

AIM: SHELL SCRIPTING

Shell scripting refers to writing scripts using a shell (command-line interpreter) to automate tasks or execute commands in a specific sequence. The most common shell used for scripting on Unix-like systems is the Bash shell (Bourne Again SHell).

Here's an overview of the topics you mentioned in relation to Bash shell scripting:

1. Bash Syntax:

- Bash scripts start with a shebang (`#!/bin/bash`) at the top to specify the interpreter.
- Commands and statements are written on separate lines or separated by semicolons(`;`).
- Comments start with a hash symbol (`#`) and are ignored by the shell.
- Variables are referenced using the `$` symbol (e.g., `$variable`).

2. Environment Variables:

- Environment variables are global variables accessible to all processes in the shell session.
- You can set environment variables using the `export` command (e.g., `export VARIABLE_NAME=value`).
- Common environment variables include `PATH` (directories to search for executables), `HOME` (user's home directory), etc.
- Access environment variables using the `$` symbol (e.g., `$PATH`).

3. Variables:

- Variables in Bash are assigned using the `=` sign without any spaces (e.g., `variable=value`).
- Variable names are case-sensitive and can contain letters, numbers, and underscores.
- To access the value of a variable, prefix it with `$` (e.g., `$variable`).
- It's good practice to enclose variables in double quotes (`"$variable"`) to handle spaces and special characters correctly.

4. Control Constructs:

- `if` statement: Executes commands based on a condition. It has the syntax:
- `for` loop: Iterates over a list of values. It has the syntax:

`if condition; then`

`# commands to execute if condition is true`

`else`

commands to execute if condition is false

fi

- while loop: Executes commands repeatedly until a condition is false. It has the syntax:

while [condition]

do

Code to be executed repeatedly as long as the condition is true

done

- for loop: Iterate over a sequence of values and perform a set of commands repeatedly. It Allows you to automate repetitive tasks and process multiple items in a collection.0

for variable in value1 value2 ... valueN; do

commands to execute

done

5. Aliases and Functions:

- Aliases allow you to create shortcuts for commands or command sequences. They are defined in the shell startup files (e.g., .bashrc) using the alias command.
- Functions are reusable blocks of code. They can be defined in shell scripts or shell startup files using the function keyword or just a name and parentheses.

6. Accessing Command Line Arguments:

- Command line arguments passed to a shell script can be accessed using the variables \$1, \$2, and so on, where \$1 represents the first argument, \$2 represents the second, and so on.
- \$0 represents the name of the script itself.
- \$# holds the total number of arguments passed.
- \$@ represents all the arguments as a list.

The following concepts form the foundation of shell scripting in Bash and provide the necessary tools to create powerful and efficient scripts to automate tasks or perform complex operations.

Startup Scripts: Startup scripts are executed during the boot process of a system. They are used to configure and start various services, daemons, and applications that need to run automatically when the system starts up. In Linux, the most common startup script mechanism is provided by Systemd.

2. Login and Logout Scripts: Login scripts are executed when a user logs into a system, typically after entering their username and password. These scripts can be used to set up the user's environment, define aliases, customize the shell prompt, and perform other tasks specific to that user.

Logout scripts, on the other hand, are executed when a user logs out of the system. They can be used to clean up temporary files, perform logging tasks, or execute any necessary actions before the user session ends.

Both login and logout scripts are typically written in shell scripting languages like Bash and are executed automatically by the system when the respective events occur.

3. Systemd: Systemd is a system and service manager for Linux systems that provides a range of features, including the management of services, startup processes, and system resources. It replaces the traditional init system used in older versions of Linux.

Systemd introduces the concept of service units, which are configuration files that define how a service should be started, stopped, and managed. These service units can include startup and shutdown scripts, dependencies, environment variables, and other parameters necessary for managing the service.

Systemd uses a dependency-based approach to start and stop services, ensuring that they start in the correct order and can handle dependencies between different services. It also provides tools like systemctl for managing and controlling services.

4. System Init Scripts: System init scripts are scripts that are executed during the boot process of a Linux system using the init system. Init scripts are typically written in shell scripting languages and are stored in specific directories, such as /etc/init.d/ or /etc/rc.d/init.d/.

These init scripts define actions to be performed for different runlevels or system states, such as starting or stopping services, mounting file systems, initializing devices, and performing other system-related tasks.

