George Vogas, Sriiman Visuvanathar, Jia-Yu Joy Ho

Abstract

This project is an interactive Unix shell script that helps administrators monitor and manage the system. Through a command-based interface, users can check system status, schedule backups, manage networks, services, users and files. All features and commands are documented and the full project is available in the GitHub repository

UNIX FINAL PROJECT

Table of Contents

[Project Overview 1](#_Toc215952271)

[Project Graph 1](#_Toc215952272)

[Project Requirements and Deliverables 3](#_Toc215952273)

[Task Assignment 4](#_Toc215952274)

[Project Components and Solutions 4](#_Toc215952275)

[Detailed Descriptions 5](#_Toc215952276)

[System Status: 5](#_Toc215952277)

[Backup Management 8](#_Toc215952278)

[Network Management: 10](#_Toc215952279)

[**Service Management:** 12](#_Toc215952280)

[**User Management** 14](#_Toc215952281)

[File Management 19](#_Toc215952282)

## Project Overview

This project allows the user to control and view vital information about the Linux computer system, specifically Ubuntu. Its purpose is to mimic a fully functional menu of essential commands, just like Windows's or Linux's Settings page. The precise objectives are the following:

* Allowing users to modify and view users, files, and network-related information and variables.
* Allowing users to create a backup for a specific file successfully.
* Allowing users also to view important information related to the system’s CPU, Memory Usage, and even System Processes that are running in the background
* Overall, all of this sums up to what we have learned in the Unix class. It is a clever combination of all the commands we have learned, tested through a single main menu we have created.

### Project Graph

MainMenu

├─ **SystemStatus**

│  ├─ getMemoryInfo

│  ├─ temperatureCheck

│  ├─ listActive

│  ├─ killProcess

│  ├─ MainMenu        (option 5: return)

│  └─ exit            (option 6)

│

├─ **BackupManagement**

│  ├─ backupSchedule

│  │  ├─ mkdir -p backup

│  │  ├─ mkdir -p "$dest"

│  │  ├─ readlink -f (filename, dest)

│  │  └─ crontab      (adds backup job)

│  ├─ lastBackup

│  ├─ MainMenu        (option 3: return)

│  └─ exit            (option 4)

│

├─ **NetworkManagement**

│  ├─ displayInterfaces

│  │  └─ ip -o -4 addr show <interface>

│  ├─ toggleInterfaces

│  │  └─ ip link set <interface> up/down

│  ├─ setIPAddress

│  │  └─ ip addr add <ip> dev <interface>

│  ├─ listWifi

│  │  └─ nmcli dev wifi / connect

│  ├─ MainMenu        (option 5: return)

│  └─ exit            (option 6)

│

├─ **Server\_Management**

│  ├─ **display**

│  │  └─ systemctl list-units --type=service --state=running

│  ├─ startStop

│  │  ├─ **confirmation**

│  │  │  └─ systemctl status <service>

│  │  └─ systemctl start/stop <service>

│  ├─ MainMenu        (option 3: return)

│  └─ **exit**            (option 4)

│

├─ **UserManagement**

│  ├─ **createUser**

│  │  └─ useradd / passwd

│  ├─ **rootPerm**

│  │  └─ usermod -aG sudo <user>

│  ├─ **deleteUser**

│  │  └─ userdel <user>

│  ├─ **displayUsers**

│  │  └─ who

│  ├─ **killRemote**

│  │  ├─ who | grep pts

│  │  └─ pkill -HUP -t <pts>

│  ├─ groupsUser

│  │  └─ groups <user>

│  ├─ **groupSudo**

│  │  ├─ **addGroup**

│  │  │  ├─ usermod -aG <group> <user>

│  │  │  └─ getent group

│  │  ├─ **removeGroup**

│  │  │  ├─ groups <user>

│  │  │  ├─ getent group

│  │  │  └─ gpasswd -d <user> <group>

│  │  ├─ **UserManagement**   (back to user menu)

│  │  ├─ MainMenu         (back to main menu)

│  │  └─ **exit**             (option 5)

│  ├─ **MainMenu**            (option 8: return)

│  └─ **exit**                (option 9)

│

└─ **FileManagement**

├─ **userAndFileCheck**

│  ├─ getent passwd <user> (home dir)

│  └─ find <home> -name <file>

├─ display\_large

│  └─ ls -lSh <home> | head -n 10

├─ **display\_old**

│  └─ ls -ltr <home> | head -n 10

├─ MainMenu              (option 4: return)

└─ **exit**                  (option 5)

## Project Requirements and Deliverables

System Status: We needed to display memory usage and CPU temperature, list all active systems, and allow users to terminate processes. To do this, we had to rely on many Linux commands we learned in class, which made the task less challenging. But finding the CPU temperature was by far the most difficult task in this section, right before the backup process.

Backup Management: We had to ask users for a lot of info, like the date, time, file name, destination, and then create the backup for their set date. We also had to log this in a file called MyLog. This was a bit more challenging than System Status because we were introduced to new concepts that we didn’t spend much time on. We had to rely on the textbook and online articles that provided backup tutorials.

Network Management: We had to display the network cards, IP addresses, and default gateways of the system, allow the ability to enable or display a specific network card, and to also set an IP address for any network cards, and finally, we had to create a system that allows us to connect to a nearby network. All of these were not challenging as it was the last thing we learned in class, which made our heads very clear about the subject itself.

Service Management: We had to display a list of all running services and offer the user the option to start and stop them. This also required confirmation afterward to ensure the commands indeed worked.

User Management: We were supposed to display the active users in the system and the list of groups that a user belongs to. We were also supposed to create functions that allow the user to create another one, to delete one, disconnect them if they are a remote user, and also change their group membership, so remove them from a group or add them to a group. Most of these tasks were easy, as the code was just a few lines, but it got a bit challenging when we had to disconnect a remote user because we simply did not remember learning that in class.

File Management: We had to display a specific file that the user wanted to know the path of, and to do this, we had to confirm the identity of the user and then check if that file existed to display the path. In addition, we also had to create two commands to show the 10 largest files in the same directory, and also the 10 oldest. These two were straightforward, like most of the tasks in the other category, using base commands like grep, head, ls, and awk.

## Task Assignment

|  |  |  |
| --- | --- | --- |
| **George Vogas** | **Sriiman Visuvanathar** | **Jia-Yu Joy Ho** |
| Service Management | System Status | Backup Management |
| User Management | File Management | Network |
| File Management | User Management | Documentation |
| Documentation | Documentation |  |

## Project Components and Solutions

The project uses a single main file that contains the functions needed to run. Here are the functions that make the program run:

**Main Menu:**

* Displays the program's main options.

**System Status:**

* Displays detailed information about memory usage.
* Checks the CPU temperature.
* Lists all active system processes.
* Let the user stop or end a specific process.

**Backup:**

* Creates a backup schedule.
* Shows the most recent backup process.

**Network**

* Shows network cards, IP addresses, and default gateways.
* Enables or disables a network card.
* Sets an IP address for a network card.
* Connect to a nearby wifi network.

**Services:**

* View the current services.
* Starts or stops a service.

**User Management:**

* Adds a user.
* Grants root permission to a user.
* Deletes a user.
* Shows active users.
* Disconnects a remote user.
* Lists all groups a user belongs to.
* Changes a user's group membership.

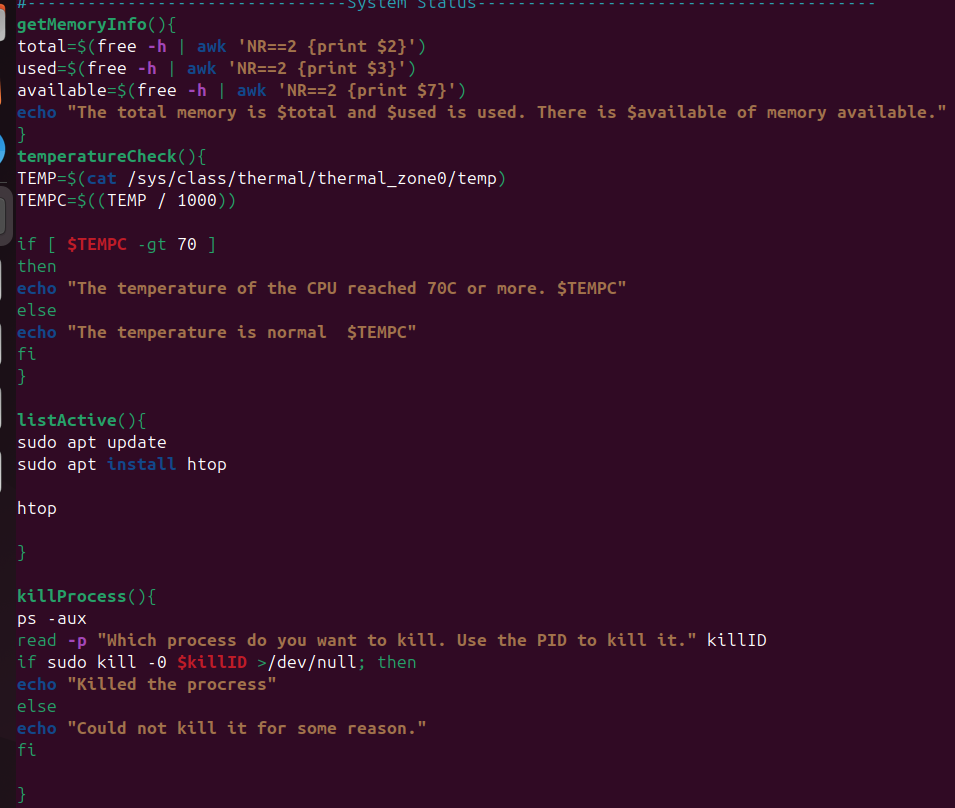
**File Management:**

* Path to a file in the user's home directory
* Shows the 10 largest files in the user's home directory.
* Shows the 10 oldest files in the user's home directory.

