

A vibrant, cartoon-style illustration of a group of hackers. They are depicted as figures in various colored hoodies (green, pink, yellow, orange, blue) with penguin logos on the chest. Some are standing and holding mobile devices, while others are seated at desks with multiple computer monitors. The monitors display code, system logs, and a Linux penguin. The scene is filled with tech-related icons like a floppy disk, a location pin, and a USB drive. The overall atmosphere is one of intense digital activity.

A Document series by VIEH Group

Shell Scripting

Power of linux command line

VIEH GROUP

Disclaimer

Dear readers,

This document is provided by VIEH Group for educational purposes only. While we strive for accuracy and reliability, we make no warranties or representations regarding the completeness, accuracy, or usefulness of the information presented herein. Any reliance you place on this document is at your own risk. VIEH Group shall not be liable for any damages arising from the use of or reliance on this document. We acknowledge and appreciate the contribution of the source person.

also,

This document is not created by a professional content writer so any mistake and error is a part of great design

Happy learning !!!

This document is credited to **Unknown(Can Mail us for Credit)**, whose exceptional insights elevate its value. Their contribution is deeply appreciated, underscoring their significant role in its creation.

Our newsletter: **Cyber Arjun**

Scan QR:



Shell scripting:

A shell is a type of computer program called a command-line interpreter that lets Linux and Unix users control their operating systems with command-line interfaces. Shells allow

users to communicate efficiently and directly with their operating systems.

Objectives:

The aim of this laboratory is to learn and practice SHELL scripts by writing small SHELL programs.

The following are the primary objectives of this session:

1. SHELL keywords
2. Arithmetic in SHELL script
3. Control Structures
 - i. Decision control
 - ii. Repetition control
4. More UNIX commands
5. Executing commands during login time

Handling shell variables:

The shell has several variables which are automatically set whenever you login. The values

of some of these variables are stored in names which collectively are called your user environment. Any **name** defined in the user environment, can be accessed from within a shell script. To include the value of a shell variable into the environment you must **export** it.

Suppose you have a shell variable called **MY_VARIABLE** with the value **Hello World!**, and you want to use this variable in a shell script. You can access the value of a shell variable

by putting a dollar sign (\$) before the variable name.

```
$ MY_VARIABLE="Hello World!"
```

Now, if you want to access the value of **MY_VARIABLE** within a shell script, you need to export it:

```
$ export MY_VARIABLE
```

Passing arguments to the shell (with predefined variables):

Script

```
#!/bin/bash
```

```
# Usage: SS1 param1 param2 param3 param 4 param5
```

```
# Script to accept 5 numbers and display their sum.
```

```
echo the parameters passed are : $1, $2, $3, $4, $5
```

```
echo the name of the script is : $0
```

```
echo the number of parameters passed are : $#
```

```
echo the process number of this shell : $$
```

```
echo the string containing all the arguments to the shell, starting at s1 All parameters : $*
```

```
echo same as above, except when quoted : $@
```

```
((sum= $1 + $2 + $3 + $4 + $5))
```

```
echo The sum is : $sum
```

```
GNU nano 2.5.3 File: script2.sh
#!/bin/bash
# Usage: SS1 param1 param2 param3 param 4 param5
# Script to accept 5 numbers and display their sum.
echo the parameters passed are : $1, $2, $3, $4, $5
echo the name of the script is : $0
echo the number of parameters passed are : $#
echo the process number of this shell : $$
echo the string containing all the arguments to the shell, starting at s1 All parameters : $*
echo same as above, except when quoted : $@
((sum= $1 + $2 + $3 + $4 + $5))
echo The sum is : $sum

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit ^R Read File ^_ Replace ^U Uncut Text ^T To Linter ^_ Go To Line ^V Next Page

learn@ubuntu:~/Desktop$ ./script2.sh 100 985 467 7889 33245
the parameters passed are : 100, 985, 467, 7889, 33245
the name of the script is : ./script2.sh
the number of parameters passed are : 5
the process number of this shell : 14018
the string containing all the arguments to the shell, starting at s1 All parameters : 100 985 467 7889 3
3245
same as above, except when quoted : 100 985 467 7889 33245
The sum is : 42686
```

Why need of shift command?

In shell scripting, when you run a script and pass arguments to it, those arguments are stored in special variables called "positional parameters." You can access these

parameters using **\$1**, **\$2**, **\$3**, and so on, where **\$1** represents the first argument, **\$2** represents the second argument, and so forth.

However, there's a limitation: the shell only recognizes positional parameters up to **\$9**. If you pass more than 9 arguments to a script, you can't access them directly using **\$10**, **\$11**, and so on.

