UNIX FINAL PROJECT

Due date: December 2nd, 2025

Teacher: Samad Rostampour

Team Members: Adil Molla, Deven Shah-Phan, Arend Markies

Team Members: Adil Molla, Arend Markies, Deven Shah-Phan

1.PROJECT OVERVIEW:

This project is meant to fully acknowledge the different commands that can be found in bash scripting. It's useful for being able to display important information such as the cpu's temperature, services that are running or need to be stopped, user management such as deleting, granting certain privileges and managing files by size. The main purpose of the project is to be able to navigate bash scripting in real world applications.

2.PROJECT REQUIREMENTS AND DELIVERABLES:

DELIVERABLE 1 (Deven Shah-Phan)
1.System Management
3.Network Management
Deadline: Nov 27, 2025
Summary of Tasks:
Did the system status and network management scripts. The System status script allows us to show the memory usage, check the CPU temperature, show the active processes, and allows the user to stop or terminate a process. And the network management allows the user to show the network interfaces, ips and default gateways, it also allows users to enable or disable a network interface, users can set an IP address to a network card, and display wifi networks.

DELIVERABLE 2 (Arend Markies):
2.Backup Management
5.User Management
Deadline: Nov 27, 2025
Summary of Tasks:
The Backup Management handles the scheduling of automated backups based on the user-provided day, time, file name and destination. It will also display the most recent backup for the selected file. User Management provides tools to create and delete users, as well as assigning them sudo privileges and group memberships. As well as displaying the connected users and disconnecting them via remotely from online sessions.

DELIVERABLE 3 (Adil Molla):
4.Service Management
6.File Management
Deadline: Nov 27, 2025
Summary of Tasks:
The deliverable 3 takes care of the service management and file management parts of the project. Service management in particular has a goal of showing services that are currently “running” and give the ability to stop or start any service “running”. On the other hand File Management is used for displaying the largest and oldest files after verifying if the other conditions such as username and file name are valid.

TABLE VIEW:

Deven Shah-Phan	**System Management**	**Network Management**
Arend Markies	**Backup Management**	**User Management**
Adil Molla	**Service Management**	**File Management**

3.TASK ASSIGNMENT:

Deven Shah-Phan:
1. **System Management**
3. **Network Management**

Arend Markies:
2. **Backup Management**
5. **User Management**

Adil Molla:
4.**Service Management**
6. **File Management**

4.Project Components and Solutions

DELIVERABLE 1 (Deven Shah-Phan)

1.System Management
3.Network Management

In the system status management in order to display the memory i just had to use the command free -m. The free command just displays detailed information about the memory and the -m just converts the numbers from kilobytes to megabytes.

Then for the cpu temperature, the system stores the CPU temperature inside /sys/class/thermal/ so,
cpuTemp=$(cat /sys/class/thermal/thermal_zone0/temp)
Gets the cpu temperature of the computer, and then after i have to convert the value by dividing by 1000. Then we do an if / else condition to check whether or not the cpu temperature is greater than 70C.

And to list the active list system processes i just use top |head -n 20 which displays the first 20 processes.

And to allow a user to stop a process or terminate one, i ask the user to enter the PID of the process to make sure that there isn't any mix ups or like name conflicts with other processes, then we kill -SIGSTOP or -SIGTERM "$process" which will stop or terminate the process that the user chose.

For the network management, to display the network interfaces you have to list all the interfaces that are in /sys/class/net/ using a for loop, and then get the ip address of each interface. And we need to check if there are interfaces with no ip addresses so we use an if condition, and display the interfaces. And then we display the default gateways with ip route show default.

To be able to enable or disable an interface you just have to do sudo ip link set “$interface” up (or down). First i show the available interfaces, ask them to choose an interface then ask them to enable or disable that interface.

And to assign and IP address to a network interface/card i had to first display all the available interfaces to the user then ask them to choose one, then ask them to enter an IP address inorder to assign it to the interface using: sudo ip addr add “$ip” dev “$interface”.

To display the WI-FI Networks and allow the user to connect to one, i have to show the user the available Wifi-Networks by doing nmcli device wifi list, and then ask them to enter the ssid of the wifi network they want to connect to, and connect them to it with : nmcli device wifi connect “$ssid” password “$pass”.

Both of these scripts mainly use functions and a while true; loop to display their menu and to read the users choice of function

DELIVERABLE 2 (Arend Markies):
2.Backup Management
5.User Management

The backup management allows the user to schedule automated backups and view the most recent backup for the set file selected. It relies heavily on the user input to schedule when and where backups occur.

1 User Input Collection
The script prompts the user to enter:

- Backup day (e.g., Monday)

- Backup time (HH:MM format)
- File name or file path
- Destination directory

  This ensures full flexibility and prevents hard-coded values.

2 Cron Job Creation
After collecting the required inputs, the script automatically generates a valid cron entry in the system's crontab.

- The time and day are parsed into the appropriate cron format.
- A `cp` command is scheduled to copy the specified file to the destination.

3 Backup History Display
A function checks the backup destination directory and displays the timestamp of the most recent backup.

- This helps verify that the scheduled task is functioning correctly.

4 Error Handling(Additional stuff)

- Checks for missing files or invalid directories.

- Validates time input before scheduling.

- Prevents duplicate or malformed cron entries.

The user management provides administrative controls for managing system accounts and permissions. It centralizes user creation, deletion, privilege assignment and monitoring there session.

1 the User Management System

Menu-based interface for system administration

Options include:

    - Create user
    - Delete user
    - Grant sudo privileges
    - Show logged-in users
    - Disconnect users
    - View and change group memberships

Designed to simplify tasks normally done with multiple Linux commands

2 Safety & System Integrity Controls

- Checks if the filesystem is read-only before modifying accounts
- Detects and removes stale lock files to prevent corruption
- Prevents operations if conflicting user-management processes are active
- Ensures operations run safely without damaging system files

3 Administrative Tools

- Allows adding users to sudo group
- Displays active user sessions
- Terminates user processes when needed
- Supports primary and secondary group modifications
- Provides essential tools for system administrators in one place

4 Reliable User Operations

- Validates system state before performing account changes
- Handles errors cleanly and reports clear messages
- Terminates active user processes to allow safe deletion
- Prevents partial account creation or deletion issues

5 Enhanced Error Handling

- Notifies user of failed actions (e.g., password set failures, missing permissions)
- Provides explanations for possible errors
- Ensures the admin always knows the outcome of each operation

6 System Protection Measures

- Avoids simultaneous account-modifying operations that can corrupt system files
- Ensures secure, stable handling of `/etc/passwd`, `/etc/group`, and related files
- Reduces the risk of breaking user accounts or system authentication

7 User-Friendly Design

- Menu-driven layout makes system administration simpler
- Reduces reliance on remembering command syntax
- Groups operations logically for easy navigation
- Minimizes mistakes by guiding the user through each step

DELIVERABLE 3 (Adil Molla):

4.Service Management
6.File Management

Main Components Service Management:

- **1. PS3 And Select**

The use of PS3 and select is to invoke a message to the user to allow them to choose what options they'd like to proceed with. It acts as a sort of echo that's interactive. The select statement is used to prompt to the user the number of options and what they each do. In this case asking the user to stop, start or see the status of services.

- **2. Switch-case**

The switch case for service management is called serviceMgmt and is used to switch in between the different options depending on the user's input, as number 1 would bring up the services that are currently active, number 2 would ask the user to start a service and number 3 would ask a user to stop the service, anything else would throw an error as it's not a valid option.

- **3. IF/ELSE Statements**

IF/ELSE statements are very important in this script as they're the way errors are handled, in terms what this means is that if the user starts a service but it does not exist it will use an echo to let the user know that it does not exist and same goes for stopping a service, the way it checks for if the service exist is with the help of grep which filters out the name of the service and adds .service at the end of it to check if any service with that name exists.

- **4. Invalid Number Handling**

The switch case statement takes care of any options that is not included with the * which indicates everything else typed in besides the options given will throw an echo telling the user it's invalid and exit out with the help of a break so that the program does not crash.

