

# Password Project report

The requirement for this project was to create password management system. The password management has a server which is always running until a shutdown command is sent to it. The role of the server was execute instruction from different client concurrently.

## Introduction

In this report I will talk about how I created a simple password management system using bash. The password managers allows a user to create a user, insert login details into the folder. It also allows users to update their login details, remove login details and show all of the login details they have saved. My password also works currently, only one user can has access to any give folder at a time.

We had to make the followings:

Init.sh # user make a new file

Insert.sh #user create a file in its folder

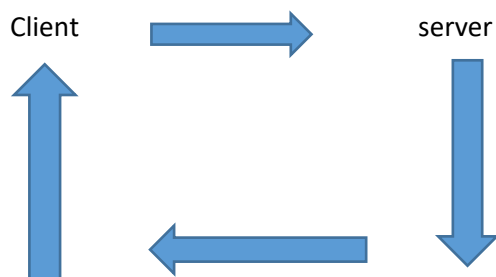
Ls.sh #shows users all the file in the directory

Rm.sh #remove a file if the user no longer needs it

Client.sh #this is GUI for the client that allows it to use the password management system

Server.sh #this is what takes request from client and execute it.

The server and the client communicate this following way:



The client and the server communicate through a pipe. The client send a request to the server using a pipe and the server echo a message back to the client using a pipe.

Before they can send a message to the server it must pass some parameter test, all of which I will discuss in the "Design" section.

## Design of the system.

### Init.sh

The init.sh script takes one argument. The argument taken will be the name of the directory. The requirement of the script is to make this directory and echo "The directory has been made". If the directory already exist it will also tell the user.

### Insert.sh

The insert.sh is the script that input a fold and file or just a file into the directory. When this script is running on the client side it takes in 4 argument in this following format

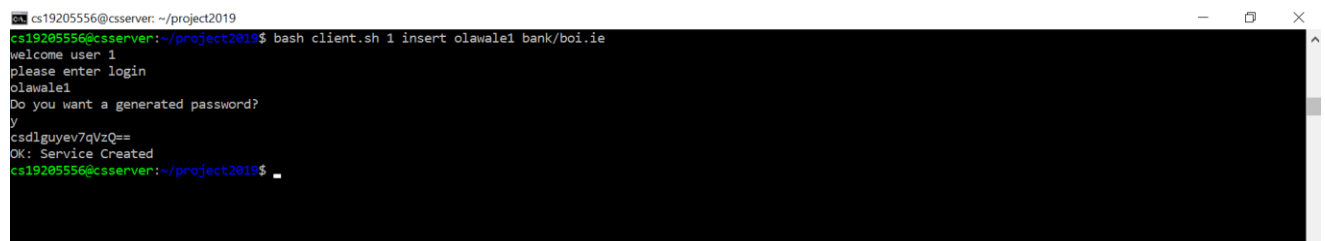
- Client-id #This is required so the user can communicate with the server
- Insert #the keyword that give a signal to the server on what it should do.
- Username #the name of the folder the user is trying to insert into
- Service # the service the user is trying to create e.g. aib.ie .google.com

If the argument don't come in this format the client.sh exits.

If client.sh receives the argument in that format. Then it does the followings

- Ask the user to enter the login details #this is the login detail for the service user is trying to create
- Give the user an option whether they want a random generated password, if they do it generates a password for the user.
- If the user don't want a generated password. It asks the user to input their own password.

Once this is done the client.sh script send it over to the server.sh. The server.sh execute it and echo's the outcome to the client.

A terminal window screenshot showing the execution of a script. The prompt is 'cs19205556@csserver: ~/project2019'. The user enters 'bash client.sh 1 insert olawale1 bank/boi.ie'. The script outputs 'welcome user 1', 'please enter login', and 'olawale1'. It then asks 'Do you want a generated password?' and the user enters 'y'. The script outputs a generated password 'csd1guyev7qVzQ==', 'OK: Service Created', and returns to the prompt 'cs19205556@csserver:~/project2019\$'.

## Rm.sh

This script remove a service file if the user doesn't want it anymore. When trying to execute this on the client side it takes 3 Argument

1. Client-id #this is needed in other to a pipe so it can communicate with the server
2. Rm #The keyword that indicates to the client.sh and server.sh on what it should do
3. Username #The name of the directory
4. Service name #the name of the service file the user is trying to remove

If the request don't come in, in that order in won't work.