## Detailed Descriptions

### System Status:

* In this section, you can display detailed information about the memory usage, check the CPU temperature and also set an alert if it reaches a certain celsius, list all the active system processes and also let the user stop/end a specific process.
* Overall, this section has four different methods. All reached through the Main Menu, clicking 1.

 **Figure** 1**: Full code of the System Status program.**

**getMemoryInfo:**

In this method, I am using the free command to show general information about memory usage. I am then filtering it by using AWK to get only the ones I need, which is the total GB used,the available GB, and the total GB.

An echo is used to display the info to the member.

**Figure** 2**: Output of the command. Memory Information successfully showed.**

**TemperatureCheck:**

In this method, I am using two variables, TEMP and TEMPC. We can find the temperature of the CPU by using cat to access the file that has it. However, it is not in celsius, it is in milli-celius so you have to divide it by a thousand, according to Google, to make it into Celsius. It then checks if the temperature in celsius is more than 70 and if it is it will say it is more than 70. And if its not, then it will just say the normal temperature of the CPU.To display the message, once again, and like always, we are using echo. Below is the example of output.

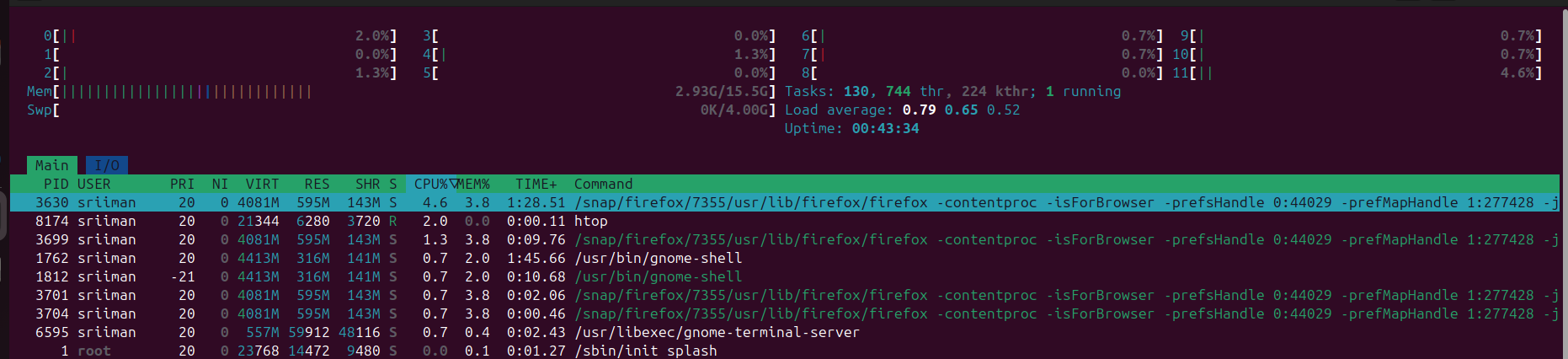


**Figure** 3**: Output of the command. Temperature is correctly shown.**

And well, my CPU wasn’t hot til 70 Celcius so it didn't show the other message but it just says that it's over 70.

**listActive:**

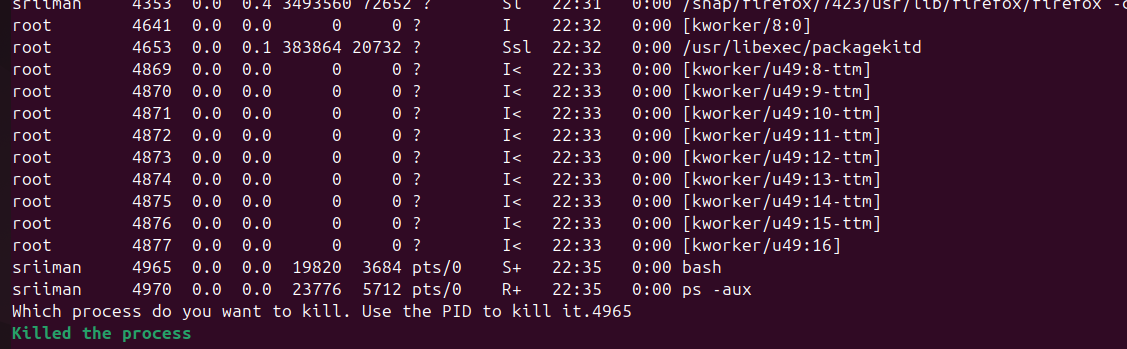
This method simply returns a list full of active system processes. It uses the htop command to show them. But because it is not installed in all systems of Linux, when you run the command it is going to sudo apt update and install htop. And then it will display the command. This is actually displayed by the htop command itself, there is no echo included.

**Figure** 4**: Output of the command. All active processes are shown. Memory information is an add-on for this command.**

**killProcess:**

This method is used to terminate any processes that are currently running. I used kill and also ps aux to show all of the processes with their PID.

When entered, it will ask the user to choose a PID to kill and they must write the number. It will then check if it did work or not, if it didn’t work then it is probably non-existent or the task is a vital one needed for the system to continue running. If it worked then it would just send an echo message saying it killed the process. The second column is the PID so in my case I chose the 6603, and then it just killed it using kill -9 killID.

**Figure** 5**: Output of the command. Successfully killed a process using PID.**

### Backup Management

This section is accessed from the Main Menu by entering **2.**

Once in it, the user will be prompted to choose between one of these four options:

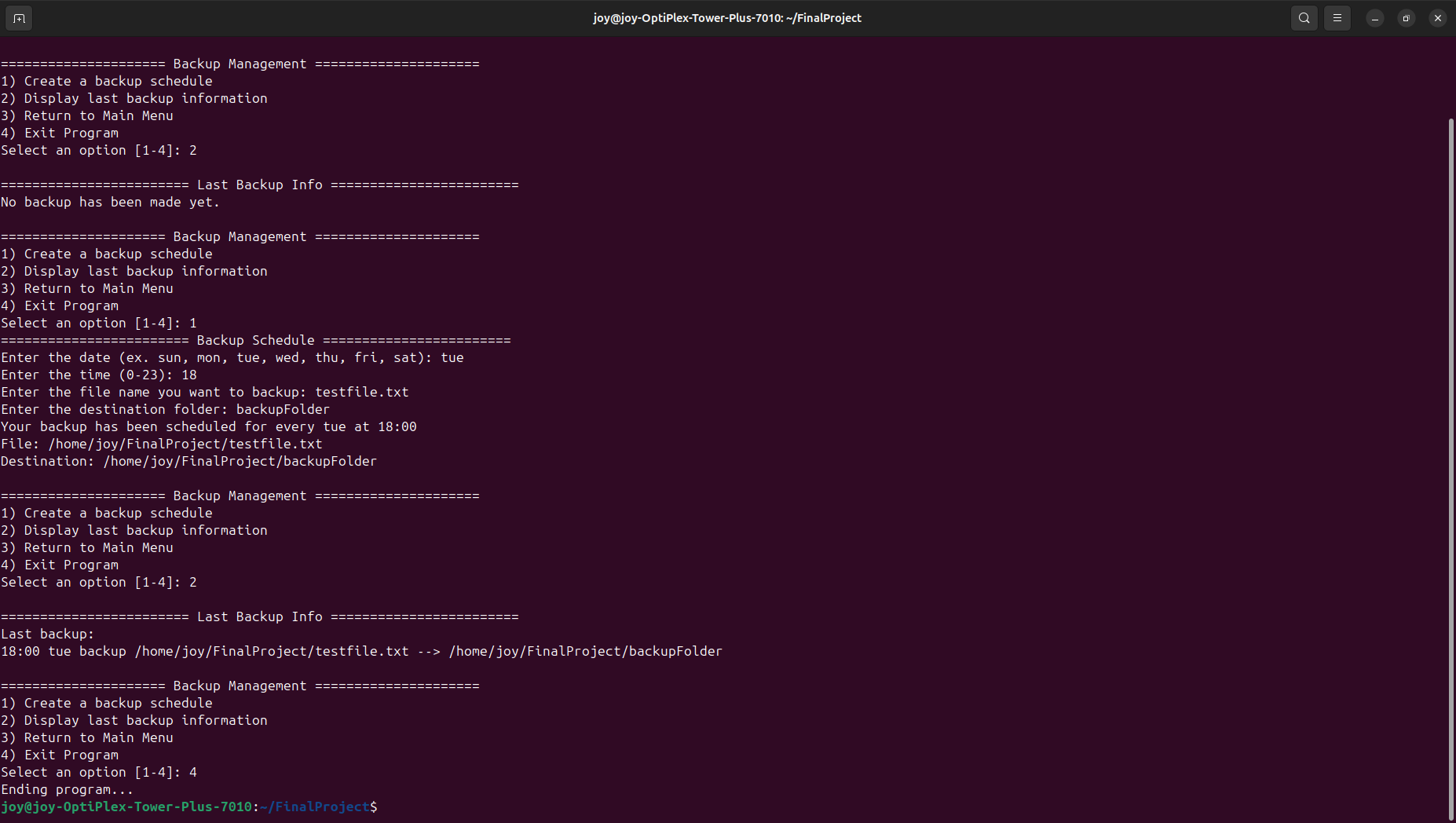


Figure 1: Backup management menu

The program uses a case statement to call the appropriate method.

