Design for Gitlet

Dennis Zhong, Haodong Li

July 13, 2020

0 Overview

0.1 List of commands

- ☑ **init**
- ☐ **add**
- ☐ **commit**
- ☐ **rm**
- ☐ **log**
- ☐ **global-log**
- ☐ **find**
- ☐ **status**
- ☐ **checkout**
- ☐ **branch**
- ☐ **rm-branch**
- ☐ **reset**
- ☐ **merge**
- ☐ **rebase**

0.2 Attention

- Always exit with **exit code 0**
- **.gitlet** contains all old files and meta files
- General **failure cases**:
  doesn't input any arguments------Please enter a command.
  inputs a command that doesn't exist------No command with that name exists.
  inputs a command with the wrong number or format of operands------Incorrect operands.
  inputs a command not in such a directory------Not in an initialized Gitlet directory.

1 Classes and Data Structures

1.1 Commit

- String timestamp
- String message
- Blob[ ] blobs
- Commit pa1
- Commit pa2

1.2 Blob

- String name
- String contents

1.3 Branch

2 Algorithms

2.1 TBD

3 Persistence

We will be saving a representation of the object created to file. Thus, the settings provided in the first run should persist across multiple executions of the program, as we will be able to reload the object.

In order to persist the state of files, we must save the necessary data elements. That is: write the Commit and Blob to disk. We can serialize them into bytes that we can eventually write to a file on disk, named based on a set naming convention. This can be done with the writeObject() method from the Utils class.

In order to retrieve our state, before executing any code, we need to search for the saved files in the working directory (folder in which our program exists) and load the objects that we saved in them. Since we set on a file naming convention (TBD) our program always knows which files it should look for. We can use the readObject() method from the Utils class to read the data of files and deserialize the associated objects we previously wrote to these files