The specific init system and directory structure for init scripts may vary depending on the Linux distribution. Common init systems include System V init (SysVinit) and Upstart, which have been largely replaced by systemd.

5. Five Examples of Init Scripts: Here are five examples of init scripts that you might encounter in Linux systems:

- /etc/init.d/apache2: Controls the Apache HTTP server service.
- /etc/init.d/sshd: Manages the SSH server service.
- /etc/init.d/networking: Configures network interfaces during system startup.
- /etc/init.d/cron: Handles the cron daemon, which executes scheduled tasks.
- /etc/init.d/mysql: Controls the MySQL database server service.

These init scripts would typically include functions to start, stop, restart, and manage the respective services.

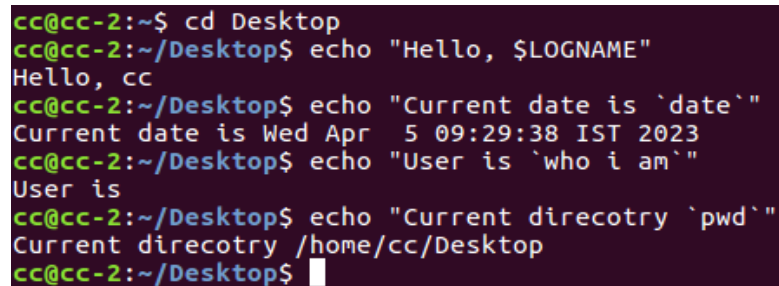
RESULT: The script has been familiarized successfully.

1. Write a script to show current date, time and current directory.

SOURCE CODE:

```
#!/bin/bash
echo "Hello, $LOGNAME"
echo "Current date is `date`"
echo "User is `who i am`"
echo "Current direcotry `pwd`"
```

OUTPUT:



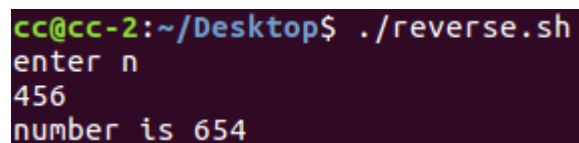
```
cc@cc-2:~$ cd Desktop
cc@cc-2:~/Desktop$ echo "Hello, $LOGNAME"
Hello, cc
cc@cc-2:~/Desktop$ echo "Current date is `date`"
Current date is Wed Apr  5 09:29:38 IST 2023
cc@cc-2:~/Desktop$ echo "User is `who i am`"
User is
cc@cc-2:~/Desktop$ echo "Current direcotry `pwd`"
Current direcotry /home/cc/Desktop
cc@cc-2:~/Desktop$
```

2. Write a script to reverse of a number.

SOURCE CODE:

```
#!/bin/bash
echo enter n
read n
num=0
while [ $n -gt 0 ]
do
num=$(expr $num \* 10)
k=$(expr $n % 10)
num=$(expr $num + $k)
n=$(expr $n / 10)
done
echo number is $num
```

OUTPUT:



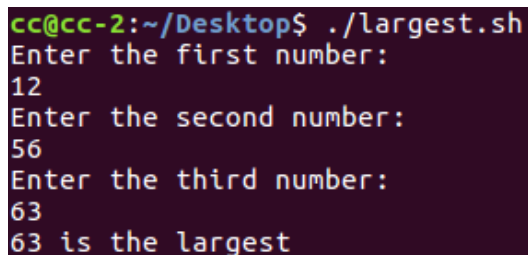
```
cc@cc-2:~/Desktop$ ./reverse.sh
enter n
456
number is 654
```

3. Write a script to largest among three numbers.

SOURCE CODE:

```
#!/bin/bash
echo "Enter the first number:"
read num
echo "Enter the second number:"
read num1
echo "Enter the third number:"
read num2
if [ $num -gt $num1 ] && [ $num -gt $num2 ]
then
    echo $num" is the largest"
elif [ $num1 -gt $num ] && [ $num1 -gt $num2 ]
then
    echo $num1" is the largest"
else
    echo $num2" is the largest"
fi
```

OUTPUT:

A screenshot of a terminal window with a dark purple background. The prompt is 'cc@cc-2:~/Desktop\$' followed by the command './largest.sh'. The output shows three prompts: 'Enter the first number:', 'Enter the second number:', and 'Enter the third number:'. The user has entered '12', '56', and '63' respectively. The final output is '63 is the largest'.

