Where's the file? (WTF)

Authors:
Michael Ziegler
Eshaan Gandhi

WTFServer and WTF are a server and client that are used for version control. There are a variety of different things you can do with this server in terms of file version control.

How to Use:

Run the Makefile first to compile the project
The default target would give two executables the server executable and the client executable.
If you want to delete the server executables just run clean
If you want to build a test file run the test file executable

make - This should produce two executables WTF and WTFserver. They're the client and the server executables respectively.

make clean - This would delete the executables and "clean the folder"

make test - This creates the test executable that tests the folder and goes through with the commands one by one

To start the server use
./WTFserver <#port>

To use a client use
./WTF <ip_host_name> <#port>

WTFtest

This executable would test various commands. You can see the changes in the directories after the test executable has run. You would also be able to see all the messages from the client-side.

The test file has to find an empty port or else it would be a failure. The client would not be able to connect and there would not be a pleasant outcome

WTFtest runs through a basic usage of the client-server interface. It configures the client, creates projects, destroys projects, adds files, removes files, commits changes, pushes commits, rolls back to a prior version, updates, upgrades, performs history, and performs checkout.

The test has two options:

1. If it is run with no arguments in the command line the server would be hosted at 8080 and the client will connect with localhost
2. If an ip and port number are fed into the executable in the format ./WTFtest <ip> <port> then it would run using those credentials

The following are the commands and what they do:

1. /WTF configure <iphost> <#port> - Configures the file that helps connect with the server. None of the commands will work if this command is not run.

2. ./WTF checkout <project name> - If it does run it will request the entire project from the server, which will send over the current version of the project .Manifest as well as all the files that are listed in it. The client will be responsible for receiving the project, creating any subdirectories under the project and putting all files in to place as well as saving the .Manifest.

3. ./WTF update <project name> - Update's purpose is to fetch the server's .Manifest for the specified project, compare every entry in it to the client's .Manifest and see if there are any changes on the server side for the client. If there are, it adds a line to a .Update file to reflect the change and outputs some information to STDOUT to let the user know what needs to change/will be changed. This is done for every difference discovered. If there is an update but the user changed the file that needs to be updated, update should write instead to a .Conflict file and delete any .Update file (if there is one). If the server has no changes for the client, update can stop and does not have to do a line-by-line analysis of the .Manifest files, and should blank the .Update file and delete any .Conflict file (if there is one), since there are no server updates.

4. ./WTF upgrade <project name> - It makes the version of the client equal to the server

5. ./WTF commit <project name> - It prepares the project to be pushed onto the server. The commit command will fail if the project name doesn't exist on the server, if the server can not be contacted, if the client can not fetch the server's .Manifest file for the project, if the client has a .Update file that isn't empty (no .Update is fine) or has a .Conflict file. After fetching the server's .Manifest, the client should should first check to make sure that the .Manifest versions match. If they do not match, the client can stop immediatley and ask the user to update its local project first. If the versions match, the client should run through its own .Manifest and compute a live hash for each file listed in it. Every file whose live hash is different than the stord hash saved in the client's local .Manifest should have an entry written out to a .Commit with its file version number incremented.

6. ./WTF push <project name> - It makes the servers version equal to the client's version

7. ./WTF create <project name> - It creates a project on a remote repository. It initializes a manifest and sends the manifest over to the client.

8. ./WTF destroy <project name> - It locks the repository online on the server, cancels all the commits and deletes all the files.

9. ./WTF add <project name> <filename> - Adds a file to be tracked to the system. The file gets added to the Manifest and is given a unique Hash

10. ./WTF remove <project name> <filename> - Removes a file to be untracked by the system. Removes the file from the manifest

11. ./WTF currentversion <projectname> - Returns the current version of all the files from the manifest

12. ./WTF history <project name> - Returns the history of all the pushes and commits

13. ./WTF rollback <project name> <version> - Allows the user to rollback to the version they want to

The way to quit the server is using CTRL C - This would send the stop signal and quit with a message

Design

The server is built in a way that would allow multiples users to connect to it simultaneously by multithreading. Every time a new client connects to the server a new thread is spawned and all the instructions take place through that.

Our program hashes using the MD5 protocol of OpenSSL to generate a 32-byte hashed string for a given file.

.Manifest files

The manifest files contain the file path, the metadata, and the version number to be used. This is implemented in the following format <file path> <file hash> <version number>.

.History files

These files are updated every time something is pushed onto the server. They help when the history command is called on this server.