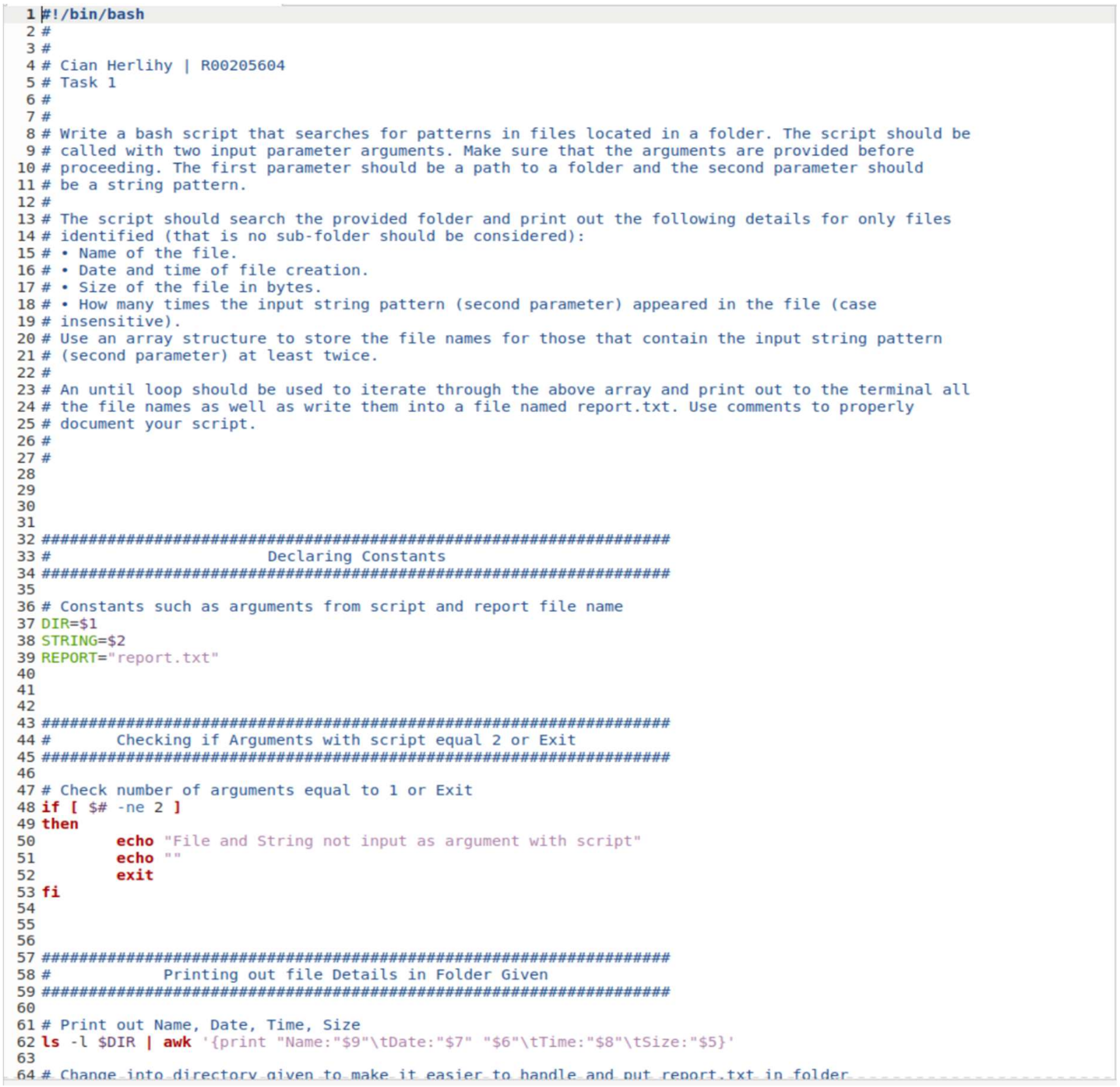
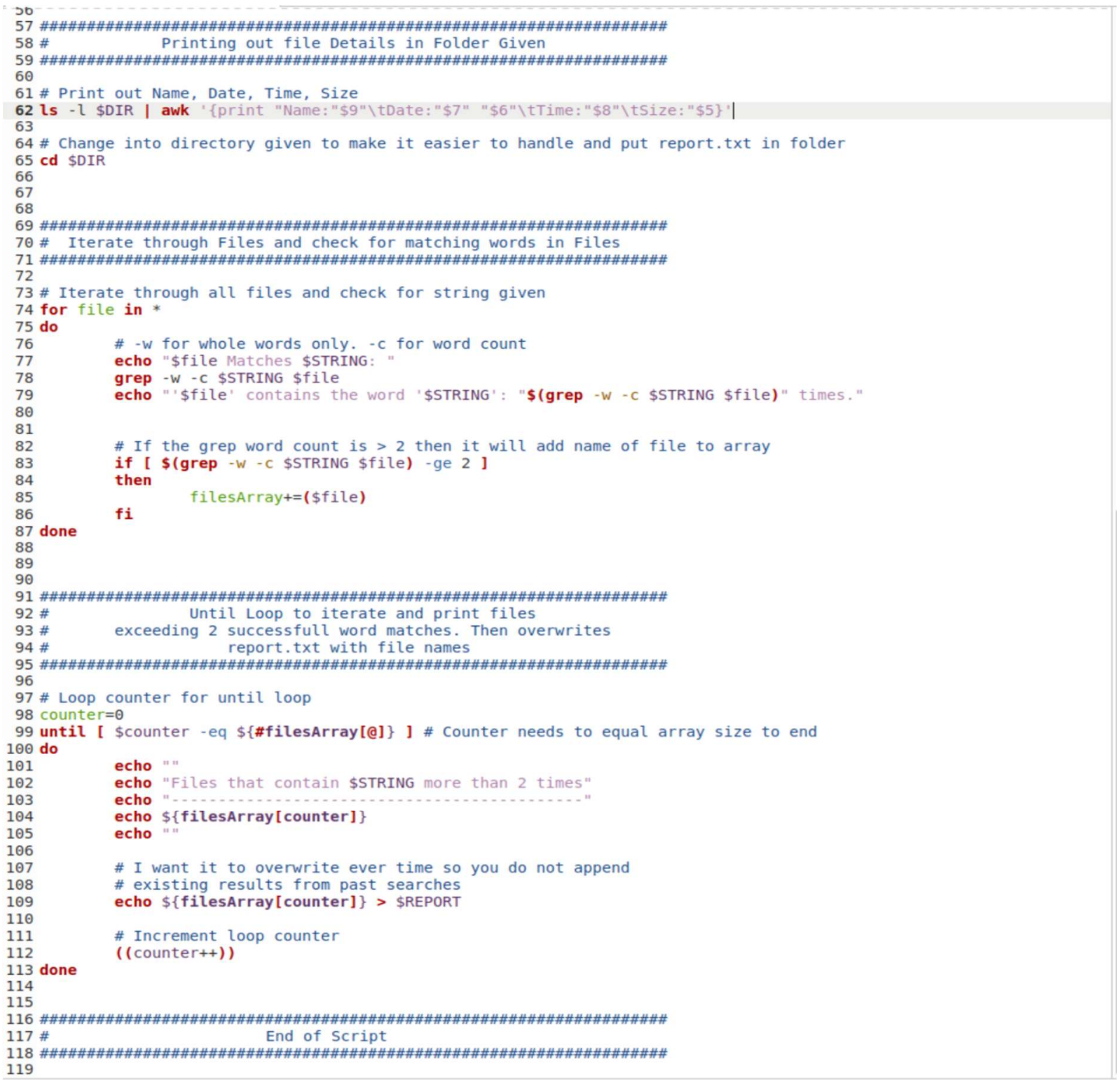
Assignment 1 – System Scripting