4. Write a script check whether the number is Armstrong or not.

SOURCE CODE:

```
#!/bin/bash
echo "Enter a number: "
read c
x=$c
sum=0
r=0
n=0
while [ $x -gt 0 ]
do
    r=`expr $x % 10`
    n=`expr $r \* $r \* $r`
    sum=`expr $sum + $n`
    x=`expr $x / 10`
```

```

done
if [ $sum -eq $c ]
then
echo "It is an Armstrong Number."
else
echo "It is not an Armstrong Number."
fi

```

OUTPUT:

```

cc@cc-2:~/Desktop$ chmod +x armstrong.sh
cc@cc-2:~/Desktop$ ./armstrong.sh
Enter a number:
125
It is not an Armstrong Number.
cc@cc-2:~/Desktop$ ./armstrong.sh
Enter a number:
153
It is an Armstrong Number.

```

5. Write a script to check password and login

SOURCE CODE:

```

#!/bin/bash
expected_username="admin"
expected_password="password123"
read -p "Enter your username: " username
read -sp "Enter your password: " password
echo
if [[ $username == $expected_username && $password ==
$expected_password ]]; then
    echo "Login successfull!"
else
    echo "Login failed!"
fi

```

OUTPUT:

```

exam23@cc-13:~/Desktop$ chmod +x login.sh
exam23@cc-13:~/Desktop$ ./login.sh
Enter your username: admin
Enter your password:
Login successfull!

```

6. Write a script to count the prime numbers in specific range.

SOURCE CODE:

```

#!/bin/bash
low=1
count=0
while [ $low -eq 1 ]

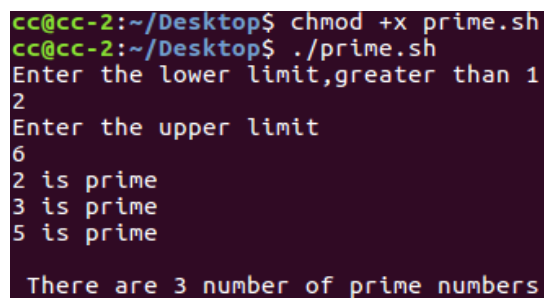
```

```

do
echo "Enter the lower limit,greater than 1"
read low
done
echo "Enter the upper limit"
read upper
for mun in `seq $low $upper`
do
ret=$(factor $mun | grep $mun | cut -d ":" -f 2 | cut -d " " -f 2)
if [ "$ret" -eq "$mun" ]
then
echo "$mun is prime"
((count++))
fi
done
echo -e "\n There are $count number of prime numbers"

```

OUTPUT:



```

cc@cc-2:~/Desktop$ chmod +x prime.sh
cc@cc-2:~/Desktop$ ./prime.sh
Enter the lower limit,greater than 1
2
Enter the upper limit
6
2 is prime
3 is prime
5 is prime

There are 3 number of prime numbers

```

7. Write a script to convert the contents of a given file from uppercase to lowercase and also count the number of lines, words and characters of the resultant file. Also display the resultant file in descending order.

SOURCE CODE:

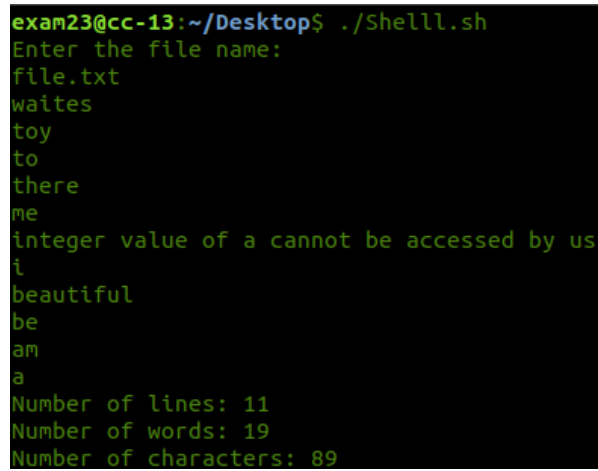
```

#!/bin/bash
echo "Enter the file name:"
read filename
if [ ! -f "$filename" ]; then
echo "File '$filename' does not exist."
exit 1
fi
cat "$filename" | tr '[:upper:]' '[:lower:]' > lowercase.txt
lines=$(wc -l lowercase.txt | cut -d ' ' -f 1)
words=$(wc -w lowercase.txt | cut -d ' ' -f 1)
characters=$(wc -c lowercase.txt | cut -d ' ' -f 1)
sort -r lowercase.txt > sorted.txt
cat sorted.txt
echo "Number of lines: $lines"
echo "Number of words: $words"