Main Components File Management:

- **1. PS3/SELECT**
  The PS3 prompts a message to the user so that they could pick from the following options: findFile, largestFiles, oldestFiles and Exit. Select statement allows them to pick the option they'd prefer.
- **2. Switch-case**
  The switch cases here are used for the different options such as findFile, this case will take the user input to the find if such a file exist and if will also ask the user for the username to validate and will prompt to the user that it has been found, we use options such as -f for finding files and dev/null to check the existence of the user. LargestFiles will look for the files that take the most amount of space on the system and the oldestFiles is the same but with older files instead, they both use their different ways of checking such as du for disk usage and ls -lt to check last modified date.

- **3. IF/ELSE Statements**

The if/else statements are a way of handling edge cases in particular with case one if the user is found or not found it will prompt a different message and the same can be said for if the file exists or not.

- **4. Usage of Pipe**

  The pipe command is used a lot for this script to allow for an easier time reading what the script outputs and to reduce the number of lines shown on the script, in particular the use of head and tail help with limiting the number of lines to 10 on the screen.

- **5. Exit and Error handling**

  The case to exit exists if the user would like a clear way out to exit the program and to not be confused, the exiting is done with the help of a break statement and unused numbers are handled by the *) case which prompts a message and stops the program.

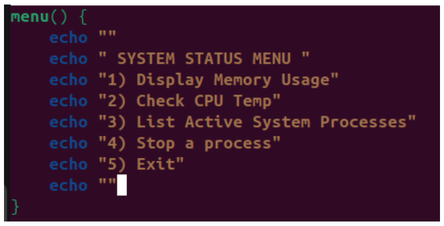
5.Detailed Descriptions

DELIVERABLE 1 (Deven Shah-Phan)

1.System Management

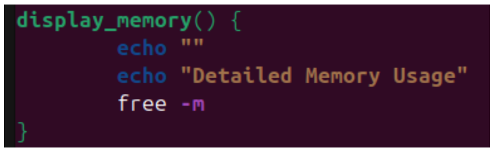
3.Network Management

System Status Management



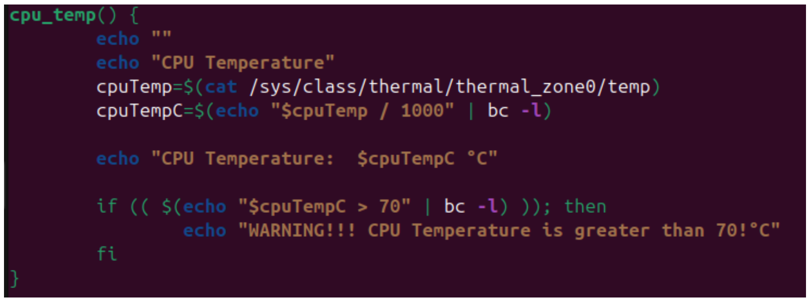
```
menu() {
    echo ""
    echo " SYSTEM STATUS MENU "
    echo "1) Display Memory Usage"
    echo "2) Check CPU Temp"
    echo "3) List Active System Processes"
    echo "4) Stop a process"
    echo "5) Exit"
    echo ""
}
```

Menu function to call in the while loop



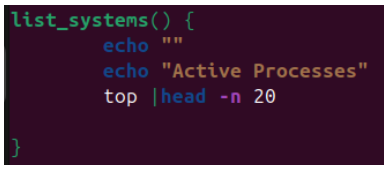
```
display_memory() {
        echo ""
        echo "Detailed Memory Usage"
        free -m
}
```

using free command to display detailed memory



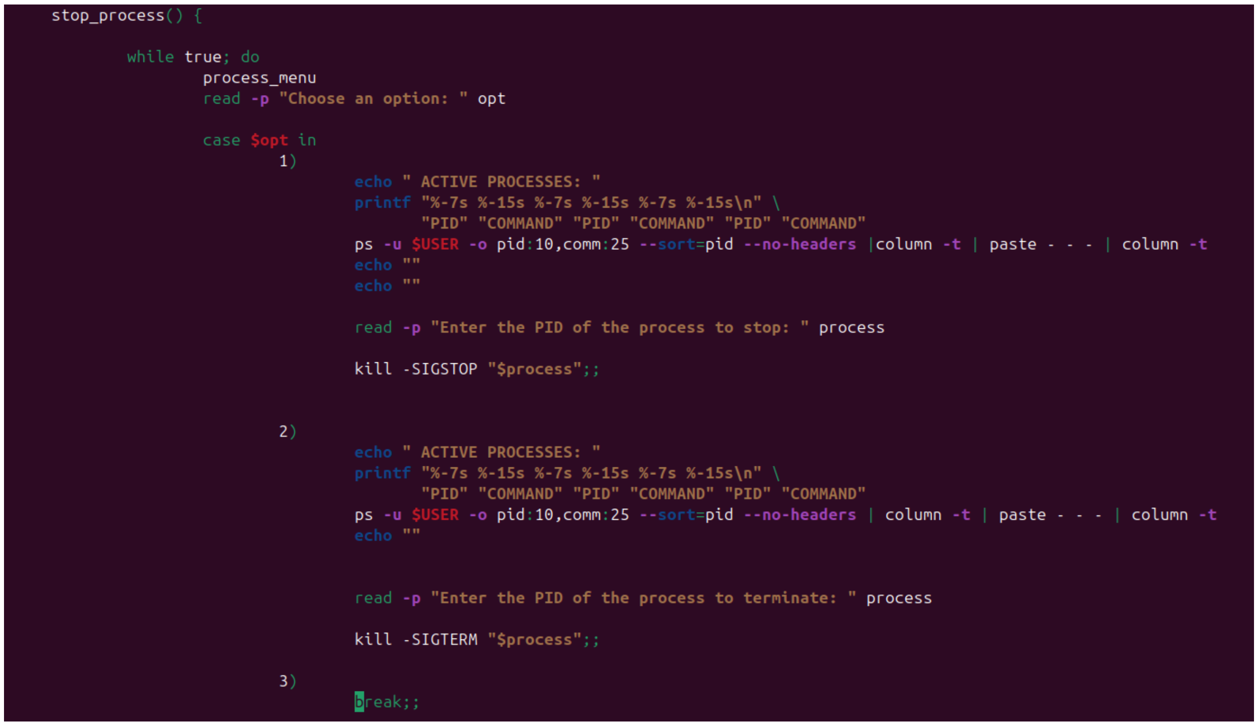
```
cpu_temp() {
        echo ""
        echo "CPU Temperature"
        cpuTemp=$(cat /sys/class/thermal/thermal_zone0/temp)
        cpuTempC=$(echo "$cpuTemp / 1000" | bc -l)

        echo "CPU Temperature:  $cpuTempC °C"

        if (( $(echo "$cpuTempC > 70" | bc -l) )); then
                echo "WARNING!!! CPU Temperature is greater than 70!°C"
        fi
}
```

Cpu function to get the cpu temperature of the system, from the /sys/class/thermal/thermal_zone0/temp.
Because it stores the temp in millidegree we need to convert by dividing by 1000
Then we have the if condition to check the cpu temp and display a warning if greater than 70.