Cian Herlihy | R00205604

Task 1



```
#!/bin/bash
#
#
# Cian Herlihy | R00205604
# Task 1
#
#
# Write a bash script that searches for patterns in files located in a folder. The script should be
# called with two input parameter arguments. Make sure that the arguments are provided before
# proceeding. The first parameter should be a path to a folder and the second parameter should
# be a string pattern.
#
# The script should search the provided folder and print out the following details for only files
# identified (that is no sub-folder should be considered):
# • Name of the file.
# • Date and time of file creation.
# • Size of the file in bytes.
# • How many times the input string pattern (second parameter) appeared in the file (case
# insensitive).
# Use an array structure to store the file names for those that contain the input string pattern
# (second parameter) at least twice.
#
# An until loop should be used to iterate through the above array and print out to the terminal all
# the file names as well as write them into a file named report.txt. Use comments to properly
# document your script.
#
#




###################################################################
#                      Declaring Constants
###################################################################

# Constants such as arguments from script and report file name
DIR=$1
STRING=$2
REPORT="report.txt"



###################################################################
#       Checking if Arguments with script equal 2 or Exit
###################################################################

# Check number of arguments equal to 1 or Exit
if [ $# -ne 2 ]
then
        echo "File and String not input as argument with script"
        echo ""
        exit
fi



###################################################################
#            Printing out file Details in Folder Given
###################################################################

# Print out Name, Date, Time, Size
ls -l $DIR | awk '{print "Name:"$9"\tDate:"$7" "$6"\tTime:"$8"\tSize:"$5}'

# Change into directory given to make it easier to handle and put report.txt in folder
```



