Readme.pdf

Design/Implementation:

Adam and I implemented our version controlling system with the exception of the push, upgrade and history commands. We were able to make a multi-threaded server/client relationship through TCP sockets that allows multiple users to make changes to files and repositories on the server.
We used a Linked List of projects to keep track of each repository on the server, and used a Linked List of manifest lines in order to parse through files and compare their input efficiently. In the commit and update functions, these Linked Lists of manifest entries are compared with each other, at a worst case of O(n^2). To avoid this worst case in the majority of circumstances, we used break' to get out of the inner loop when a match was found, for efficiency purposes. We implemented add, remove, create, destroy similarly, all building out paths in order to fulfill their responsibilities. While we understand the importance of mutexes and how they are necessary to protect race conditions, we were unable to implement them into this project due to time constraints. If we could have, we would have had the project lock for push, upgrade, destroy, so that no other thread could alter the repository while the user is attempting to access information or make changes to it.

Time/Space Usage:

Linked List (LL) = O(n) for searching, but O(n^2) while nested searching
Searching through files as well would be around O(n^2) as well as the user would have to look through n items, n times. Since the files that are used in this project are not very large, this implementation can hold strong.

Complexity:

Overall, this is a very long project, and our algorithm reflects this. There are many moving parts and due to this, it took a lot of modularization on our part to make the code readable and understandable. Organization was essential with this project, and although we were unable to finish the last few functions, we made sure that the ones we did do were done very well. We stored all our client functions in clientfunctions.c and our client would run through WTF.c and WTF.h. On the server side, our functions were in serverfunctions.c and it ran through WTFserver.c and WTFserver.h.

Finally, for our functions that require filenames, we require the user to use a full path, therefore if a user wants to add testfile.txt to project1, they would have to enter the command:

```
./WTF add project1 <project>/<version+version#>/<filename>

Example: ./WTF add project1 testfile.txt
```

Overall, we felt that we did a good job with a difficult task and we know we will do better on the final. Thank you for taking the time to grade this!