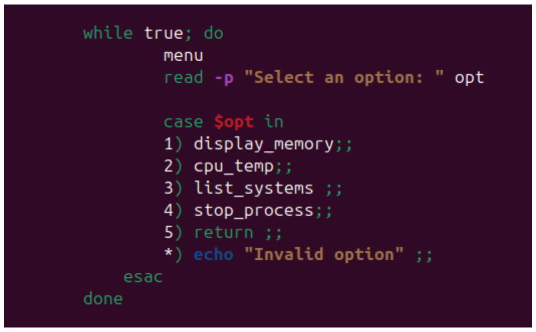
```
list_systems() {
        echo ""
        echo "Active Processes"
        top |head -n 20

}
```

Using top to display the first 20 processes



```
    stop_process() {

            while true; do
                    process_menu
                    read -p "Choose an option: " opt

                    case $opt in
                            1)
                                    echo " ACTIVE PROCESSES: "
                                    printf "%-7s %-15s %-7s %-15s %-7s %-15s\n" \
                                           "PID" "COMMAND" "PID" "COMMAND" "PID" "COMMAND"
                                    ps -u $USER -o pid:10,comm:25 --sort=pid --no-headers |column -t | paste - - - | column -t
                                    echo ""
                                    echo ""

                                    read -p "Enter the PID of the process to stop: " process

                                    kill -SIGSTOP "$process";;


                            2)
                                    echo " ACTIVE PROCESSES: "
                                    printf "%-7s %-15s %-7s %-15s %-7s %-15s\n" \
                                           "PID" "COMMAND" "PID" "COMMAND" "PID" "COMMAND"
                                    ps -u $USER -o pid:10,comm:25 --sort=pid --no-headers | column -t | paste - - - | column -t
                                    echo ""


                                    read -p "Enter the PID of the process to terminate: " process

                                    kill -SIGTERM "$process";;

                            3)
                                    break;;
```

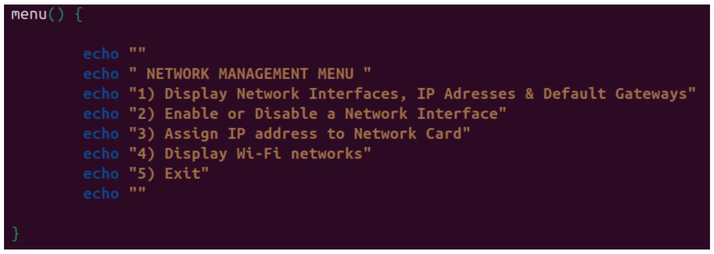
Displays the active processes in three columns and asks the user to enter the PID of the process they want to stop or terminate, and we do that by kill -SIGSTOP “$process” and kill -SIGTERM “$process”



```
	while true; do
		menu
		read -p "Select an option: " opt

		case $opt in
		1) display_memory;;
		2) cpu_temp;;
		3) list_systems ;;
		4) stop_process;;
		5) return ;;
		*) echo "Invalid option" ;;
	    esac
	done
```

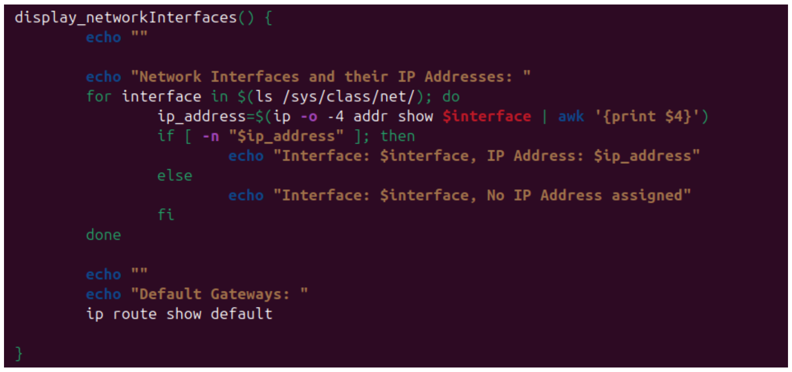
while loop to display system status menu

Network Management



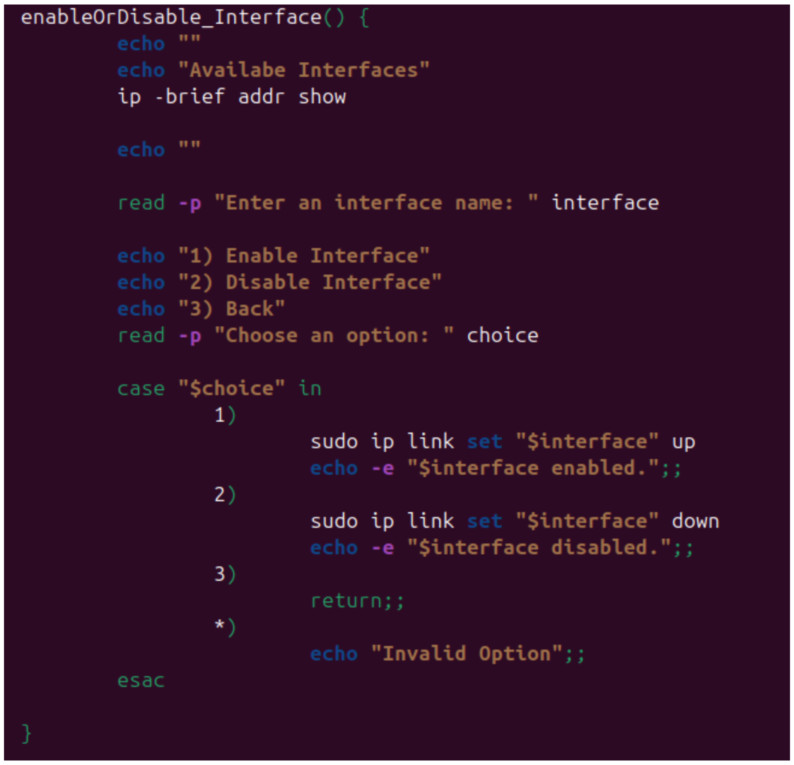
```
menu() {

	echo ""
	echo " NETWORK MANAGEMENT MENU "
	echo "1) Display Network Interfaces, IP Adresses & Default Gateways"
	echo "2) Enable or Disable a Network Interface"
	echo "3) Assign IP address to Network Card"
	echo "4) Display Wi-Fi networks"
	echo "5) Exit"
	echo ""

}
```

Menu function to display to user



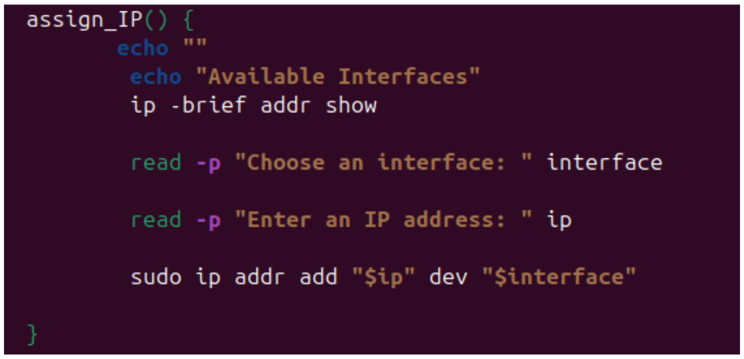
```
display_networkInterfaces() {
	echo ""

	echo "Network Interfaces and their IP Addresses: "
	for interface in $(ls /sys/class/net/); do
		ip_address=$(ip -o -4 addr show $interface | awk '{print $4}')
		if [ -n "$ip_address" ]; then
			echo "Interface: $interface, IP Address: $ip_address"
		else
			echo "Interface: $interface, No IP Address assigned"
		fi
	done

	echo ""
	echo "Default Gateways: "
	ip route show default

}
```

Displaying the interfaces from sys/class/net/ using a for loop and getting each of their IP addresses, and checking if the interface has an ip. And then displaying the default gateways.



