

Cheat Codes

Follow the instructions below to avoid any headaches:

1. From the desktop, press `CTRL+ALT+T` on the keyboard to open a terminal.
2. In the terminal, type `git clone https://github.com/daytonschuh/CPSC449Project1`.
3. Next, type `cd CPSC449Project1` to change into working directory.
4. Next, type `bash autoinstall.sh` to install all necessary libraries.
5. Next, open 2 more terminals using `CTRL+SHIFT+N` twice.
6. Choose a terminal to dedicate to caddy, then type `ulimit -n 8192` and `caddy`.
7. Next, choose a different terminal to dedicate to foreman, then type `foreman start -m posts=3,accounts=3`
8. Finally, go to the last terminal and type `bash testuser.sh` to test users then `bash testpost.sh` to test posts.

Introduction

The goal of this runbook is to safeguard the user from a deep hatred of technology and early hair loss. We will attempt to set up and configure our system to run the project before we encounter issues. Hopefully, this preemptive approach will provide the necessary guidance to simplify the process.

This project is built solely for use in the Tuffix environment, and will only be explained in that capacity. It is not impossible to implement something similar on a different operating system, but we will not detail steps to do such. As a Linux based project, there are certain libraries and dependencies that must be installed for the program to work. We will cover all those details (and commands) very soon.

This project was developed with python in the PyCharm IDE using Flask and SQLAlchemy. It implements posting and user account microservices, such as creating a post or user and deleting a post or user. Additional functionalities of our implemented microservices will be discussed later.

Early development testing was done using Postman, but we have since moved to automated testing through curl commands in bash scripts. We are using Foreman to start all the processes associated with our project to better accommodate testing within our WSGI container, Unicorn.

Getting Started

From our desktop, we should be able to press `CTRL+ALT+T` to open a terminal. Once the terminal is open, we can begin installing all the libraries for the project.

This project uses a few different libraries to run. For simplicity, they've been alphabetically arranged in a table. Ensure that the following libraries and their dependencies are installed on your system:

Library/Dependency	Command
Caddy	<code>curl https://getcaddy.com bash -s personal</code>
Flask	<code>sudo pip3 install flask</code>
Flask-SQLAlchemy	<code>sudo pip3 install flask_sqlalchemy</code>
Flask-Marshmallow	<code>sudo pip3 install flask_marshmallow</code>
Foreman	<code>sudo gem install foreman</code>
Gunicorn3	<code>sudo apt install --yes gunicorn3</code>
Marshmallow-SQLAlchemy	<code>sudo pip3 install marshmallow-sqlalchemy</code>
Pytz	<code>sudo pip3 install pytz</code>

Setup and Run

To keep things clean and less confusing, it would be wise to create a new directory for our project. Using the command `mkdir CPSC449_DDM_1`, we can create a directory named `CPSC449_DDM_1`. After we've created our directory, we will need to navigate to that directory. Use the command `cd CPSC449_DDM_1` to enter our newly created directory (Figure 3).

Ensure that git is installed with `sudo apt-get install git`. We are ready to grab the project and can do so by running the command (Figure 4):

```
git clone https://github.com/daytonschuh/CPSC449Project1
```

This will create another directory with all the files from the project. Again, we will need to navigate into that directory by using the command `cd CPSC449Project1` (Figure 5). We are now in our working directory where all the magic will happen.

**Note: You may name the directory whatever you wish by changing `CPSC449_DDM_1` to something different. However, this will affect all following commands that use the directory name.*

Files

If we run the command `ls`, we will see all the files that are included in our project (Figure 6). The files that we should see in our project folder are:

- `app.py`
- `autoinstall.sh`
- `Caddyfile`
- `posts.db`

- Procfile
- README.md
- Runbook.pdf
- testpost.json
- testpost.sh
- testscript.sh
- testuser.json
- testuser.sh
- wsgi.py

If there are any missing or additional files before we proceed, there may have been edits to the github and this runbook is outdated. We will continue to our microservices and testing section to see if things are still working.

Microservices and Testing

Now we are ready to run a test. Press **CTRL+SHIFT+N** to open two more terminals in the same directory. Make sure that the directories are the same. If this does not work, we will have to open another terminal and navigate to the working directory.

It may be easiest to resize the terminals and place them side by side to watch the testing process. In the first terminal, run the command **foreman start -m posts=3,accounts=3**. This reads all the commands from our Procfile and names those instances according to their associated labels. In the second terminal, run the command **caddy**. This reads the information from our Caddyfile and starts it.