1. **Create a backup schedule**

The **backupSchedule** method will ask the user to enter a day of the week (e.g. tue), an hour at which they want the backup to be done, the name of the file they want to backup and finally the name of the folder they want the backup file (.tar) to be stored in.

Each inputted value will be stored in a variable which will later be used in the cron command to automate the backup.

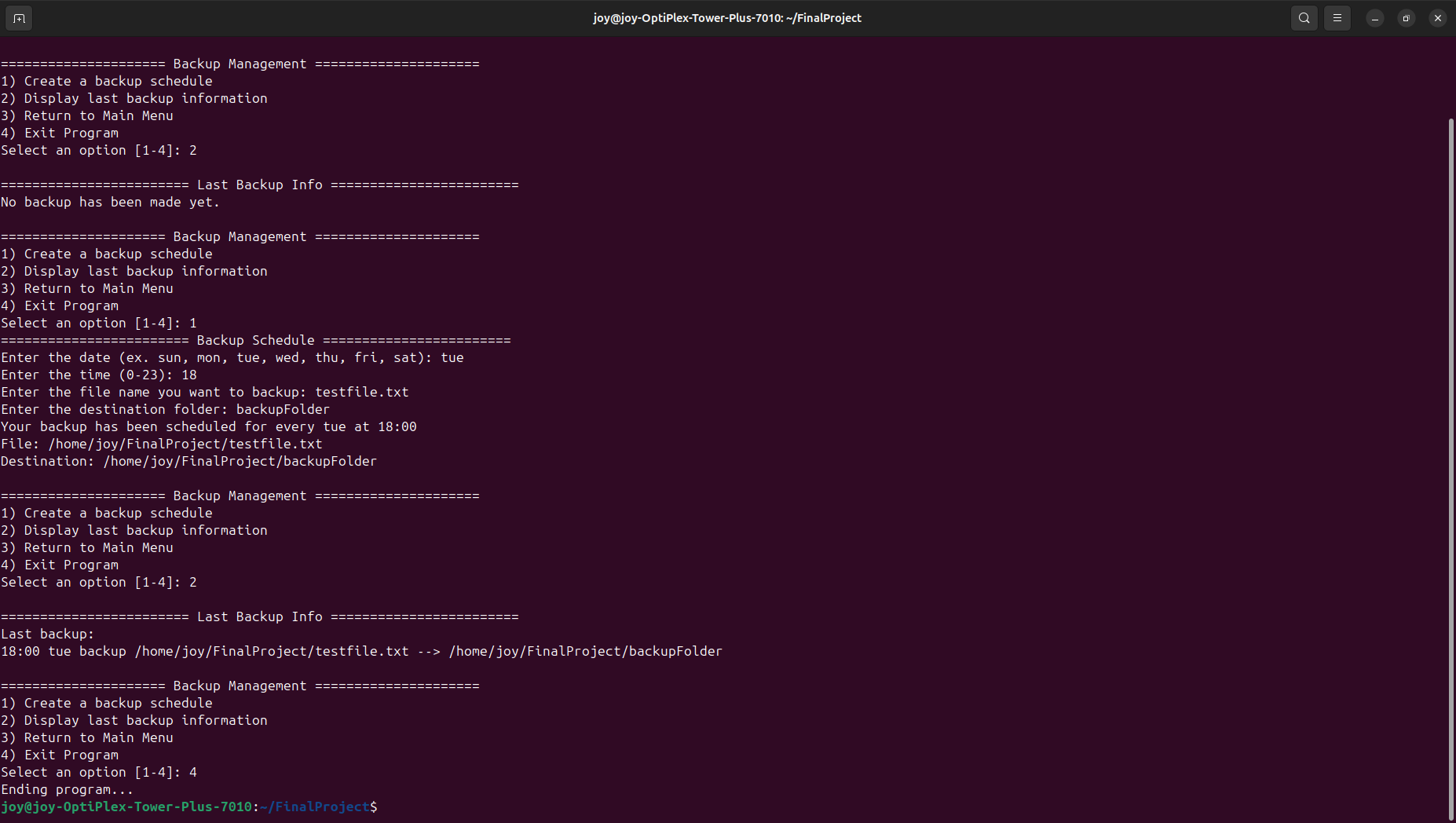


Figure 2: Prompts and example of inputs for backup schedule

Since the cron command uses numbers to interpret days of the week, I used a case statement to convert the string input (“sun”, “mon”, “tue”, etc) to numbers (0-6).

To make sure that the cron command can locate the file we want to backup as well as the destination folder, I used the **readlink** command to turn the inputted names into absolute paths.

When the backup is done, a .tar file will appear inside the designated folder:



Figure 3: Successfully created a backup file automatically

Once the backup has been scheduled, the program will log the information into *lastBackup.txt.*

1. **Display last backup information**

When the user selects the second option, **lastBackup** function will execute.

This function simply outputs the content of *lastBackup.txt* using **cat** command.

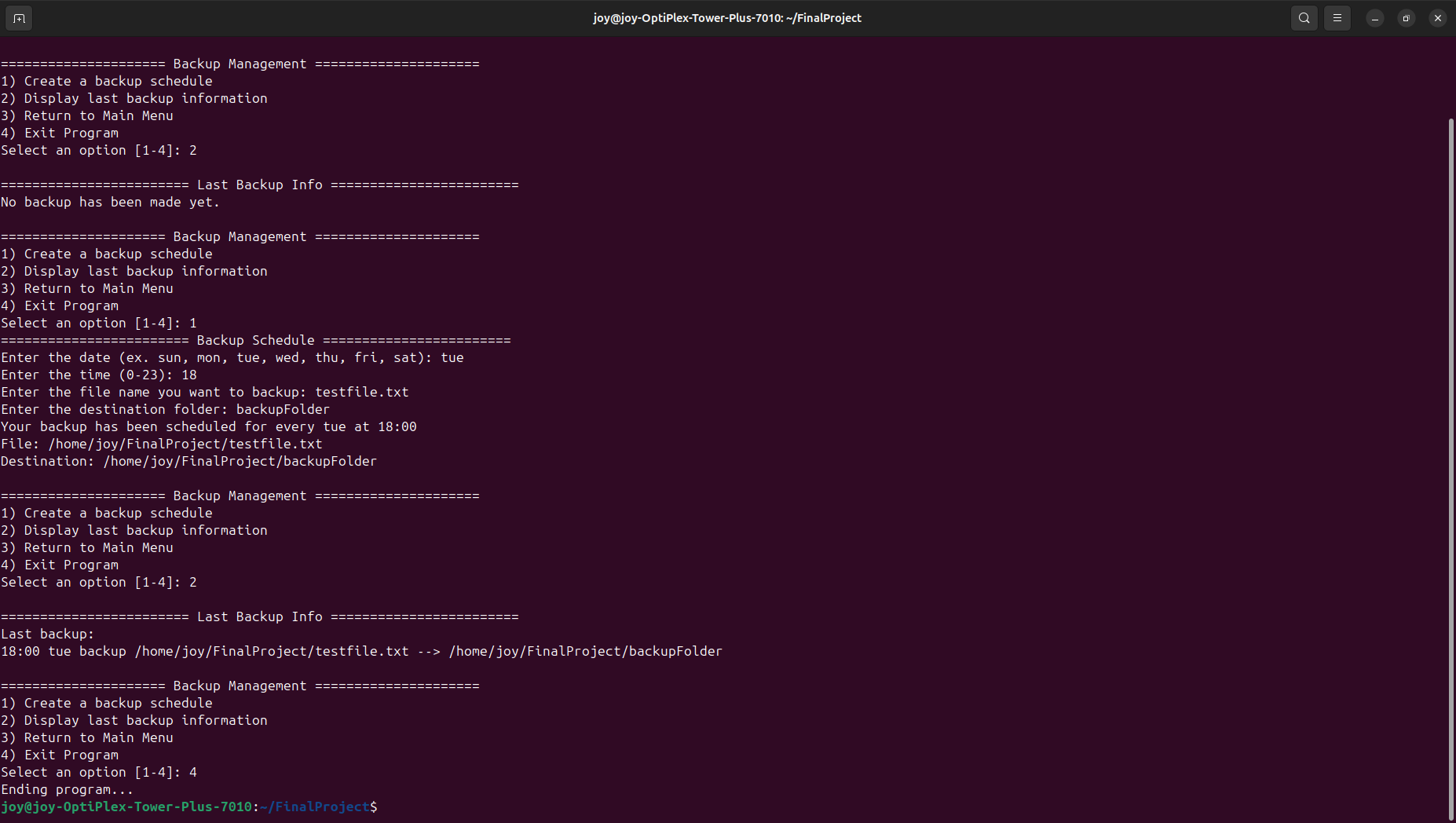


Figure 4: Displaying last backup information

If *lastBackup.txt* is empty because no backup was previously made, this message will show up instead:

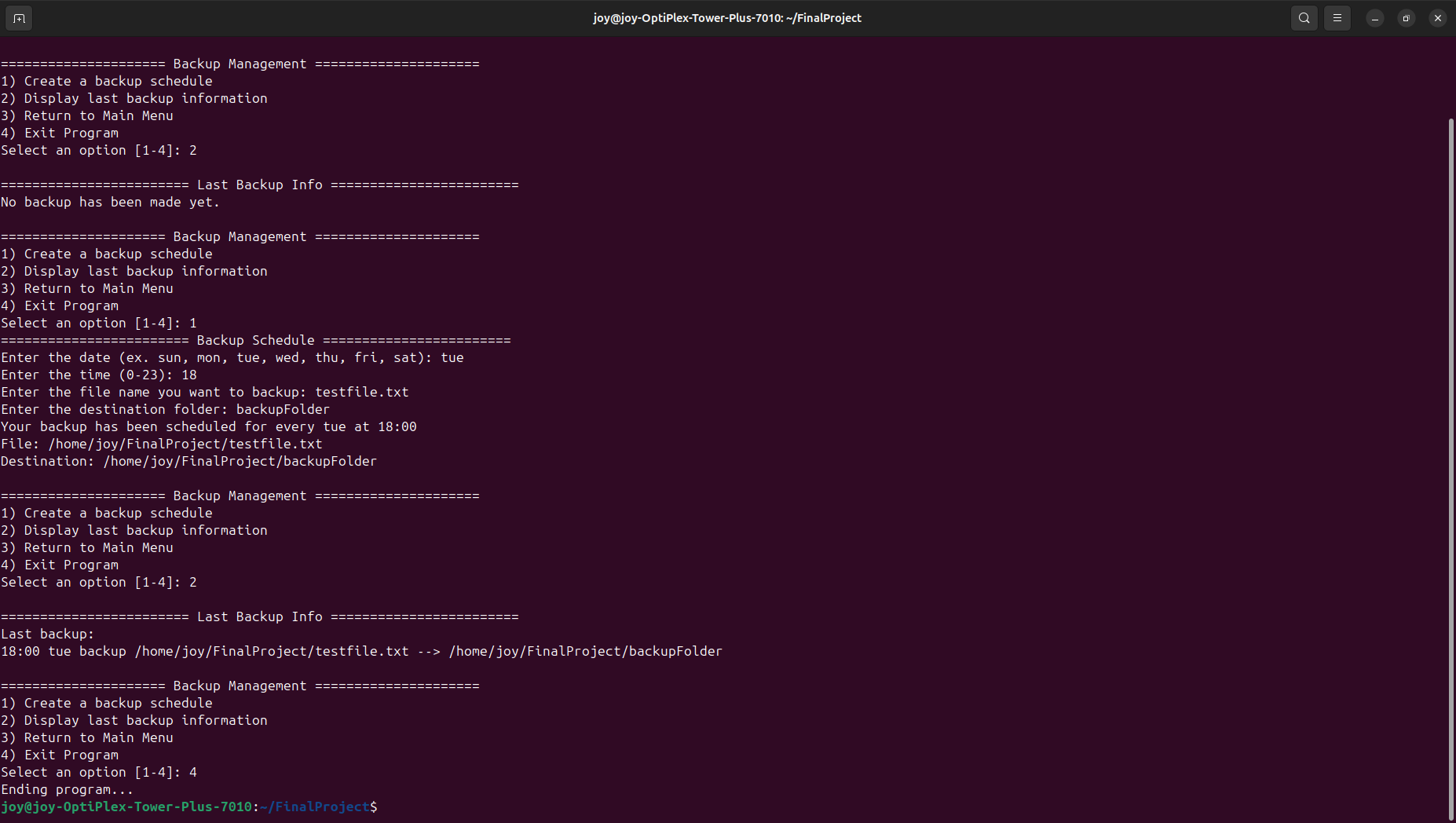


Figure 5: Displaying no last backup information

### Network Management:

This section is accessed from the Main Menu by entering 3**.**

Once in it, the user will be prompted to choose between one of these six options:

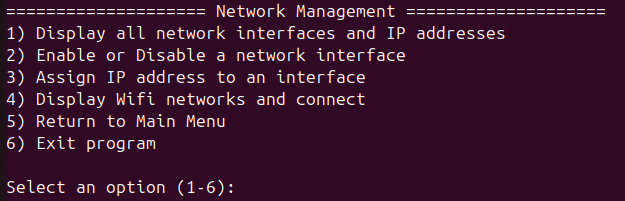


Figure 1: Network management menu

1. **Display all network interfaces and IP addresses**

The **displayInterfaces** function uses a for loop to go through all available interfaces, and uses **ip -o -4 addr show $interface | awk '{print $4}'** to get its IP address.

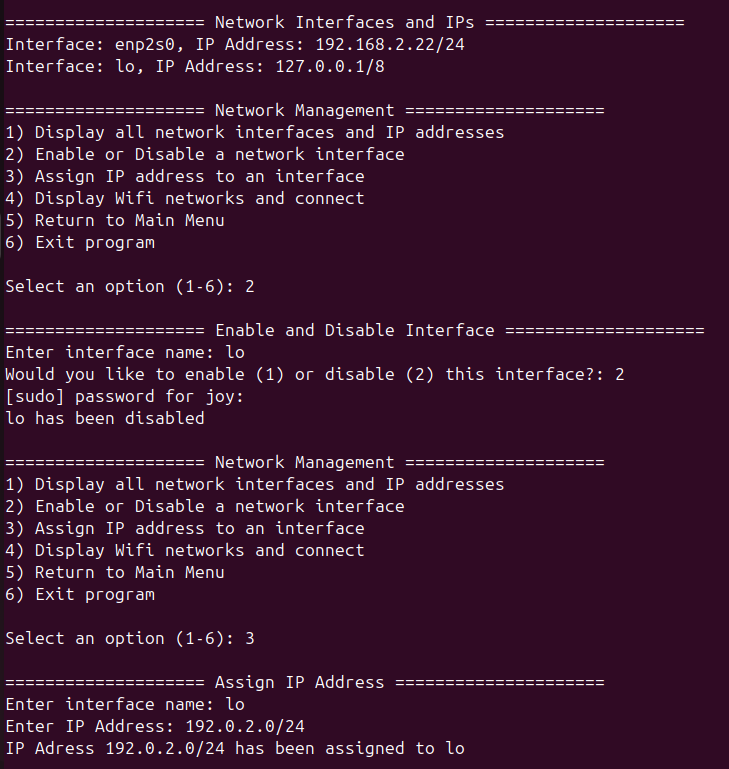


Figure 2: Displaying all interfaces and IP addresses

1. **Enable or Disable a network interface**

The **toggleInterfaces** function asks the user to input which interface they’d like to toggle, and then asks which action to perform (enable/disable) using a case statement. If the user chooses to enable it, the function uses **sudo ip link set $interface up**, otherwise it uses **sudo ip link set $interface down**

Since that command requires sudo, the user will be prompted to enter the password first.

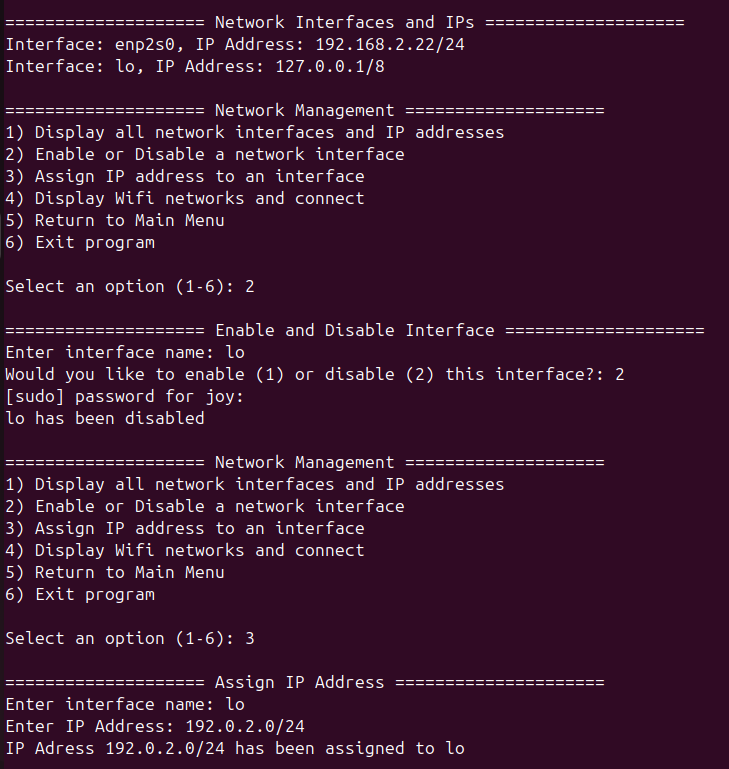


Figure 3: Successfully disabled

1. **Assign IP address to an interface**

The **setIPAddress** function prompts the user to enter the name of the interface and the IP address they want to assign. Those inputs will be stored in variables to run this command **sudo ip addr add $ip\_num dev $interface**