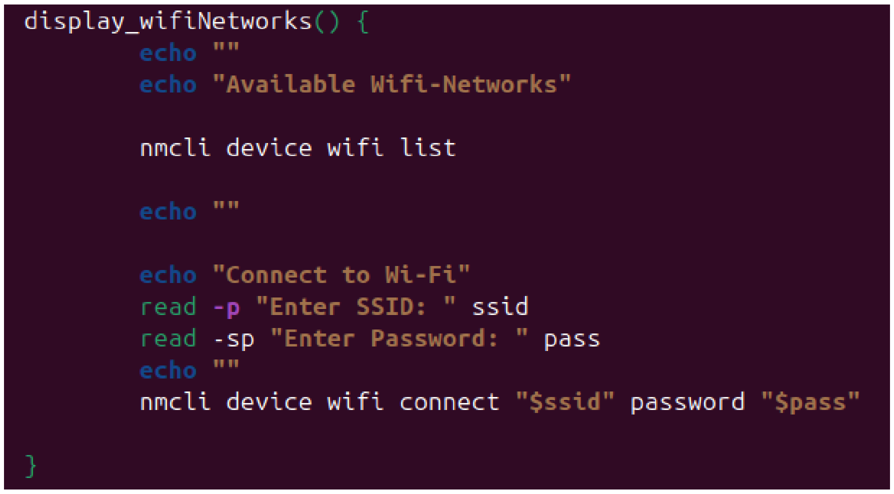
```
enableOrDisable_Interface() {
        echo ""
        echo "Availabe Interfaces"
        ip -brief addr show

        echo ""

        read -p "Enter an interface name: " interface

        echo "1) Enable Interface"
        echo "2) Disable Interface"
        echo "3) Back"
        read -p "Choose an option: " choice

        case "$choice" in
                1)
                        sudo ip link set "$interface" up
                        echo -e "$interface enabled.";;
                2)
                        sudo ip link set "$interface" down
                        echo -e "$interface disabled.";;
                3)
                        return;;
                *)
                        echo "Invalid Option";;
        esac

}
```

Display all available interfaces to the user, then asking them to choose one. And then giving them the choice to stop or terminate it by using sudo ip link set (interface) up/down



```
assign_IP() {
        echo ""
         echo "Available Interfaces"
         ip -brief addr show

         read -p "Choose an interface: " interface

         read -p "Enter an IP address: " ip

         sudo ip addr add "$ip" dev "$interface"

}
```

Displays the available interfaces, then asks the user to choose one, and then enter the ip address to assign to the interface. And then we assign it using: Sudo ip addr add “$ip” dev “$interface”



```
display_wifiNetworks() {
        echo ""
        echo "Available Wifi-Networks"

        nmcli device wifi list

        echo ""

        echo "Connect to Wi-Fi"
        read -p "Enter SSID: " ssid
        read -sp "Enter Password: " pass
        echo ""
        nmcli device wifi connect "$ssid" password "$pass"

}
```

Display wifi networks, then asks the user to enter the SSID that they want to connect to and their password to be able to connect to it by using:
nmcli device wifi connect “$ssid” password “$pass”

DELIVERABLE 2 (Arend Markies):
2.Backup Management
5.User Management

Backup Management:
The BackupLog insures that there is a record of all the saves for all the selected files run by the program
BackupLog="$HOME/backup_log.txt"

```
read -p "Enter the day of the week (0-6, 0=Sunday, 1=Monday,...): " Day
case "$Day" in
  0) Day="Sun"
  echo "Sunday selected";;
  1) Day="Mon"
  echo "Monday selected";;
  2) Day="Tue"
  echo "Tuesday selected";;
  3) Day="Wed"
  echo "Wednesday selected";;
  4) Day="Thu"
  echo "Thursday selected";;
  5) Day="Fri"
  echo "Friday selected";;
  6) Day="Sat"
  echo "Saturday selected";;
  *)
    echo "Invalid day"
    exit 1
    ;;
esac
read -p "Enter the hour for backup (0-23): " Hour
read -p "Enter the minute for backup (0-59): " Minute
echo "the time you have selected $Hour:$Minute"
read -p "Enter the full path of the file to back up: " File
echo "you selected $File"
read -p "Enter the destination directory: " Destination
echo "you selected $Destination"
```

This checks to see if the file exists and that if it doesn’t it closes the program / script file

```
if [ ! -f "$File" ]; then
   echo "The file '$File' does not exist."
   exit 1
fi
```

The script prompts the user to enter data so it’s able to schedule a backup date and after entering a selected it then displays to the user what it is they have selected

This part makes the destination if it doesn’t already exist

```
mkdir -p "$Destination"
```

The script will first check for spaces in the file path and destination then replaces all the spaces with \ so the cron job and cp command can read the paths correctly.

```
FileEscaped=$(echo "$File" | sed 's/ /\\ /g')
DestinationEscaped=$(echo "$Destination" | sed 's/ /\\ /g')
```

Using the day, hour, and min that has been entered by the user the script will then create the cron job, the file is then copied to the destination folder and 'boom' a timestamp has been added to the file name to that there is no overwriting on past backups. Once the backup has been successful the script logs the completion time in back_log.txt. The next part has the script check to see the current cron job for duplicates for the same file name and removes any existing backup jobs for that file. It will then add the new cron job to the user's crontab so the backup runs automatically by the scheduled time, it basically makes sure that there is only one scheduled backup per file.

```
CronJob="$Minute $Hour * * $Day cp $FileEscaped
$DestinationEscaped/$(basename "$File")_$(date +\%Y\%m\%d_\%H\%M\%S)
&& echo \"Backup completed at \$(date)\" >> $BackupLog"
(crontab -l 2>/dev/null | grep -Fv "$File" ; echo "$CronJob") | crontab
```

The set of echos just prints to the user when and where the data and/or file is being backuped to

```
echo ""
echo "Backup scheduled:"
echo "Day of week: $Day"
echo "Time: $Hour:$Minute"
echo "File: $File"
echo "Destination: $Destination"
echo ""
```

This checks to see if the file has been backed up before and if it has, it prints to the user when the last backup happened (and only the latest backup)

```
if [ -f "$BackupLog" ]; then
   echo "Last completed backup:"
   tail -n 1 "$BackupLog"
fi
```

User Management:

When you first run the script you are going to be denied access due to not running the script as root and will prompt you to write: sudo bash [name of script] so that the script will be able to run

```
if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root. Use: sudo bash \$0"
   exit 1
```

```
fi
```

The function check_fs() is to ensure that the root filesystem is writable, this plays an important part when wanting to create, modify, or delete user accounts

```
check_fs() {
    if mount | grep " / " | grep -q "(ro,"; then
        echo "Root filesystem is read-only!"
        echo "Fix with: sudo mount -o remount,rw /"
        exit 1
    fi
}
```

The function clear_locks() removes stale lock files and ensures no other user modifying process is active so that we are able to go ahead and modify user accounts without any difficulties

```
clear_locks() {
    lock_files=("/etc/passwd.lock" "/etc/shadow.lock" "/etc/group.lock")

    for f in "${lock_files[@]}"; do
        if [[ -e $f ]]; then
            if pgrep -f "useradd|usermod|userdel" >/dev/null; then
                echo "Another user-modifying process is running, try again later"
                exit 1
            fi
            echo "Removing stale lock: $f"
            rm -f "$f"
        fi
    done
}
```

The function create_user allows administrators to add a new user account to the system safely without there being any issues, as well as including a password and ensuring all prerequisites are met. It as well creates a directory for the user as well

```
create_user() {
    check_fs
    clear_locks
    read -p "Enter new username: " username
    read -s -p "Enter password: " password
    echo

    if useradd -m "$username"; then
        echo "$username:$password" | chpasswd || {
            echo "Failed to set password for '$username'"
            return
        }
        echo "User '$username' created"
    else
        echo "Failed to create user"
    fi
}
```

The function grant_root() prompts the user to enter a username, then checks to see if the user has or doesn’t have sudo privileges and then grants them sudo permissions

```
grant_root() {
    read -p "Enter username: " username
    usermod -aG sudo "$username" && echo "The user $username currently has
no sudo privileges"
    echo "Now adding sudo privileges"
    echo "$username now has sudo privileges"
}
```

The function delete_user() prompts to enter a username, once a user has been selected it then safely removes the users account from the system including their home directory and any active processes. The if statement checks to see if the user has been deleted, if it has not it prompts the user not being deleted and the possible causes for it, else it would print that the deletion was successful.

```
delete_user() {
    read -p "Enter username to delete: " username

    check_fs
    clear_locks
    echo "Checking for active processes"

    if pgrep -u "$username" >/dev/null 2>&1; then
        echo "User '$username' has running processes, now terminating"
        pkill -KILL -u "$username"
        sleep 1
    fi

    echo "Deleting user '$username' (suppressing harmless warnings)"
    userdel -r "$username" 2>/dev/null

    if id "$username" >/dev/null 2>&1; then
        echo "Failed to delete user '$username'"
        echo "Possible causes: "
        echo "The user is running a systemd service"
        echo "Another process recreated the user"
        echo "Filesystem or permission issue"
    else
        echo "User '$username' deleted successfully"
    fi
}
```

