**LAB CYCLE - 4**

# EXPERIMENT NO:6 DATE:

SHELL SCRIPTING

**AIM:**

1. study bash syntax, environment variables, variables, control constructs such as if, for and while, aliases and functions, accessing command line arguments passed to shell scripts.

# 

# DESCRIPTION

**Shell Scripting** is an open-source computer program that can be executed/ run on the Unix/Linux shell. **Shell Scripting** is a set of instructions to write a set of commands for the shell to execute. A **shell script** is a set of instructions / commands (program) designed to be run by the Unix/Linux shell. It is a command-line interpreter and typical operations performed by **shell scripts** include printing text & values, file manipulation, program execution, and printing text, etc. The most common shell used for scripting on Unixlike systems is the Bash shell (Bourne Again Shell).

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).

1. **Environment variables**

In Linux and Unix based systems environment variables are a set of dynamic named values, stored within the system that are used by applications launched in shells or subshells. In simple words, an environment variable is a variable with a name and an associated value. Environment variables allow you to customize how the system works and the behavior of the applications on the system. For example, the environment variable can store information about the default text editor or browser, the path to executable files, or the system locale and keyboard layout settings.

There are several commands available that allow you to list and set environment variables in Linux:

* env – The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.
* printenv – The command prints all or the specified environment variables.
* set – The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.
* unset – The command deletes shell and environment variables.
* export – The command sets environment variables.

1. **Variables**

A shell variable is a character string in a shell that stores some value. It could be an integer, filename, string, or some shell command itself. Basically, it is a pointer to the actual data stored in memory. We have a few rules that have to be followed while writing variables in the script.

• 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).

1. **Control Constructs**

Control constructs allow you to repeat commands and to select certain commands over others. If the test is successful, then the commands are executed. In this way, you can use control constructs to make decisions as to whether commands should be executed. There are two different kinds of control constructs**: loops** and **conditions**. A loop repeats commands, whereas a condition executes a command when certain conditions are met. The BASH shell has three loop control structures: **while**, **for**, and **for-in**. There are two condition structures: **if** and **case**.

**if**   
This block will process if specified condition is true.

**Syntax:**

if [ expression ]

then

statement

fi

**for**

The for loop repeats a set of commands for every item in a list.

**Syntax:**

for <var> in <value1 value2 ... valuen>

do

<command 1>

<command 2>

<etc>

done

**while statement**

Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated.

while <condition>

do

<command 1>

<command 2>

<etc>

done

1. **Aliases and functions**

**Aliases:** User-defined shortcuts for commands or command sequences .Provide custom names or abbreviations for existing commands .Save time by reducing the need for typing long or complex commands .Particularly useful for frequently used or complicated commands. Enhance productivity and efficiency in executing commands.

**Functions:** Reusable blocks of code that perform specific tasks .Help organize and structure code logic .Accept input parameters (arguments) and can return results or perform actions .Promote code reuse and maintainability .Improve readability by encapsulating code into modular units .Widely used in programming languages to create flexible and reusable code.

1. **Accessing command line arguments**

Accessing command-line arguments allows programs to receive input directly from the command line when they are executed. Here's a short note on accessing command-line arguments:

* Command-line arguments are values or parameters provided to a program when it is run from the command line or terminal.
* They allow users to pass specific information to a program without modifying the source code.

**Result:** Program run successfully and output is obtained.

1. Study of startup scripts, login and logout scripts, familiarity with systemd and system 5 init scripts is expected
2. **Startup scripts**

A startup script is a set of instructions or commands that are executed automatically when a system or application starts up. It is a script or program designed to perform specific tasks during the initialization phase. Startup scripts can be used to set up system-wide settings, launch background processes, initialize services or modules, or perform any necessary tasks to prepare the environment for operation. They are commonly used in operating systems, web servers, database servers, and other software applications. Startup scripts are often executed in a specific order or sequence, ensuring that dependencies and prerequisites are met before proceeding.

Startup scripts can be customized and modified to suit specific needs, allowing administrators and developers to tailor the startup behavior of a system or application

1. **Login and logout scripts**

Login and logout scripts are scripts that are executed when a user logs into or logs out of a computer system. Here's a short note on login and logout scripts:

**Login Scripts:** Login scripts are executed when a user successfully logs into a system, typically after entering their username and password. They can be used to configure user-specific settings, perform authentication checks, set environment variables, or launch applications or services. They help customize the user experience and provide a consistent environment for each user. They can automate tasks such as mounting network drives, configuring network settings, or displaying personalized messages

**Logout Scripts:** Logout scripts are executed when a user logs out of a system or terminates their session.They are useful for performing cleanup tasks, saving user-specific data, or logging session statistics.Logout scripts can terminate background processes or services associated with the user's session.They allow administrators to enforce policies or execute actions when a user ends their session, ensuring security and system stability.Logout scripts can be used to perform tasks like clearing temporary files, closing network connections, or updating user activity logs.