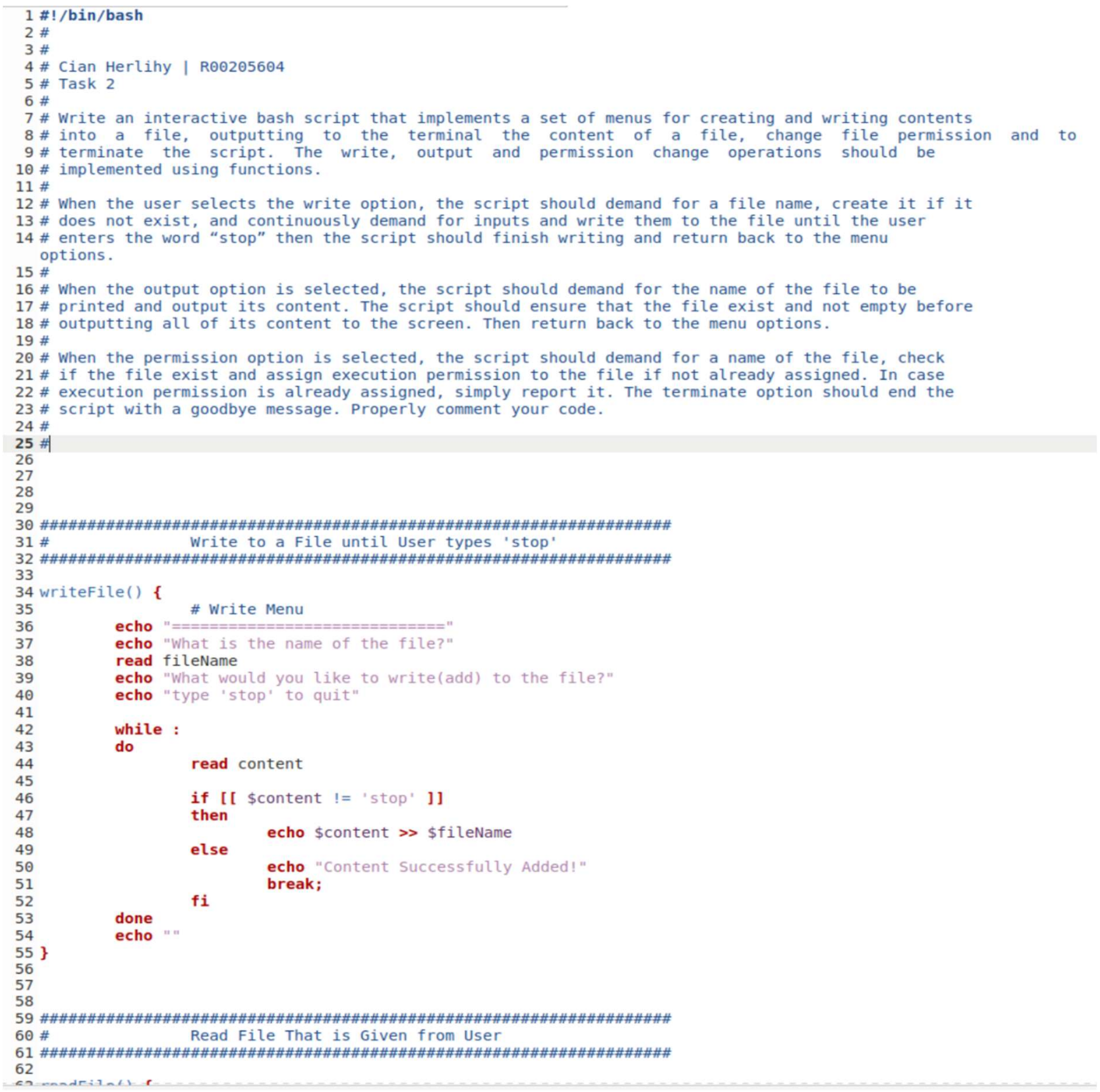
```
56
57 ######################################################################
58 #            Printing out file Details in Folder Given
59 ######################################################################
60
61 # Print out Name, Date, Time, Size
62 ls -l $DIR | awk '{print "Name:"$9"\tDate:"$7" "$6"\tTime:"$8"\tSize:"$5}'
63
64 # Change into directory given to make it easier to handle and put report.txt in folder
65 cd $DIR
66
67
68
69 ######################################################################
70 #  Iterate through Files and check for matching words in Files
71 ######################################################################
72
73 # Iterate through all files and check for string given
74 for file in *
75 do
76         # -w for whole words only. -c for word count
77         echo "$file Matches $STRING: "
78         grep -w -c $STRING $file
79         echo "'$file' contains the word '$STRING': "$(grep -w -c $STRING $file)" times."
80
81
82         # If the grep word count is > 2 then it will add name of file to array
83         if [ $(grep -w -c $STRING $file) -ge 2 ]
84         then
85                 filesArray+=($file)
86         fi
87 done
88
89
90
91 ######################################################################
92 #               Until Loop to iterate and print files
93 #       exceeding 2 successfull word matches. Then overwrites
94 #                   report.txt with file names
95 ######################################################################
96
97 # Loop counter for until loop
98 counter=0
99 until [ $counter -eq ${#filesArray[@]} ] # Counter needs to equal array size to end
100 do
101         echo ""
102         echo "Files that contain $STRING more than 2 times"
103         echo "-----------------------------------------"
104         echo ${filesArray[counter]}
105         echo ""
106
107         # I want it to overwrite ever time so you do not append
108         # existing results from past searches
109         echo ${filesArray[counter]} > $REPORT
110
111         # Increment loop counter
112         ((counter++))
113 done
114
115
116 ######################################################################
117 #                       End of Script
118 ######################################################################
119
```

For task 1, I needed to take in 2 arguments, so I made sure to check if only 2 arguments were given. If it was not exactly 2 then it will give an error. I declared constants for the arguments because it gives the script more understanding than just seeing '$1' and '$2'.

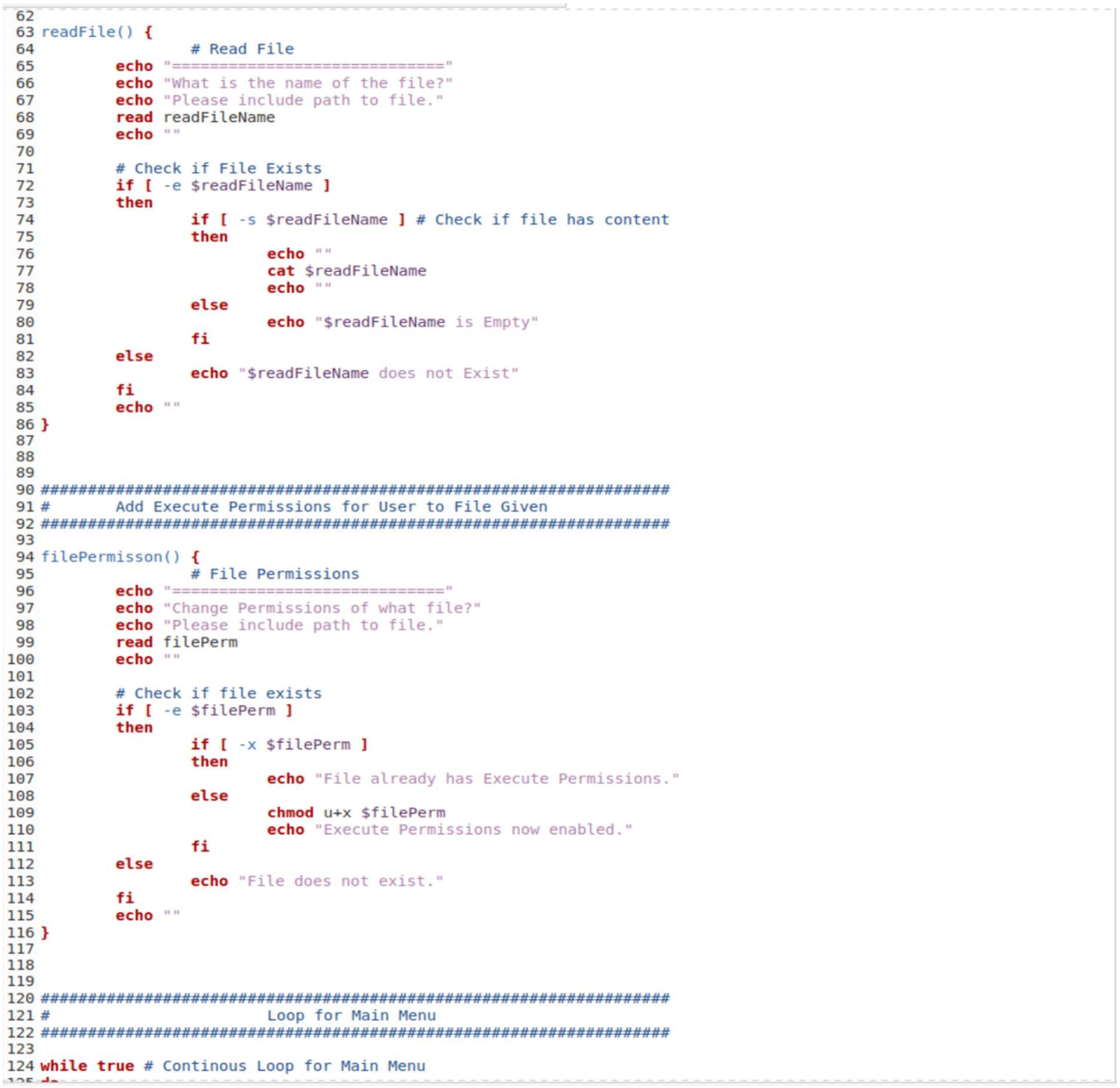
I iterate through the files using a for loop and in this for loop I check within the files using grep if the file contains any matching patterns to the string and I made sure to add -w next to -c to match the whole word. For example, I caught 6 matches in my file with the word test when there was only 5. This was because it counted the word 'testing' as a match for 'test'. I did not want that outcome, so I fixed that error. On lines 77 -79 could be excluded since you do not need to print out each files word count but for testing purposes, I had it printing but simply commenting this out would work just as good.