The show_connected_users() just prints to the user all the current users that are logged into the system

```
show_connected_users() {
    echo "All logged-in users: "
    who
}
```

The function disconnect_user() prompts to the user to enter a user they wish to disconnect them from the system without choice terminating all their active processes

```
disconnect_user() {
    read -p "Enter username to disconnect: " username
    pkill -KILL -u "$username"
    echo "User '$username' disconnected"
}
```

The function show_user_groups() asks you to enter a user you wish to check their groups and then displays the groups (ex making a user Adil will create a group Adil and should be the only group that he has)

```
show_user_groups() {
    read -p "Enter username: " username
    groups "$username"
}
```

The function change_membership() allows to modify the user's primary and secondary group memberships of an existing user account, giving control over their system permissions. You would need to be careful though if you were to change their primary group to something else then delete the user the home directory will still exist and will not let you recreate the previous user

```
change_membership() {
    read -p "Enter username: " username
    read -p "Enter primary group: " pgroup
    read -p "Enter additional groups (comma-separated): " agroups

    usermod -g "$pgroup" -G "$agroups" "$username" \
        && echo "Updated group membership for '$username'"
}
```

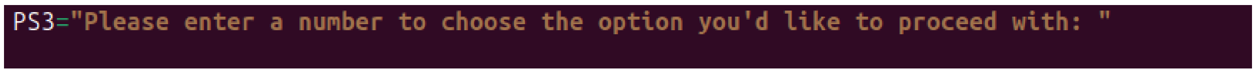
The last part of the script provides a menu of all the services and has them prompted to the user as a select and when you choose one of the cases it then calls the set function to it

```
while true; do
    echo ""
    echo "===== User Management (Arend Markies) ====="
    echo "1) Create new user"
    echo "2) Grant root privileges"
    echo "3) Delete user"
    echo "4) Show connected users"
    echo "5) Disconnect a user"
    echo "6) Show user's groups"
    echo "7) Change user's group membership"
    echo "8) Exit"
    read -p "Choose an option (1–8): " choice
    echo ""

    case "$choice" in
        1) create_user ;;
        2) grant_root ;;
        3) delete_user ;;
        4) show_connected_users ;;
        5) disconnect_user ;;
        6) show_user_groups ;;
        7) change_membership ;;
        8) echo "Goodbye!"; exit 0 ;;
        *) echo "Invalid option" ;;
    esac
done
```

Detailed Description:

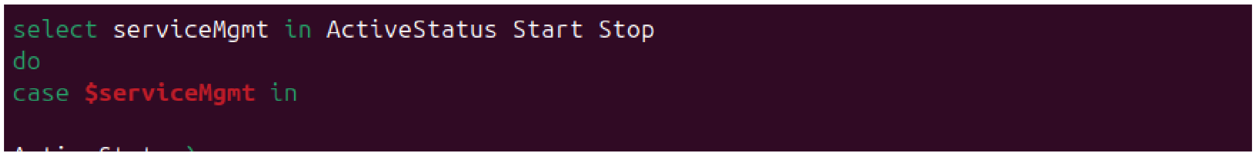
DELIVERABLE 3 (Adil Molla):
4.Service Management
6.File Management

Service Management:
1- The use of PS3 was invoked to prompt a message to the user in particular to which option they'd like to pick.
Example: PS3="Please enter a number to choose the option you'd like to proceed with:"

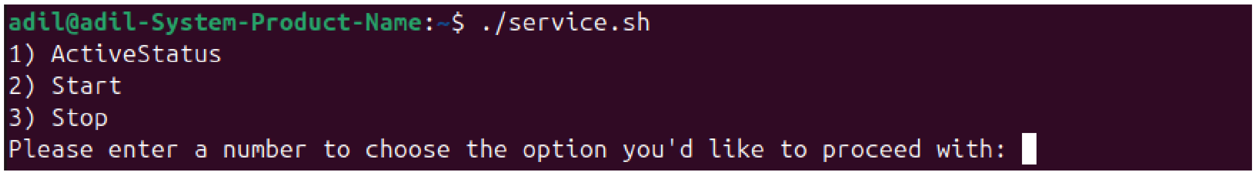


```
PS3="Please enter a number to choose the option you'd like to proceed with: "
```

2- Select statement is used to determine the name of the switch case we're gonna use and to show off the options the user could pick such as ActiveStatus Start Stop.

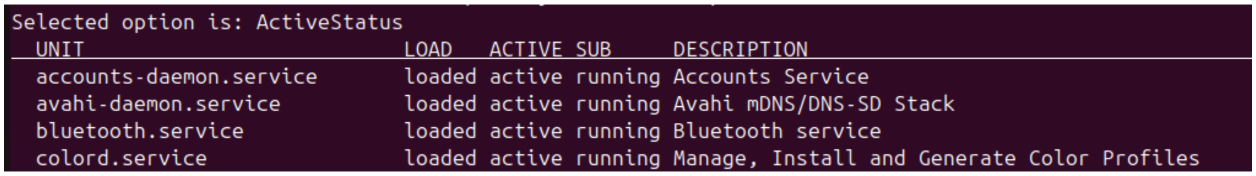


```
select serviceMgmt in ActiveStatus Start Stop
do
case $serviceMgmt in
```

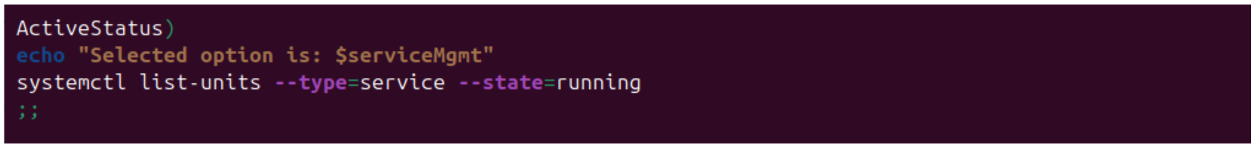


```
adil@adil-System-Product-Name:~$ ./service.sh
1) ActiveStatus
2) Start
3) Stop
Please enter a number to choose the option you'd like to proceed with: 
```

3- The first case ActiveStatus shows off the services that are currently active with the help of systemctl command which uses in particular the list unit to list all the units and specific types of unit such as services and running services with the help of - - type and - - state. This part of the code gives the user a good idea of what services they can start or stop.

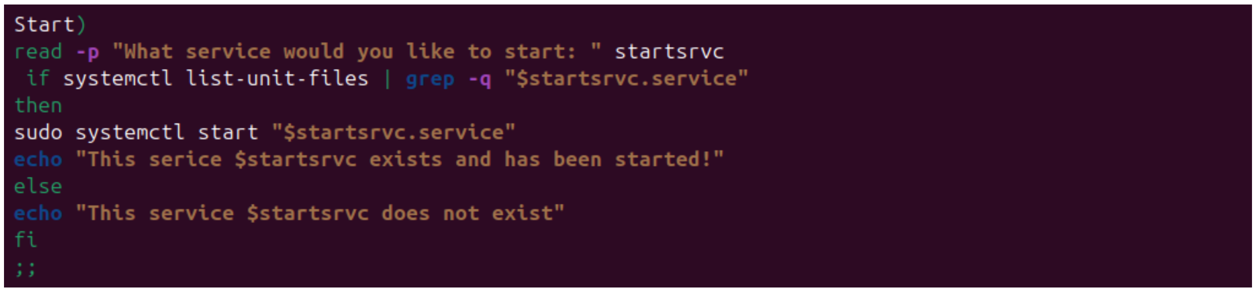


```
Selected option is: ActiveStatus
  UNIT                          LOAD   ACTIVE SUB     DESCRIPTION
  accounts-daemon.service       loaded active running Accounts Service
  avahi-daemon.service          loaded active running Avahi mDNS/DNS-SD Stack
  bluetooth.service             loaded active running Bluetooth service
  colord.service                loaded active running Manage, Install and Generate Color Profiles
```

