

CT213 Computer Systems & Organization

Assignment 1: BashBook version 1.0

Team – Maxwell Maia & Cathal Lawlor

Maxwell - 21236277

Cathal - 21325456

Introduction

We were tasked with this project of BashBook.

BashBook – A rudimentary version of Facebook built entirely with Bash scripts. It includes the features of an early version of Facebook, creating a user account, adding other users as friends, posting to each other's walls and displaying the various messages left on these walls by each other.

We took an approach of pair programming together in the computing systems labs. We found this was the most productive as we could bounce ideas off each other instantly and not struggle with explaining it. We coded the program on Maxwell's laptop but had similar and fair amounts of time spent on this project. We don't have clear contributions over each part of the script other than Maxwell starting with the user creation & adding friend base template.

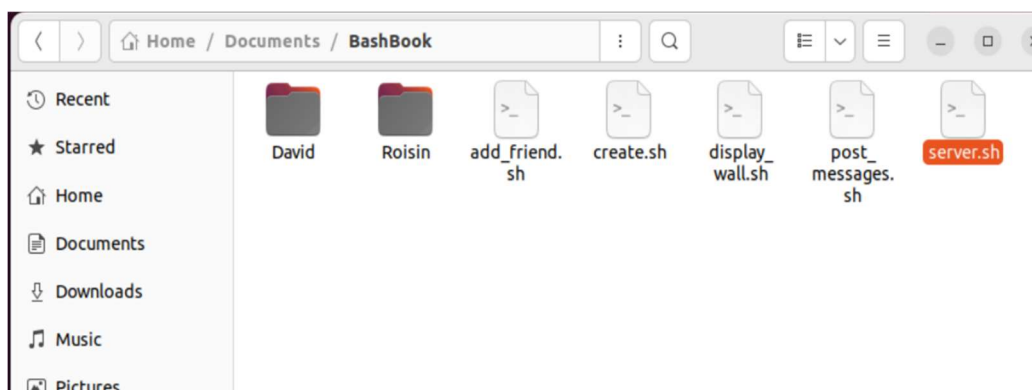
Organisation

The program's server is currently hosted locally on the machine. All commands are run through the ubuntu terminal.

All the .sh files can be called upon by the server.sh bash script which. The is a server script that can call on the rest of the scripts to do operations within the program. This makes the user interface / commands a lot more uniform for the user, e.g. no need to keep typing new filepaths.

All the files are in the one directory, making it much easier for us to work with. Our bash scripts and user directories are found in the "BashBook" directory. Each user has their own directory in the "BashBook" directory.

This organisation approach is not scalable at all, but it works currently for its desired purpose.



Features

Script - Creating a user.

Create command `./create Maxwell` will run the `create.sh` script.

Arguments: [new user id]

A user exists if a directory named the same as the username exists. Certain conditions need to be met before a user is created. The script checks these first.

1. Exactly one parameter needs to be given to the script. Else the script will give an error message and exit 1 (unsuccessful). The parameter is the new user's name.
2. If the user already exists, the script will give an error message and exit 1 (unsuccessful).

When all conditions are met, the script makes a directory inside the server. Inside the directory it creates two empty files. "wall" and "friends."

The wall file will store each message in the user's wall - each line will be a new message. The friends file will store the friends of the user. For now, these files are empty.

Script – Adding a friend.

Add friend command `./add_friend.sh Maxwell Cathal`

Arguments: [user id] [friend user id]

If there is no directory for the user, then an error message is given, and the script is exit 1 (unsuccessful).

If there is no directory for the friend, then an error message is given, and the script is exit 1 (unsuccessful).

Otherwise, the friend will be added to the user's friend list file, and the script is exit 0 (successful).

"Cathal" will be appended on a new line to the `/BashBook/Maxwell/friends` file.

Script – Posting a message to another user's wall

`./post_messages.sh Maxwell Cathal Hi how are you?"`

Arguments: [sender user id] [receiver user id] [message]

Certain conditions need to be met before a message is posted in someone's wall. The script checks these first.

1. If there is no directory for the sender (the sender is not a valid user), then the script will give an error message and exit 1 (unsuccessful).
2. If there is no directory for the receiver (the receiver is not a valid user), then the script will give an error message and exit 1 (unsuccessful).
3. If the sender is in the receiver's friend file (the users are friends), then put the message in the receiver's wall and exit 0 (successfully) with a message. Else the script will give an error message and exit 1 (unsuccessful).

The wall file will store each message in the user's wall - each line will be a new message. The friends file will store the friends of the user. If someone is trying to post a message in a wall, first the sender's username needs to be in the friend list of the user whose wall is being posted to.

* A note of friends. Friendships in our app are asymmetric. The sender must be in the friend list of the receiver for a message to be posted in the receiver's wall.

Script – Displaying a user's wall

`./display_wall.sh maxwell`

Arguments: [user id]

Conditions that need to be met -

If there is no directory for the requested user (the user with the wall is not a valid user), then the script will give an error message and exit 1 (unsuccessful).

If there is a directory for the user, the script will print to the screen the user's wall.

Script – Server

`./server.sh`

Arguments: none

No conditions are needed.

It keeps the computer in a constant loop, checking for our exact requests that we have created.