This is where the **shift** command comes into play. It allows you to shift the positional parameters, effectively "moving" them one position to the left. When you use **shift**, the value of **\$2** becomes **\$1**, the value of **\$3** becomes **\$2**, and so on. The first positional parameter **\$1** is lost in this process.

```
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script3.sh

#!/bin/bash

#Usage: SS3 param1 param2 param3 param4 param5 param6 param7 param8 param9 param10 param11 param12
echo "arg1=$1 arg2=$2 arg3=$3"

shift

echo "After shfiting"

echo "arg1=$1 arg2=$2 arg3=$3"
shift

echo "After shfting"

echo "arg1=$1 arg2=$2 arg3=$3"

[ Read 21 lines
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line ^V Next Page

learn@ubuntu:~/Desktop$ ./script3.sh 1444 555 888 55 5 6 4 5 6
arg1=1444 arg2=555 arg3=888
After shfiting
arg1=555 arg2=888 arg3=55
After shfting
arg1=888 arg2=55 arg3=5
After shfiting
arg1=55 arg2=5 arg3=6
```

Sum of all with more than 9

arguments: `#!/bin/bash`

`# Initialize a variable to hold the sum`

`sum=0`

`# Loop through all the arguments`

`for arg in "$@"; do`

`# Add each argument to the sum`

`sum=$((sum + arg))`

`done`


```
# Print the sum
```

```
echo "The sum of all arguments is: $sum"
```



```
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script3.sh

#!/bin/bash

# Initialize a variable to hold the sum
sum=0

# Loop through all the arguments
for arg in "$@"; do
    # Add each argument to the sum
    sum=$((sum + arg))
done

# Print the sum
echo "The sum of all arguments is: $sum"

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit ^R Read File ^_ Replace ^U Uncut Text ^T To Linter ^_ Go To Line ^V Next Page

learn@ubuntu:~/Desktop$ ./script3.sh 1444 555 888 55 5 6 4 5 6
The sum of all arguments is: 2968
```

Control Structures:

Every UNIX command returns a value on exit which the shell can interrogate. This value is held in the read-only shell variable \$? A value of 0 (zero) signifies success; anything other than 0 (zero) signifies failure.

The **test** command in UNIX is used to check conditions and returns either a 0 (zero) if the condition is true or a 1 (non-zero) if the condition is false. This result indicates success or failure respectively.

Here's how it works in simpler terms:

1. **Purpose of the test Command:** The **test** command helps you evaluate conditions in shell scripts. It checks whether a condition is true or false.

2. **Exit Status:** After running the **test** command, it sets its exit status based on whether the condition is true or false. If the condition is true, **the test** returns a 0. If the condition is false, it returns a non-zero value (typically 1).

3. **Integration with if Statement:** You can use the exit status of the **test** command directly within an **if** statement. If the condition succeeds (exit status is 0), the corresponding block of code within the **if** statement is executed. If the condition fails (exit status is non-zero), the code within the **else** block (if present) is executed.

Operators on Numeric Variables used with test command:

Operators on Numeric Variables used with test

command: -eq : equal to

-ne : not equals to

-gt : greater than

-lt : less than

-ge : greater than or equal to

-le : less than equal to

Script:

```
#!/bin/bash
```

```
# Get two variables from the user
```

```
read -p "Enter the First integer " num1
```

```
read -p "Enter the second integer " num2
```

```
# Check if num1 is equal to num2
```

```
if test "$num1" -eq "$num2"
```

```
then
```

```
echo "$num1 is equal to $num2"
```

```
else
```

```
echo "$num1 is not equal to $num2"
```

```
fi
```

```
# Check if num1 is not equal to num2
```

```
if test "$num1" -ne "$num2"
```

```
then
```

```
echo "$num1 is not equal to $num2"
```



```
else
echo "$num1 is equal to $num2"
fi

# Check if num1 is greater than num2
if test "$num1" -gt "$num2"
then
echo "$num1 is greater than $num2"
else
echo "$num1 is less than $num2"
fi

# Check if num1 is less than num2
if test "$num1" -lt "$num2"
then
echo "$num1 is less than $num2"
else
echo "$num1 is not less than $num2"
fi

# Check if num1 is greater than or equal to num2
if test "$num1" -ge "$num2"
then
echo "$num1 is greater than or equal to $num2"
else
echo "$num1 is less than $num2"
fi

# Check if num1 is less than or equal to num2
if test "$num1" -le "$num2"
```

