CT213 University of Galway
Computing Systems & Organisation 2022/2023

*Lab Project*

# BashBook: Building Your Facebook-Like Social Media Using Bash

Assignment 1. Version v1.1

## 1 Project Description

In this project, you will be implementing a basic Facebook-like social media server (called *BashBook*) in Bash.

In *BashBook*, all users have their respective walls, users can become friends, and write on each other's walls.

## 2 The Server Side

We will now describe the system you will have to build and its different features.

### 2.1 Data Storage

The server is composed of a set of users–each with a unique ID (i.e., a username). A user whose ID is `$id` (e.g., `takfarinas`) possesses a directory with the same name as `$id`.

Each user has two files in their directory:

- a wall file made of a sequence of text lines–each of them being a message posted by a friend. Each message has the following structure: id of the friend: message.
- a friends file which contains all the friends of the user (an id per line). The only users who can post messages on a user's wall are the friends of this user.

An example for the user "takfarinas": in the directory `takfarinas` there are two files: `takfarinas/wall.txt` and `takfainas/friends.txt`.

```
$> less takfarinas/wall.txt
sean: great show last night
niamh: it was ok
sean: just ok?

$> less takfarinas/friends.txt
john
niamh
sean
max
```

The server needs to implement the following 4 operations.

1. Create a user, i.e., create a directory for a user and initialise the two files for this user.
2. Add a friend to a user
3. Post a message on someone's wall
4. Display the content of a wall

### 2.1.1 Creating a User

First write a script `create.sh $id` that requires one and only one parameter (`$id`) and creates the directory and the files of the user `$id`. Your script will print the following messages in the terminal to notify what has happened during the execution of the script:

- `nok: no identifier provided` if the script did not get any parameter.
- `nok: user already exists` if user `$id` is already on the server.
- `ok: user created!` if everything went well.

Those 3 messages are the only allowed outputs of the script. This will be important for the assignment 2 (an extension of this lab project).

### 2.1.2 Adding a Friend

Now you need to add the feature that allows to add a friend (`$friend`) to a user (`$id`)–this will be done in a script `add_friend.sh $id $friend`. First, check that the user exists (i.e., if there is a directory called `$id`). A friend is also a user, therefore, you also need to check that the friend exists as well (i.e., if the directory `$friend` exists). Finally, you need to check that the user `$friend` is not already in the list of friends of the user `$id` before adding them. To check that, look up for the expression `$friend` in the file `$id/friends`. For instance you could check the exit code of the grep command:

```
if grep "^$friend" "$id"/friends > /dev/null; then
    ...
fi
```

The exit code is 0 (true) if `$friend` is in the user `$id`'s friend file and 1 (false) otherwise. The redirection to `/dev/null` is important not to spam the standard output of your script. Your script add friend.sh has three possible output messages:

- `nok: user '$id' does not exist` if the user `$id` does not exist (i.e., there is no repository `$id`).
- `nok: user '$friend' does not exist` if the user `$id` does not exist (i.e., there is no directory `$friend`).
- `ok: friend added!` otherwise.

Make sure your script `add_friend.sh` has only those 3 outputs–in particular make sure that the error messages are redirected to `/dev/null`.

Note: There is one thing I have not mentioned yet: is friendship symmetric or not? [This is really your call :), but it is easier to keep it asymmetrical.]

### 2.1.3 Posting Messages

Next, your program has to be able to post messages on to user's walls.

Implement a script `post_messages.sh $sender $receiver message` that takes at least three parameters. The first two parameters are users of the server: `$sender` is posting the `message` and `$receiver\verb` receives the message on their wall. Next parameters compose the message. Adding a message on a wall consists in writing the line `$sender: message\verb` on the file `$receiver/wall.txt`. Your script must output the following messages depending on what happened:

- `nok: user '$sender' does not exist` if the `$sender` does not exist.
- `nok: user '$receiver' does not exist` if the `$receiver` does not exist.
- `nok: user '$sender' is not a friend of '$receiver'` if the two users (`$sender` and `$receiver`) are not friends.
- `ok: message posted!` otherwise.

Again, make sure to have only those outputs. You can obviously copy a lot of the code used in the previous scripts.

### 2.1.4 Displaying Walls

Finally, you need to add the possibility to display users' walls. Implement a script `display_wall.sh $id` that gets only one parameter (the user ID). The script needs to output the following depending on the context:

- `nok: user '$id' does not exist` if the user `$id` does not exist.
- otherwise, your script prints start of file, followed by the content of `$id`'s wall, followed by the line end of file.

```
$> ./display_wall.sh Balthasar
    start_of_file
    Abraham: Do you bite your thumb at us, sir?
    Sampson: I do bite my thumb, sir.
    Abraham: Do you bite your thumb at us, sir?
    Sampson: Is the law of our side, if I say ay?
    Gregory: No.
    Sampson: No, sir, I do not bite my thumb at you, sir, but I bite my thumb, sir.
    Gregory: Do you quarrel, sir?
    Abraham: Quarrel sir! no, sir.
    Sampson: If you do, sir, I am for you: I serve as good a man as you.
    end_of_file
```

### 2.1.5 Server Script

Now you will set up the server of your application. As a first step, your server will read commands from the prompt and will execute them.

Write a script `server.sh`. This script is composed of an endless loop. Every time the script enters the loop it reads a new command from the prompt. Every request follows the structure `req id [args]`. There are four different types of requests, corresponding to the four scripts you have written so far:

- `create $id`: which creates user id.
- `add $id $friend`: adds a friend to the user id.
- `post $sender $receiver message`: posts a message on receiver's wall.
- `display $id`: displays id's wall.

If the request does not have the right format, your script prints an error message: `nok: bad request`.
A good programmimng structure for your script could be the case one. Your script will look like that:

```
# do something
while true; do
    # do something
    case "$request" in
        create)
            # do something
            ;;
        add)
            # do something
            ;;
        post)
            # do something
            ;;
        display)
            # do something
            ;;
        *)
            echo "Accepted Commands: {create|add|post|display}"
            exit 1
done
```

# 3 General Instructions

- This assignment is to be completed in pairs.
- You are encouraged to collaborate with your peers on this project, but all written work must be your own. In particular we expect you to be able to explain every aspect of your solution if asked.
- **Both pair members must submit an archive** (zip or tar.gz) of your solution on Blackboard:
  1. Code/scripts
  2. README.txt file describing how to run your programs
  3. PDF report of your work (5 pages max, no need to include code in it).
- The report should include the following sections:
  1. A short introduction
  2. A section that describes the organisation of your system.
  3. A list of implemented features highlighting the contribution of each group member. If you wish, you could describe how you would implement the features you did not implement in words (it might get you some points).
  4. A description the different challenges you faced and your solutions. For instance, for each script, describe the difficulty you faced and your solution. These sections can be short–the objective here is to show how you crafted the solutions with the tools you have learned so far.
  5. A short conclusion
- Due Date: November $4^{th}$ 2022, at 5pm (**hard deadline!**). There will be no possible extensions as the solution for Assignment 1 will be release after the due date.
- Both members of the group must submit their project.

# 4 Note on Next Project/Assignment

Assignment 2 will be a continuation of Assignment 1. While the exact instructions for Assignment 2 will be posted at later stage, the general goal will be to:

- implement a client script in bash
- adapt the server to communicate with the client script
- make sure that the server is able to handle correctly and efficiently multiple clients at the same time

Therefore, to make your code from assignment 1 extendable, I would advise you to:

- write a clean and commented code
- divide your code into small scripts
- do not attempt to proposed advanced features at the moment as they might be incompatible with Assignment 2