Requests

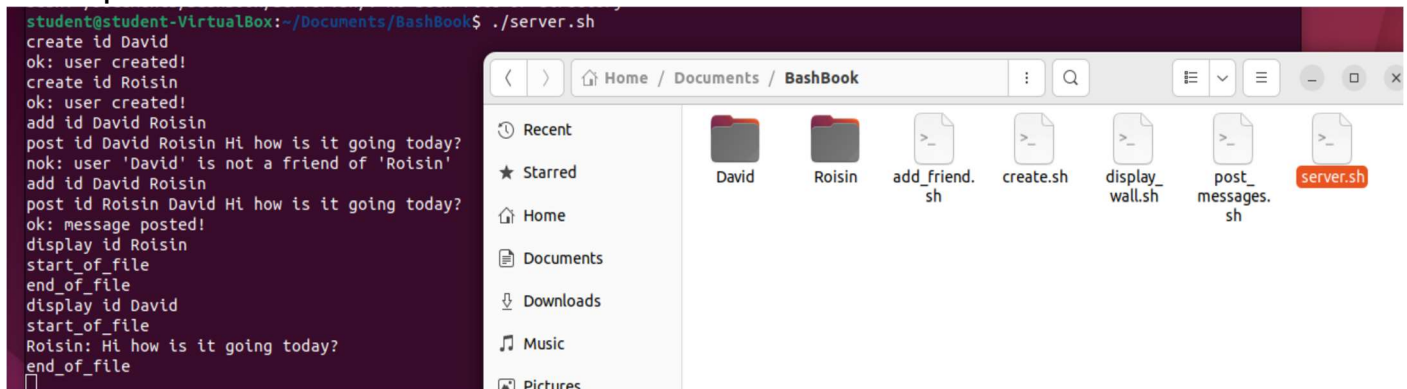
Create a user – `create [id] [username]`

Add a friend – `add [id] [user] [newFriend]`

Post a message – `post [id] [sender] [receiver] [message]`

Display a wall – `display [id] [username]`

Example



The screenshot shows a terminal window on the left and a file explorer on the right. The terminal window displays the output of the `./server.sh` script, including the creation of users David and Roisin, adding Roisin as a friend of David, posting a message from Roisin to David, and displaying the walls of both users. The file explorer shows the directory structure created by the script, including folders for David and Roisin, and files for the scripts `add_friend.sh`, `create.sh`, `display_wall.sh`, `post_messages.sh`, and `server.sh`.

```
student@student-VirtualBox:~/Documents/BashBook$ ./server.sh
create id David
ok: user created!
create id Roisin
ok: user created!
add id David Roisin
post id David Roisin Hi how is it going today?
nok: user 'David' is not a friend of 'Roisin'
add id David Roisin
post id Roisin David Hi how is it going today?
ok: message posted!
display id Roisin
start_of_file
end_of_file
display id David
start_of_file
Roisin: Hi how is it going today?
end_of_file
```

You can see when we run `./server.sh` we can now use the specified **requests** above to run the features that we desire.

We create users David and Roisin. This creates two directories – “David” & “Roisin” in the BashBook directory.

We then have Roisin added to David's friends list.

We try to have David post to Roisin's wall, however, David is not on Roisin's friends list so the message is not added to Roisin's wall and an error message is displayed.

We then successfully have Roisin post to David's wall. Roisin is in David's friends list so the message is added to David's wall and a success message is displayed.

From there we can display both of their walls. Roisin's wall is empty. David's wall has Roisin's message – “Roisin: Hi how is it going today?”

Challenges

1.

In the `post_messages.sh` script, Maxwell struggled to understand the logic needed to make a check that the two users are friends. Cathal helped Maxwell understand this logic much better through offering different perspectives and a rough idea for how to implement it.

We talked out how we want to implement friends in our program. We decided at this point that the friends would be asymmetric, this means that only one user would need to be friends with the other to post a message in their wall.

We also had to take a moment of deciding which user had to be friends with the other to post messages on the each's walls due to the nature of friendships being asymmetric. It was tricky putting the theory into practise in the code.

We sorted this out through using hypothetical examples. Labelling the two parties as "sender" and "receiver".

Through brainstorming this we figured out that the "sender" must be in the friends list of the "receiver" to post a message to the "receiver's" wall.

→ "I send a message to you, for you to receive the message (on your wall), I must be your friend (I must be in your friend list)."

2.

In the `post_messages.sh` script, Maxwell struggled to write code to check the error condition where the sender's name had to be in the friend list of the receiver.

Initially we tried to check the error condition first (if sender ID is not in receiver's friend list), but couldn't get that code to work.

Our solution was to check for the success condition first (if sender ID **IS** in receiver's friend list), with an else statement for the failure condition.

This refactoring of the code made it a lot easier to write and housed the same functionality.

For the working solution of this `post_messages.sh` error check, we repurposed a command from the assignment instructions of the "add friend" feature that uses the "grep" function. In this case grep is used to check if the sender ID is in the receiver's friends list.

We put this in an if statement holding the following logic: If the sender's ID is in receiver's friend file, the code will post the message in the receiver's wall and exit successfully with a message stating that the operation was successful.

Everything afterwards is effectively the "else" of that if statement. So, else: exit and display a message stating that the operation was unsuccessful.

Honestly the biggest challenge for us was writing this report. We don't like words.

Conclusion

We have made a working rudimentary version of Facebook.

With our program you are able to:

1. Create a user, with a friend list and a wall.
2. Add other users to your friend list. (Asymmetric – Adding a friend puts them on your friend list, they can now post on your wall).
3. Post a message to a user's wall with the name of the user who posted the message.
4. Display a user's wall.
5. Do any of the above operations by sending requests to the server.sh bash script.