le chemin du contenu permet notamment de sauvegarder toutes les différentes versions du fichier. En effet, modifier le contenu du fichier va modifier le chemin vers lequel le sauvegarder et ainsi créer différentes sauvegardes correspondant à différents états du fichier. Quand on réalise une telle sauvegarde, on dit communément qu'on "enregistre un instantané" ou encore qu'on crée un "enregistrement instantané". L'objectif de cette première partie du projet est d'écrire un programme permettant d'enregistrer un instantané, comme décrit dans l'exemple.

Exercice 1 – Prise en main du langage Bash

Les données sur le disque sont stockées dans des fichiers (file en anglais), qui sont des structures de données qui apparaissent aux programmes comme des suites finies d'octets. La structure de données qui organise les fichiers sur le disque s'appelle le système de fichiers (file system). Le système de fichiers est l'une des fonctionnalités principales d'un système d'exploitation. Si généralement on y a recours en utilisant une interface graphique, nous apprendrons ici à le contrôler par du code. Pour cela, nous allons commencer par quelques exercices de prise en main du langage Bash permettant d'utiliser une interface en ligne de commande.

Q 1.1 Pour vous familiariser avec les commandes usuelles, commencez par suivre la séquence de d'instructions suivantes, et utiliser à chaque fois la commande `man` pour afficher la documentation de la commande concernée :

- Ouvrez un terminal (sous linux, cela positionne par défaut sur le chemin principal "∼/").
- Utilisez la commande `pwd` pour afficher le répertoire courant.
- Utilisez la commande `cd` pour vous positionner sur votre répertoire de travail pour cette UE.
- Utilisez la commande `mkdir` pour créer un répertoire nommé "projet_scv".
- Positionnez vous dans le répertoire nouvellement créé, puis utilisez la commande `touch` pour créer les fichiers "main.c", "Makefile" et "test.txt".

— Utilisez `ls` pour lister les fichiers de votre répertoire.
— Après avoir modifié le fichier "main.c" via votre éditeur de texte préféré, utilisez la commande `cat` pour en afficher le contenu.
— Utilisez la commande `rm` pour supprimer le fichier "test.txt".

Linux met en place des flux globaux permettant à des commandes appelées, de dialoguer avec le programme appelant, en écrivant ou en lisant depuis ces flux. Ces flux globaux sont :
— `stdin` (entrée standard) : flux d'entrée du programme. Par défaut, il s'agit des données saisies au clavier. Ce flux permet notamment de définir des programmes interactifs, récupérant des données saisies sur le terminal par l'utilisateur, par le biais de la fonction `scanf` (par exemple).
— `stdout` (sortie standard) : ce flux correspond à la sortie du programme. Par défaut, il s'agit du terminal ayant lancé le programme. Il permet notamment d'afficher des données sur le terminal.
— `stderr` (sortie standard d'erreur) : ce flux sert à récupérer les messages d'erreurs, qui par défaut sont affichés sur le terminal qui a lancé le programme.

Sous linux, il est possible, après avoir appelée une commande, de rediriger une des sorties vers un fichier. Par exemple, avec la commande `ls > list.txt`, vous obtenez un fichier "list.txt" contenant la liste des fichiers et répertoires présents dans le répertoire courant. Il est également possible de rediriger la sortie d'une commande vers une autre, en utilisant ce que l'on appelle une "pipeline". Par exemple, la commande `cat noms.txt | sort` permet de lire la liste de noms présente dans le fichier noms.txt, et de la transmettre à la fonction sort qui va la trier (par ordre alphabétique).

Q 1.2 Sous linux, la commande `sha256sum` permet de hacher le contenu d'un fichier en utilisant la fonction de hachage SHA256. En utilisant une redirection et une pipeline, écrivez une commande qui transmets le contenu du fichier "main.c" à la commande `sha256sum` puis écrit le hash correspondant dans un fichier (temporaire) appelé "file.tmp".
Ces commandes peuvent aussi être utilisées à travers un code C, par le biais de la fonction `system` qui, comme son nom l'indique, permet de faire des appels système. Pour bien comprendre le fonctionnement, commencez par exécuter le programme C suivant :

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    system("ls");
}
```

Q 1.3 Écrivez une fonction `int hashFile(char* source, char* dest)` qui, étant donné le chemin de deux fichiers, calcule le hash du contenu du premier fichier et l'écrit dans le deuxième fichier.
Bien que la fonction `system` permette d'exécuter des commande Bash, ce n'est pas le seul moyen de manipuler le système de fichiers. Un autre moyen, qui est préférable quand il est disponible, est d'utiliser des librairies C prévues à cet effet. Par exemple, on préférera ce code :

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    FILE* f = fopen("test.txt", "w");
    fprintf(f, "Test");
}
```

à son équivalent suivant :

