CT213 Assignment 1

Michael Mc Curtin, Tomasz Gruca

Introduction

In this assignment, we created a social network, *BashBook*, using bash scripts.

This social network is comprised of users who can interact with each other by forming friendships and posting messages on each other's profiles.

These functions are performed by bash scripts which run on a central server. To demonstrate this functionality, the user can give commands to this server via a command-line interface. In a real-world scenario, these commands would likely be triggered by a backend following user interaction with a *Bashbook* application or website.

System Organisation

The social network revolves around four main functions, each with its own script:

1. Creating users (**create.sh**)
2. Adding users to a user's friends list (**add_friend.sh**)
3. Posting messages on a user's wall (**post_messages.sh**)
4. Displaying a user's wall (**display_wall.sh**)

To simplify the user experience, these scripts are all triggered via commands given to the server script **server.sh**, which runs constantly.

```
Accepted Commands: {create|add|post|display}
create Joseph
User created
Accepted Commands: {create|add|post|display}
create Francine
User created
Accepted Commands: {create|add|post|display}
add Francine Joseph
Friend added
Accepted Commands: {create|add|post|display}
post Joseph Francine hi:)
ok: message posted!
Accepted Commands: {create|add|post|display}
display Francine
start_of_file
Joseph: hi:)
end_of_file
Accepted Commands: {create|add|post|display}
```

example Bashbook interaction

Implementation

As this was a pair project, Tomasz and I decided to split responsibility for the code based around the ideas of 'users/friends' and 'wall'. Therefore, he mainly handled create.sh and add_friend.sh, while I mainly handled post_messages.sh and display_wall.sh. We worked together on server.sh to integrate our code seamlessly.

Users/Friends

create.sh

Creates a directory for each user containing two files: friends.txt and wall.txt.

If the input is invalid or the user already exists, no action will be performed.

add_friend.sh

Adds a friend to another user's friends.txt.

If either user does not exist or the users are already friends, no action will be performed.

Wall

post_messages.sh

Adds a specified message from one user to another user's wall.txt.

If either user does not exist or the sender is not a friend of the receiver, no action will be performed.

display_wall.sh

Prints out a user's wall, line by line.

If the user does not exist, no action will be performed.

server.sh

Runs an infinite loop which requests input from the user.

This input is then checked for a valid request, if one is found then the desired script will be executed.

Challenges Faced

Getting used to scripting in the bash environment was an interesting challenge for me as before this assignment I did not regularly use Linux and I tend to write my code in IDEs.

I used Linux's Nano code editor for this project – the lack of code autocompletion left me with a greater understanding of bash as I pieced together the code, referring to bash documentation and years of online reference material along the way.

Navigating the Linux file structure via commands in post_messages.sh and display_wall.sh was initially quite tricky and took some trial and error to understand.

Accepting the arguments for post_messages.sh was quite a struggle for me, indeed after much trial and error I could not implement a user-friendly way to pass multi-word strings as a parameter for the message. As a workaround, the user must separate words in the message with a backslash.

Reading the four-parameter argument for server.sh initially required some thinking, but I eventually realised that it could be implement by reading the user's input as an array, then using each element of the array as the parameters.

Conclusion

Overall, Tomasz and I are quite satisfied with our first implementation of *Bashbook* – we feel it satisfies the main 4 operations as outlined in the assignment details.