In our last terminal, we can test our user management API with the command **bash testuser.sh** (Figure 7). It will attempt to create a user, update the user's email, increment the user's karma, decrement the user's karma and delete the user. It uses our testuser.json file to supply the appropriate data for the test. Since testuser.sh is a bunch of curl commands, we will have to carefully read over the output. We should be able to find a success message for each test.

After confirming that each test for testuser.sh has run properly, we can use the command **bash testpost.sh** to test our post management API (Figure 8). It will attempt to create a post, retrieve a post, delete a post, get a specified number of most recent posts from a specified community and get a specified number of most recent posts from any community. It uses our testpost.json file to supply the appropriate data for the test. Again, we will have to carefully review the output for each success message.

Figures (Outdated)

Figure 1

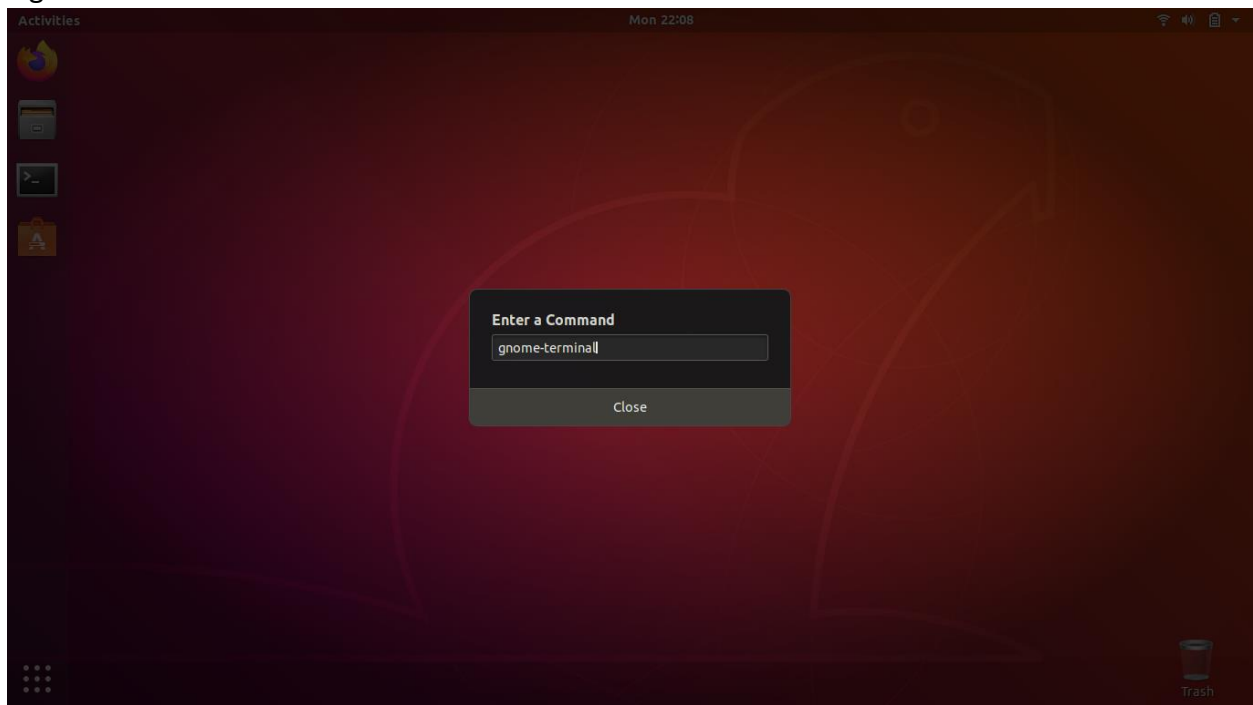


Figure 2

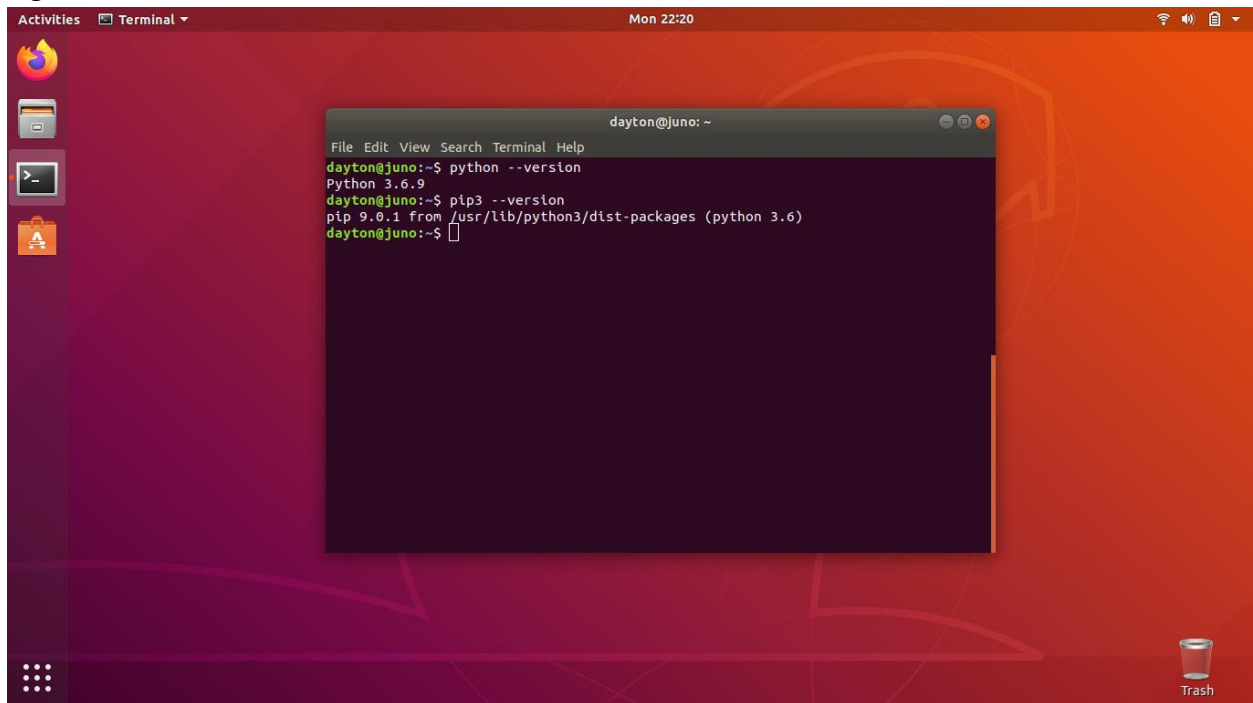


Figure 3

