CS61bl, section 104

7/17/2015

Gitlet Checkpoint 1

STRUCTURE OF .GITLET

* contains all commits that aren’t active
* contains a “stage” folder for the files that are on stage
  + this is emptied when the staged files are committed
* contains a “garbage” folder where the files that are marked as untracking are stored
* use a hashset instead?

GITLET class

* HashMaps – commit ID, branch names
* myCommitTree: COMMIT objects are stored in a tree of commits
  + activeHead instance variable : the active pointer that might change
  + curCommit instance variable : pointer that keeps track of the current commit
  + prevCommit instance variable: pointer that keeps track of the parent of the current commit
  + branchHead instance variable: pointer(s) that keep track of head commit in other branches
  + init method : creates a .gitlet directory and initializes hashMap/tree
  + If one already exists, abort and print: A gitlet version control system already exists in the current directory.
  + commit methods :
    - moveCurPointer: moves the activeHead pointer to point to the new commit
  + addToStage: first checks if the file is in the “garbage” folder, meaning it is marked for untracking: if it is, the new commit should not include the file; adds a new commit of the files in the “stage” folder to the commit tree and empty out the “stage” folder and the “garbage folder.” print “File does not exist” if error.
  + makeCommit: calls commit constructor. creates a new COMMIT object that has “snapshots” of the files in the stage folder. A commit will only update the version of a file it is tracking if that file had been staged at the time of commit. In this case the commit will now include the version of the file that was staged instead of the version it got from its parent. In addition, a commit will start tracking any new files that were staged, that its parent lacks.
  + addStage method : should add staged files to the stage folder in Gitlet folder
  + remove method : if the file is in the “stage” folder, remove it; else put the file in the “garbage” folder (mark for untracking)
  + newBranch method : creates a new pointer at the head of the commit tree
  + removeBranch method : deletes the pointer of the given branch
  + reset method : moves the current pointer (and active pointer) to the given commit. Note this doesn’t delete any commits made after the given one; instead reset creates a new branch for new changes and commits made to that older commit.
  + 3 checkout methods :

1. getPrev: makes the most recent commit active and overwrite the file in working directory
2. getCommit: a particular commit in the commit tree replaces working files,
3. getBranch: takes all files at head of specified branch, replaces working files , switches activeHead to the pointer of that branch

* Static method Main: it parses user commands, then calls that command
  + if user doesn’t input any arguments, print: “Please enter a command.”
  + if a user inputs an invalid command, print “No command with that name exists.”
* globalLog method: gives the history of all the commits made, ever
* commitLog method: gives history of particular commit
* findMsg method : searches for all the commits that have a particular message
* showStatus method: displays what branches currently exist, what files are staged, what files are marked for untracking
* rebase method: unlinks branch from the master at the split point, then reattaches a copy of the branch to the front of the master. if the specified branch has its own branch, rebase does NOT copy that second branch over to the master.
* merge method : first check if the file at the head of the given branch is different from the one at the split point (we can do this by checking the time/date stamps of the commits); also check if the files in the current head is modified
  + if the branch is modified and current is NOT, then move the files in the branch head to the “stage” folder
  + if the branch is NOT modified and the current is, then do nothing
  + if both are modified, modified files in the branch head are copied into the working directory with name [oldfilename].conflicted

COMMIT class

* Many commit objects will be made, each can have multiple files
* Instance variables: commitID that keeps track of the number of commits made; a user-specified commitMsg; a time/date stamp commitTime; myFiles (keeps track of all files), myParent (myPrev), myChild (myNext);
* log method : gives the history of this commit (all of its inherited parents)