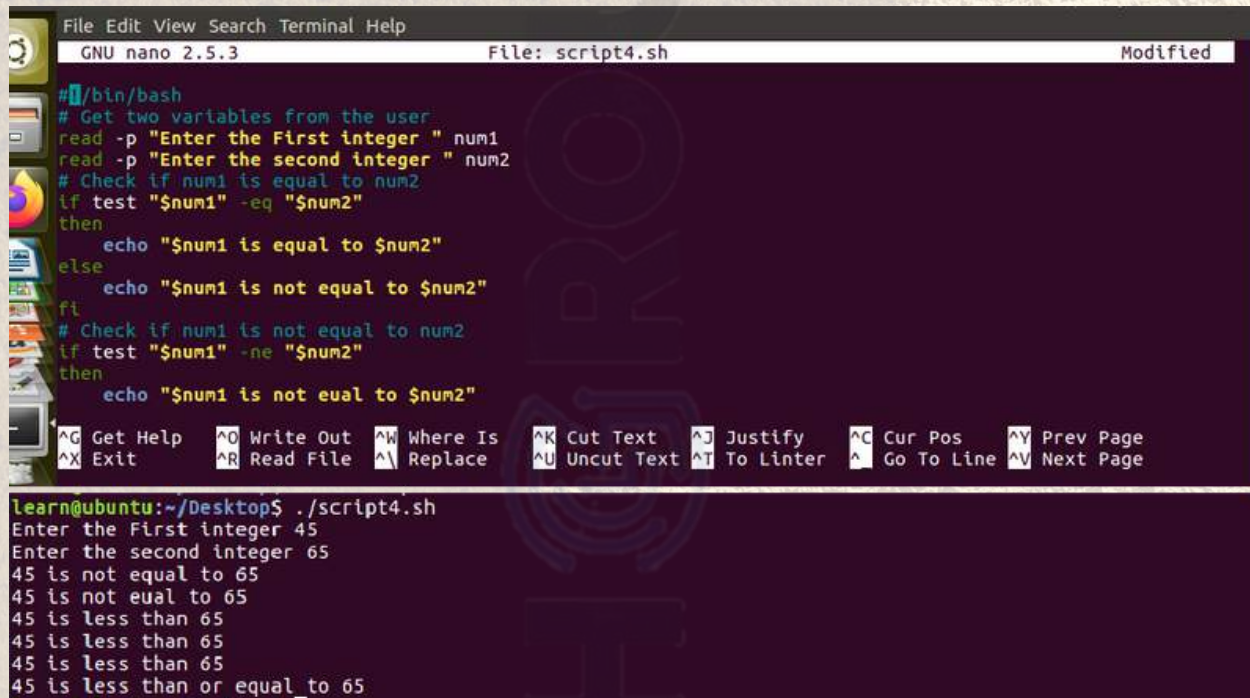

then

echo "\$num1 is less than or equal to

\$num2" else

echo "\$num1 is greater than \$num2"

fi



```
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script4.sh Modified

#!/bin/bash
# Get two variables from the user
read -p "Enter the First integer " num1
read -p "Enter the second integer " num2
# Check if num1 is equal to num2
if test "$num1" -eq "$num2"
then
    echo "$num1 is equal to $num2"
else
    echo "$num1 is not equal to $num2"
fi
# Check if num1 is not equal to num2
if test "$num1" -ne "$num2"
then
    echo "$num1 is not equal to $num2"
fi

learn@ubuntu:~/Desktop$ ./script4.sh
Enter the First integer 45
Enter the second integer 65
45 is not equal to 65
45 is not equal to 65
45 is less than 65
45 is less than 65
45 is less than 65
45 is less than 65
45 is less than or equal to 65
```

Operators on String Variables used with test command:

= : equality of strings

!= : not equal

-z : zero length string (i.e. string containing zero character i.e. null string). -n : String length is non zero.

Script:

```
#!/bin/bash
```

```
# Define two string variables
```

```
read -p "Enter the First String " str1
```



```
read -p "Enter the second String "
str2 empty_str=""

# Check if str1 is equal to str2
if test "$str1" = "$str2"
then
echo "$str1 is equal to $str2"
else
    echo "$str1 is not equal to $str2"
fi

# Check if str1 is not equal to str2
if test "$str1" != "$str2"
then
    echo "$str1 is not equal to $str2"
else
echo "$str1 is equal to $str2"
fi

# Check if str1 has zero length
if test -z "$str1"
then
echo "String str1 is empty"
else
echo "String str1 is not empty"
fi

# Check if str2 has non-zero length
if test -n "$str2"
then
```



```

echo "String str2 is not empty"

else

echo "String str2 is empty"

fi

# Check if empty_str has zero length

if test -z "$empty_str"

then

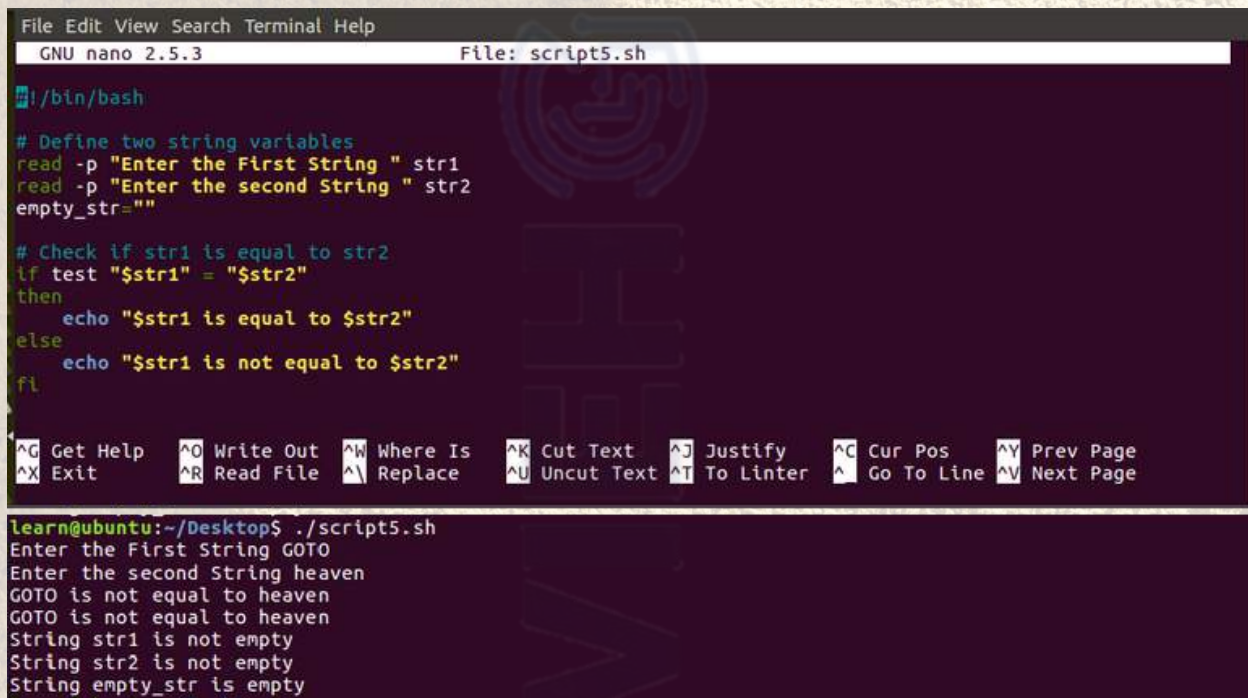
echo "String empty_str is empty"

else

echo "String empty_str is not empty"

fi

