Hands-On Bash Scripting: Automating Linux Tasks with Shell Scripts

Thrilled to share my hands-on journey exploring the fundamentals and real-time implementation of **Bash Scripting** in a Linux environment. From creating users to implementing logic with conditions and loops, this task helped strengthen my command over Linux automation and scripting logic.

What is Scripting?

Scripting refers to writing a sequence of commands in a file (called a script) that the operating system can execute automatically. Unlike manually typing each command, scripting lets you bundle them together to run in one go, making processes efficient and error-free.

Why Scripting?

Scripting is essential because it helps:

- **Automate repetitive tasks** (e.g., creating users, backups, file operations)
- **Reduce human error** by minimizing manual intervention
- **Enhance productivity** by executing complex workflows with a single command
- **Save time** in system administration, DevOps tasks, and server management
- **Support conditional logic** and looping — making it flexible and intelligent

Whether you're managing hundreds of users or deploying a web service, scripting turns commands into repeatable and scalable automation.

How Bash Scripting Works?

1. **You create a text file** (usually ending in .sh, but not mandatory).
2. At the top of the script, you define the **interpreter** using the **shebang (#!)**.
3. Inside the script, you write one or more Linux commands (including variables, logic, loops).
4. You make the script **executable** using chmod.
5. Finally, you **run the script**, and it executes the listed commands in order.

Example Flow:

```
#!/bin/bash       # Shebang defines the interpreter

echo "Hello, World" # Command inside the script
```

What is a Shebang (#!)?

The **shebang** is the first line in any shell script that defines which interpreter should execute the script.

Examples:

#!/bin/bash # Executes the script using Bash

#!/bin/sh # Executes the script using Bourne shell

◈ Steps to Write & Run a Shell Script

1️⃣ Create a script file

vim script_name.sh

2️⃣ Add a shebang and commands inside the file

#!/bin/bash

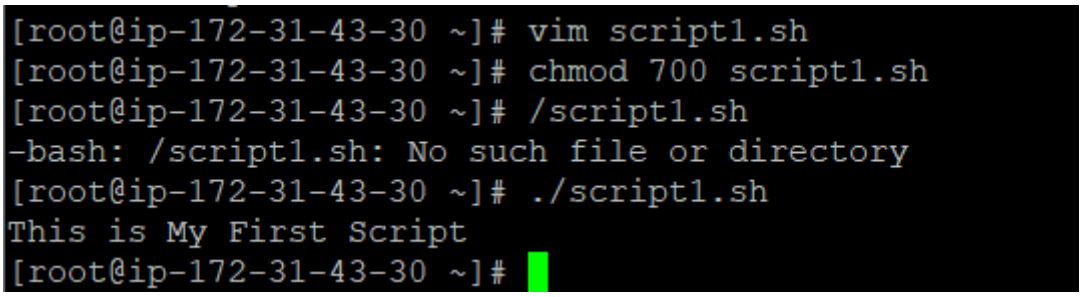
echo "This is a sample script"

3️⃣ Make the file executable

chmod 700 script_name.sh

4️⃣ Run the script

./script_name.sh



Screenshot 1:Script file creation, editing, permission change, and execution

◈ Bash Script Components: Operators & Variables

Operators in Bash

Type	Example
Arithmetic	+ - * /
Relational	-eq -ne -gt -lt -ge -le
Logical	-a -o !
Assignment	=

```
[root@ip-172-31-43-30 ~]# cat script1.sh
#! /bin/bash
read -p "Enter the 1st number: " a
read -p "Enter the 2nd number: " b

c= expr $a + $b
d= expr $a - $b
e= expr $a \* $b
f= expr $a / $b

echo "The addition is $c"
echo "The subtraction is $d"
echo "The multiplication is $e"
echo "The division is $f"
[root@ip-172-31-43-30 ~]# ./script1.sh
Enter the 1st number: 70
Enter the 2nd number: 30
100
40
2100
2
The addition is
The subtraction is
The multiplication is
The division is
```

Screenshot 2: Sample operator output

◈ Variables in Bash

- **Local/User-defined**: a=10
- **Environment/System**: PATH, USER
- **Special/Positional**: $0, $1, $#, $@, $*

```
[root@ip-172-31-43-30 ~]# cat /file1.sh
#! /bin/bash
echo "filename: $0"
echo "1st argument: $1"
echo "2nd argument: $2"
echo "3rd argument: $3"
echo "total no of arguments: $#"
echo "All arguments: $@"
echo "All arguments: $*"
[root@ip-172-31-43-30 ~]# /file1.sh HII THIS IS JAYSHRI HOW ARE YOU
filename: /file1.sh
1st argument: HII
2nd argument: THIS
3rd argument: IS
total no of arguments: 7
All arguments: HII THIS IS JAYSHRI HOW ARE YOU
All arguments: HII THIS IS JAYSHRI HOW ARE YOU
```