```

```
echo "Number of characters: $characters"
```

OUTPUT:

A terminal window with a black background and green text. The prompt is 'exam23@cc-13:~/Desktop\$./Shelll.sh'. The script prompts 'Enter the file name:' and the user enters 'file.txt'. The script then prints the contents of the file: 'waites', 'toy', 'to', 'there', 'me', and a multi-line error message 'integer value of a cannot be accessed by us'. It then prints 'beautiful', 'be', 'am', and 'a'. Finally, it prints the statistics: 'Number of lines: 11', 'Number of words: 19', and 'Number of characters: 89'.

```
exam23@cc-13:~/Desktop$ ./Shelll.sh
Enter the file name:
file.txt
waites
toy
to
there
me
integer value of a cannot be accessed by us
i
beautiful
be
am
a
Number of lines: 11
Number of words: 19
Number of characters: 89
```

8. Write a script to perform following basic math operation as: Addition, subtraction, multiplication, division

SOURCE CODE:

```
echo "Enter a="
read a
echo "Enter b="
read b
val=`expr $a + $b`
echo "a + b : $val"
val=`expr $a - $b`
echo "a - b : $val"
val=`expr $a \* $b`
echo "a * b : $val"
val=`expr $b / $a`
echo "b / a : $val"
val=`expr $b % $a`
echo "b % a : $val"
if [ $a == $b ]
then
    echo "a is equal to b"
fi
if [ $a != $b ]
then
    echo "a is not equal to b"
fi
```


OUTPUT:

```
cc@cc-2:~/Desktop$ ./aoprtn.sh
Enter a=
12
Enter b=
25
a + b : 37
a - b : -13
a * b : 300
b / a : 2
b % a : 1
a is not equal to b
```

9. Read 3 marks of a student and find the average. Display the grade of the student based on the average. (if..then..elif..fi)

S >= 90%
A < 90%, but >= 80%
B < 80%, but >= 60%
P < 80%, but >= 40%
F < 40%

SOURCE CODE:

```
#!/bin/bash
echo "Enter the marks for the student:"
read -p "Mark 1: " mark1
read -p "Mark 2: " mark2
read -p "Mark 3: " mark3
average=$(( ($mark1 + $mark2 + $mark3) / 3 ))
echo "Average: $average"
if (( average >= 90 )); then
    grade="S"
elif (( average >= 80 )); then
    grade="A"
elif (( average >= 60 )); then
    grade="B"
elif (( average >= 40 )); then
    grade="P"
else
    grade="F"
fi
echo "Grade: $grade"
```

OUTPUT:

```
exam23@cc-13:~/Desktop$ ./marks.sh
Enter the marks for the student:
Mark 1: 50
Mark 2: 36
Mark 3: 23
Average: 36
Grade: F
```

10. Read the name of an Indian state and display the main language according to the table. For other states, the output may be "Unknown". Use "|" to separate states with same language (case..esac).

State	Main Language
Andhra Pradesh	Telugu
Assam	Assamese
Bihar	Hindi
Himachal Pradesh	Hindi
Karnataka	Kannada
Kerala	Malayalam
Lakshadweep	Malayalam
Tamil Nadu	Tamil

SOURCE CODE:

```
#!/bin/bash
echo "Enter the name of an Indian state:"
read state
state_lowercase=$(echo "$state" | tr '[:upper:]' '[:lower:]')
case $state_lowercase in
"andhra pradesh")
main_language="Telugu"
;;
"assam")
main_language="Assamese"
;;
"bihar")
main_language="Hindi"
;;
"himachal pradesh")
main_language="Hindi"
;;
"karnataka")
main_language="Kannada"
```

```
;;
"kerala" | "lakshadweep")
main_language="Malayalam"
;;
"tamil nadu")
main_language="Tamil"
;;
*)
main_language="Unknown"
;;
esac
echo "The main language of $state is $main_language."
```

OUTPUT:

```
exam23@cc-13:~/Desktop$ ./india.sh
Enter the name of an Indian state:
kerala
The main language of kerala is Malayalam.
```