```



```

File Edit View Search Terminal Help
GNU nano 2.5.3 File: script5.sh

#!/bin/bash

# Define two string variables
read -p "Enter the First String " str1
read -p "Enter the second String " str2
empty_str=""

# Check if str1 is equal to str2
if test "$str1" = "$str2"
then
    echo "$str1 is equal to $str2"
else
    echo "$str1 is not equal to $str2"
fi

Learn@ubuntu:~/Desktop$ ./script5.sh
Enter the First String GOTO
Enter the second String heaven
GOTO is not equal to heaven
GOTO is not equal to heaven
String str1 is not empty
String str2 is not empty
String empty_str is empty

```

Operators on files used with test command:

- f : the file exists.
- s : the file exists and the file size is non zero.
- d : directory exists.

- r : file exists and has read permission.
- w : file exists and has write permission.
- x : file exists and has execute permission.

Script:

```
#!/bin/bash

# Set the file path to your drive
mydrive="/home/learn/Desktop/mydrive"

# Check if the file exists
if test -f "$mydrive"
then
echo "The file $mydrive exists."
else
echo "The file $mydrive does not exist."
fi

# Check if the file exists and has non-zero size
if test -s "$mydrive"
then
echo "The file $mydrive exists and has a non-zero size."
Else
echo "The file $mydrive either does not exist or has zero size."
fi

# Check if the path is a directory
if test -d "$mydrive"
then
echo "$mydrive is a directory."
```



```
else
echo "$mydrive is not a directory."
fi

# Check if the file exists and has read permission
if test -r "$mydrive"
then
echo "The file $mydrive exists and has read permission."
else
echo "The file $mydrive either does not exist or does not have read permission."
fi

# Check if the file exists and has write permission
if test -w "$mydrive"
then
echo "The file $mydrive exists and has write permission."
else
echo "The file $mydrive either does not exist or does not have write permission."
fi

# Check if the file exists and has execute permission
if test -x "$mydrive"
then
echo "The file $mydrive exists and has execute permission."
else
echo "The file $mydrive either does not exist or does not have execute permission."
fi
```



```
Terminal
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script6.sh

#!/bin/bash

# Set the file path to your drive
mydrive="/home/learn/Desktop/mydrive"

# Check if the file exists
if test -f "$mydrive"
then
    echo "The file $mydrive exists."
else
    echo "The file $mydrive does not exist."
fi

# Check if the file exists and has non-zero size
if test -s "$mydrive"

Learn@ubuntu:~/Desktop$ ./script6.sh
The file /home/learn/Desktop/mydrive exists.
The file /home/learn/Desktop/mydrive exists and has a non-zero size.
/home/learn/Desktop/mydrive is not a directory.
The file /home/learn/Desktop/mydrive exists and has read permission.
The file /home/learn/Desktop/mydrive exists and has write permission.
The file /home/learn/Desktop/mydrive exists and has execute permission.

Learn@ubuntu:~/Desktop$ ls -la mydrive
-rwxrwxrwx 1 learn learn 96 Mar 16 04:38 mydrive
```

Logical Operators used with test command:

Combining more than one condition is done through the logical AND, OR and

NOT

operators.