Screenshot 3: Script using positional arguments

◈ Looping Constructs in Bash

Loop Type	Use Case
for	Repeat over a list
while	Repeat while condition is true
until	Repeat until condition becomes true
Infinite loop	Run tasks continuously

```
[root@ip-172-31-43-30 ~]# cat /file2.sh
#! /bin/bash
for n in user1 user2 user3
do
  useradd $n
done

[root@ip-172-31-43-30 ~]# id user1
uid=1001(user1) gid=1001(user1) groups=1001(user1)
[root@ip-172-31-43-30 ~]# id user2
uid=1002(user2) gid=1002(user2) groups=1002(user2)
[root@ip-172-31-43-30 ~]# id user3
uid=1003(user3) gid=1003(user3) groups=1003(user3)
[root@ip-172-31-43-30 ~]#
```

Screenshot 4: Script using for loop

```
[root@ip-172-31-43-30 ~]# cat /file3.sh
#! /bin/bash
n=5
while [ $n -eq 5 ]
do
echo "no is $n"
n=`expr $n + 1`
done
[root@ip-172-31-43-30 ~]# /file3.sh
no is 5
[root@ip-172-31-43-30 ~]#
```

Screenshot 5: Script using while loop

```
[root@ip-172-31-43-30 ~]# cat /file5.sh
#! /bin/bash
a=1
until [ $a -eq 10 ]
do
echo "$a"
a=`expr $a + 1`
done

[root@ip-172-31-43-30 ~]# /file5.sh
1
2
3
4
5
6
7
8
9
[root@ip-172-31-43-30 ~]#
```

Screenshot 6: Script using until loop

```
[root@ip-172-31-43-30 ~]# cat /file6.sh
#! /bin/bash
for ((n=2;n>1;n++))
do
  echo "number is $n"
done
[root@ip-172-31-43-30 ~]# /file6.sh
number is 59575
number is 59576
number is 59577
number is 59578
number is 59579
```

Screenshot 7: Script using infinity loop

◈ Conditional Statements

1. if
2. if-else
3. if-elif-else
4. Nested if

These control flow structures allow decision-making inside scripts.

```
[root@ip-172-31-43-30 ~]# cat /script2.sh
#! /bin/bash
read -p "enter 1st value:" a
read -p "enter 2nd value:" b

if [ $a -gt $b ]
then
    useradd rohan
fi

[root@ip-172-31-43-30 ~]# /script2.sh
enter 1st value:80
enter 2nd value:50
[root@ip-172-31-43-30 ~]# id rohan
uid=1004(rohan) gid=1004(rohan) groups=1004(rohan)
[root@ip-172-31-43-30 ~]#
```

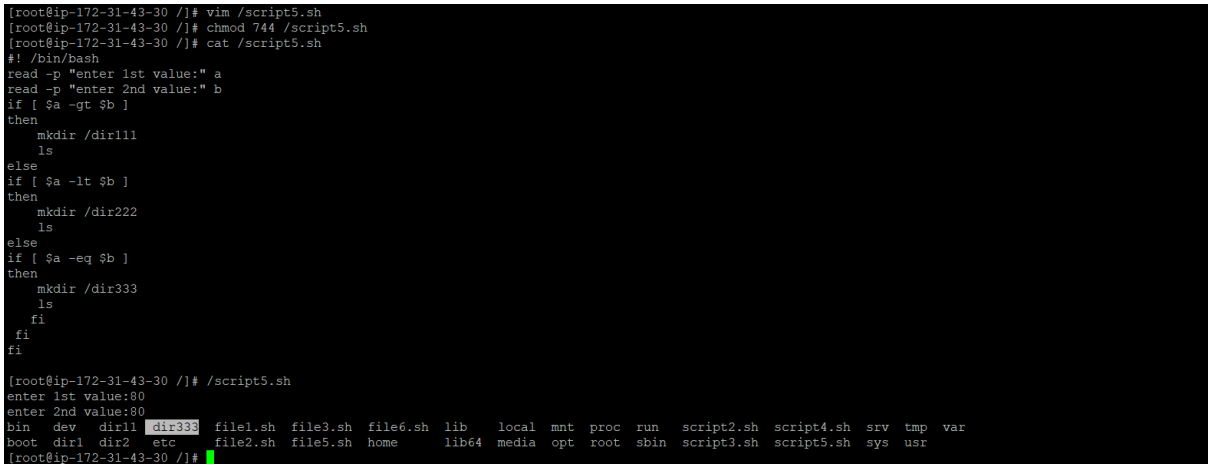
Screenshot 8: Script output using if condition type

```
[root@ip-172-31-43-30 ~]# cat /script3.sh
#! /bin/bash
read -p "enter 1st number:" a
read -p "enter 2nd number:" b
if [ $a -lt $b ]
then
echo "smaller number is $a"
else
echo "smaller number is $b"
fi

[root@ip-172-31-43-30 ~]# /script3.sh
enter 1st number:100
enter 2nd number:20
smaller number is 20
```

