README - Asst3 WTF

In this assignment we are to create a version control system. The project is based on a server client architecture where the server is multithreaded to accept multiple clients. On the client side we can do the following commands:

1. checkout:

    Client requests the entire project from the server. The server will look for the project in the server_repo and will send the latest version (.Manifest and all corresponding files). The client will create the project, all of the project's sub-directories, and the .Manifest.

2. update:

    Client will request the server's current .Manifest file for a given project. It will then compare the its own .Manifest file with the server's .Manifest file, recording all the differences into a .Update file, which will be stored on the client side. The files will be recorded with specific tags signalling what the changes are:

    - U (upload) : the file on the client side is a greater version than the server's side or the file doesn't exist on the server's side
    - M (modify) : the server has a greater version of the file than the client and the hash of the files are different
    - A (add) : the file exists on the server side but not on the client side
    - D (delete) : the file exists on the client side but not on the server side

    The update command will only record only M, A and D changes. and will be in the format:

    <M, A, or D> <version> <path> <hashcode>

    If there are no changes then the .Update file will be empty and the user will be informed that the project is up to date.

3. upgrade:

    Using the .Update the client will do three things based on the tags of the file:

    - M : fetch the file from the server and replace the client side file with it
    - A : same as above
    - D : delete the client side file

4. commit:

    Similar to update in the client will receive the server's manifest and compare them; however, now we are marking changes based on the server. There are three tags:

    - 0 : File is in the server side but not in the client side
    - 1 : File is in the client side but not in the server side
    - 2 : Files are on both sides but the client side is a higher version
        if the client side is a lower version there must be an error sent

    The client will make a .Commit file and send it over to the server.

5. push:

Similar to upgrade only this time the changes will be made on the server side.
The server will look at the tags and make changes to the project files based on them.

- 0 : delete the file
- 1 : Fetch from client side and replace the server side's corresponding file
- 2 : same as above

The server will send a success message to the client. Upon receiving the message the client will delete the .Commit file.

6. create:

Client will send the command create with project name. Server will create a subdirectory in .server_repo with the given project name. Inside the project directory a version1 subdirectory will be created and within that directory a .Manifest will be created. This .Manifest will be sent to the client and the client will create a project directory with the same name and place the .Manifest in there.

7. destroy:

The client will tell the server which project to destroy. The server will go in the project file and delete every file within. If there are subdirectories for example the version directories, the server will go inside them and delte all files then proceed to to delete the subdirectory. Then once the project directory is empty the server will delete the project form the .server_repo.

8. add:

The client side will add a file entry to the .Manifest of the project with a new version and hash.

9. remove:

The client will remove the file entry from the .Manifest file.

10. currentversion:

The client will request the current version from the server. The server will send its latest version .Manifest. The client will output a list of files under the project's name along with their version.

11. history:

The client will request a history. The server will send a file that contains all pushes since the project was created. The client will output something similar to what was outputted to STDOUT in update. However, there will be version numbers and a new line separating each push log.

12. rollback:

    The client will request a rollback. The server will find the version specified and delete all greater versions.