-a : logical AND

-o : logical OR

! : logical NOT

Script:

```
#!/bin/bash
```

```
# Prompt the user to enter a number
```

```
read -p "Enter any integer " num
```

```
# Check if the number is greater than 10 and less than 20(-a : logical AND)
```

```
if test "$num" -gt 10 -a "$num" -lt 20
```

```
then
```

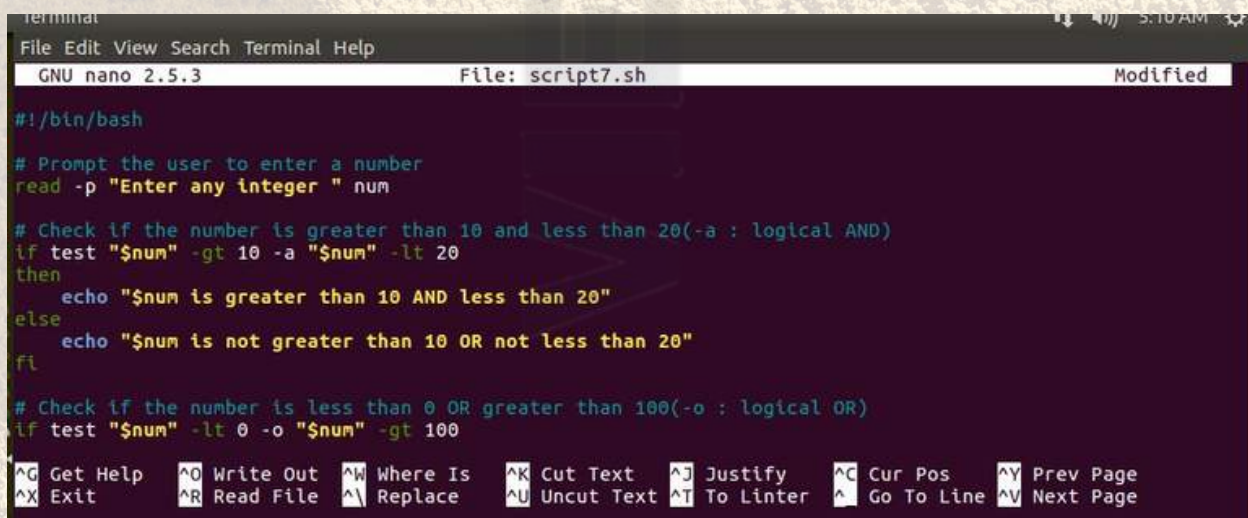
```
echo "$num is greater than 10 AND less than 20"
```



```
else
echo "$num is not greater than 10 OR not less than 20"
fi

# Check if the number is less than 0 OR greater than 100(-o : logical
OR) if test "$num" -lt 0 -o "$num" -gt 100
then
echo "$num is less than 0 OR greater than 100"
else
echo "$num is not less than 0 AND not greater than 100"
fi

# Check if the number is NOT equal to 50(! : logical NOT)
if test ! "$num" -eq 50
then
echo "$num is NOT equal to 50"
else
echo "$num is equal to 50"
fi
```



```
terminal
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script7.sh Modified

#!/bin/bash

# Prompt the user to enter a number
read -p "Enter any integer " num

# Check if the number is greater than 10 and less than 20(-a : logical AND)
if test "$num" -gt 10 -a "$num" -lt 20
then
    echo "$num is greater than 10 AND less than 20"
else
    echo "$num is not greater than 10 OR not less than 20"
fi

# Check if the number is less than 0 OR greater than 100(-o : logical OR)
if test "$num" -lt 0 -o "$num" -gt 100
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line ^V Next Page
```



```
Learn@ubuntu:~/Desktop$ ./script7.sh
Enter any integer 13
13 is greater than 10 AND less than 20
13 is not less than 0 AND not greater than 100
13 is NOT equal to 50
```

list of users currently logged

Script:

```
if who | grep -s student > /dev/null
```

```
then
```

```
echo student is logged in
```

```
else
```

```
echo student is not available
```

```
fi
```

Explanation:

This lists who is currently logged on to the system and pipes the output through grep to search for the username student.

The -s option causes grep to work silently and any error messages are directed to the file /dev/null instead of the standard output.

If the command is successful i.e. the username student is found in the list of users currently logged

in then the message student is logged on is displayed, otherwise the second message is displayed.

The Case Statement:

Script:

```
#!/bin/bash
```

```
while true; do
```

```
echo "Choose an option: "
```

```
echo "1. Display current date and time"
```

```
echo "2. Display calendar"
```

```
echo "3. List files in current directory"
```



```
echo "4. Display currently logged-in users"
echo "5. Print current working directory"
echo "6. Display system information"
echo "7. Show disk usage"
echo "8. Display network information"
echo "9. Show running processes"
echo "10. Display system memory usage"
echo "11. Show user information"
echo "12. Display kernel version"
echo "13. Show available disk space"
echo "14. Show system load"
echo "15. Display system uptime"
echo "16. List installed packages"
echo "17. Show hostname"
echo "18. Show active network connections"
echo "19. Show file system information"
echo "20. Show logged-in users and their processes"
echo "21. Exit"
read choice
case $choice in
1)
echo "Current date and time: $(date)"
;;
2)
echo "Calendar:"
cal
```



```
;;
3)
echo "Files in current directory:"
ls -l
;;
4)
    echo "Currently logged-in users:"
who
;;
5)
    echo "Current working directory:"
pwd
;;
6)
    echo "System Information:"
uname -a
;;
7)
echo "Disk Usage:"
df -h
;;
8)
    echo "Network Information:"
ifconfig
;;
9)
```



```
echo "Running Processes:"  
ps aux  
;;  
10)  
    echo "System Memory Usage:"  
free -m  
;;  
11)  
    echo "User Information:"  
id  
;;  
12)  
echo "Kernel Version:"  
uname -r  
;;  
13)  
echo "Available Disk Space:"  
du -h --max-depth=1 /  
;;  
14)  
echo "System Load:"  
uptime  
;;  
15)  
echo "System Uptime:"  
uptime -p
```



```
;;
16)
echo "Installed Packages:"
dpkg --get-selections
;;
17)
echo "Hostname:"
hostname
;;
18)
echo "Active Network Connections:"
netstat -tuln
;;
19)
echo "File System Information:"
df -T
;;
20)
echo "Logged-in Users and Their Processes:"

w
;;
21)
echo "Exiting program."
break
;;
*)
```

