LU2IN006 Rapport du projet

Oleksandr Hoviadin, Tianxiang Xia

April 16, 2023

Sujet synthétique

On crée un outil de gestion des versions des fichiers qui possède les fonctionnalités principales de git : sauvegarde des instantanés des fichiers, navigation entre différentes versions, création et gestion des branches afin de travailler simultanément sur plusieures versions des projets et avoir la possibilité de réunir les branches en une version finale.

Structure de développement

```
bin/ # fichiers compilés/exécutables
src/ # fichiers source
tests/ # jeu de tests
tests/freespace/ # lieu de tests
Makefile
```

Structure de code

- **hashFunc.c** : contient les fonctions pour créer et manipuler les hashes.
- **cellList.c** : contient les fonctions pour manipuler des listes des chaînes des caractères : `Cell`, `List` (liste simplement chaînée), etc.
- **fsop.c** : contient les fonctions pour manipuler des fichiers et dossiers : `listdir`, `blobFile`, etc.
- **workTree.c** : contient les fonctions pour manipuler ensembles de fichiers : `WorkFile`, `WorkTree` (arbre des dossiers et des fichiers comme feuilles), `saveWorkTree`, `restoreWorkTree`, etc.
- **gestionCommits.c** : contient les fonctions permettant naviguer entre instantanés, construire et maintenir arborescence des différentes versions de projet : `Commit` (table de hachage), etc.
- **myGit.c** : les codes sur l'interface utilisateur et les interactions : `main`, etc.
- **misc.c** : les fonctions/macros supplémentaires qui sont pourtant très utiles.

Sur l'implémentation

- Selon l'énoncé, après avoir ajouté les fichiers dans la zone de préparation on peut encore les modifier et quand on fait commit c'est la version la plus récente qui sera soumise. Pour cette raison, on a défini les fonctions `wfts_upd` et `wtts_upd` qui permettent de retouren l'état actuel des fichiers en chaîne de caractères.

- On stocke tous les nombres concernants mode en octal, e.g. `"%o"`.
- On stocke tous les fichiers auxilliaires du programme dans le dossier .mygit du répertoire utilisateur. Une fois pour tout, on l'ignore dans `listdir`.
- L'utilisation des macros dans `misc.h` rend le programme très configurable.
- On fait un maximum d'abstraction que possible: on réutilise des fonctions existantes et on fait des petits outils en dehors de l'énoncé qui peuvent beaucoup aider e.g. `s2f`, `f2s`, `blobStringExt` (consulter les commentaires pour savoir l'usage des fonctions).
- On écrit des commentaires brèves pour les fonctions.
- On fait attention aux ownerships de la mémoire: chaque fonction, si elle est donnée de la mémoire pointée par un pointeur, cette mémoire doit être librérée ou donnée à une autre fonction avant qu'elle termine de sorte que quand le programme termine, on puisse garantir que toute la mémoire allouée est libérée. En pratique, on utilise `const` pour indiquer l'information sur ownership, dans des cas spécifiques, on utilise des commentaires ou il est clair par l'usage de la fonction.
- On a ajouté command `myGit log` pour pouvoir consulter l'information du dernier commit (ainsi on peut utiliser l'entrée `second_predecessor`, qui est demandé d'être stockée par l'énoncé mais n'est pas utilisé).
- Dans cette application, l'éfficacité n'est pas la plus importante, on fait quand même beaucoup d'optimisations sans changer que l'application soit robust : on utilise `buf[MAXL]` où la taille de mémoire est moins importante (la pile est plus rapide que le tas) vice versa; la complexité temporelle est toujours optimale sous les structures données, etc.

Mode de test et démonstration d'utilisation

Test des unités `tests/tests.c`

Test rassemblé/démonstration d'utilisation `tests/assembled_tests*.sh`, on lance les scripts dans `tests/freespace`

Pour consulter toutes les commandes possibles, veuillez lancer `myGit` sans paramètre.