1. **Systemd**

systemd refers to the systemd service management and initialization system available on Linux-based operating systems. Although shell scripts are typically separate from systemd, they can interact with systemd in a few ways:

* Service Management: Shell scripts can be used to create systemd service unit files or modify existing ones. These unit files define the behavior and configuration of a service
* Service Control: Shell scripts can use the systemctl command, which is part of systemd, to start, stop, restart, enable, disable, or query the status of services. By invoking systemctl with appropriate arguments, shell scripts can control the lifecycle of systemd-managed services.
* Service Dependencies: Shell scripts can define dependencies between services within systemd unit files. These dependencies determine the order in which services are started or stopped. Shell scripts can generate or modify unit files to establish dependencies between services, ensuring proper service coordination during system startup or shutdown.
* Systemd Environment Variables: Systemd allows the definition of environment variables that can be accessed by services during execution. Shell scripts can set or modify these environment variables to provide configuration or runtime information to systemd-managed services.
* Logging: Systemd captures log data from services using the Journal. Shell scripts can leverage systemd's journal logging capabilities to redirect or filter log messages generated by the script or services invoked by the script.

1. **System 5 init**

System V (SysV) Init is a traditional and widely used init system in Unix-like operating systems. It is named after the original System V release of Unix. SysV Init follows a sequential and script-based initialization process during system startup and shutdown. Here are some key characteristics of SysV Init:

* Initialization Scripts: SysV Init relies on shell scripts located in the /etc/init.d/ directory. These scripts are responsible for starting, stopping, restarting, and managing system services and resources. Each script corresponds to a specific service or task.
* Run levels: SysV Init defines different run levels, which represent specific system states or modes. Common run levels include 0 (halt), 1 (single-user mode), 2-5 (multi-user mode with different configurations), and 6 (reboot). Each run level has associated scripts that are executed during system startup or shutdown.
* Run level Configuration: Run level configuration files determine which scripts are executed in each run level. These files, typically located in the /etc/rc.d/ or /etc/rc.d/rc\*.d/ directories, consist of symbolic links to the corresponding init scripts. The naming convention of the symbolic links indicates the order in which the scripts are executed.
* Service Management: SysV Init provides commands such as service, chkconfig, or update-rc.d to manage services. The service command is used to start, stop, restart, or check the status of a service. Other commands handle service configuration and run level management.

**Limitations od system5 init**

* slower startup times due to sequential execution.
* Lack of parallelization.
* Complex run level configuration management.

**Result:** Program run successfully and output is obtained.

# DATE:

# AIM: SHELL SCRIPT PROGRAM QUESTIONS

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

**SOURCE CODE**

# #!/bin/bash

# date

date +"%FORMAT"

var=$(date)

var=`date`

# echo $pwd

# OUTPUT

# 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

3. Write a script to largest among three numbers

**SOURCE CODE**

# #!/bin/bash

# echo "Enter Num1"

# read num1

# echo "Enter Num2"

# read num2

# echo "Enter Num3"

# read num3

# if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]

# then

# echo "The largest number is" $num1

# elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]

# then

# echo "The largest number is" $num2

# else

# echo "The largest number is" $num3

# fi

# 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

# 5. Write a script to check password and login

**SOURCE CODE**

# #!/bin/bash

# read -p "Enter The username:" username

# read -sp "Enter The Password:" password

# if [[ $username == "mca" && $password == "mca" ]]

# then

# echo -e "\nYou're Logged In\n"

# elif [ $username != "mca" ]

# then

# echo -e "\nInvalid User Name\n"

# else

# echo -e "\nInvalid Password\n"

# fi

# 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

# echo "The prime numbers are:"

# 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"

# ((count++))

# fi

# done

# echo -e "\n There are $count 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.

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

# SOURCE CODE

# #!/bin/bash

# echo Enter the number

# read a

# echo Enter the number

# read b

# c=`expr $a + $b`

# echo Addition= $c

# d=`expr $a - $b`

# echo subtraction=$b

# e=`expr $a \\* $b`

# echo Multipliccation=$e

# 

# 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 "Name of student:"

read name

echo "student registration number:"

read student registration number

echo "Enter Marks obtained in DFS: "

read m1

echo "Enter marks obtained in OOP: "

read m2

echo "Enter marks obtained in OS: "

read m3

total=`expr $m1 + $m2 + $m3`

avg=`expr $total / 3`

echo "Total: $total"

echo "Average: $avg"

if [ $avg -ge 90 ]

then

echo "Grade = S"

elif [ $avg -le 90 ] && [ $avg -ge 50 ]

then

echo "Grade = A"

elif [ $avg -le 80 ] && [ $avg -ge 60 ]

# then

# echo "Grade = B"

# elif [ $avg -le 80 ] && [ $avg -ge 40 ]

# then

# echo "Grade = P"

# else

# echo "Grade = F"

# fi

# 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 n

# read nnum=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

# 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

# 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

# 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

# echo "Multiplication table for $number:"

# for ((i = 1; i <= 10; i++));

# do

# result=$((number \* i))

# echo "$number x $i = $result"

# done

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

# #!/bin/bash

**SOURCE CODE**

# #!/bin/bash

# read -p "Enter a decimal number: " decimal

# if ! [[ "$decimal" =~ ^[0-9]+$ ]];

# then

# echo "Invalid input. Please enter a valid decimal number."

# exit 1

# fi

# binary=""

# while [ "$decimal" -gt 0 ];

# do

# remainder=$(( decimal % 2 ))

# binary="$remainder$binary"

# decimal=$(( decimal / 2 ))

# done

# echo "Binary representation: $binary"