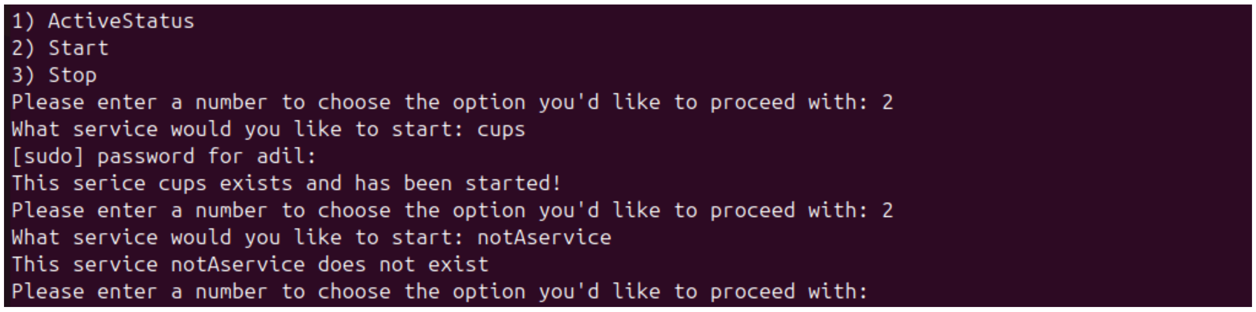


```
ActiveStatus)
echo "Selected option is: $serviceMgmt"
systemctl list-units --type=service --state=running
;;
```

4- The second case Start as the name implies starts a service that the user precises with the command systemctl start and the name of the service, although if and else statements are very important here because we first need to check if the service even exists with the help of systemctl lis-unit-files and the grep which allows us to filter out the name of the service and .service which checks if it’s a service or else it will echo the fact that the service does not exist and bring them back to the menu of options.

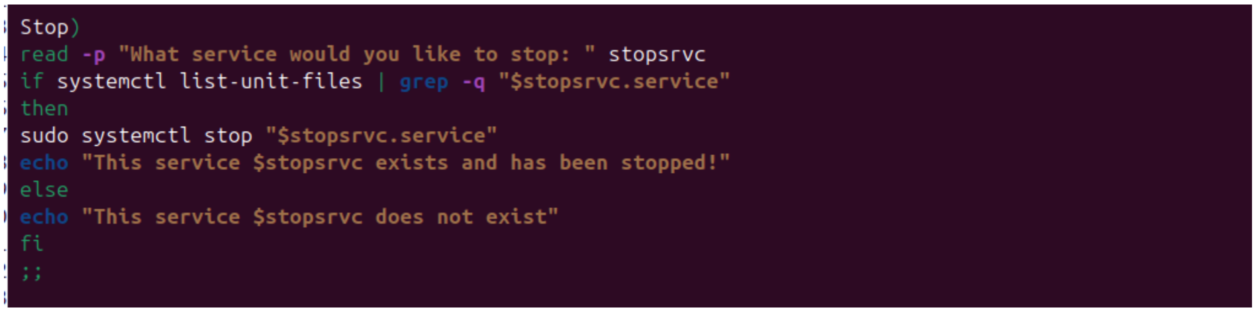


```
Start)
read -p "What service would you like to start: " startsrvc
 if systemctl list-unit-files | grep -q "$startsrvc.service"
then
sudo systemctl start "$startsrvc.service"
echo "This serice $startsrvc exists and has been started!"
else
echo "This service $startsrvc does not exist"
fi
;;
```

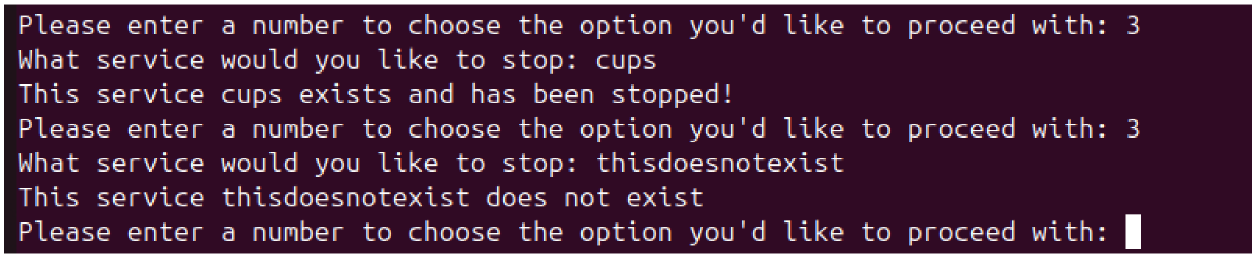


```
1) ActiveStatus
2) Start
3) Stop
Please enter a number to choose the option you'd like to proceed with: 2
What service would you like to start: cups
[sudo] password for adil:
This serice cups exists and has been started!
Please enter a number to choose the option you'd like to proceed with: 2
What service would you like to start: notAservice
This service notAservice does not exist
Please enter a number to choose the option you'd like to proceed with:
```

5-The third case Stop as the name implies stops a service that the user types with systemctl stop and the name of the service, although it will once again check if it even exists and the first place before it stops it with systemctl list-unit-files and the grep that filters out the name of that specific service again and will let the user know if it stops with the echo implying that the service has stopped or else it will tell the user it does not exist and bring them back to the options they have.

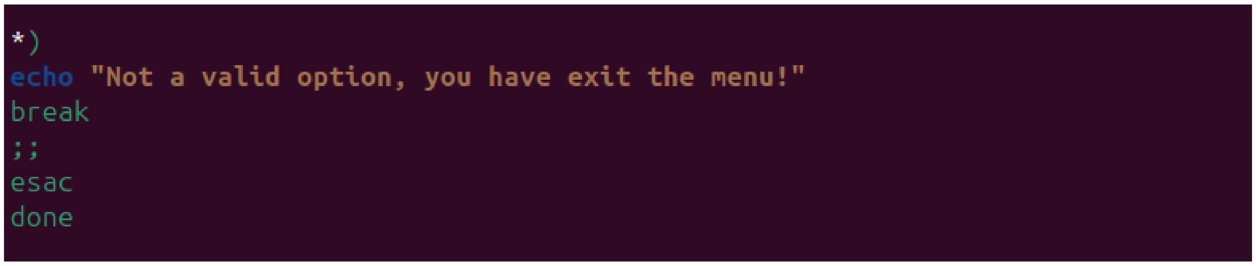


```
Stop)
read -p "What service would you like to stop: " stopsrvc
if systemctl list-unit-files | grep -q "$stopsrvc.service"
then
sudo systemctl stop "$stopsrvc.service"
echo "This service $stopsrvc exists and has been stopped!"
else
echo "This service $stopsrvc does not exist"
fi
;;
```

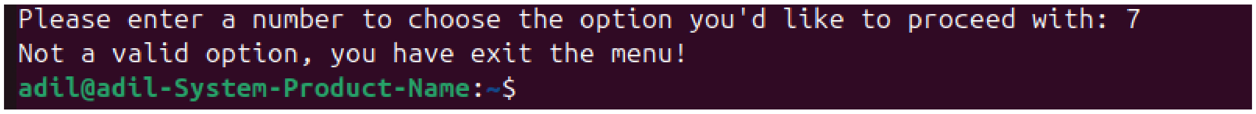


```
Please enter a number to choose the option you'd like to proceed with: 3
What service would you like to stop: cups
This service cups exists and has been stopped!
Please enter a number to choose the option you'd like to proceed with: 3
What service would you like to stop: thisdoesnotexist
This service thisdoesnotexist does not exist
Please enter a number to choose the option you'd like to proceed with:
```