Screenshot 9: Script output using if-else condition type

```
[root@ip-172-31-43-30 ~]# cat /script4.sh
#! /bin/bash
read -p "enter 1st value:" a
read -p "enter 2nd value:" b
if [ $a -gt $b ]
then
    mkdir /dir11
    ls
elif [ $a -lt $b ]
then
    mkdir /dir22
    ls
else
    mkdir /dir33
    ls
fi

[root@ip-172-31-43-30 ~]# /script4.sh
enter 1st value:70
enter 2nd value:40
[root@ip-172-31-43-30 ~]# cd /
[root@ip-172-31-43-30 /]# ls
bin   dev   dir11  etc       file2.sh  file5.sh  home  lib64  media  opt   root  sbin        script3.sh  srv  tmp  var
boot  dir1  dir2   file1.sh  file3.sh  file6.sh  lib   local  mnt    proc  run   script2.sh  script4.sh  sys  usr
[root@ip-172-31-43-30 /]#
```

Screenshot 10: Script output using if-elif-else condition type



Screenshot 11: Script output using nested-if condition type

◈ Practical Task Implementations

🔹 Create Multiple Users

#!/bin/bash

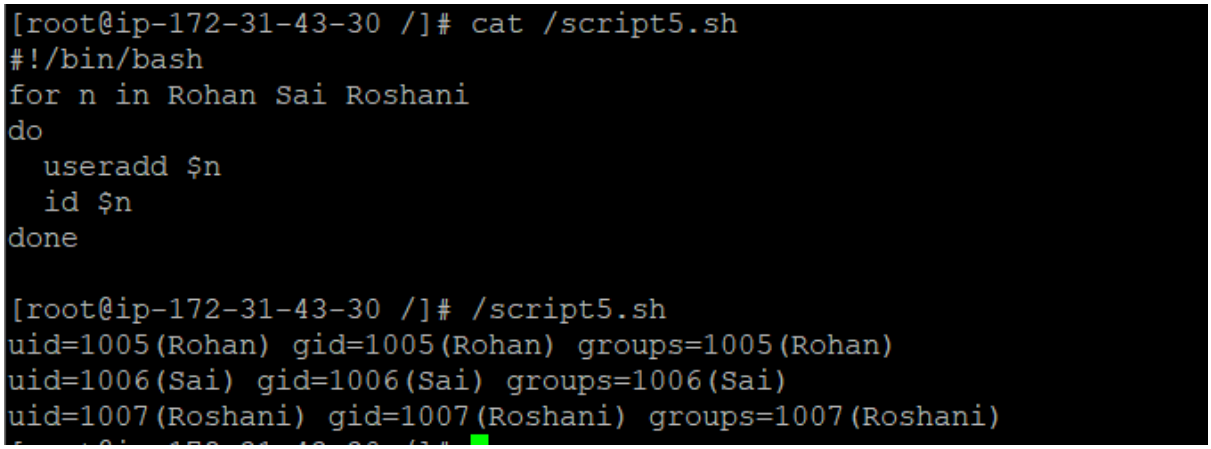
for n in Rohan Sai Roshani

do

useradd $n

id $n

done



Screenshot 12: Create Multiple Users

- **Delete Multiple Users**

```
#!/bin/bash
for n in Rohan Sai Roshani
do
 userdel -r $n
 id $n
done
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#!/bin/bash
for n in Rohan Sai Roshani
do
  userdel -r $n
  id $n
done

[root@ip-172-31-43-30 /]# /script6.sh
id: 'Rohan': no such user
id: 'Sai': no such user
id: 'Roshani': no such user
```

Screenshot 13: Delete Multiple Users

- **Create Users & Set Passwords**

```
#!/bin/bash
for n in user1 user2 user3
do
 useradd $n
 echo "Android@50" | passwd --stdin $n
done
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#!/bin/bash
for n in user1 user2 user3
do
  useradd $n
  echo "Android@50" | passwd --stdin $n
done

[root@ip-172-31-43-30 /]# /script6.sh
Changing password for user user1.
passwd: all authentication tokens updated successfully.
Changing password for user user2.
passwd: all authentication tokens updated successfully.
Changing password for user user3.
passwd: all authentication tokens updated successfully.
[root@ip-172-31-43-30 /]#
```

