Aditi Singh and Rumeet Goradia (as2811 and rug5)
Asst3
Where's the File

Read Me

In this assignment we designed a program that builds a version control system. The resulting program contains a server side that maintains all the projects, while the client side maintains all the commit and fetches updates and communicates them to the server. This program uses multithreading so that the server can accept multiple clients.

WTF.c serves as the main function. It calls on a number of helper methods (located within helpermethods.c) that assist in the running of the program itself. For the function **add**, it tests to see whether the file that is called upon is already there. If it is already there (within the .Manifest file) and there have been no changes made to the file, the add function does not do anything. It checks this by comparing the hashes of the file's contents. If changes have been made, this function increments the version number and then updates the hash. In the case that the file was not already there in .Manifest, this function adds the file to the end of .Manifest with a version number of 0.

The method **checkout** lets the client request the project from the server. The server will then search for the project within the server repository and will send the latest version. The client will then create the project, its subdirectories, as well as the .Manifest.

The method **update** will let the client request the current .Manifest file from the server. The client will then compare the .Manifest file to that present in the server, recording all the differences in a .Update file which will be stored with the client. The files will be labeled with specific tag to signal what the changes are.

The method **upgrade** uses the .Update within the client to do three tasks: (M): fetch the file from the server and replace the client sides file with it; (A): is the same as (M); (D): will delete the client sides file.

The function **commit** receives the server's .Manifest and will compare it to that of the client's side, however unlike update it will be marking changes based on the server.

The function **remove** simply checks if the file is in the .Manifest and if it is not there, this function does not do anything. If the file is present, it changes the hash code to 64 hyphens so that the version number remains the same and does not revert to 0 in the case that the same file is added again.

The function **push** is similar to upgrade but this time the changes will be made to the server side. The server will look at tags and make changes to the project files based on them. The server will then send a success message to the client and upon getting this message the client will delete the .Commit file.

The function **create** will create a .Manifest file as requested and send it over to the client. The client then sets up a folder with the file and places the .Manifest that the server sent. This function will send back a specific char based on whether the function was successful or not. This function will fail to create the file if the a file with the same name already exists or if the client fails to communicate with the server.

The function **destroy** will delete a file as requested by locking the repository and deleting all files and subdirectories. This function will also send back a specific char based on whether the function was successful or not. If the project name does not exist or if the client cannot reach the server this function will fail.

The function **current version** sends the size of the server's .Manifest file for a specified project to the client so that the client is aware of how many bytes are expected to be received. The function then sends the input of the same .Manifest file to the client. The client then passes through the received input and then subsequently prints out the project version and the version for each file along with the file's path. The function skips over the hash code, however, because it is not necessary for this function. If the project does not exist on the server the function fails.

The function **history** sends over a file with all the operations performed to the project folder. The client does not need to have a local copy of the project in order for this function to work. If the project does not exist or if the client cannot successfully communicate to the server this function will fail to work.

The command **rollback** will revert a project back to the version number requested by the user by deleting all the most recent versions on the server's side. The client does not need to have a local copy of the project in order for this function to work. If the project does not exist, the version number does not exist, or if the client cannot successfully communicate to the server this function will fail to work.

The command **configure** saves the IP address and port of the server. This function does not connect to the server and instead just saved the IP address and port number.

This program has comprehensive error checking that checks all different types of errors, including, but not limited to, errors that would be a result of miscommunication between the client and the server as well as scenarios in which the project folder does not exist or if the same project is trying to be created.