I then have gathered all the files that contain more than or equal to 2 matching word counts to an array and I iterate through the array to then print off what files met this requirement. I then redirected the output to a report.txt file but I purposefully left it as overwrite so I did not need to clear it every time and get mixed up results with past running of the script.

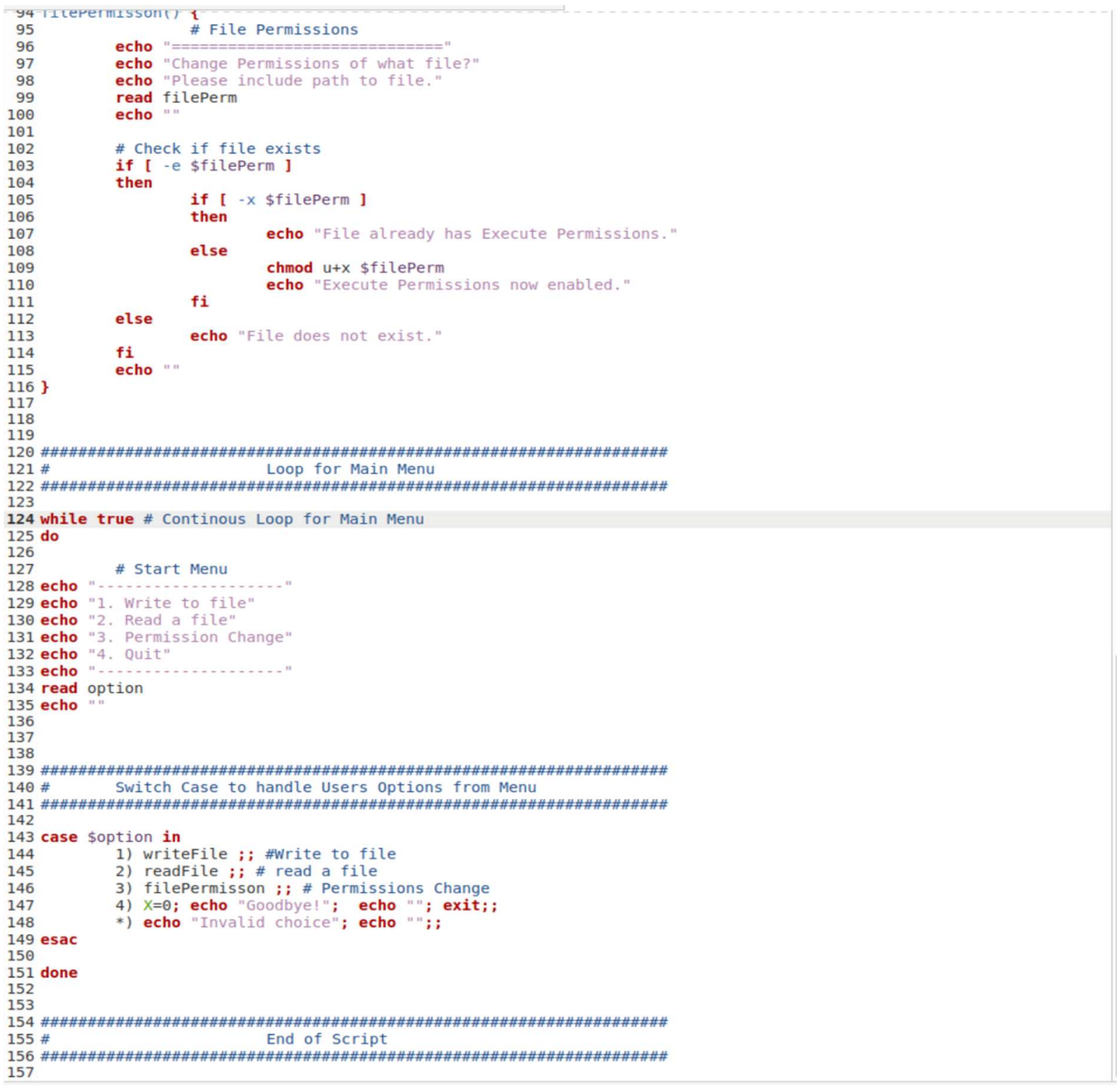
Task 2



```bash
#!/bin/bash
#
#
# Cian Herlihy | R00205604
# Task 2
#
# Write an interactive bash script that implements a set of menus for creating and writing contents
# into  a  file,  outputting  to  the  terminal  the  content  of  a  file,  change  file  permission  and  to
# terminate  the  script.  The  write,  output  and  permission  change  operations  should  be
# implemented using functions.
#
# When the user selects the write option, the script should demand for a file name, create it if it
# does not exist, and continuously demand for inputs and write them to the file until the user
# enters the word “stop” then the script should finish writing and return back to the menu options.
#
# When the output option is selected, the script should demand for the name of the file to be
# printed and output its content. The script should ensure that the file exist and not empty before
# outputting all of its content to the screen. Then return back to the menu options.
#
# When the permission option is selected, the script should demand for a name of the file, check
# if the file exist and assign execution permission to the file if not already assigned. In case
# execution permission is already assigned, simply report it. The terminate option should end the
# script with a goodbye message. Properly comment your code.
#
#




#####################################################################
#               Write to a File until User types 'stop'
#####################################################################

writeFile() {
                # Write Menu
        echo "=============================="
        echo "What is the name of the file?"
        read fileName
        echo "What would you like to write(add) to the file?"
        echo "type 'stop' to quit"

        while :
        do
                read content

                if [[ $content != 'stop' ]]
                then
                        echo $content >> $fileName
                else
                        echo "Content Successfully Added!"
                        break;
                fi
        done
        echo ""
}



#####################################################################
#               Read File That is Given from User
#####################################################################