6- Finally if the user types a number or letter that does not exist when selecting and option the script is gonna echo to the user that it is not valid and exit the program with the help of a break.



```
*)
echo "Not a valid option, you have exit the menu!"
break
;;
esac
done
```

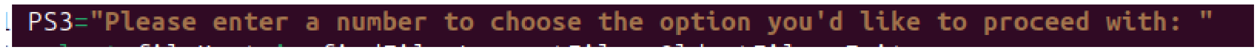


```
Please enter a number to choose the option you'd like to proceed with: 7
Not a valid option, you have exit the menu!
adil@adil-System-Product-Name:~$
```

File management:

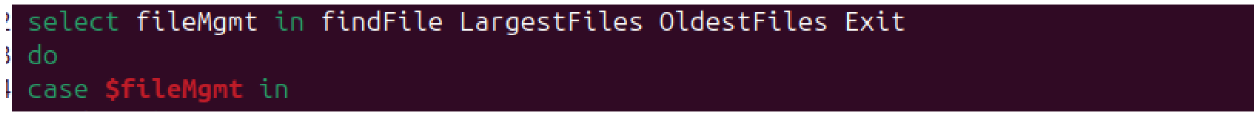
1- The use of PS3 was invoked to prompt a message to the user in particular to which option they'd like to pick just like in service management.

Example: PS3="Please enter a number to choose the option you'd like to proceed with:”



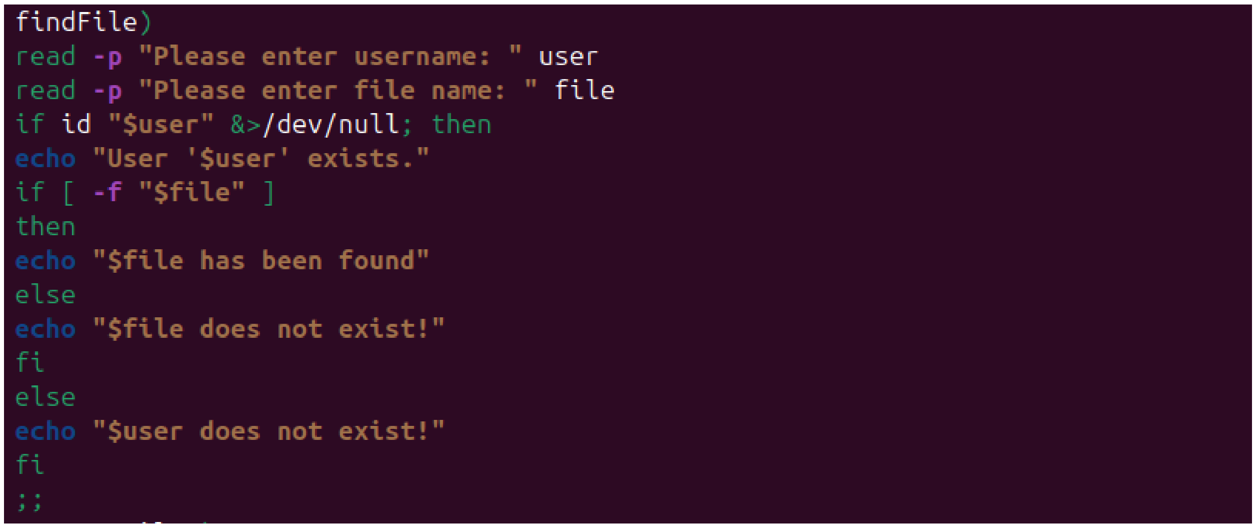
```
PS3="Please enter a number to choose the option you'd like to proceed with: "
```

2- Select statement is used to determine the name of the switch case we’re gonna use and to show off the options the user could pick such as findFIle, LargestFiles, OldestFiles and Exit.

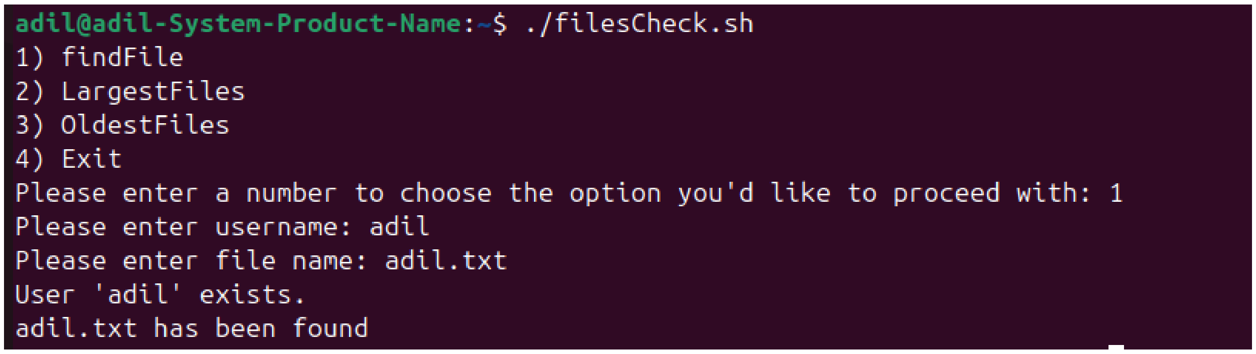


```
select fileMgmt in findFile LargestFiles OldestFiles Exit
do
case $fileMgmt in
```

3- The first case findFile asks for the username and fileName to check for both of their existence with the help of if and else statements in particular for the files it’s going to be [-f $file] to see if it exists and if it does it will throw an eco indicating it exists and for user it’s “$user” &>/dev/null; to check.

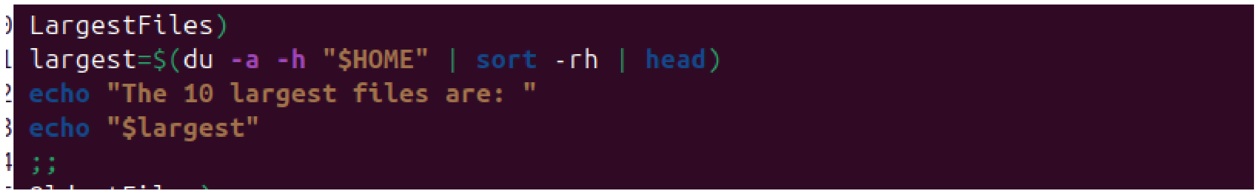


```
findFile)
read -p "Please enter username: " user
read -p "Please enter file name: " file
if id "$user" &>/dev/null; then
echo "User '$user' exists."
if [ -f "$file" ]
then
echo "$file has been found"
else
echo "$file does not exist!"
fi
else
echo "$user does not exist!"
fi
;;
```

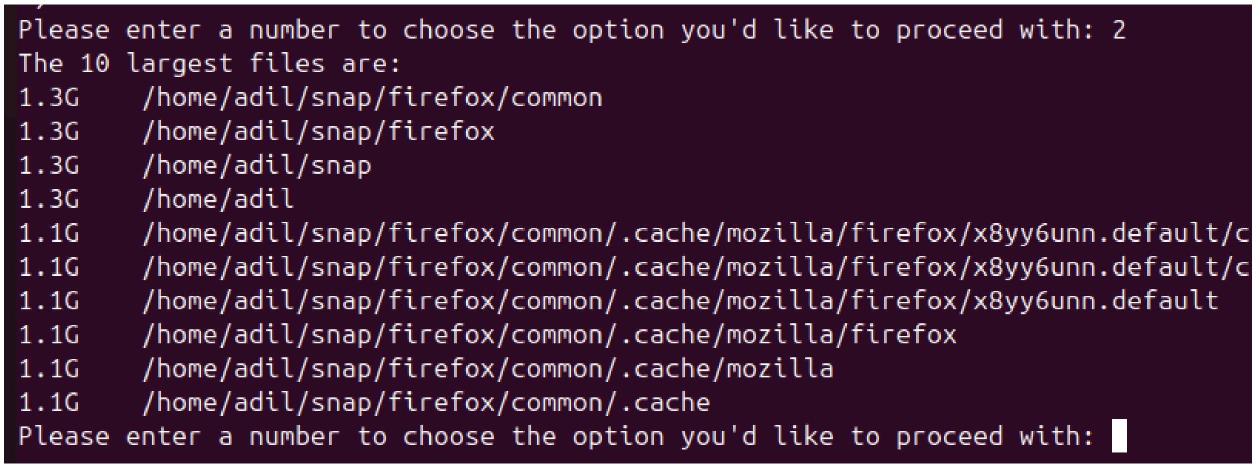


```
adil@adil-System-Product-Name:~$ ./filesCheck.sh
1) findFile
2) LargestFiles
3) OldestFiles
4) Exit
Please enter a number to choose the option you'd like to proceed with: 1
Please enter username: adil
Please enter file name: adil.txt
User 'adil' exists.
adil.txt has been found
```