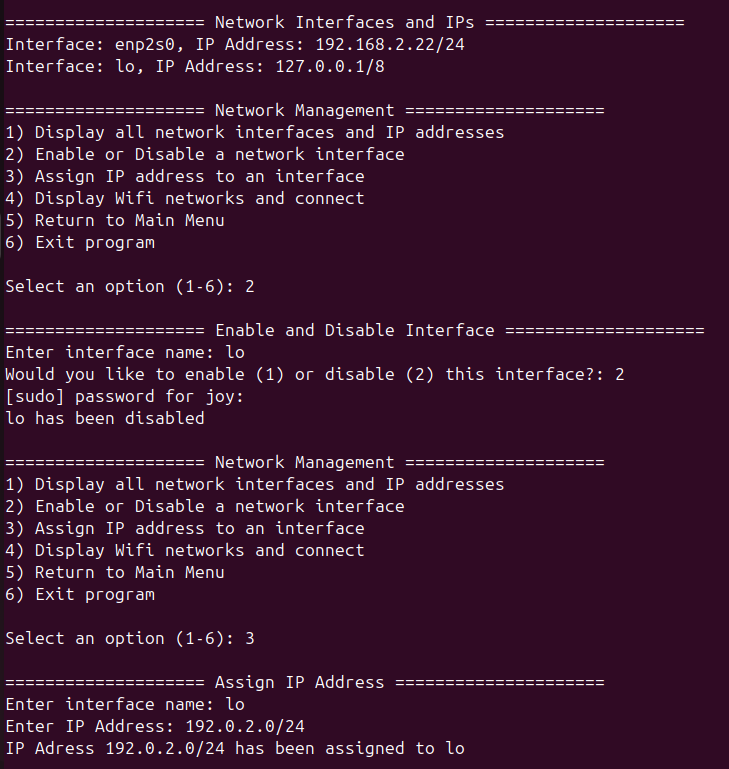


Figure 4: Successfully started a service

1. **Display Wifi networks and connect**

The **listWifi** function will first display all of the available Wifi networks using **sudo nmcli dev wifi**. The computer used in this demonstration only has a wired connection which is why no wifi networks was displayed.

Once everything is displayed, the user will be asked to enter the name of the Wifi they would like to connect to, as well as its password.



Figure 5: Successfully started a service

### **Service Management:**

* In this section, you can see all running services and choose to stop or start any of them.
* This section uses four different methods.
* This section is accessed from the Main Menu by entering 4.

This method asks the user to choose one of four options, then uses a case statement to call the right method.

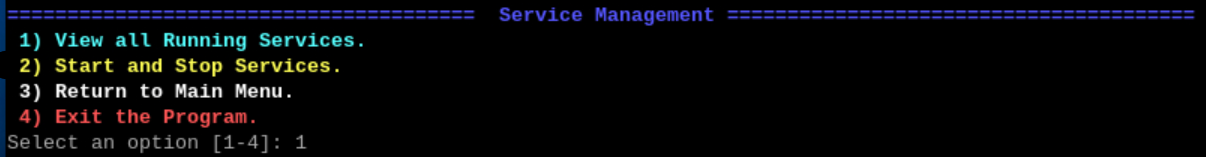


Figure 1: Service Management Menu Output

#### **Display:**

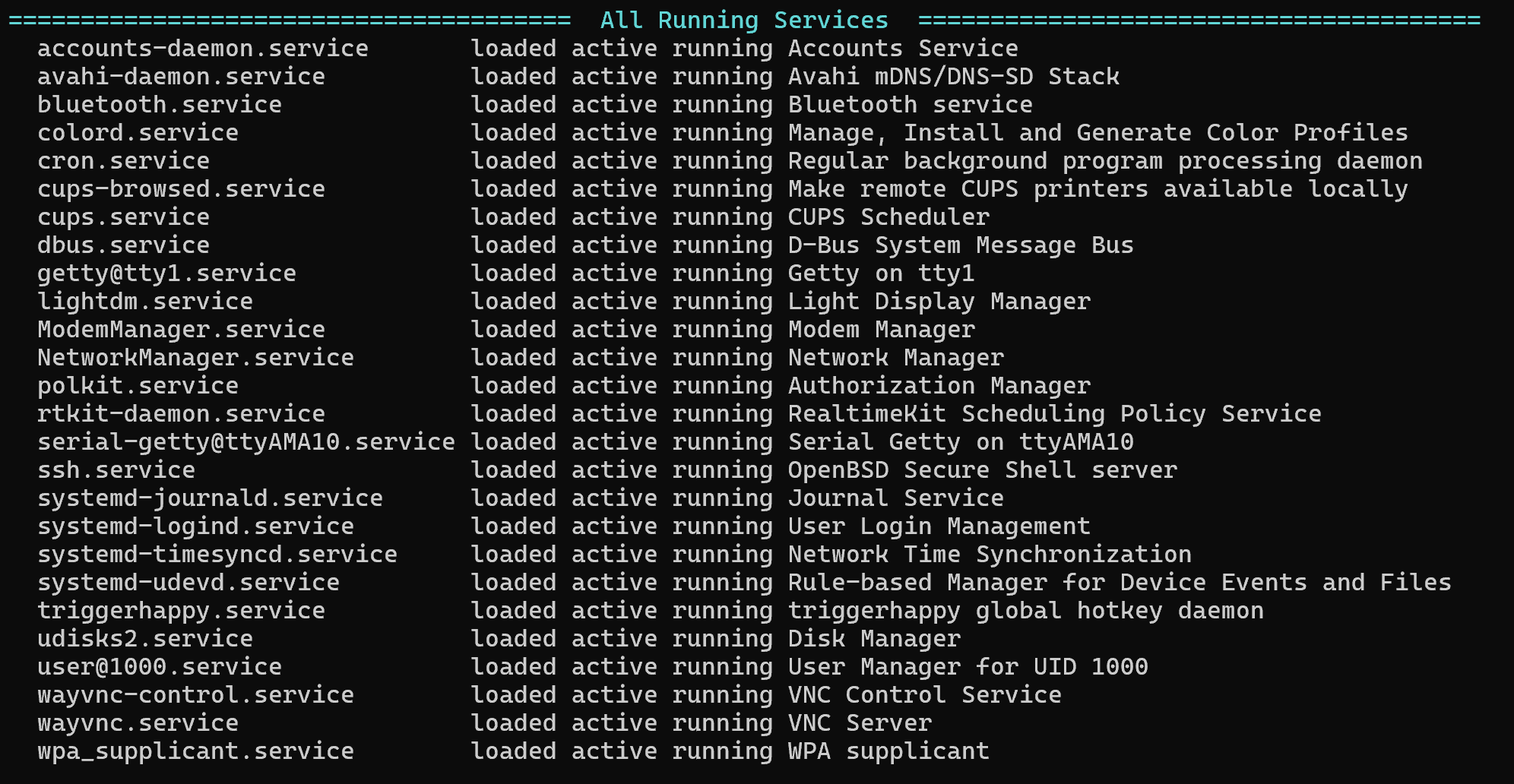
I use systemctl list-units --type=service --state=running to show all the running services.

Figure 2: Output of all runnig services

#### **StartStop:**

This method asks the user if they want to start or stop a service, then prompts for the service name after confirming the choice.

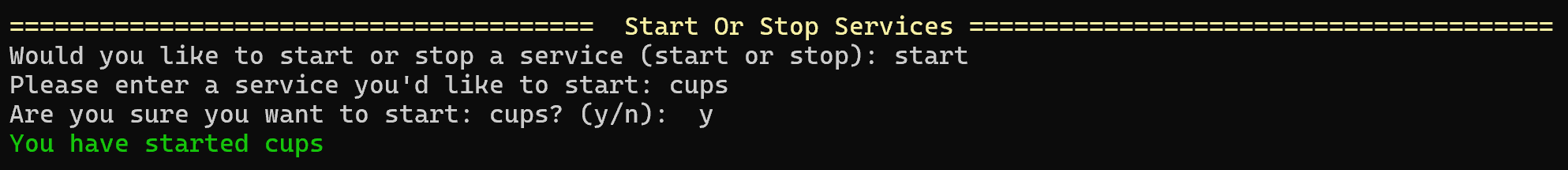
The function first asks whether the administrator wants to start or stop a service, checks whether the user entered start or stop, and then prompts for the service name. It checks whether the service actually exists using systemctl show ‘service’ &>/dev/null. If it is valid, the script asks for confirmation. Based on the choice, it either starts the service with systemctl start or stops it with systemctl stop, and then calls the confirmation function. If the service does not exist or invalid input is given, it prints an error message.

Figure 3: Successfully started a service

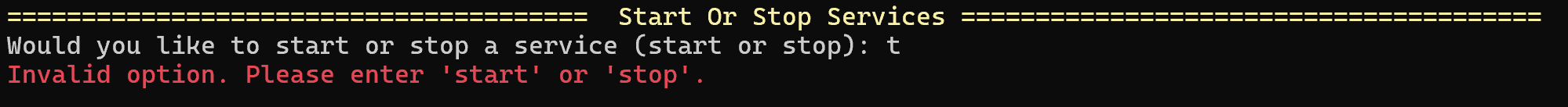


Figure 4: Invaild option

#### **Confirmation:**

* This helper method checks the status of the service given as a parameter.

Using check=$(systemctl status $1 | awk 'NR==3 {print $2}'). This command gets the status, and then I check if it is active.