```



```

readFile() {
                # Read File
        echo "=============================="
        echo "What is the name of the file?"
        echo "Please include path to file."
        read readFileName
        echo ""

        # Check if File Exists
        if [ -e $readFileName ]
        then
                if [ -s $readFileName ] # Check if file has content
                then
                        echo ""
                        cat $readFileName
                        echo ""
                else
                        echo "$readFileName is Empty"
                fi
        else
                echo "$readFileName does not Exist"
        fi
        echo ""
}



####################################################################
#       Add Execute Permissions for User to File Given
####################################################################

filePermisson() {
                # File Permissions
        echo "=============================="
        echo "Change Permissions of what file?"
        echo "Please include path to file."
        read filePerm
        echo ""

        # Check if file exists
        if [ -e $filePerm ]
        then
                if [ -x $filePerm ]
                then
                        echo "File already has Execute Permissions."
                else
                        chmod u+x $filePerm
                        echo "Execute Permissions now enabled."
                fi
        else
                echo "File does not exist."
        fi
        echo ""
}



####################################################################
#                       Loop for Main Menu
####################################################################

while true # Continous Loop for Main Menu
```



```bash
filePermisson() {
		# File Permissions
	echo "============================="
	echo "Change Permissions of what file?"
	echo "Please include path to file."
	read filePerm
	echo ""

	# Check if file exists
	if [ -e $filePerm ]
	then
		if [ -x $filePerm ]
		then
			echo "File already has Execute Permissions."
		else
			chmod u+x $filePerm
			echo "Execute Permissions now enabled."
		fi
	else
		echo "File does not exist."
	fi
	echo ""
}



###################################################################
#                        Loop for Main Menu
###################################################################

while true # Continous Loop for Main Menu
do

	# Start Menu
echo "--------------------"
echo "1. Write to file"
echo "2. Read a file"
echo "3. Permission Change"
echo "4. Quit"
echo "--------------------"
read option
echo ""



###################################################################
#        Switch Case to handle Users Options from Menu
###################################################################

case $option in
	1) writeFile ;; #Write to file
	2) readFile ;; # read a file
	3) filePermisson ;; # Permissions Change
	4) X=0; echo "Goodbye!";  echo ""; exit;;
	*) echo "Invalid choice"; echo "";;
esac

done


###################################################################
#                        End of Script
###################################################################

```

The task 2 script should allow the user to write to a file, read a file and change the permissions to allow for a user to have execute permissions. I accomplished this by using a switch case for my menu. I echoed out the menu and then let the user select using numbers 1-4. I feel like this is a very simple way.

I then use a switch case statement to call on functions to do the work that was intended for that option. I started with option 1 being “Write to a File” which then calls on the function write File. This function is located above the switch case statements because bash would not have seen the function if it was below it causing an error.