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    system("echo Test $>$ test.txt");
}
```

En particulier, la librairie `<unistd.h>` permet une gestion efficace des fichiers temporaires, ce qui nous sera très utile pour stocker le hash d'un fichier en attente de lecture par notre programme. Voici un exemple de code illustrant comment créer un fichier temporaire avec la fonction `mkstemp` de cette librairie :

```c
static char template[] = "/tmp/myfileXXXXXX";
char fname[1000];
strcpy(fname, template);
int fd = mkstemp(fname);
```

Dans cet exemple, l'appel à `mkstemp(fname)` crée un fichier temporaire dont le nom sera stocké dans `fname`. Ce nom sera généré de manière unique à partir du motif `template`. Plus précisément, il faut que les six derniers caractères de `template` soient "XXXXXX" pour que la fonction `mkstemp` les remplacent de sorte à créer un nom de fichier unique. La fonction `mkstemp` ouvre ensuite le fichier temporaire nouvellement créé, et renvoie un descripteur de fichier ouvert (en lecture et écriture).

Q 1.4 En utilisant la commande `sha256sum`, un pipe, une redirection, et un fichier temporaire (qu'il faudra supprimé après usage), écrivez une fonction `char* sha256file(char* file)` qui renvoie une chaîne de caractères contenant le hash du fichier donné en paramètre.

Exercice 2 – Implémentation d'une liste de chaînes de caractères

Dans cet exercice, il s'agira d'implémenter les fonctions permettant de gérer une structure de données de type liste chaînée dont la définition est la suivante :

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

typedef Cell* List;
```

Q 2.1 Écrivez une fonction `List* initList()` qui initialise une liste vide. Veillez par la suite à ne plus initialiser de liste autrement que par cette fonction.

Q 2.2 Écrivez une fonction `Cell* buildCell(char* ch)` permettant d'allouer et de retourner une cellule de la liste.

Q 2.3 Écrivez une fonction `void insertFirst(List *L, Cell* C)` permettant d'ajouter un élément en tête d'une liste.

Q 2.4 Écrivez une fonction `char* ctos(Cell* c)` qui retourne la chaîne de caractères qu'elle représente, puis utilisez cette fonction pour écrire la fonction `char* ltos(List* L)` qui transforme une liste en une chaîne de caractères avec le format suivant : chaîne1|chaîne2|chaîne3|...

Q 2.5 Écrivez une fonction `Cell* listGet(List* L, int i)` qui renvoie le $i^{\text{ème}}$ élément d'une liste.

Q 2.6 Écrivez une fonction `Cell* searchList(List* L, char* str)` qui recherche un élément dans une liste à partir de son contenu et renvoie une référence vers lui ou NULL s'il n'est pas dans la liste.

Q 2.7 Écrivez une fonction `List* stol(char* s)` qui permet de transformer une chaîne de caractères représentant une liste en une liste chaînée.

Q 2.8 Écrivez la fonction `void ltof(List* L, char* path)` permettant d'écrire une liste dans un fichier, et la fonction `List* ftol(char* path)` permettant de lire une liste enregistrée dans un fichier.

Exercice 3 – Gestion de fichiers sous git

L'objectif final de cet exercice est de produire une fonction qui enregistre un instantané d'un fichier dont le nom est donné en paramètre.

Q 3.1 Les fonctions `opendir` et `readdir` de la librairie `<dirent.h>` permettent d'explorer un répertoire. Par exemple, on peut afficher le noms des fichiers et des répertoires qui le composent en procédant de la manière suivante :

```
1 DIR *  dp = opendir (root_dir);
2 struct dirent *ep;
3 if (dp != NULL)
4 {
5     while ((ep = readdir (dp)) != NULL)
6     {
7         printf("\%s␣\n", ep->d_name);
8     }
9 }
```

Écrivez une fonction `List* listdir(char* root_dir)` qui prend en paramètre une adresse et renvoie une liste contenant le noms des fichiers et répertoires qui s'y trouvent.

Q 3.2 Utilisez la question précédente pour écrire une fonction `int file_exists(char *file)` qui retourne 1 si le fichier existe dans le répertoire courant et 0 sinon.

Q 3.3 Écrivez une fonction `void cp(char *to, char *from)` qui copie le contenu d'un fichier vers un autre, en faisant une lecture ligne par ligne du fichier source.
Indication : pensez à vérifier que le fichier source existe avant.

Q 3.4 Écrivez une fonction `char* hashToPath(char* hash)` qui retourne le chemin d'un fichier à partir de son hash (on rappelle que le chemin s'obtient en insérant un "/" entre le deuxième et le troisième caractères du hash).

Q 3.5 En utilisant la commande Bash `mkdir` (qui permet de créer un répertoire), écrivez une fonction `void blobFile(char* file)` qui enregistre un instantané du fichier donné en entrée.