Synopsis - These are the ways a project can be modified:
./WTF configure <host ip> <port number> - Sets up server connection for the rest of the
./WTF create <project name> - Creates a project on the server.
./WTF destroy <project name> - Deletes a project on the server.
./WTF add <project name> <file> - Adds a project to commit on client.
./WTF remove <project name> <file> - Removes a project to commit on client.
./WTF currentversion <project name> - Gets the version of project and every file on server
./WTF checkout <project name> - Makes clone of the project on the client.
./WTF commit <project name> - Notes what changes to push.
./WTF push <project name> - Pushes changes to server.
./WTF update <project name> - Notes what changes to pull.
./**WTF upgrade <project name>** - Pulls changes from server.
./WTF history <project name> - Gets push history of project from server.
./WTF rollback <project name> <version> - Rolls project back to specified version.

Description

The server is a multithreaded application. Each time a client connects to the server, a new thread is created and the specified command is executed within the thread. Any time a client is accessing the repo, the entire repository is locked with a mutex. This ensures that at the time of connection, the server is getting only one set of changes and by the time another client connects, the server is definitively up to date, and no merge issues are generated.

Since each client connection was generated within a thread, every operation is fully contained in that thread. As a result of this, there is no reason to store a list of every thread.

Every file that we had to send over the network other than the actual files of the project (i.e .Conflict .Update .Manifest .Commit .History) all had the same structure of version number (if necessary) and then one line for each file in the form: **<status> <version> <file path> <md5 hash>**

This made transferring those files simpler than having to worry about the format of each file. Any other files/directories were compressed using system calls and sent over the socket as a compressed file. Additionally, on push, old versions of the project are compressed and stored in a hidden .Backups folder in the project. This folder is ignored when sending changes to the client, so the client does not have access to the backups, only the server does.

<u>Errors</u>

Note: Any functions that do not appear on this failure state list explicitly only fail on one or multiple of the common error states listed as the first bullet point.

- Common Failure Cases
  - Client cannot connect to server
    - Prints “Failed to connect to server” to STDOUT
  - Project **doesn’t** exist on server (EXCEPTION: CREATE)
    - Prints “Project doesn’t exist on server’” to STDOUT
- Create
  - Project already exists on server.
    - Prints “Project already exists on server” to STDOUT.
- Add
  - File is in client manifest
    - Prints “File is already added to manifest” to STDOUT
- Remove
  - File is **not** in client manifest
    - Prints “File is not in manifest, cannot be removed” to STDOUT.
- Checkout
  - Project already exists on client.

        - Prints “Project exists on client” to STDOUT.
- Commit
    - Client has non-empty .Update file
        - Prints “Client is behind the server, upgrade before you commit” to STDOUT
    - Client has .Conflict file
        - Prints “Conflicts exist” to STDOUT
    - Client project version does not match server project version.
        - Prints “Server and Client project versions do not match, please update local project” to STDOUT
    - File versions are not synced
        - Prints “” to STDOUT
- Push
    - .Commit does not exist on server/ .Commits do not match
        - Prints “Push failed, please commit again. .Commit either expired or doesn’t exist on server.” to STDOUT
- Update
    - Conflicts are found (NOTE: COMPLETES EXECUTION)
        - Prints “Conflicts exist. They must be resolved before upgrading the project” to STDOUT.
- Upgrade
    - .Conflict exists
        - Prints “Conflicts exist. They must be resolved before upgrading the project.
    - There is no .Update
        - Prints “Client does not have .Update file. Run update before upgrading.” to STDOUT.
- Rollback
    - Specified version does not exist on server.
        - Prints “Specified version does not exist on server” to STDOUT.