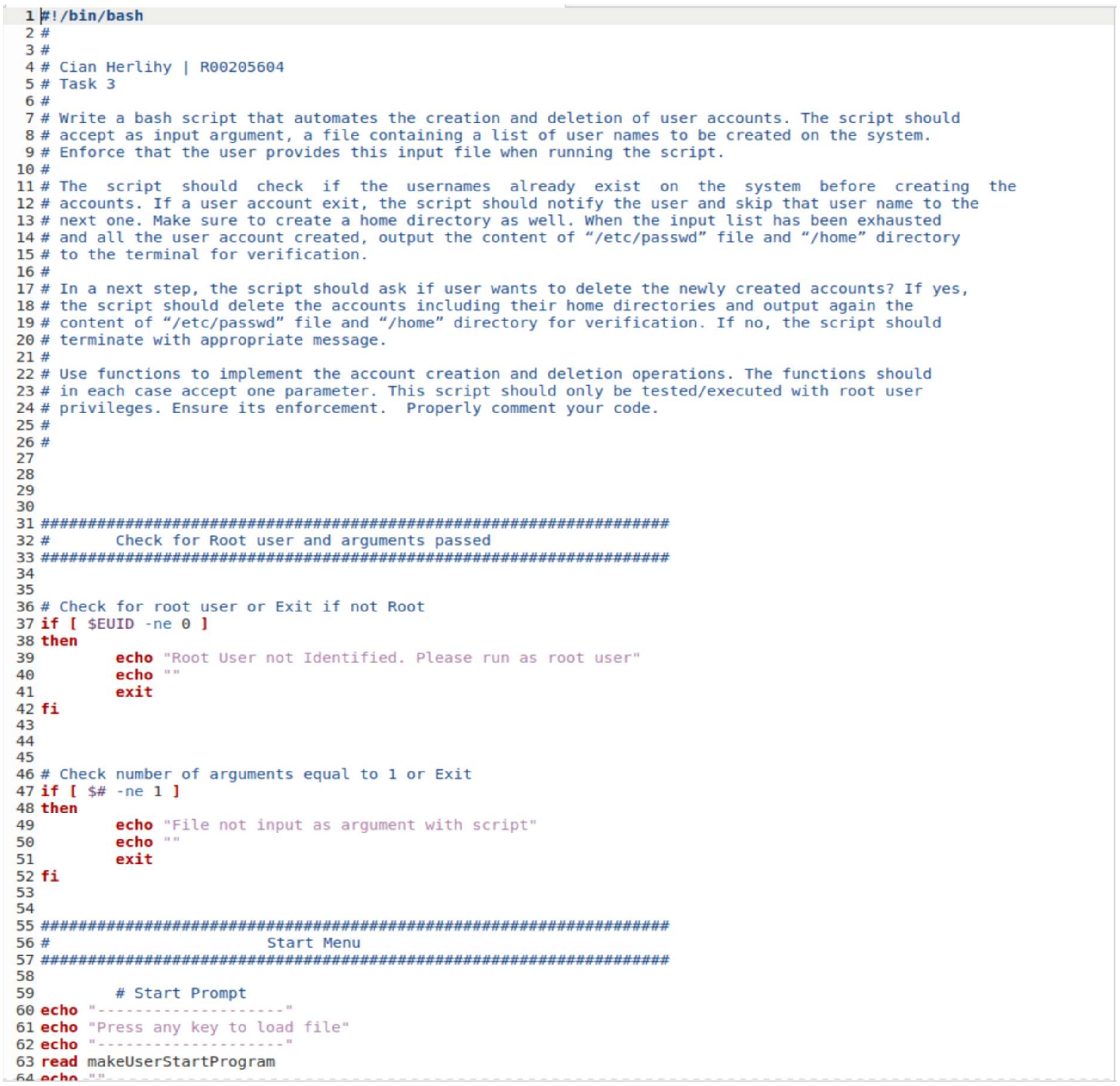
In this function I append content to the file if it already exists or it will create the file if it does not exist. The user can keep typing while skipping some lines too. The way to stop the script from looping when you are done inserting information into a file is to type ‘stop’ on its own line. It will not recognise the word in the middle of the line allowing you to be unrestricted from using that word.

The second function then reads a file and firstly it checks if it exists. If it doesn’t exist, then it will try read the file. If it is empty, it will prompt the user that it is empty. Otherwise, it will read the file to the terminal.

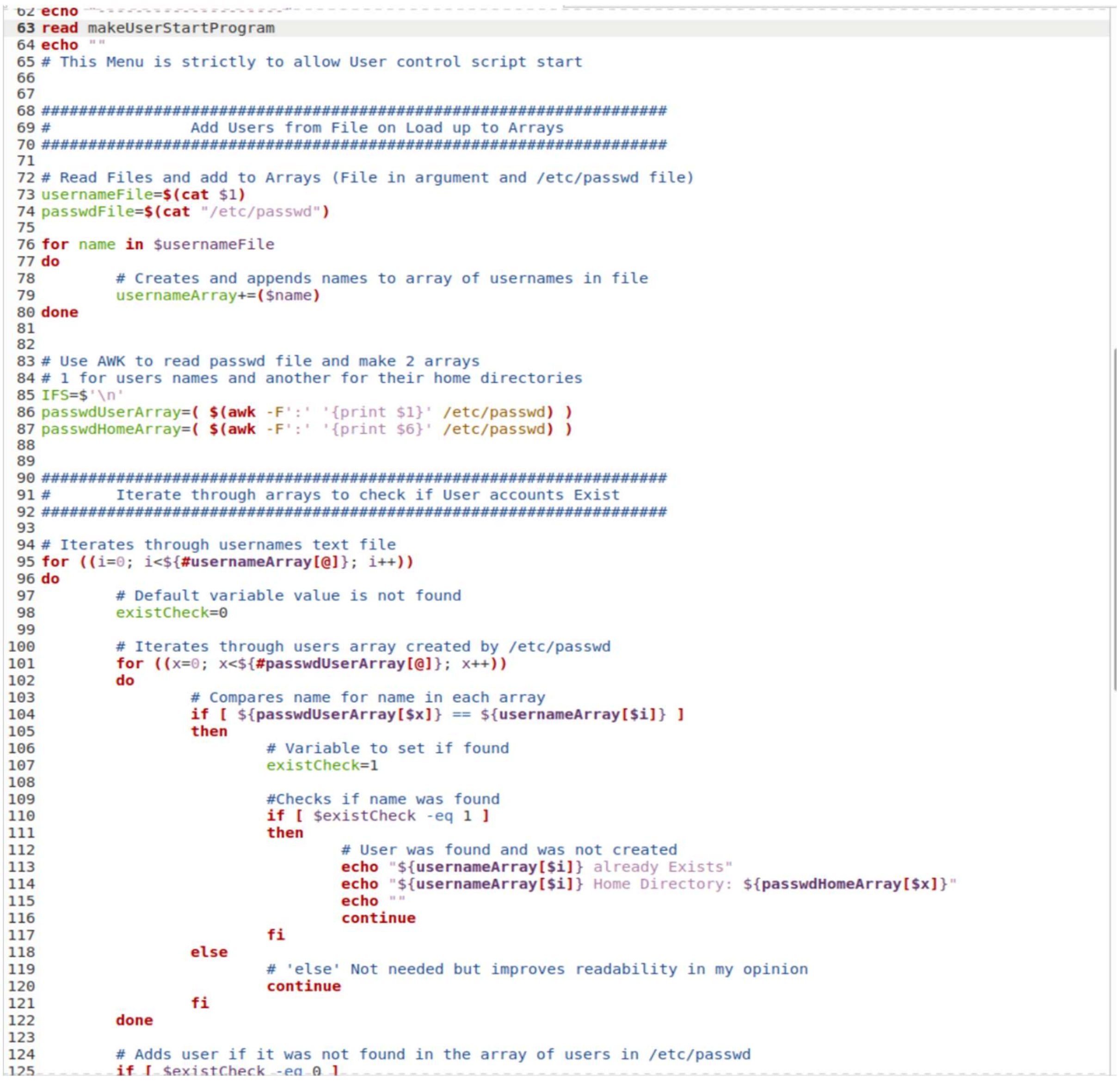
3rd function then changes the permissions of a given file to allow execute permissions for the user. If the file already has execute permissions, it will prompt the user that it already has the permissions.