11. Change the home folder of all users whose name start with stud from /home/username to /usr/username. Also change the password of username to username123 (e.g., /home/stud25 changes to /usr/stud25 and his/her password changes to stud25123) - (Use for .. in)

SOURCE CODE:

```
#!/bin/bash
for username in /home/stud*;
do
    if [ -d "$username" ]; then
        new_home="/usr${username#/home}"
        new_password="${username#/home/stud}"
        new_password="${new_password} 123"

        usermod -d "$new_home" "$username"
        echo "$username:$new_password" | chpasswd
        echo "Changed home folder and password for $username"
    fi
done
```

OUTPUT:

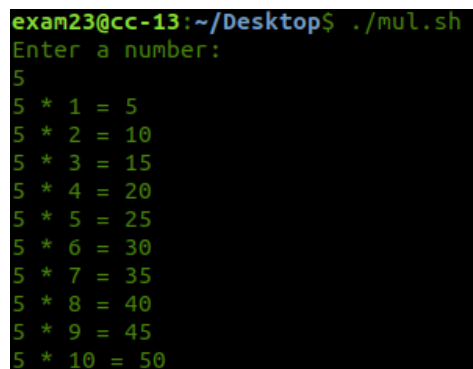
```
Changed home folder and password for /home/stud25
Changed home folder and password for /home/stud42
Changed home folder and password for /home/stud99
mca@ubuntu01:~$
```

12. Read a number and display the multiplication table of the number up to 10 lines. -
(Use for((..)))

SOURCE CODE:

```
#!/bin/bash
echo "Enter a number:"
read number
for((i=1;i<=10;i++))
do
    echo "$number * $i = $((number * $i))"
done
```

OUTPUT:



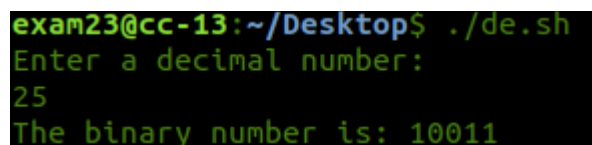
```
exam23@cc-13:~/Desktop$ ./mul.sh
Enter a number:
5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

13. Read a Decimal number. Convert it to Binary and display the result. -(Use while)

SOURCE CODE:

```
#!/bin/bash
echo "Enter a decimal number: "
read number
binary_number=""
while [ "$number" -gt 0 ]; do
    binary_number="$binary_number$((number % 2))"
    number=$((number / 2))
done
echo "The binary number is: $binary_number"
```

OUTPUT:



```
exam23@cc-13:~/Desktop$ ./de.sh
Enter a decimal number:
25
The binary number is: 10011
```

14. Look at the system log files. Write a shell script to extract the last login details of a particular user and list out all failed logins. Store the results to a file. The user name should be given as a command line argument.

SOURCE CODE:

```
#!/bin/bash
if [ $# -eq 0 ]
then
    echo "Please try again with a valid argument";
    exit
fi
lastLogin=$(last -n 1);
echo "Last logged in user is $lastLogin"
loginAttempts=$(sudo cat /var/log/auth.log | grep $1 | grep failed)
echo "Failed login attempts of $1 are:"
echo "Here: $loginAttempts"
```

OUTPUT:



```
ubuntu@ubuntu:~$ ./lastlogin.sh Desktop
Last logged in user is ubuntu      :0                :0                Wed Jun 28 13:03
gone - no logout

wtmp begins Wed Jun 28 13:01:27 2023
Failed login attempts of Desktop are:
Here:
```

15. Write a shell script to display the details of a particular process currently running. Assume that you have necessary permissions. The process name/id is to be given as a command line argument.

SOURCE CODE:

```
#!/bin/bash
if [ $# -eq 0 ]
then
    echo "Please try again with a valid argument";
    exit
fi
echo "Selected process ID is: $1"
ps -q $1 -axu
```

OUTPUT:

```
ubuntu@ubuntu:~$ ps
  PID TTY          TIME CMD
 17848 pts/1    00:00:00 bash
 22615 pts/1    00:00:00 ps
ubuntu@ubuntu:~$ ./auth.sh 17848
Selected process ID is: 17848
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
ubuntu   17848  0.0  0.1  19660  5332 pts/1    Ss   03:51   0:00 bash
ubuntu@ubuntu:~$
```

RESULT: The scripts have been run successfully and output has obtained.