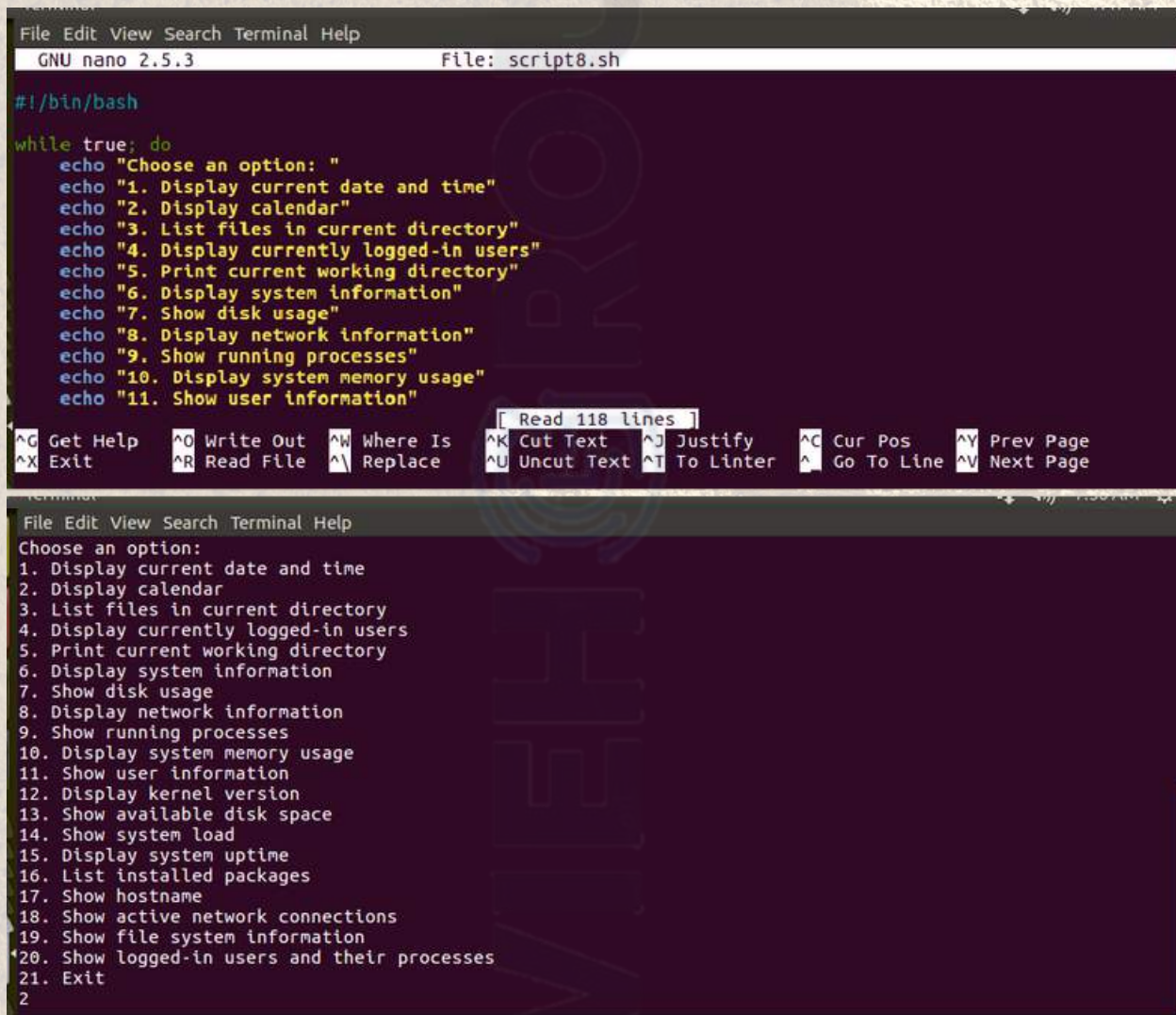


```
echo "Invalid option. Please select a valid option (1-21)."
```

```
;;
```

```
esac
```

```
done
```



```
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script8.sh

#!/bin/bash
while true; do
  echo "Choose an option: "
  echo "1. Display current date and time"
  echo "2. Display calendar"
  echo "3. List files in current directory"
  echo "4. Display currently logged-in users"
  echo "5. Print current working directory"
  echo "6. Display system information"
  echo "7. Show disk usage"
  echo "8. Display network information"
  echo "9. Show running processes"
  echo "10. Display system memory usage"
  echo "11. Show user information"
  [ Read 118 lines ]
  ^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text   ^J Justify    ^C Cur Pos   ^Y Prev Page
  ^X Exit       ^R Read File  ^\ Replace   ^U Uncut Text ^I To Linter  ^_ Go To Line  ^V Next Page

File Edit View Search Terminal Help
Choose an option:
1. Display current date and time
2. Display calendar
3. List files in current directory
4. Display currently logged-in users
5. Print current working directory
6. Display system information
7. Show disk usage
8. Display network information
9. Show running processes
10. Display system memory usage
11. Show user information
12. Display kernel version
13. Show available disk space
14. Show system load
15. Display system uptime
16. List installed packages
17. Show hostname
18. Show active network connections
19. Show file system information
20. Show logged-in users and their processes
21. Exit
2
```



```

17. Show hostname
18. Show active network connections
19. Show file system information
20. Show logged-in users and their processes
21. Exit
2
Calendar:
  March 2024
Su Mo Tu We Th Fr Sa
    1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
Choose an option:
1. Display current date and time
2. Display calendar
3. List files in current directory