Lastly the program exits the infinite menu loop by choosing 4 as their option. If they choose an option that doesn't exist like '5' then it will prompt the user that it is not a valid option and then loop it again for a better choice.

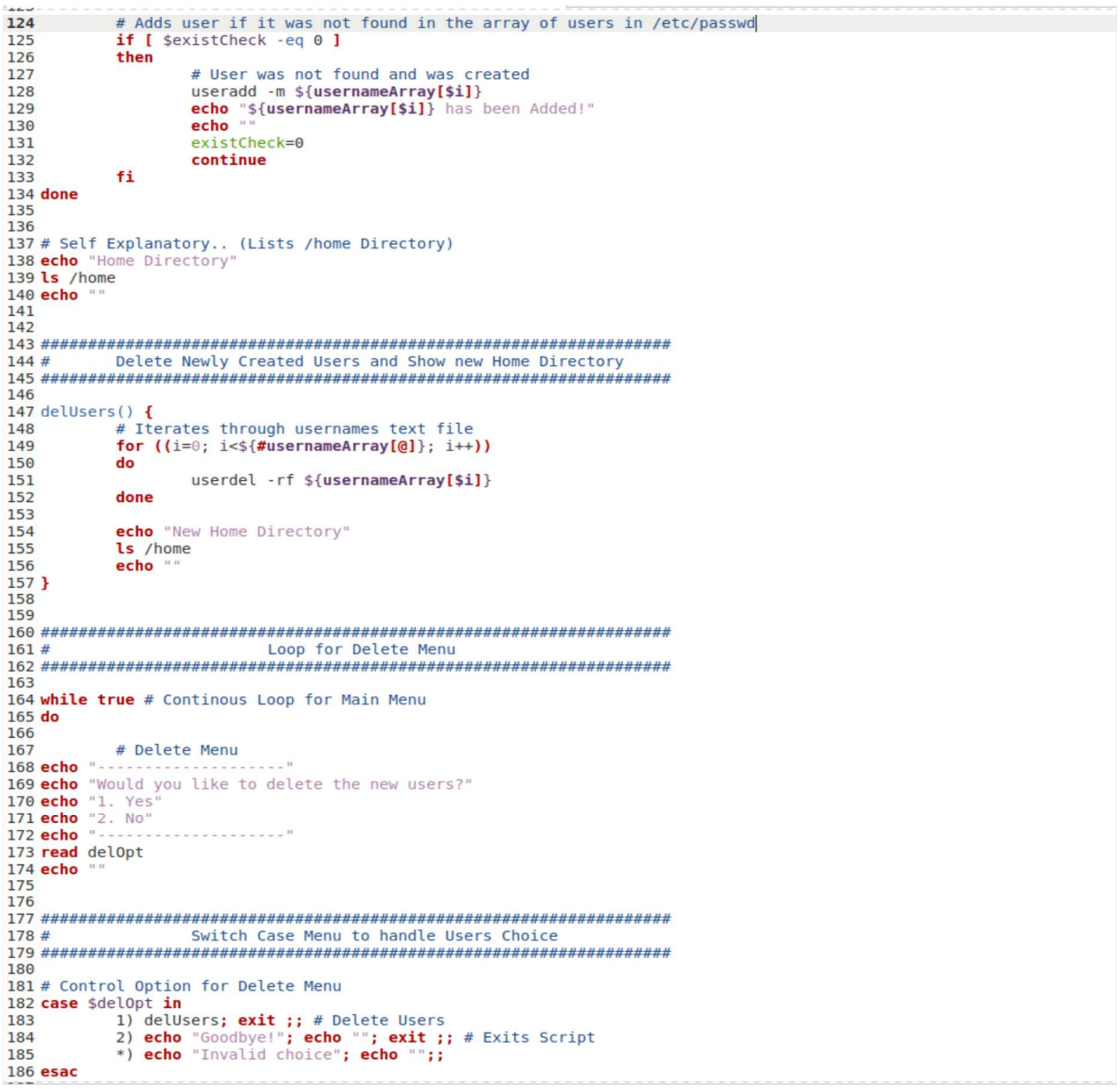
Task 3



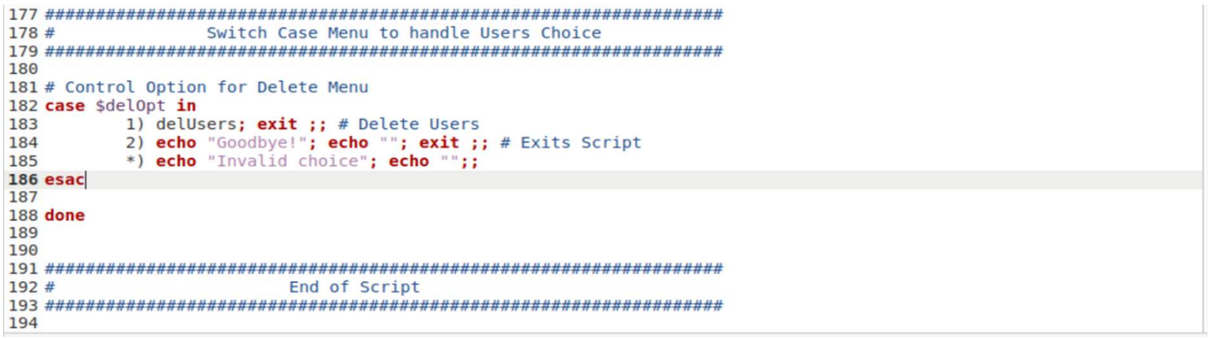
```
#!/bin/bash
#
#
# Cian Herlihy | R00205604
# Task 3
#
# Write a bash script that automates the creation and deletion of user accounts. The script should
# accept as input argument, a file containing a list of user names to be created on the system.
# Enforce that the user provides this input file when running the script.
#
# The  script  should  check  if  the  usernames  already  exist  on  the  system  before  creating  the
# accounts. If a user account exit, the script should notify the user and skip that user name to the
# next one. Make sure to create a home directory as well. When the input list has been exhausted
# and all the user account created, output the content of "/etc/passwd" file and "/home" directory
# to the terminal for verification.
#
# In a next step, the script should ask if user wants to delete the newly created accounts? If yes,
# the script should delete the accounts including their home directories and output again the
# content of "/etc/passwd" file and "/home" directory for verification. If no, the script should
# terminate with appropriate message.
#
# Use functions to implement the account creation and deletion operations. The functions should
# in each case accept one parameter. This script should only be tested/executed with root user
# privileges. Ensure its enforcement.  Properly comment your code.
#
#




###################################################################
#       Check for Root user and arguments passed
###################################################################


# Check for root user or Exit if not Root
if [ $EUID -ne 0 ]
then
        echo "Root User not Identified. Please run as root user"
        echo ""
        exit
fi



# Check number of arguments equal to 1 or Exit
if [ $# -ne 1 ]
then
        echo "File not input as argument with script"
        echo ""
        exit
fi


###################################################################
#                       Start Menu
###################################################################

        # Start Prompt
echo "--------------------"
echo "Press any key to load file"
echo "--------------------"
read makeUserStartProgram
echo ""
```



```
echo "-------------------"
read makeUserStartProgram
echo ""
# This Menu is strictly to allow User control script start


#####################################################################
#               Add Users from File on Load up to Arrays
#####################################################################

# Read Files and add to Arrays (File in argument and /etc/passwd file)
usernameFile=$(cat $1)
passwdFile=$(cat "/etc/passwd")

for name in $usernameFile
do
        # Creates and appends names to array of usernames in file
        usernameArray+=($name)
done


# Use AWK to read passwd file and make 2 arrays
# 1 for users names and another for their home directories
IFS=$'\n'
passwdUserArray=( $(awk -F':' '{print $1}' /etc/passwd) )
passwdHomeArray=( $(awk -F':' '{print $6}' /etc/passwd) )


#####################################################################
#       Iterate through arrays to check if User accounts Exist
#####################################################################

# Iterates through usernames text file
for ((i=0; i<${#usernameArray[@]}; i++))
do
        # Default variable value is not found
        existCheck=0

        # Iterates through users array created by /etc/passwd
        for ((x=0; x<${#passwdUserArray[@]}; x++))
        do
                # Compares name for name in each array
                if [ ${passwdUserArray[$x]} == ${usernameArray[$i]} ]
                then
                        # Variable to set if found
                        existCheck=1

                        #Checks if name was found
                        if [ $existCheck -eq 1 ]
                        then