### **User Management**

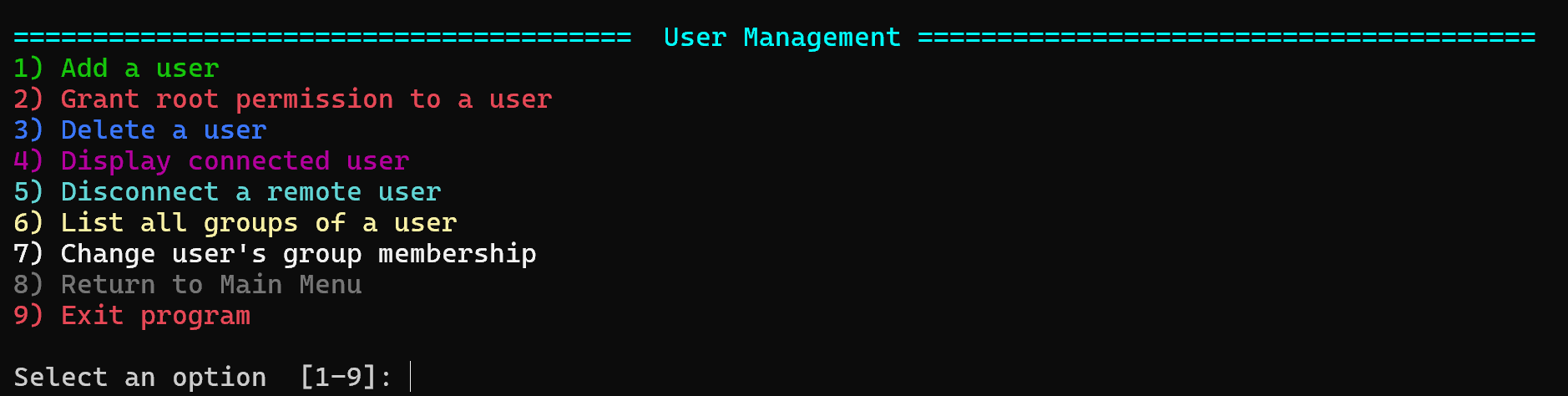
The function shows a menu for managing system users. Options include creating a new user, granting root permissions, deleting a user, listing connected users, disconnecting a remote user, listing a user's groups, or changing a user's group membership. You can also return to the main menu or exit. After an option is selected, a case statement calls the right function. Invalid input gets an error message, and the menu repeats until you exit. 

Figure 1: User Management Menu output

#### Create User

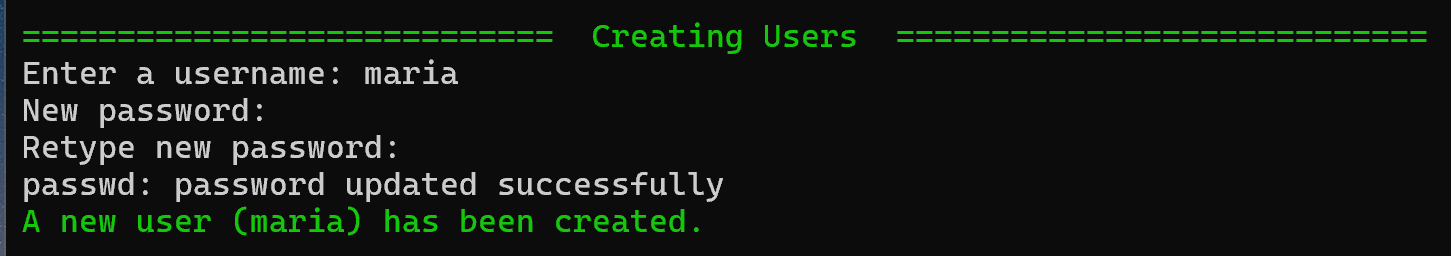
First, prompt the user to enter a username. Use if id "$username" &>/dev/null; to check if the username exists. If not, use sudo useradd $username and sudo passwd $username to create a new user with a home directory and set a password.

Figure 2: Successfully created a new user

#### Root permission

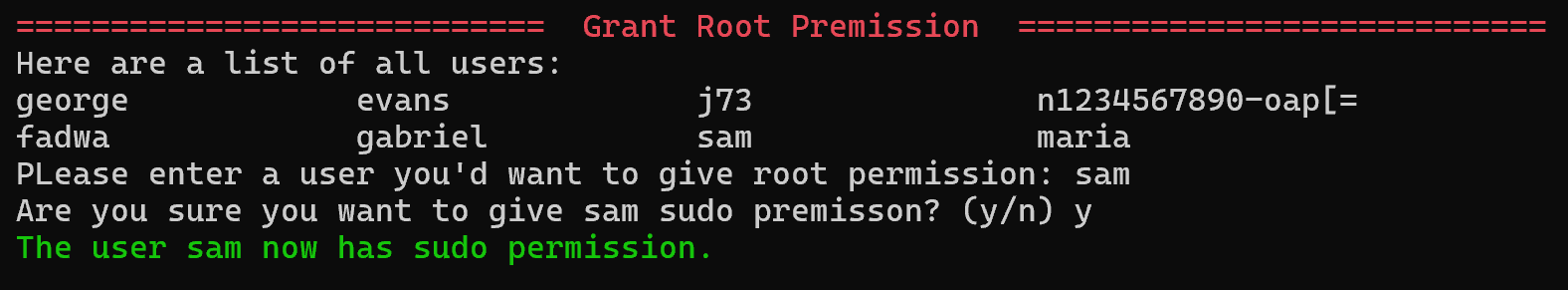
First, I show all users in a formatted list using: cut -d: -f1 /etc/passwd | pr -t -a -4. Then I prompt for a username to give root permission. I check if the user exists with if id "$username" &>/dev/null;. If not, I display “The user does not exist.” I then ask for confirmation. If the answer is “y,” I add the user to the root group with sudo usermod -aG sudo $user. If the answer is “n,” the user is not added.

Figure 3: SUCCESSFULLY gave sudo premission

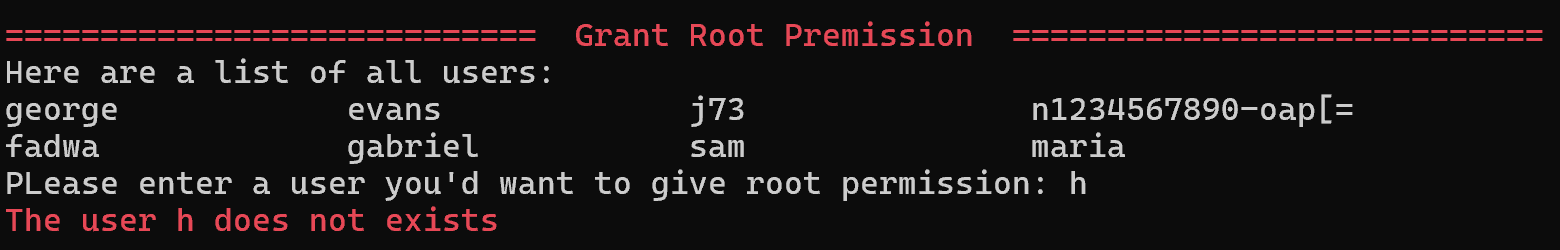


Figure 4: User does not exists error

#### Delete User

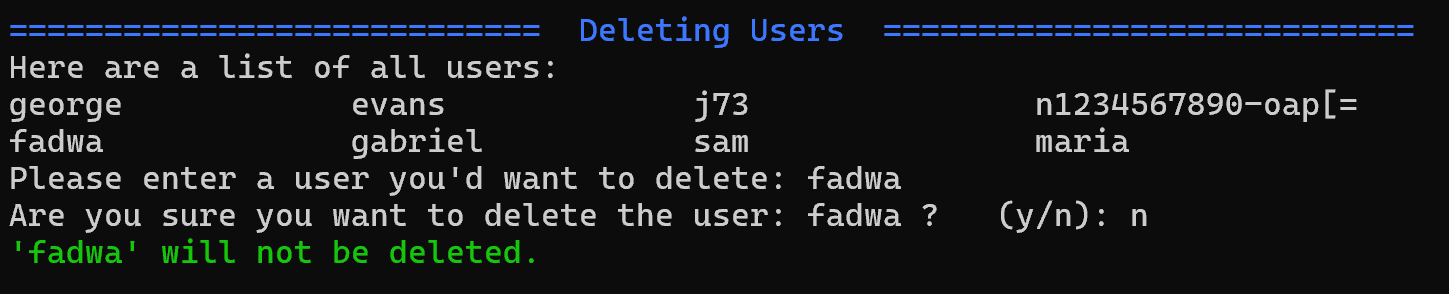
First, I show all users with a home directory in a formatted list using: cat /etc/passwd | grep home | cut -d: -f1 | pr -t -a -4. Then I prompt for a username to delete. I check if the user exists with if id "$username" &>/dev/null;. If not, I display “The user does not exist.” I then ask for confirmation. If the answer is “y,” I delete the user with sudo userdel "$user". If the answer is “n,” the user is not deleted.

