Project 2: Gitlet Design Document

Group members
- Jie Zhang (s270)
- Xu Ke (s271)

Data Structures and Functions

Classes

Repository

Repository stores all the data, which includes blobs, tree objects, commit objects, tag objects, references.

GitletObject

```
public abstract class GitletObject implements Serializable {
    /*This is a abstract class for other gitlet objects, including blobs and commits. */
    public String sha1() { }            //Generate the SHA-1 for the gitlet object.
    @Override
    public int hashCode() { }           //return the hash code from SHA-1.
    @Override
    public boolean equals(Object o) { } //compare two gitlet object.
    ...
}
```

Blobs

```
public class Blob extends GitletObject {
    /* a binary representation of a file.*/
```

```
    //Constructor
    public Blob(byte[] content) {}
    //Fields
    private byte[] content;
    private String name;
    //method
    public byte[] getContent() {} //return the blob's binary con
tent
    public int hashCode() {}
}
```

Tree Objects

```
public class Tree extends GitletObject{
    /* Tree object is a directory, which contains pointers to bl
obs, and other tree objects. */
    //Constructor
    public Tree(){}
    //Fields
    private String[] hashPointers;
    private String _name;
    private String _type = "tree";
    //Method
    public String get_type() {}
    public String get_name() {}
    public String[] getHashPointers() {}
    public void add_HashPointer() {} //to add tree or blob to th
e list of hash pointers

}
```

Commit

```java
public class Commit extends GitletObject{
    /*a pointer that contains some metadata. It has a hash that built from all the metadata it contains. */
    //Fields
    private String id;
    private Commit parent;
    private Long timeStamp;
    private String message;
    private HashMap<String, String> blobs;

    //Constructor: initialize a commit with message, tme stamp, its parent, and files.
    public Commit(String message, Long timeStamp, Commit parent, HashMap<String, String> blobs) { }

    //Getter:
    public String getId() { return id; }
    public Commit getParent() { return parent; }
    public Long getTimeStamp() { return timeStamp; }
    public String getMessage() { return message; }
    public HashMap<String, String> getblobs() { return blobs; }

    //Methods:
    @Override
    public int hashCode() {
        return Objects.hash(id, parent, timeStamp, message, blobs);
    } //return the hash of the commit.
}
```

References

```
public class Reference extends GitletObject {
    /*
    Since it is difficult for user to memorize the hash for each object,
    gitlet has a file stored in .gitlet/refs , containing the hash of a commit object.
    */
    //Constructor
    public Reference(String date, String message, String tag, String pointer) {}

    //Fields
    private String date;
    private String message;
    private String tag;
    private String pointer;

    //Methods
    public String[] get_info() {} //This returns date, message and tag
    public String get_pointer() {} //This returns the pointer because it might be used more often
}
```

GitletCommand

```
public abstract class GitletCommand {
    /*
    An abstract class that should be extended by all commands
     */
```

```
5
6       // Constructor that takes in arbitrary number of argument
7       public GitletCommand(String... args ) {}
8
9       private String settings;
10
11      public boolean isDangerous() { return false; }
12      public void execute() {}
13  }
```

Init

```
1  //initialize gitlet, creating repository, make a staging area an
   d a working directory
2  public void init(Path file_path) {}  //When calling, provide the
   file path.
3  public void generate_repo() {}
4  public void generate_stage() {}
5  public void genetate_working_dir() {}
```

Add

```
1  //put the binary file into the staging area
2  public void add(String... args) {}
```

Commit Command

```
1  public class CommitCommand extends GitletCommand {
2      /* commit a file. */
3      private String message;
4
5      public CommitCommand (String message) { this.message = messa
   ge; }
```

```
    @Override
    public void execute() {}
}
```

Log Command

```
/* recursively calling the ancestor of current commit from the HEAD reference to the
very first commit
 */
public void log() {}
```

Branch Command

```
//create a new reference object using the name in args and the current commit
public void branch(String[] args) {}
```

CheckoutCommand

```
public class CheckoutFileCommand extends GitletCommand{
    /* head for a file point to the commitId */
    private String commitId;
    private String fileName;
    public CheckoutFileCommand (String fileName) {}

    @Override
    public void execute() {}
}
```

Status Command

```
public class StatusCommand extends GitletCommand{
```

```
    private Repo repository;
    //Show the current branch, all branches and the Staging area
.
    @Override
    public void execute() { }
}
```

Checkout Branch Command

```
//input a branch name, then move HEAD to that branch
public void checkout_branch(String args) {}
```

Algorithms

Blob

- `getContent()`: get the content of the bolb.
- `hashCode()`: using Objects.hash(name, content) to generate a hash.

Tree

- String[] hashPointers: record all the hash codes of what it contains.

Example:
We had a simple repository, with a `README` file and a `dir1/` directory containing a HelloWorld.java file.

> *README*
> *dir1/*
> *HelloWorld.java*

This is the representation of two tree objects: one for the root directory, and another for the `dir1/` directory.

tree 79054025...

Type	Hash	Name

blob	255fb1a2...	README
tree	624bc422...	dir1

tree 624bc422...

Type	Hash	Name
blob	aef54eba	HelloWorld.java

Since the root tree node contains the children tree nodes, gitlet can use recurse through every tree object to get the entire working directory.

Commit

- `hashCode()`: using Objects.hash(id, parent, timeStamp, message, blobs) to generate a hash.

Reference

- Store date, message, tag related to a commit, which is the variable “pointer”.

GitletObject

- `hashCode()`: not implement.
- `equals(Object obj)`: comparing this and obj with operator `==`, if equal return true; Comparing `this.hashCode()` and `obj.hashCode()`, if equal return true; otherwise, return false.

Init

- initialize everything, take in a file path for creating a repository and staging area in it.

Add

- Using `hashCode()` in Blob to transform a file into binary form, then put it in the staging area.

CommitCommand

- `execute()`: if there is no message, ask user to enter the commit message; create a new commit object with id, most recent commit, time stamp, commit message and blobs.

Log

- print the current commit(including the date, message, and SHA1) under HEAD, then call it's parent and recursively run the printing until hit the first one created at initialization.

Branch

- Create a reference object to a commit(current commit by default), let HEAD point on the newly created branch(by default).

Checkout Branch

- search for the input branch in all branches, and checkout that branch.

CheckoutFile

- `execute()`: if the commit id is null, checkout the file to current head pointer; if the commit id does not exist, throw error; otherwise, checkout file to the commit id.

StatusCommand

- `execute()`: using the repository to get the current branch and print; get all branches and print; get all files in staging area and print;

Persistence

Working Directory

Since every commit has a parent, Gitlet can use a single commit to build up the entire working directory. It only needs to store one commit for a branch, when users open the Gitlet, the head will point to that commit id.

Staging Area

Since every commit has a unique hash code, Gitlet can store all the hash in .gitlet/index. The **index** is a single, large, binary file, which lists all files in the current branch with their hash, time stamps and the file names.

Local Repository

When a commit is made, Gitlet takes a snapshot of the current branch. If only small changes have been made, other files' hash will not change, and Gitlet will only keep one copy of files with same content and let new commit id point to the pervious one.

Rationale

Log and branching

We are using a tree as our data structure, but when implementing the log command, it takes Θ(N) if we consider N as the length of the commits from current to the initial one. Checkout branch also takes Θ(N) for N is the number of total branches while branching only takes constant time.

Commit

When user create a commit, gitlet will provide the hash of that commit. Gitlet can use the partial hash (first few characters) to represent the entire hash. If with the given partial hash, gitlet cannot identified which hash it represent, it will show all commits that start with the partial hash, or simply path not in the working tree.