                                # User was found and was not created
                                echo "${usernameArray[$i]} already Exists"
                                echo "${usernameArray[$i]} Home Directory: ${passwdHomeArray[$x]}"
                                echo ""
                                continue
                        fi
                else
                        # 'else' Not needed but improves readability in my opinion
                        continue
                fi
        done

        # Adds user if it was not found in the array of users in /etc/passwd
        if [ $existCheck -eq 0 ]
```



```
        # Adds user if it was not found in the array of users in /etc/passwd
        if [ $existCheck -eq 0 ]
        then
                # User was not found and was created
                useradd -m ${usernameArray[$i]}
                echo "${usernameArray[$i]} has been Added!"
                echo ""
                existCheck=0
                continue
        fi
done


# Self Explanatory.. (Lists /home Directory)
echo "Home Directory"
ls /home
echo ""


####################################################################
#       Delete Newly Created Users and Show new Home Directory
####################################################################

delUsers() {
        # Iterates through usernames text file
        for ((i=0; i<${#usernameArray[@]}; i++))
        do
                userdel -rf ${usernameArray[$i]}
        done

        echo "New Home Directory"
        ls /home
        echo ""
}


####################################################################
#                       Loop for Delete Menu
####################################################################

while true # Continous Loop for Main Menu
do

        # Delete Menu
echo "--------------------"
echo "Would you like to delete the new users?"
echo "1. Yes"
echo "2. No"
echo "--------------------"
read delOpt
echo ""


####################################################################
#               Switch Case Menu to handle Users Choice
####################################################################

# Control Option for Delete Menu
case $delOpt in
        1) delUsers; exit ;; # Delete Users
        2) echo "Goodbye!"; echo ""; exit ;; # Exits Script
        *) echo "Invalid choice"; echo "";;
esac
```



```
####################################################################
#               Switch Case Menu to handle Users Choice
####################################################################

# Control Option for Delete Menu
case $delOpt in
        1) delUsers; exit ;; # Delete Users
        2) echo "Goodbye!"; echo ""; exit ;; # Exits Script
        *) echo "Invalid choice"; echo "";;
esac
```

```

done


####################################################################
#                       End of Script
####################################################################

```

For Task 3 I start off by checking if the user is root and then if it has 1 argument passed and only 1. Once it meets these requirements it can then move on to the actual functionality of the code. I auto load up the file given as an argument and create the users with the names in the file. I make the user press a key to proceed with the loading to give the user more control. They could exit the program by pressing 'CTRL' + 'C' and this would exit the script abruptly. When the user presses a button, it will proceed to load and create all the users. It will first check if there's any users with that name already existing and if so, then do not try and create another with the same name.

If it has added the user or found a duplicate, then it will notify the user. The user will then be prompted with the delete menu. This give them the option of deleting the users that were just added to the system all while displaying the /home folder to prove the users were added correctly. If the user selects yes, then it will delete the users off the system and exit the script. If they want to keep the users, then they select no, and it exits for them. I have a switch case statement in a while loop that filters out invalid answers/selections from the menu.