Project 2: Gitlet

Data Structures and Functions

Classes

Commit *implements* ***Serialiazable***{

//Fields

private final String parent // The SHA-1 identifier of my parent, or null if I am the initial commit
Private final String message // Log message
Private Date commitDate // My timeStamp
Private HashMap<String, String> contents = new HashMap<>(); //A mapping uniqueCommitMessage and message
Private HashMap<String, String> commits = new HashMap<>();/ acts like the root of the commit
TreeNode init;
Private Branches[] branches; // An array of all branches

//Methods

Public String toString()
/**Get SHA-1 identifier of my parent, or null if I am the initial commit. */
String getParent()
String getMessage()
Date getCommitDate()
addCommit(String msg) // add a commit
log() // displays the commits in the current branch
globalLog() // display all the commits from the init
rm(filename) // Untrack a file
find(String commitMsg)
checkout(filename) // find the file in commit where the HEAD pointer points to and then overwrite the file in working directory
checkout(commit_id,filename)
reset() // Removes tracked files that are not being tracked in the given commit, and set the HEAD to the current commit. Staging area is cleared.
status() // gets an overview
}

Branch implements Serializable{

//Constructor

public Branch (String branchname, Head headpointer)

//Fields

private final String branchname // The name of a branch
private final Head headbranch //The head pointer of the branch

//Methods
public checkout() ///Set the HEAD pointer to the current branch's head pointer
public rm-branch() //Set the instance variable headbranch as null(the branch will lose its track)
public checkout() //Takes all the files in the commit where current branch head points to, then overwrite the files in the working directory. Set the current branch head as HEAD.
Public getBranch() // returns the name of the branch
}
Blobs implements **Serializable**{
//Fields
/** The SHA-1 identifier of the content file I refer to - same as content name*/
private final String blodId;
/** blob current version */
Private int version;
//Methods
public String getBlodId()
public getCurrentVersion()
}

Head implements Serializable {
// Constructor
Public Branch getbranchedPointed() //get the current branch that head is being pointed to
//
//Methods
/** set the head to point to this branch */
Public set branchedPointed(Branch branch)
}

StagingArea implements Serializable {
//Fields
Static StringList staged
//Method
add(String fileID)   //(1.Find the file in the working directory by its file name and get the ID of that file, then put the fileID into staged variable in Staging Area Class)
clear() //Set the StringList staged as null
}

UnstageArea implements Serializable {
//Fields
Static StringList unStaged;
//Methods
add(String fileID)
}

Gitletcommand implements Serializable {
}

RemoveFiles implements Serializable {
}

Algorithms

Log

- **log():** prints the reverse of the tree of commits of the branch where the head points to
    1. Get the current branch
    2. Prints commit's metadata using a recursive reverse

Global Log

- **globalLog():** prints all the existing commit made to the repository
    1. Have an array variable that stores every commit regardless of where the branch or the head points to
    2. Print all the elements in that array

Add

Add (String filename):

1. Check if the file exists
2. Make a copy of the file and save it with a unique ID with a SHA-1 hashing
3. Update the staging area to reflect the newly added file
4. Write file to the staging area to keep track of our state.
5. If there are more files no added, update the unstageArea to reflect for these files

commitAdd(String commitMsg)

1. Check if there is something in the staging area. If there is:
    a. Make a copy of the committed message
    b. Check where the head
    c. Check if tree is null,
        i. If null, add the default init
        ii. If not null, go to the end of the list and add a new node at the end of the tree
2. If there's nothing in the staging area, print error msg

Find

- find(String commitMsg):
    1. Convert commitMsg to a HashValue using the SHA-1
    2. Iterate through the commit hashMap and print the values of the commitMsg (keys = uniqueHashing, values = messages themselves)
    3. If value does not exist print the error message

Status

1. For each branch in the Branch array
    a. If the head points to that branch print a * before the branch
    b. Otherwise print in a new line

2. For each item in the stringList Staged print item
3. For each item in the stringList Unstage
    a. If item has been removed, print item followed by the (delete) string
    b. If item has been modified, print item followed by the (modified) string

Checkout

checkout(branchname)

1. Knowing the header of the branch, take out all the files and compare them with the files in the working directory by name. Overwrite the working directory files.
2. Set the HEAD static variable branchPointed as the current branch instance variable headbranch.

checkout(commit_id,filename)

checkout(filename)

Merge

1. Compare the serializable objects, if they are the same commit. Otherwise, display contents and save the local changes

Rm

rm(filename)

1. In the current commit object, check whether the file is tracked by its ID.
2. If it is, delete the file from the working directory. Then, check whether the ID is in the current staged area to see whether it has been staged or not. If it is, remove the ID from the class variable string list.
3. If it is not, check whether the ID is in the current staged area. If it is, remove the ID from the class variable string list.

rm-branch(branch name)

1. Set the instance variable headbranch as null.

Reset

reset()

1. Find the commit according to the input parameter commit id. Removes tracked files that are not being tracked in the given commit. Set the HEAD pointer to the current commit object. Call clear() method in the StagingArea class.

Persistence

In order to persist the status of the gitlet, we will need to save the state of the commit trees, staging area and HEAD pointer. To do this,

1. Write the commit trees to disk. Write StagingArea instance to disk.Write the Head instance to disk.

In order to retrieve our state, we need to search for the saved files.
To do this,

1. Look for the specially named files, such as “StagingArea”, “CommitTree”.
2. Use the Files.readAllBytes to read the data of files or deserialize to convert files into corresponding Java objects.