```

```

20. Show logged-in users and their processes
21. Exit
12
Kernel Version:
4.15.0-142-generic
Choose an option:
1. Display current date and time
2. Display calendar

```

Example of using for statement:

```

Terminal
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script8.sh

#!/bin/bash

# Loop through each argument
for username in "$@"; do
    # Check if the user is logged in
    if who | grep -q "$username"; then
        echo "$username is logged in"
    else
        echo "$username is not logged in"
    fi
done

learn@ubuntu:~/Desktop$ ./script8.sh learn
learn is logged in
learn@ubuntu:~/Desktop$ ./script8.sh ubuntu
ubuntu is not logged in

```

Another Example:

Script:

```

#!/bin/bash

# Display a prompt asking the user to enter a command
echo "Please enter a command:"

# Start a loop to continuously read and execute
# commands while true; do

```



```
# Read user input into the variable 'response'
read response

# Check the value of 'response'
case "$response" in
'done')
# If 'done' is entered, exit the loop
break
;;
"")
# If no command is entered (empty input), prompt again
continue
;;
*)
# For any other command, execute it using 'eval'
eval "$response"
;;
esac
done
```



```
GNU nano 2.5.3 File: script9.sh

#!/bin/bash

# Display a prompt asking the user to enter a command
echo "Please enter a command:"

# Start a loop to continuously read and execute commands
while true; do
    # Read user input into the variable 'response'
    read response
    # Check the value of 'response'
    case "$response" in
        'done')
            # If 'done' is entered, exit the loop
            break
            ;;
        "")
            # If no command is entered (empty input), prompt again
            continue
    esac
done

learn@ubuntu:~/Desktop$ ./script9.sh
Please enter a command:
ls
a.out          gcc2ndPrograme.cpp  mydrive      script3.sh  script6.sh  script9.sh
commandexe.sh  gcc3Programe.c     script1.sh   script4.sh  script7.sh  script.sh
example.sh     gccPrograme.cpp    script2.sh   script5.sh  script8.sh
date
Sat Mar 16 10:47:40 PDT 2024
time

real    0m0.000s
user    0m0.000s
sys     0m0.000s
done
learn@ubuntu:~/Desktop$
```

Problem Solving:

Write a shell script that takes a keyword as a command line argument and lists the filenames containing the keyword?

```
#!/bin/bash
```

```
# Check if a keyword is provided as a command line argument
```

```
#[ -z "$1" ] checks if the first command line argument ($1) is empty.
```

```
if [ -z "$1" ]; then
```

```
echo "Please provide a keyword."
```

```
exit 1
```

```
fi
```

```
# Loop through each file in the current directory
```

```
#* is a wildcard that matches all files in the current directory.
```



```

for file in *; do
# Check if the file contains the keyword
#grep -q "$1" "$file" searches for the keyword $1 in the file $file.
#The -q option is used to suppress grep's output.
# It just checks if the keyword exists without printing any matching
lines. #If the keyword is found in the file, grep exits with a success
status,
# and the if statement evaluates to true
if grep -q "$1" "$file"; then
echo "$file"
fi

```



```

Done nano 2.5.3 File: script10.sh
#!/bin/bash
# Check if a keyword is provided as a command line argument
[ -z "$1" ] checks if the first command line argument ($1) is empty.

if [ -z "$1" ]; then
    echo "Please provide a keyword."
    exit 1
fi

# Loop through each file in the current directory
#* is a wildcard that matches all files in the current directory.

for file in *; do
    # Check if the file contains the keyword
    #grep -q "$1" "$file" searches for the keyword $1 in the file $file.
    #The -q option is used to suppress grep's output.
    # It just checks if the keyword exists without printing any matching lines.

learn@ubuntu:~/Desktop$ ./script10.sh used
a.out
script10.sh

```

2)Write a shell script that takes a command line argument and reports whether it is a directory, or a file or a link?