Screenshot 14: Create Users & Set Passwords

🔹 Arithmetic Operations

```
#!/bin/bash

a=20

b=10

echo "Addition: $(expr $a + $b)"

echo "Subtraction: $(expr $a - $b)"

echo "Multiplication: $(expr $a \* $b)"

echo "Division: $(expr $a / $b)"
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#!/bin/bash
a=20
b=10
echo "Addition: $(expr $a + $b)"
echo "Subtraction: $(expr $a - $b)"
echo "Multiplication: $(expr $a \* $b)"
echo "Division: $(expr $a / $b)"

[root@ip-172-31-43-30 /]# /script6.sh
Addition: 30
Subtraction: 10
Multiplication: 200
Division: 2
[root@ip-172-31-43-30 /]#
```

Screenshot 15: Arithmetic Operations

◈ Loop Examples

Create Directories

```
#!/bin/bash

for n in 1 2 3 4 5

do

 mkdir dir$n

done
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#!/bin/bash
for n in 1 2 3 4 5
do
  mkdir ashok$n
done

[root@ip-172-31-43-30 /]# /script6.sh
[root@ip-172-31-43-30 /]# ls
'!'   5       ashok5   dir11   dir5      file5.sh   lib64   proc        script3.sh   sys
 1    ashok1  bin      dir2    etc       file6.sh   local   root        script4.sh   tmp
 2    ashok2  boot     dir3    file1.sh  home       media   run         script5.sh   usr
 3    ashok3  dev      dir333  file2.sh  jayshri    mnt     sbin        script6.sh   var
 4    ashok4  dir1     dir4    file3.sh  lib        opt     script2.sh  srv
[root@ip-172-31-43-30 /]#
```

Screenshot 16: Create Directories

Print Numbers Using While

```
#!/bin/bash
a=1
while [ $a -le 5 ]
do
 echo "$a"
 a=$(expr $a + 1)
done
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#!/bin/bash
a=1
while [ $a -le 5 ]
do
  echo "$a"
  a=$(expr $a + 1)
done

[root@ip-172-31-43-30 /]# /script6.sh
1
2
3
4
5
[root@ip-172-31-43-30 /]#
```

Screenshot 17: Print Numbers Using While

◈ Decision Making

Script to Find Greater Number

```
read -p "Enter 1st: " a
read -p "Enter 2nd: " b
if [ $a -gt $b ]; then
 echo "$a is greater"
else
 echo "$b is greater"
fi
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#! /bin/bash
read -p "Enter 1st: " a
read -p "Enter 2nd: " b
if [ $a -gt $b ]; then
  echo "$a is greater"
else
  echo "$b is greater"
fi

[root@ip-172-31-43-30 /]# /script6.sh
Enter 1st: 100
Enter 2nd: 20
100 is greater
[root@ip-172-31-43-30 /]#
```

Screenshot 18: Script to Find Greater Number

Script to Find Equality

```
read -p "Enter 1st: " a
read -p "Enter 2nd: " b
if [ $a -eq $b ]; then
 echo "Numbers are equal"
else
 echo "Numbers are different"
fi
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#! /bin/bash
read -p "Enter 1st: " a
read -p "Enter 2nd: " b
if [ $a -eq $b ]; then
  echo "Numbers are equal"
else
  echo "Numbers are different"
fi

[root@ip-172-31-43-30 /]# /script6.sh
Enter 1st: 60
Enter 2nd: 60
Numbers are equal
[root@ip-172-31-43-30 /]#
```

Screenshot 19: Script to Find Equality

Pass/Fail Based on Marks

```
read -p "Enter marks: " a
if [ $a -gt 35 ]; then
 echo "Result: Pass"
else
 echo "Result: Fail"
fi
```

```
[root@ip-172-31-43-30 /]# cat /script6.sh
#! /bin/bash
read -p "Enter marks: " a
if [ $a -gt 35 ]; then
  echo "Result: Pass"
else
  echo "Result: Fail"
fi

[root@ip-172-31-43-30 /]# /script6.sh
Enter marks: 90
Result: Pass
[root@ip-172-31-43-30 /]#
```

Screenshot 20: Pass/Fail Based on Marks

Conclusion

This task enriched my understanding of **Linux automation using Bash scripting**, covering fundamentals like shebang, operators, and variables, and advancing through practical implementations using loops and decision-making logic. By scripting solutions for common administrative tasks, I was able to explore core principles of system automation and scripting logic—an essential step in DevOps and Linux system management.