4.The second case LargestFiles uses a variable in this case called largest and contains the following command : largest=$(du -a -h "$HOME" | sort -rh | head). All of these are very important as du is used for checking the disk usage, -a is used for checking the files and directories everywhere, h makes it more human readable, sort is used to sort the files and -rh is because the r sorts it in reverse order therefore the biggest to smallest, h converts the format to be easier to read and head displays the 10 biggest files.

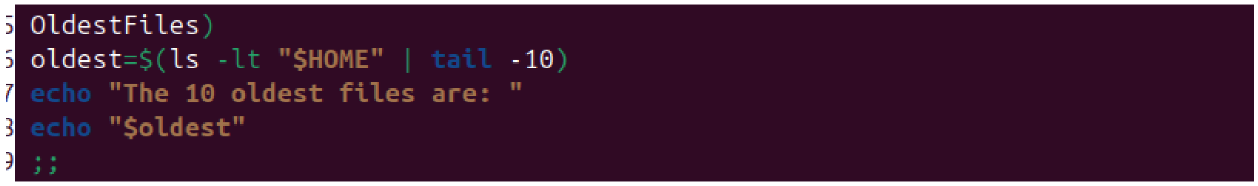


```
LargestFiles)
largest=$(du -a -h "$HOME" | sort -rh | head)
echo "The 10 largest files are: "
echo "$largest"
;;
```

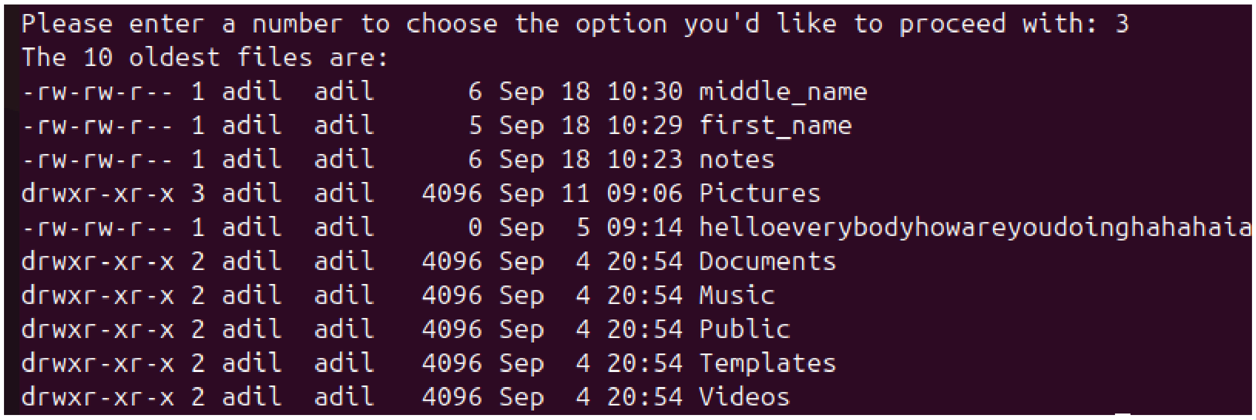


```
Please enter a number to choose the option you'd like to proceed with: 2
The 10 largest files are:
1.3G    /home/adil/snap/firefox/common
1.3G    /home/adil/snap/firefox
1.3G    /home/adil/snap
1.3G    /home/adil
1.1G    /home/adil/snap/firefox/common/.cache/mozilla/firefox/x8yy6unn.default/c
1.1G    /home/adil/snap/firefox/common/.cache/mozilla/firefox/x8yy6unn.default/c
1.1G    /home/adil/snap/firefox/common/.cache/mozilla/firefox/x8yy6unn.default
1.1G    /home/adil/snap/firefox/common/.cache/mozilla/firefox
1.1G    /home/adil/snap/firefox/common/.cache/mozilla
1.1G    /home/adil/snap/firefox/common/.cache
Please enter a number to choose the option you'd like to proceed with:
```

5- The third case OldestFiles consists of searching for the oldest files on the system and uses this in particular: oldest=$(ls -lt "$HOME" | tail -10). The variable oldest stores a sequence of commands such as ls -lt which lists all files and information by their last modification date in the home directory and then pipes with tail -10 which shows the 10 oldest files.

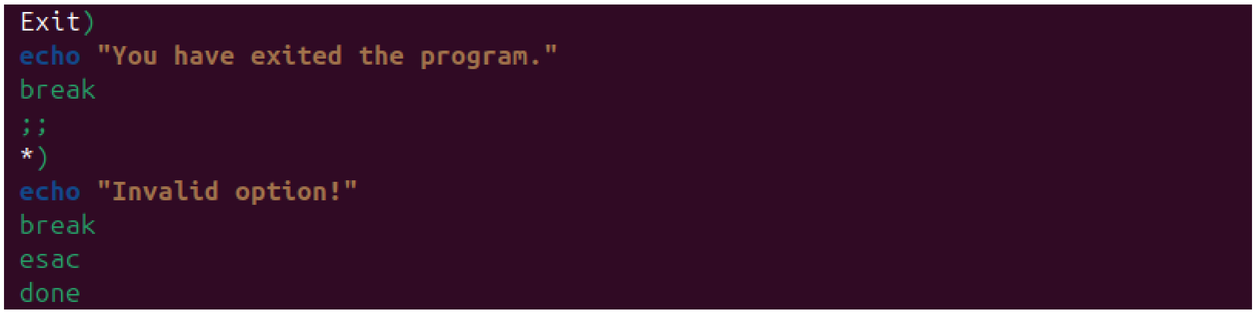


```
OldestFiles)
oldest=$(ls -lt "$HOME" | tail -10)
echo "The 10 oldest files are: "
echo "$oldest"
;;
```



```
Please enter a number to choose the option you'd like to proceed with: 3
The 10 oldest files are:
-rw-rw-r-- 1 adil  adil      6 Sep 18 10:30 middle_name
-rw-rw-r-- 1 adil  adil      5 Sep 18 10:29 first_name
-rw-rw-r-- 1 adil  adil      6 Sep 18 10:23 notes
drwxr-xr-x 3 adil  adil   4096 Sep 11 09:06 Pictures
-rw-rw-r-- 1 adil  adil      0 Sep  5 09:14 helloeverybodyhowareyoudoinghahahaia
drwxr-xr-x 2 adil  adil   4096 Sep  4 20:54 Documents
drwxr-xr-x 2 adil  adil   4096 Sep  4 20:54 Music
drwxr-xr-x 2 adil  adil   4096 Sep  4 20:54 Public
drwxr-xr-x 2 adil  adil   4096 Sep  4 20:54 Templates
drwxr-xr-x 2 adil  adil   4096 Sep  4 20:54 Videos
```

6- The fourth case exit is used so that the user can exit the program and the last case is used to handle invalid numbers or letters when prompting the user for what option they’d like to use.



```
Exit)
echo "You have exited the program."
break
;;
*)
echo "Invalid option!"
break
esac
done
```

REFERENCES:

Links to the resources used:

1- https://www.geeksforgeeks.org/linux-unix/linux-tutorial/
2- https://www.w3schools.com/bash/bash_commands.php
3- https://askubuntu.com/