Script:

```

#!/bin/bash
# Check if a command line argument is provided
if [ -z "$1" ]; then
echo "Please provide a filename or directory."

```



```
exit 1

fi

# Check if the provided argument is a directory
if [ -d "$1" ]; then
echo "$1 is a directory."
exit 0
fi

# Check if the provided argument is a regular file
if [ -f "$1" ]; then
echo "$1 is a regular file."
exit 0
fi

# Check if the provided argument is a symbolic link
if [ -L "$1" ]; then
echo "$1 is a symbolic link."
exit 0
fi

# If none of the above conditions are met, the argument is not
found echo "$1 is not found."
```



```
GNU nano 2.5.3 File: script11.sh

#!/bin/bash

# Check if a command line argument is provided
if [ -z "$1" ]; then
    echo "Please provide a filename or directory."
    exit 1
fi

# Check if the provided argument is a directory
if [ -d "$1" ]; then
    echo "$1 is a directory."
    exit 0
fi

# Check if the provided argument is a regular file
if [ -f "$1" ]; then
    echo "$1 is a regular file."
    exit 0
fi

learn@ubuntu:~/Desktop$ ./script11.sh mydrive
mydrive is a regular file.
```

3)Write a script to find the number of sub directories in a given directory? #!/bin/bash

Check if a directory path is provided as a command line argument

if [-z "\$1"]; then

echo "Please provide a directory path."

exit 1

fi

Initialize a counter variable

count=0

Loop through each entry in the directory

for entry in "\$1"/*; do

Check if the entry is a directory

if [-d "\$entry"]; then

((count++))

fi

done

Print the number of subdirectories


```
echo "Number of subdirectories in '$1': $count"
```

```
GNU nano 2.5.3 File: script12.sh

#!/bin/bash

# Check if a directory path is provided as a command line argument
if [ -z "$1" ]; then
    echo "Please provide a directory path."
    exit 1
fi

# Initialize a counter variable
count=0

# Loop through each entry in the directory
for entry in "$1"/*; do
    # Check if the entry is a directory
    if [ -d "$entry" ]; then
        ((count++))
    fi
done

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      ^Y Prev Page
^X Exit          ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Linter    ^_ Go To Line    ^V Next Page

Read 22 lines

learn@ubuntu:~/Desktop$ ./script12.sh /home/learn
Number of subdirectories in '/home/learn': 8
```

4) Write a menu driven program that has the following

options. a) Search for a given file is in the directory or not.

b) Display the names of the users logged in

```
#!/bin/bash

# Prompt the user to select an option
echo "Menu:"

echo "1. Search a file in the directory"
echo "2. Display names of users logged in"
echo "3. Exit"

# Read user's choice
read -p "Enter your choice: " choice

# Perform action based on user's choice
if [ "$choice" = "1" ]; then
    read -p "Enter the filename to search: " filename
    if [ -e "$filename" ]; then
```



```
    echo "File '$filename' exists in the directory."
else
    echo "File '$filename' does not exist in the directory."
fi
elif [ "$choice" = "2" ]; then
    if who > /dev/null; then
        echo "Users logged in:"
        who | cut -d' ' -f1 | sort -u
    else
        echo "No users are currently logged in."
    fi
elif [ "$choice" = "3" ]; then
    echo "Exiting program."
    exit 0
else
    echo "Invalid choice. Please enter a number between 1 and 3."
fi
```



```
File Edit View Search Terminal Help
GNU nano 2.5.3 File: userloggedin.sh

#!/bin/bash

# Prompt the user to select an option
echo "Menu:"
echo "1. Search a file in the directory"
echo "2. Display names of users logged in"
echo "3. Exit"

# Read user's choice
read -p "Enter your choice: " choice

# Perform action based on user's choice
if [ "$choice" = "1" ]; then
    read -p "Enter the filename to search: " filename
    if [ -e "$filename" ]; then
        echo "File '$filename' exists in the directory."
    else
        echo "File '$filename' does not exist in the directory."
    fi
fi

^G Get Help      ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos    ^Y Prev Page
^X Exit          ^R Read File    ^\ Replace     ^U Uncut Text  ^T To Linter  ^_ Go To Line  ^V Next Page

learn@ubuntu:~/Desktop$ ./userloggedin.sh
Menu:
1. Search a file in the directory
2. Display names of users logged in
3. Exit
Enter your choice: 1
Enter the filename to search: mydrive
File 'mydrive' exists in the directory.
learn@ubuntu:~/Desktop$ ./userloggedin.sh
Menu:
1. Search a file in the directory
2. Display names of users logged in
3. Exit
Enter your choice: 2
Users logged in:
learn
```

Conclusion:

In this shell scripting lab, we covered the basics of writing Bash scripts, including user input/output, conditional statements, loops, and file operations. We also explored advanced topics such as process substitution and text processing. Through progressively simplified examples, we demonstrated how to create a menu-driven program and automate tasks efficiently.

Thank you for taking the time to read through our publication. Your continued support is invaluable.

Jai Hind!