Figure 5: Did not deleting a user

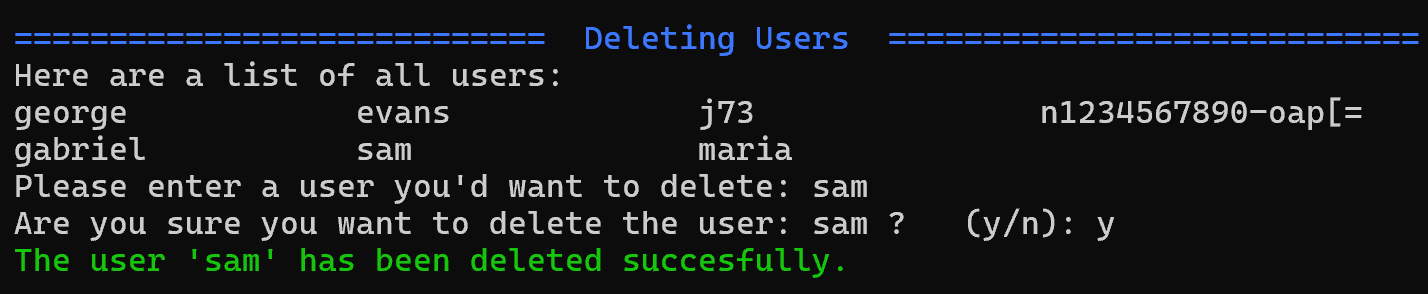


Figure 6:SUCCESSFULLY deleted a user

#### Display User

This method displays all users using: who

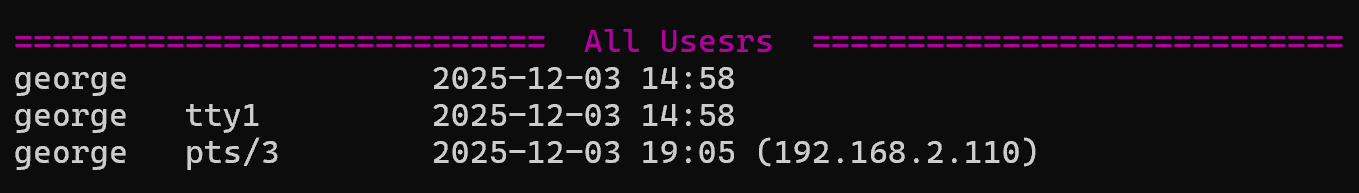


Figure 7: Display all users

#### Kill Remote

I display all remote users and their pts terminals using who | grep pts | awk '{print $1,$2}'. Then I ask which user to disconnect. I check if the user exists with id "$user" &>/dev/null. Next, I prompt for the pts session and check if it belongs to the user by comparing it with userPTS=$(who | grep "$user" | awk '{print $2}'). If it matches, I ask for confirmation and disconnect the session with pkill -HUP -t "$pts". If not, the script does not disconnect the userA screen shot of a computer

AI-generated content may be incorrect.

Figure 8:SUCCESSFULLY disconnected a remote user

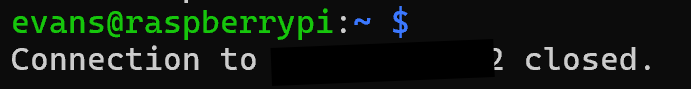


Figure 9: Showing disconnected user

#### Group User

It prints a formatted list of all system users by extracting usernames from /etc/passwd and arranging them into columns with cut -d: -f1 /etc/passwd | pr -t -a -4. Then it asks for a username and checks if the user exists with id "$user" &>/dev/null. If valid, it shows all groups the user belongs to by running groups "$user" and extracting the group names with cut -f2 -d:. If the username is invalid, it prints an error message.

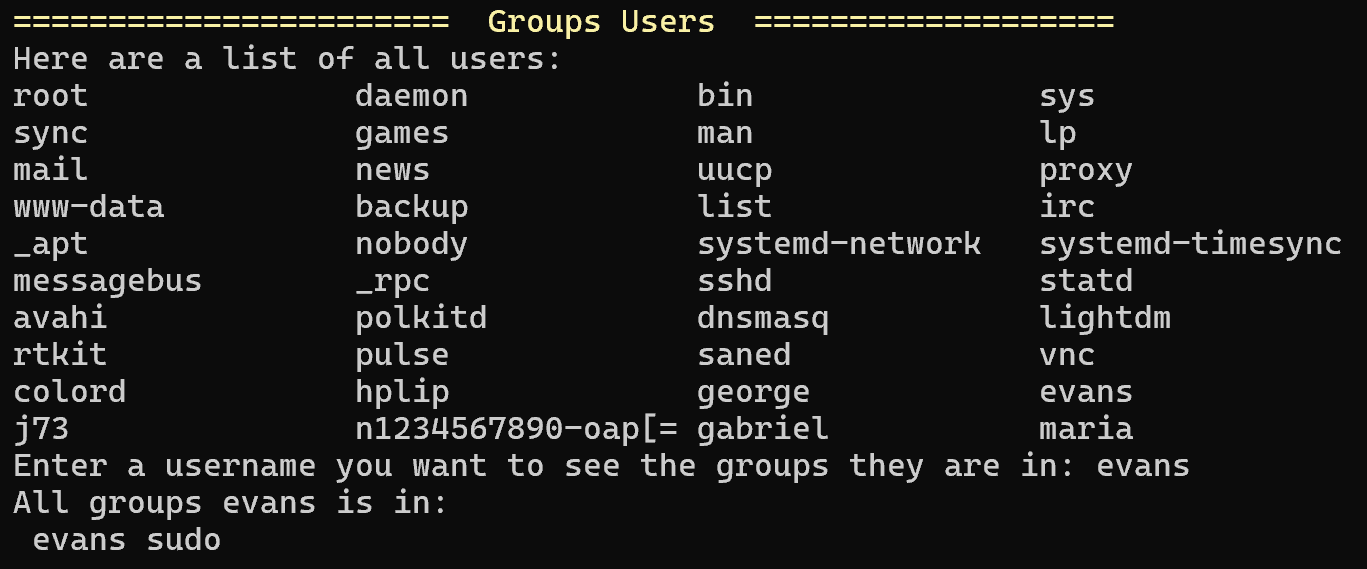


Figure 10: Display the users group

#### Add Group

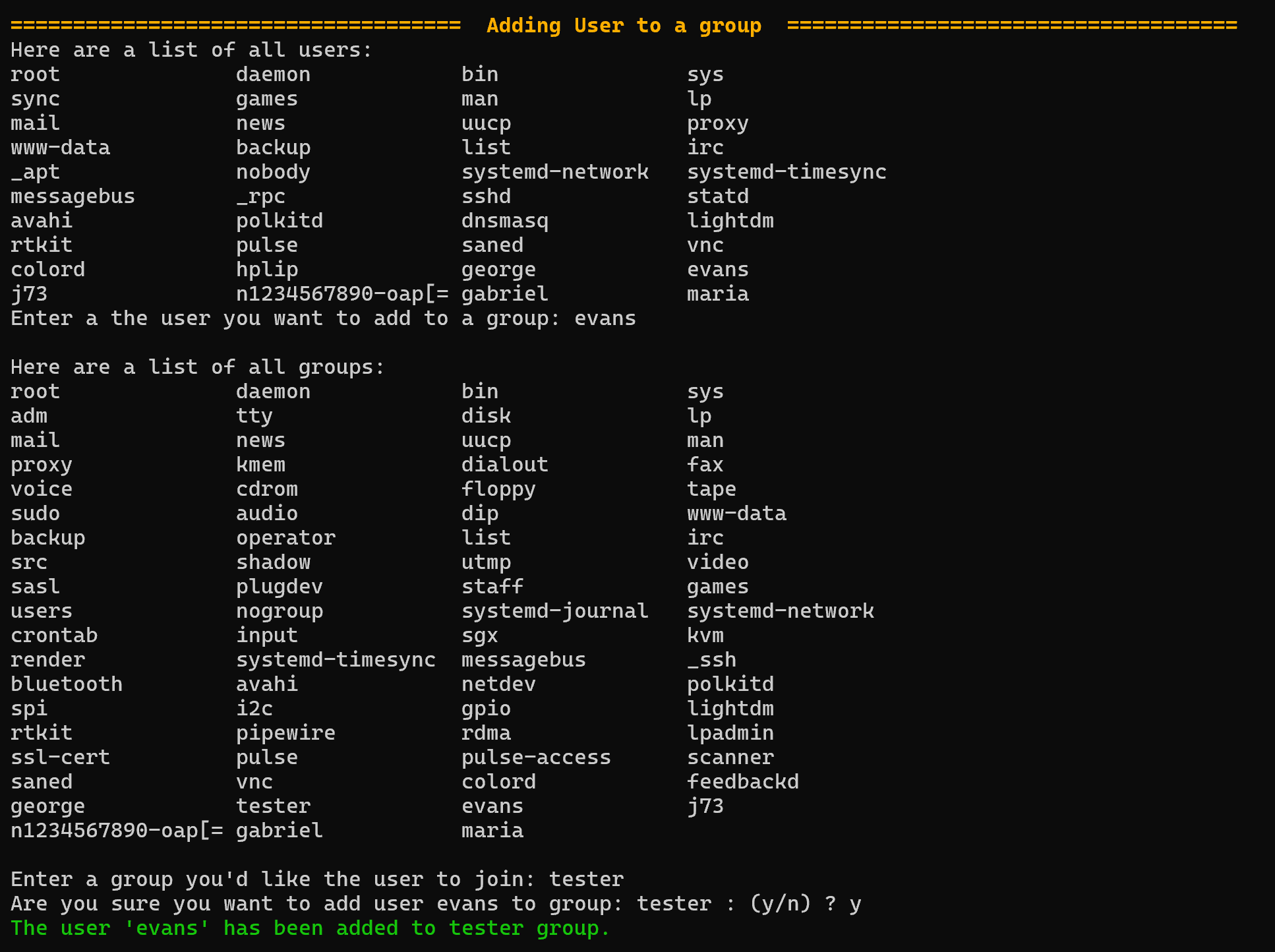
The function displays a formatted list of all system users using `cut -d: -f1 /etc/passwd | pr -t -a -4`, then prompts for the username to add to a group. It checks if the user exists with `id "$user" &>/dev/null`. If valid, it shows all available system groups, formatted with `cut -f1 -d":" /etc/group | pr -t -a -4`, and prompts for the group name. It checks if the group exists with `getent group "$group"`. If valid, it asks for confirmation. If the answer is “y,” it adds the user to the group with `usermod -aG "$group" "$user"`. If the answer is “n,” the action is canceled. Invalid input prints an error. If the user or group does not exist, it prints an error message.

