Assignment3: WTF Server

Colin Walsh
Christian Mitton

Project Summary

Our project can be split into the following three phases:

- (1.) Set up code
  - (a.) file structure
  - (b.) create manifest
- (2.) Client side code
  - (a.) ./WTF add <Project Name> <filename>
  - (b.) ./WTF remove <Project Name> <filename>
  - (c.) ./WTF create <Project Name>
- (2.) Server side code
  - (a.) ./WTF configure <IP/hostname> <port>
  - (b.) ./WTF checkout <Project Name>
  - (c.) ./WTF update <Project Name>
  - (d.) ./WTF upgrade <Project Name>
  - (e.) ./WTF commit <Project Name>
  - (f.) ./WTF history <Project Name>
  - (g.) ./WTF rollback <Project Name> <version>
  - (h.) Server notes

*** Important Note ***:

Most of the commands (listed below) work. However due to the deadline, finals, and other projects, we didn't have the time to glue all the code together to make a seamless, bug-less, fully functional project. The following commands work:

- add
- remove
- create
- configure
- checkout
- commit (still buggy)
- update

The Design

(1.) Set up code

(a.) File structure

Our assignment 3 folder has all .header/.c files, and a **server_repos** folder. "**server_repos" represents our server.** Whenever a new project is created with "create", a project will be created both on the server and locally.

Say the client has created the following 3 projects (which already exist both locally and on the server): project0, project1, and project2.

Within the server_repos folder would be 3 folders: project0_repo, project1_repo, project2_repo. Within each of these repos would be their respective project folder, and within these project folders would be all respective project files. So the server would look like this:

```
- server_repos
    -project0_repo
        -project0
            - file1.txt
    -project1_repo
        -project1
            - file1.txt
    -project2_repo
        -project2
            - file1.txt
```

(b.) Creating the manifest

Upon creation of a new project, both a server side project folder and a client side project folder will contain a new manifest. Our manifest has the following format:

```
<Manifest version number> <dummy text> <dummy text>
<ver number> <file path> <file hash>
            …
<ver number> <file path> <file hash>
```

(2.) Client Side

(a.) ./WTF add <Project Name> <filename>

Add will go in a project folder (on the client side) and append a new file to the projects manifest, as well as increment the version number.

(b.) ./WTF remove <Project Name> <filename>

remove will go in a project folder (on the client side) and remove a file from the projects manifest

(c.) ./WTF create <Project Name>

Create will create a project both on a server and locally. The creation of a project on the server will adhere to the format mentioned above (in other words, upon creation the project will have a <project name>_repo folder, and within this folder would be the <project name> folder)

(3.) Server Side

(a.) ./WTF configure <IP/hostname> <port>

Configure takes two arguments which are both strings. IP/Hostname works with a valid IP address or a hostname. Port is required to be a value between 1024 and 65535 (since 1-1023 are reserved port numbers). Configure will notify the user if the arguments are too few, too many, or if the port is beyond the ranged previously.

(b.) ./WTF checkout <Project Name>

Checkout takes a project name as an argument. Checkout will verify a project exists on the server, and if it does, will tar all files on the server and send them to the client directory. The project is then untarred into the target directory.

(c.) ./WTF update <Project Name>

Update will have the client request the servers current manifest for a project, scan through the local files, and output the differences according to the UMAD code

(d.) ./WTF upgrade <Project Name>

Upgrade will reference the update file and perform the specified actions depending on the code. Then, it will request the project repo from the server, where it will be tarred, sent over the network, and untarred in the client directory. It will not overwrite any files that are not in the tar directory.

(e.) ./WTF commit <Project Name>

For commit the servers manifest is fetched, and each hash codes in the clients manifest is recomputed. The server and client manifest is compared and every hashcode that is different has an entry written out to .Commit.

(f.) ./WTF history <Project Name>

Didn't have time to finish

(g.) ./WTF rollback <Project Name> <version>

Rollback takes a project name and version number. The server will find the specified version specified by the client, and if it exists, will reset the repo to that state and remove all versions greater than the version specified. Sends back a complete message on success.

(h.) Server Notes

The server permeates between sessions and doesn't care about new requests/old requests. All data is saved in the "server_repos" folder. Which is required for correct functionality.

Thread synchronization

There is a very simple mutex around the execute function when accessing repos. After completion it will allow the next thread through.

Extra Credit

(a.) Compress

Compress uses the tar system call to compress directories. There are a few options used in order to correctly tar specific directories.

(b.) Decompress

Decompress also uses the tar system call to decompress. Again, various arguments were used to make sure files were un-tarred to the correct directory.