# Running the code

The code was developed using IntelliJ on windows. If running through same method or another advanced IDE (VSCode, etc..), open both Server and Client java files and run each of the files on separate terminals.

## Running on Linux

1. Open Terminal and call *javac Server.java* and *javac Client.java*  to compile files.
2. Call *java Server*
3. Open second terminal and call *java Client*

The system only handles one client.

# Authenticating

To authenticate against the server,

1. Call *user* <username>. The username is checked against the database in the identification.txt file. If the username is ‘admin’ skip other steps
2. The next command can either be *acct* or *pass* followed by <account> or <password> respectively.
3. Depending on which command was entered in step 2, the other command should then be called.
4. ‘!’ should be returned if successfully logged in.

# Functionality

## Done

Calling ‘done’ command by the client breaks out of both while loops within the Server and Client and the process ends for both.

(All other commands will require authentication before use)

For commands involving files, the file name and type should be specified. I.e.; for notes.txt, simply saying ‘notes’ will not work. You must specify ‘notes.txt’

## Type

The ‘type’ command is called, the second argument is chosen between {A, B, C} Ascii, Binary and Continuous respectively. This allows the method of transmission to be changed between the different options.

## List

To use the list function, enter ‘list’ as the first argument followed by either {F, V} as the second argument. The third argument is optional and is used to specify the directory. If not given the directory shown will be that of the current working directory. Any Directory passed in as the third argument will have to be relative to the root directory which is set to the ‘files’ folder.

## Cdir

To change the working directory, call this function with the second parameter as the destination directory. The argument should be relative to the root directory, which is the files folder. Calling this command with no second argument will by default set the working directory to the root directory. This command can only be called after you have already logged in. There are no directories currently that have restricted access.

## Kill

To delete a file, call this command with the second argument as the file you wish to delete. If it exists, the file will be deleted.

## Name

To change the name of a file, enter the name command follow by the file name as the second argument. If it exists it will prompt a “+File exists” and wait for the Tobe command.

## Tobe

After Name has been called with a file name as the second argument already, Tobe can then be called with the second argument being the new name of the file. If Name has not already been called Tobe will not work.

## Retr

This command allows the client to retrieve a file from the server and store it into where the client code is running from. First argument is ‘retr’ followed by the desired file as the second argument. If the file exists, the server will wait for the send command to be entered. This command will return the number of bytes in the desired file.

## Send

After retr command is called successfully, send command can then be executed. After calling send, the file transfer process will begin. Currently the lockout functionality is not implemented for this function. Doesn’t require any further arguments.

## Stop

After retr command, if you would like to cancel the retrieval of this file, instead of calling send, call the stop command. Doesn’t require any further arguments.

## Stor

Used to store files from clients directory in to the Servers’ *files* directory. This command requires 3 arguments. The first being the *stor* itself, followed by the type {NEW, APP, OLD} and finally the client file. The server checks to see if the file that will be sent exists within the server or not. If it doesn’t exist regardless of the type selected, a new file will be created. If it already exists the NEW type will not work as the system doesn’t support generations. App will append whatever is in the clients file onto the servers version and old will replace the servers version with the clients versions, if the file does exist.

## Size

After calling *stor* and receiving a ‘+’, the user must then reply using command *size* followed the size of the file to be transferred in bytes as the second argument. For ease of use the size of the file in the client system is also printed after calling the stor function. This transferring of the file doesn’t work as intended if this size value passed in not accurate. After this is called successfully the file transfer will begin. NOTE: the size of the server storage is hardcoded to an arbitrary value of 20000, therefore file transfers of bytes greater than this cannot work.

# Testing

In the testCases directory, two test files are provided that tests the simple functionality of all the commands. Please see those txt files for order of executing tests and the expected server result. Note: before following test2, look at the current testing txt files in the directory’s to verify that they are getting transferred. In files/skmu104/notes.txt look at the current data there. This will be appended when following the tests.

# Other files

* Files directory is the root folder for the server. Acts as the Servers storage.

# Error handling

The system is set up to handle invalid inputs and too few arguments given to commands.