Figure 11: Output when added a user to a new group

#### Remove Group

The function prints a formatted list of all system users using cut -d: -f1 /etc/passwd | pr -t -a -4, then prompts for the username to modify. It checks if the user exists with id "$user" &>/dev/null. If valid, it shows all groups the user belongs to by running groups "$user" and extracting the group list with cut -f2 -d:. The script then asks which group to remove the user from and checks if the group exists with getent group "$group". If valid, it asks for confirmation. If the answer is “y,” it removes the user from the group with gpasswd -d "$user" "$group". If canceled or invalid, it prints a message. If the user or group does not exist, it reports an error.

Figure 12: output to remove user from a group

#### 

#### Group Sudo

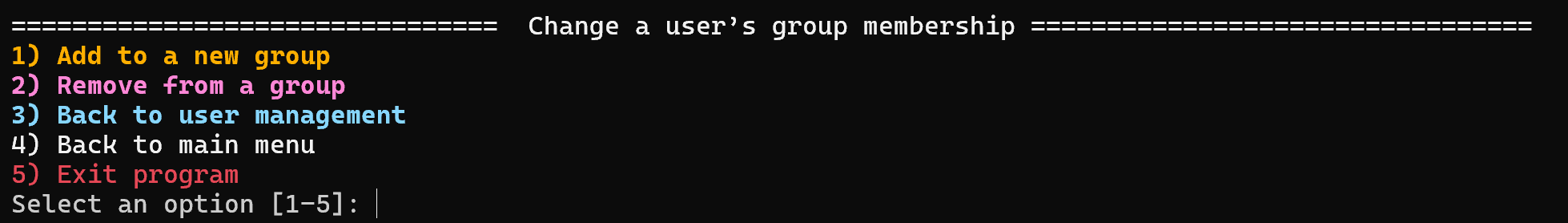
The function shows a menu for managing a user's group memberships. Options include adding or removing a user from a group, returning to the user-management menu, returning to the main menu, or exiting. After an option is selected, the script uses a case statement to call the right function or go to the correct menu. If exit is chosen, the script ends. Any input outside 1–5 shows an “invalid option” message.

Figure 13: menu for group membership

### File Management

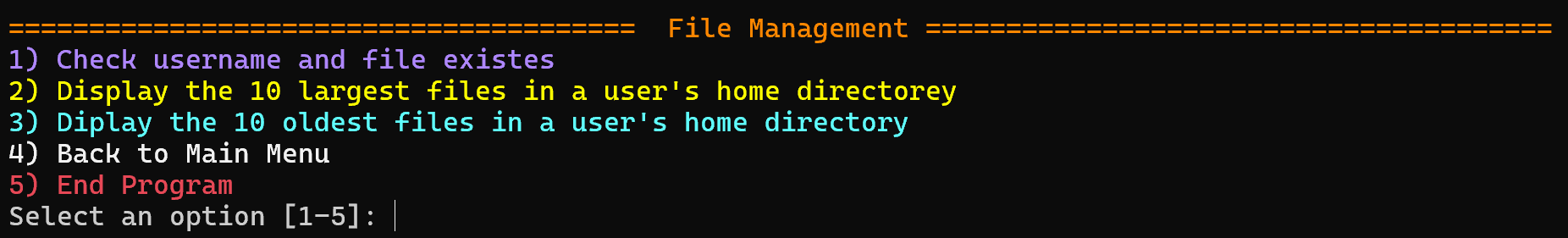
The function enters a loop that shows a menu for file operations. The administrator can check if a username and file exist, view the 10 largest files in a user's home directory, or see the 10 oldest files there. Options to return to the main menu or exit are also available. After an option is chosen, a case statement runs the right function. Invalid choices cause an error, and the menu repeats until the user exits.

Figure 1: Menu for file management

#### User and File Check

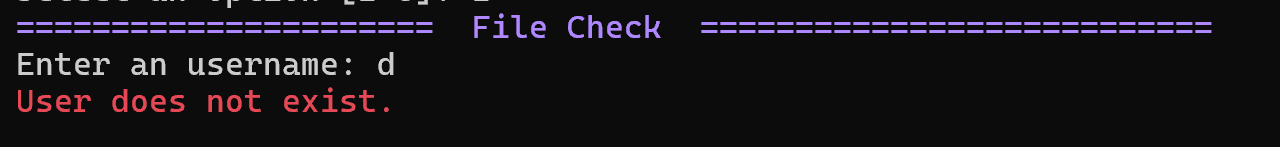
The function prompts the administrator to enter a username and checks if the account exists with id "$username" &>/dev/null. If valid, it gets the user's home directory with getent passwd and asks for the filename to search. It uses find to search the home directory for the file and stores the first match in fileResult. If fileResult is not empty, it prints the full path; otherwise, it reports that the file was not found. If the username does not exist, it prints an error and returns.

Figure 2: User does not exists error

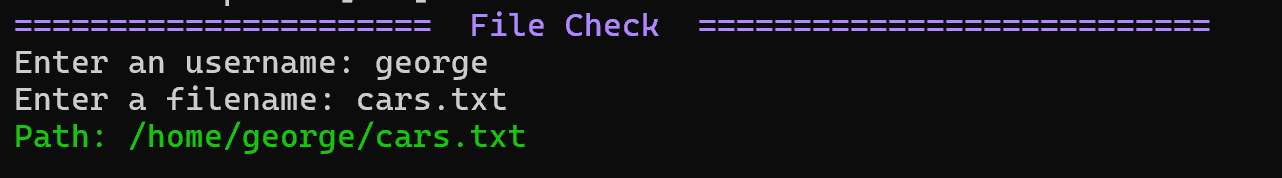


Figure 3: output of file and user found

#### Display Large

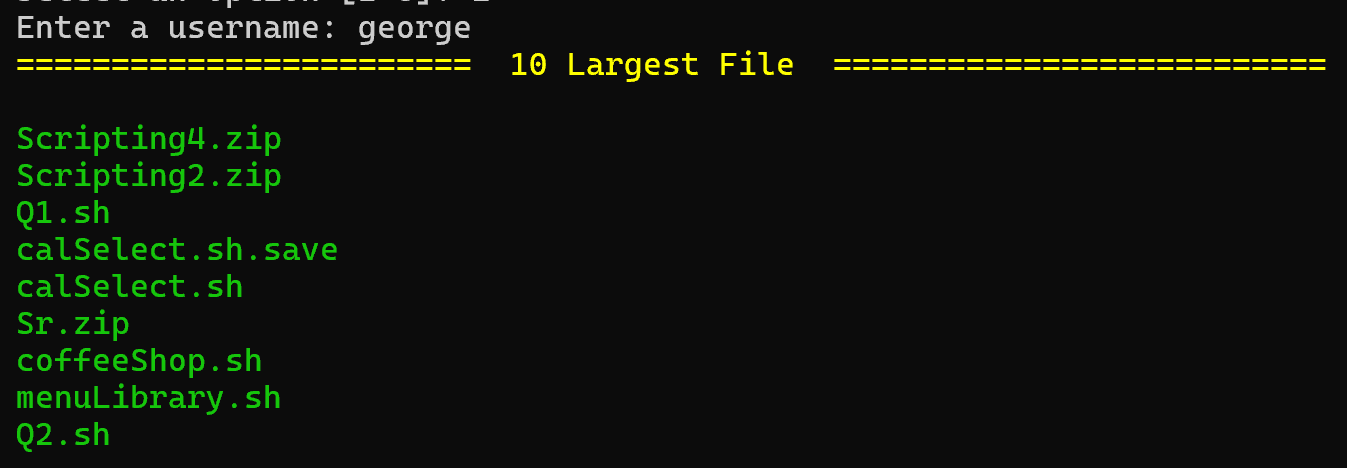
The function asks the administrator to enter a username and checks if the account exists with id "$username" &>/dev/null. If valid, it gets the user's home directory with getent passwd and shows the 10 largest files there. It runs ls -lSh to sort by size, filters out directories with grep -v d, takes the first 10 results with head -n 10, and prints filenames with awk '{print $9}'. If the username does not exist, it prints an error.

Figure 4: output of the 10 largest file

#### Display Old

The function asks for a username and checks if the account exists with id "$username" &>/dev/null. If valid, it gets the user's home directory with getent passwd. It then shows the 10 oldest files there by running ls -ltr to sort by modification time, filters out directories with grep -v d, takes the first 10 results with head -n 10, and prints filenames with awk '{print $9}'. If the username does not exist, it prints an error.

Figure 5: output of the 10 oldest file