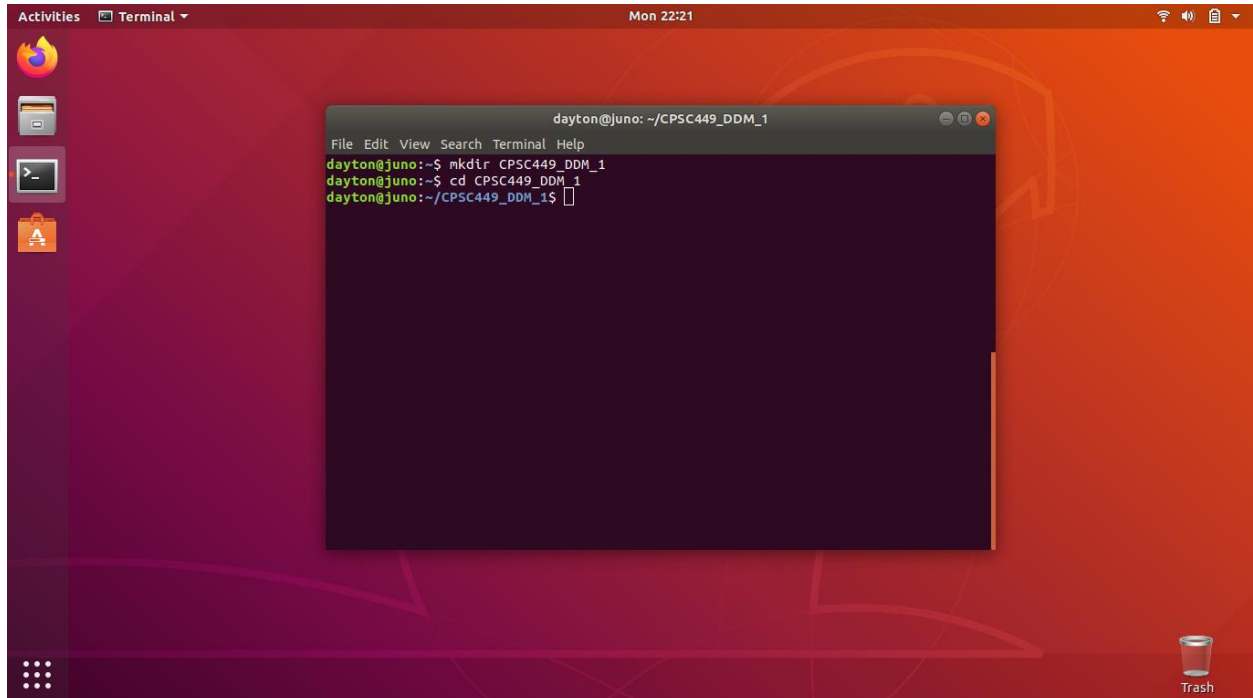


Figure 4

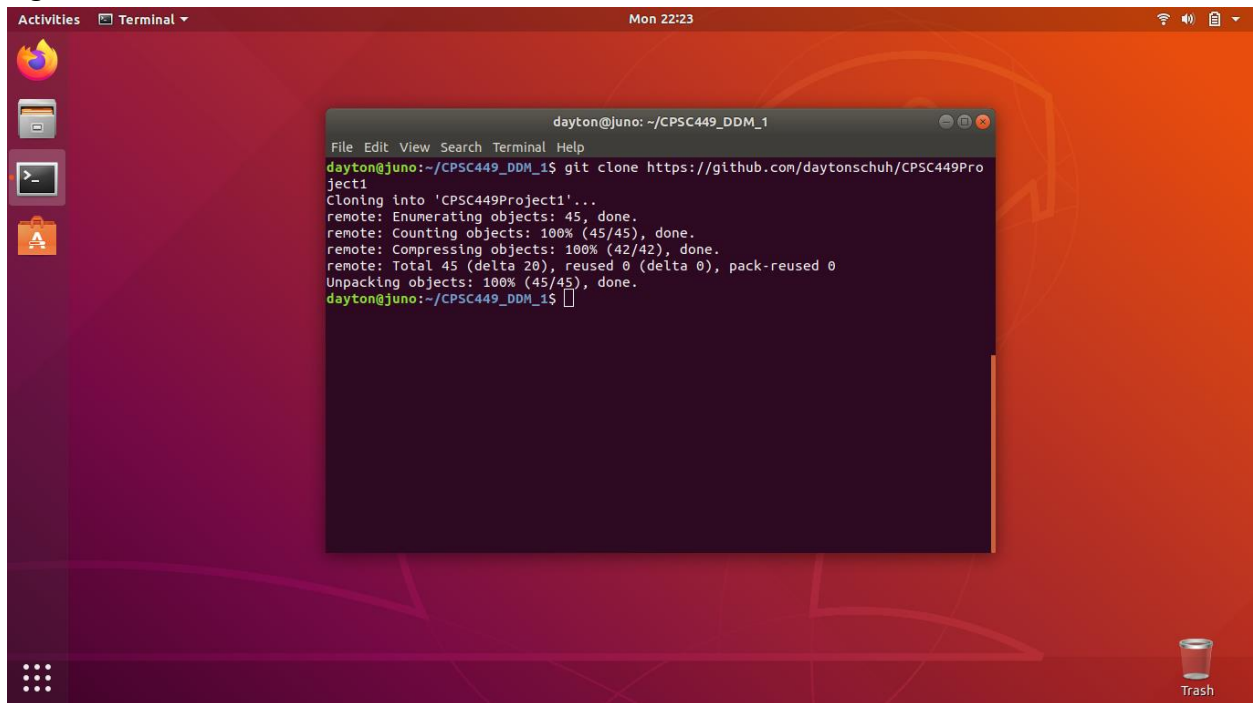


Figure 5

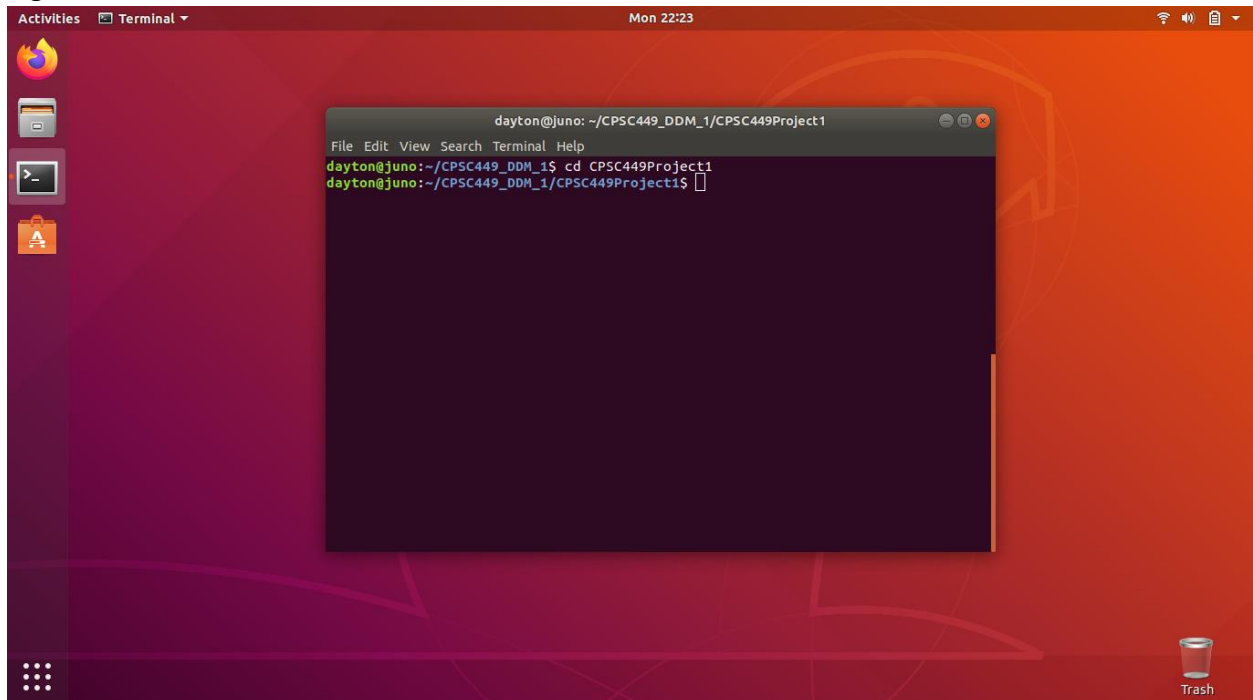


Figure 6

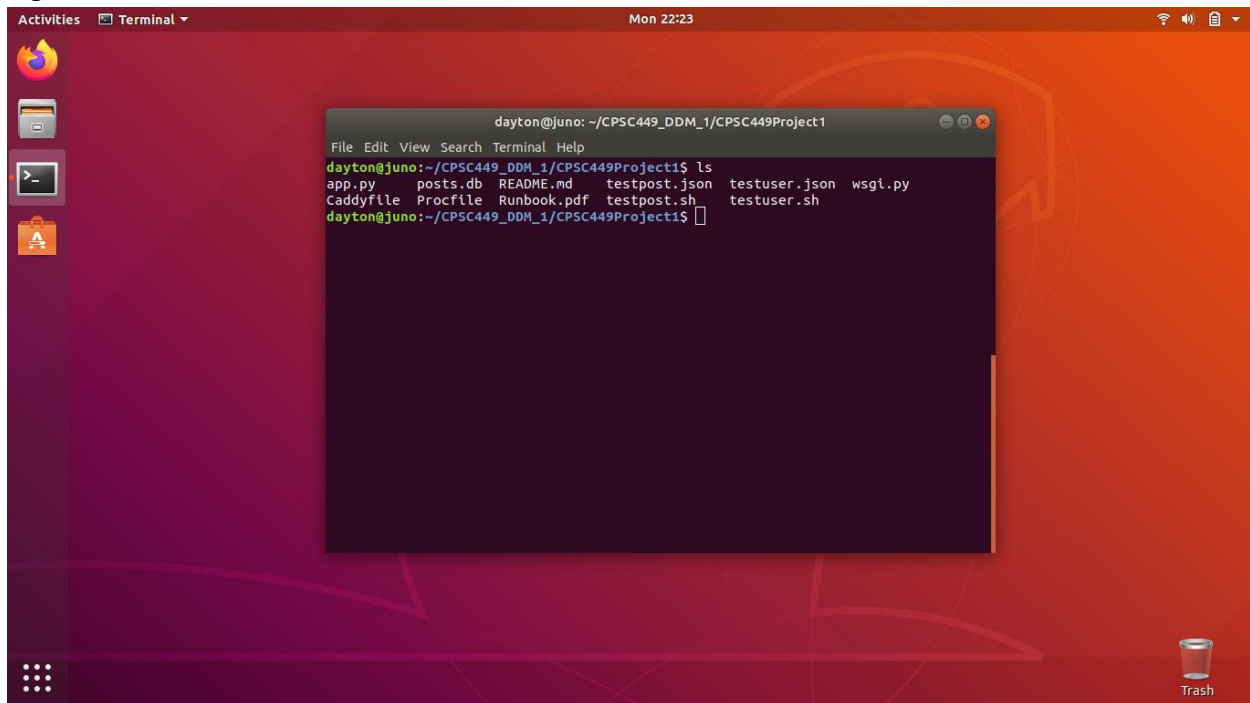


Figure 7

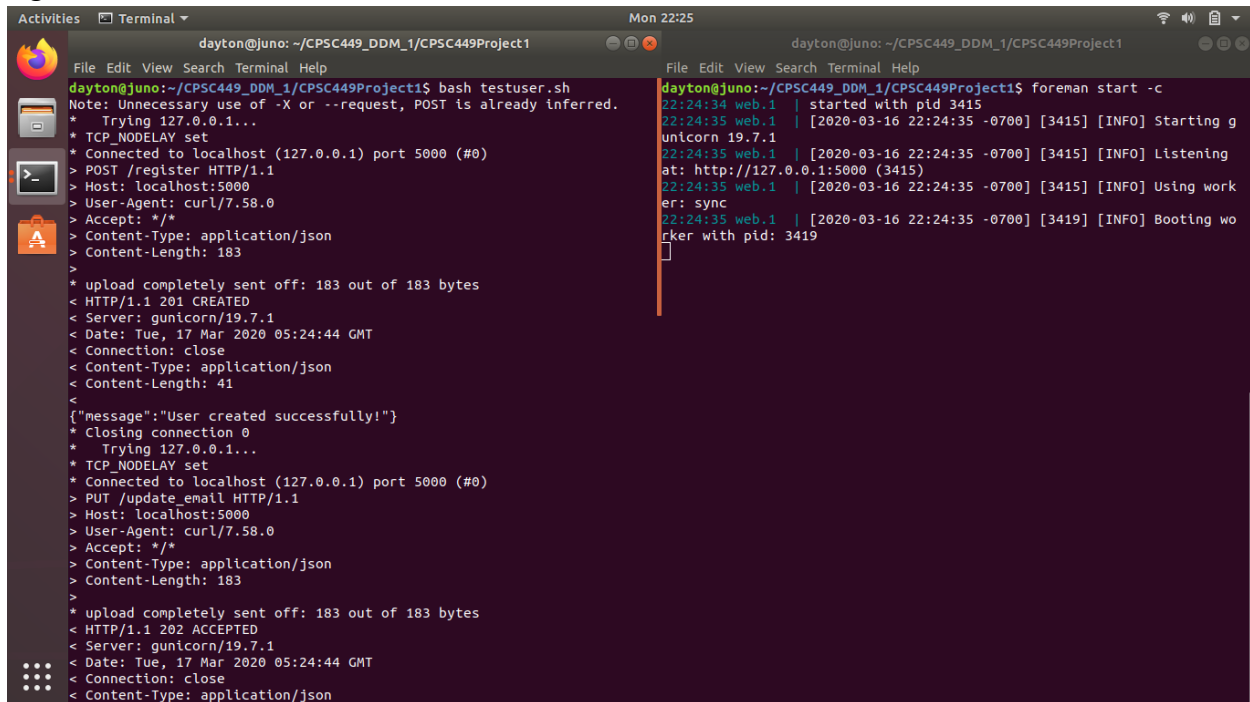
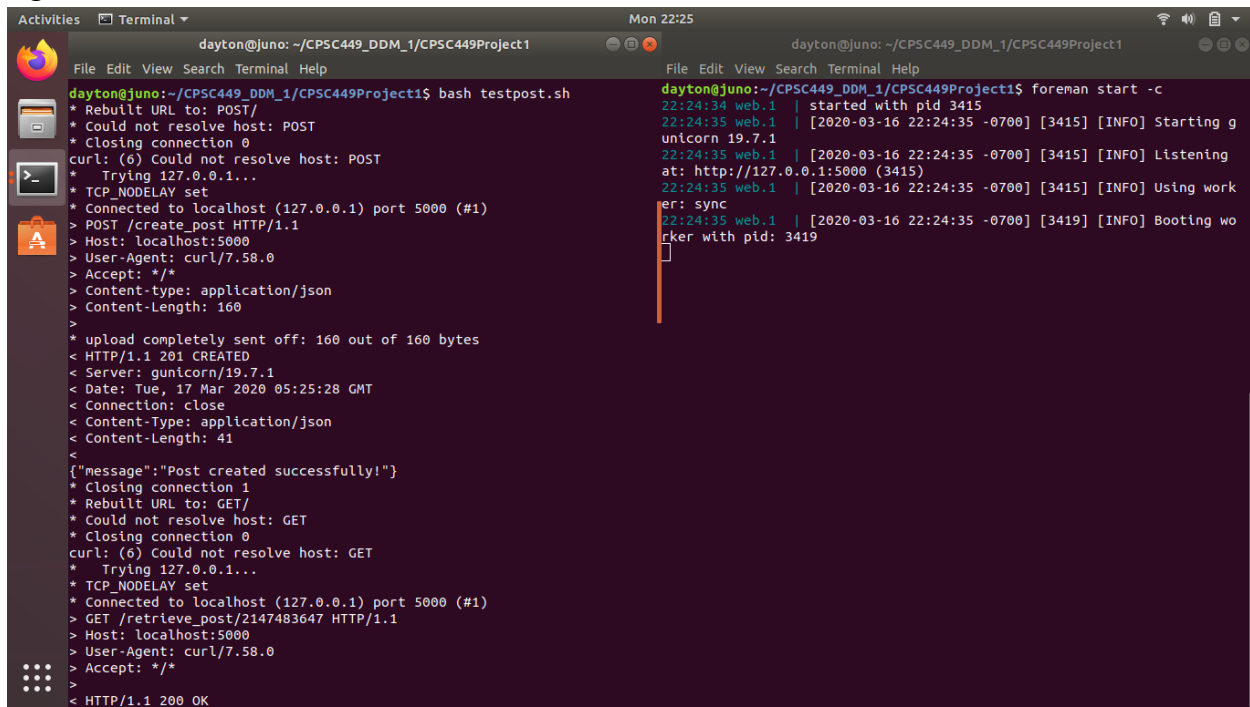