Example of the rm.sh script running on the client.sh

```
cs19205556@csserver: ~/project2019
cs19205556@csserver:~/project2019$ bash client.sh 1 olawale google.com
Error: parameter problem
cs19205556@csserver:~/project2019$ bash client.sh 1 rm olawale1 google
welcome user 1
Error: Service Does not Exist
cs19205556@csserver:~/project2019$ bash client.sh 1 rm olawale1 paypal.com
welcome user 1
The service has been removed
cs19205556@csserver:~/project2019$ bash client.sh 1 rm olawale1 paypal.com
welcome user 1
Error: Service Does not Exist
cs19205556@csserver:~/project2019$ bash client.sh 1 rm olawale paypal.com
welcome user 1
Error: User Does not Exist
cs19205556@csserver:~/project2019$ ^C
cs19205556@csserver:~/project2019$
```

The picture above shows all possible outcome's when the server try's to run the function

1. The first one came back as an error because I didn't include the Rm
2. The second one came back as in error because the service I am trying to remove doesn't exist
3. The third one came back as positive because all the argument was correct and the service was removed.
4. The fifth one came back as an error because the user doesn't exist.

## Ls.sh

The ls.sh script is to allow the user to see all of the service they have in their directory. This however doesn't show you what's in each file it only give you a tree view of the files. This function requires the following to run on the client side

1. The client-id
2. Ls
3. Username

```
cs19205556@csserver: ~/project2019
cs19205556@csserver:~/project2019$ bash client.sh 1 ls olawale1
welcome user 1
olawale1
├── bank
│   ├── ap
│   ├── apple
│   ├── applepay
│   └── boi.ie
├── facebook
├── facebook.com
├── facebookop
└── google.com

1 directory, 8 files
OK: Here are your files
cs19205556@csserver:~/project2019$
```

## Update

The update function is for when the user want to update what they have in their file already. For a user to update their file they need to give 4 arguments in the right format.

1. Client-id
2. Update
3. Username #name of file
4. Service #The account they are trying to update

Once request is giving in the right format, the user will be asked to enter their new log in, after this the user will be given the option for a random generated password or the user will be asked to input their own password.

```
cs19205556@csserver: ~/project2019
cs19205556@csserver:~/project2019$ bash client.sh 1 update olawale1 apple
welcome user 1
New login
olawale1
Do you want a generated password y/n?
y
L51lRuLF7CkSmg==
cs19205556@csserver:~/project2019$
```

## Show.sh

This is the script that that client what they have saved for a service for example google.com. if use want to see their login in details. They must give 4 arguments,

1. Client-id
2. Show
3. Username
4. Service

Example

```
cs19205556@csserver: ~/project2019
cs19205556@csserver:~/project2019$ bash client.sh 1 show olawale1 apple
welcome user 1
Error: Service Does not Exist
cs19205556@csserver:~/project2019$ bash client.sh 1 show olawale1 facebook
welcome user 1
login:olawale
Password:olawale
cs19205556@csserver:~/project2019$
```

## Shutdown

Part of the requirement was to let the client be able to shut down the server. To only 2 argument is required.

1. Client-id
2. Shutdown

## Server/client scripts

The communication between server and client happen through the use of "*pipes*". The first thing the server does is to make a **server.pipe** and then start reading from it. At the beginning, until a client sends a request, this pipe will be empty thus the server remains in a "waiting state". When the Client starts, it waits for an instruction by the user identified with a custom **user\_id**. Once the instruction is typed and sent by the user, the client checks if the **server.pipe** exists. If it doesn't, it means that the server is not running and so throw an error. Otherwise, it makes a new pipe **\$client\_id.pipe** and send the user input into the **server.pipe** and wait for a server reply on the "**\$client\_id**",**pipe** . Once the server script gets some instructions on the **server.pipe**, it triggers the appropriate script to run passing as arguments the elements of the user request then it gets back the results of the script and send them back to the client through the appropriate user pipe.

## Concurrency

For the program to run concurrently. I made semaphore\_p.sh and semaphore\_v.sh. The purpose of these two script is to lock a directory currently in use. This will ensure only one process can be in its critical section.

The semaphore\_p.sh creates a lock file if it doesn't exist. If a lock file exist that's mean the fold cant be accessed by another process.

The semaphore\_v.sh deletes the lock-file when script has run. When deleted this mean another process can enters it critical section.

## Challenges

At the start of the project I face a lot of challenges because I wasn't really good at coding in bash. In the labs never really understood what was going on. This project gave me a better understanding of bash. It made me realise Coding in bash is actually not as hard as it look.

In my show function I had issues on how to echo multiple lines to the client. To solve the issue I had to use a while loop.

Also when trying to send argument from the client.sh to server.sh at start I had problem in trying make the server.sh know which argument is which. To solve this problem I put everything the server read in into an array then index it.

## Conclusion

This project was one I really enjoyed although it was long. During this project I made my progress by solving a number of problems. Solution to each problem by myself was the most important part of the project and this provided me with experiences which will help me in future. Some important things that I learned include designing a good program architecture and converting real life situations into an efficient code, and how to write an good looking easily readable and understandable as well as time and memory efficient code. This project also taught that when solving a large problem it really important to break down into small parts.

## Bonus

In my password management system I input a command that generates a random password for the client if the want.