Figure 8



The image shows two terminal windows from a Linux desktop environment. The left window displays the output of a script named 'testpost.sh', which performs a POST request to a local server on port 5000 and then a GET request to retrieve the created post. The right window shows the output of the 'foreman start -c' command, displaying logs for the Unicorn web server and a worker process.

```
dayton@juno: ~/CPSC449_DDM_1/CPSC449Project1
File Edit View Search Terminal Help
dayton@juno:~/CPSC449_DDM_1/CPSC449Project1$ bash testpost.sh
* Rebuilt URL to: POST/
* Could not resolve host: POST
* Closing connection 0
curl: (6) Could not resolve host: POST
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#1)
> POST /create_post HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> Content-type: application/json
> Content-Length: 160
>
* upload completely sent off: 160 out of 160 bytes
< HTTP/1.1 201 CREATED
< Server: unicorn/19.7.1
< Date: Tue, 17 Mar 2020 05:25:28 GMT
< Connection: close
< Content-Type: application/json
< Content-Length: 41
<
{"message":"Post created successfully!"}
* Closing connection 1
* Rebuilt URL to: GET/
* Could not resolve host: GET
* Closing connection 0
curl: (6) Could not resolve host: GET
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#1)
> GET /retrieve_post/2147483647 HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
```

```
dayton@juno:~/CPSC449_DDM_1/CPSC449Project1$ foreman start -c
22:24:34 web.1 | started with pid 3415
22:24:35 web.1 | [2020-03-16 22:24:35 -0700] [3415] [INFO] Starting g
unicorn 19.7.1
22:24:35 web.1 | [2020-03-16 22:24:35 -0700] [3415] [INFO] Listening
at: http://127.0.0.1:5000 (3415)
22:24:35 web.1 | [2020-03-16 22:24:35 -0700] [3415] [INFO] Using work
er: sync
22:24:35 web.1 | [2020-03-16 22:24:35 -0700] [3419] [INFO] Booting wo
rker with pid: 3419
```