|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Finolex Academy of Management and Technology, Ratnagiri | | | |
| **Department of Information Technology** | | | |
| **Subject:** | **Unix Lab(**SE ITL402**)** | | | |
| **Class:** | **SE IT / Semester – IV (CBCGS) / Academic year: 2017-18** | | | |
| **Name of Student:** | **Kazi Jawwad A Rahim** | | | |
| **Roll No:** | **28** | | **Date of performance (DOP) :** | **08/03/2018** |
| **Assignment/Experiment No:** | | **06** | **Date of checking (DOC) :** |  |
| **Title: To implement basic shell script program.** | | | | |
| **Marks:** | |  | **Teacher’s Signature:** |  |

**1. Aim**: To implement basic shell script program.

**2. Prerequisites**:

C Programming Language and Operating System

**3. Hardware Requirements**:

* PC with minimum 2GB RAM

**4. Software Requirements:**

* Fedora installed.

**5. Learning Objectives:**

To learn shell script and C programming in UNIX editor environment.

**6.Course Objectives Applicable: CO1,CO4**

**7. Program Outcomes Applicable: PO2,PO3,PO4**

**8. Program Education Objectives Applicable: PEO2,PEO3,PEO4**

**Theory:**

**Shell Script**

A **shell script** is a computer program designed to be run by the Unix shell, a command-line interpreter. The various dialects of shell scripts are considered to *Editing a FreeBSD shell script for configuring ipfirewall* be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution, and printing text. A script which sets up the environment, runs the program, and does any necessary cleanup, logging, etc. is called a **wrapper**.The term is also used more generally to mean the automated mode of running an operating system shell; in specific operating systems they are called other things such as batch files (MSDos-Win95 stream, OS/2), command procedures (VMS), and shell scripts (Windows NT stream and third-party derivatives like 4NT —article is at cmd.exe), and mainframe operating systems are associated with a number of terms. The typical Unix/Linux/Posix-compliant installation includes the Korn Shell (ksh) in several possible versions such as ksh88, Korn Shell '93 and others. The oldest shell still in common use is the Bourne shell (sh); Unix systems invariably also include the C Shell (csh), Bourne Again Shell (bash), a remote shell (rsh), a secure shell for SSL telnet connections (ssh), and a shell which is a main component of the Tcl/Tk installation usually called tclsh; wish is a GUI-based Tcl/Tk shell. The C and Tcl shells have syntax quite similar to that of said programming languages, and the Korn shells and Bash are developments of the Bourne shell, which is based on the ALGOL language with elements of a number of others added as well. On the other hand, the various shells plus tools like awk, sed, grep, and BASIC, Lisp, C and so forth contributed to the Perl programming language. Other shells available on a machine or available for download and/or purchase include ash, msh, ysh, zsh (a particularly common enhanced Korn Shell), the Tenex C Shell (tcsh), a Perl-like shell (psh) and others. Related programs such as shells based on Python, Ruby, C, Java, Perl, Pascal, Rexx &c in various forms are also widely available. Another somewhat common shell is osh, whose manual page states it "is an enhanced, backward compatible port of the standard command interpreter from Sixth Edition UNIX." Windows-Unix interoperability software such as the MKS Toolkit, Cygwin, UWIN, Interix and others make the above shells and Unix programming available on Windows systems, providing functionality all the way down to signals and other interprocess communication, system calls and APIs. The Hamilton C Shell is a Windows shell that is very similar to the Unix C Shell. Microsoft distributes Windows Services for UNIX for use with its NT-based operating systems in particular, which have a Posix environmental subsystem.

Shell scripts often serve as an initial stage in software development, and are often

subject to conversion later to a different underlying implementation, most commonly being converted to Perl, Python, or C. The interpreter directive allows the implementation detail to be fully hidden inside the script, rather than being exposed as a filename extension, and provides for seamless reimplementation in different languages with no impact on end users. While files with the ".sh" file extension are usually a shell script of some kind, most shell scripts do not have any filename extension.

**Program 1: Write a Shell script program to swap values in two variables?**

**SOURCE CODE:**

echo -n “Enter the value of x: ”

read x

echo -n “Enter the value of y: ”

read y

echo -n “Before swapping X=$x and Y=$y”

t=$x

x=$y

y=$t

echo -n “After swapping X=$x and Y=$y”

**OUTPUT:**

[students@localhost ~]$ vi swap.sh

[students@localhost ~]$ sh swap.sh

Enter the value of x: 4

Enter the value of y: 5

Before swapping X=4 and Y=5

After swapping X=5 and Y=4

[students@localhost ~]$ sh sh swap.sh

**Program 2: Write a shell script program to simulate a simple calculator**

**SOURCE CODE 1:**

a=$1

op=$2

b=$3

if [ $# -lt 3 ]

then

echo "$0 num1 opr num2"

echo "opr can be +,-,\*,/"

exit 1

fi

case "$op" in

+) echo $(($a + $b));;

-) echo $(($a - $b));;

x) echo $(($a \* $b));;

/) echo $(($a / $b));;

\*) echo "Error";;

esac

**OUTPUT:**

[students@localhost ~]$ sh arith\_operation.sh 5 x 9

45

**SOURCE CODE 2:**

echo -n "Enter first number"

read a

echo -n "Enter second number"

read b

val1=`expr $a + $b`

val2=`expr $a - $b`

val3=`expr $a \\* $b`

val4=`expr $a / $b`

echo "Addition=$val1"

echo "Subtraction=$val2"

echo "Multiplication=$val3"

echo "Division=$val4"

**OUTPUT:**

[students@localhost ~]$ vi jk.sh

[students@localhost ~]$ sh jk.sh

Enter first number20

Enter second number5

Addition=25

Subtraction=15

Multiplication=100

Division=4

**Program 3: Write a shell program to display Fibonacci series.**

**SOURCE CODE using While loop :**

a=0

b=1

i=1

echo -n "Enter the nth term "

read n

echo $a

while [ $a -lt $n ]

do

b=`expr $a + $b`

echo $b

c=$a

a=$b

b=$c

i=`expr $i + 1`

done

**OUTPUT:**

[students@localhost ~]$ vi fibonacci.sh

[students@localhost ~]$ sh fibonacci.sh

Enter the nth term 6

0 1 1 2 3 5 8

**SOURCE CODE using For loop :**

a=0

b=1

echo -n "Enter nth term"

read n

for ((i=0;i<=n;i++))

do

echo -n “$a ”

c=$((a+b))

a=$b

b=$c

done

**OUTPUT:**

[students@localhost ~]$ sh JK1.sh

Enter nth term6

0 1 1 2 3 5 8

**Progarm 6: Write a shell script program to find a number is even or odd.**

**SOURCE CODE:**

echo "Enter a number"

read n

b=`expr $n % 2`

if [ $b -eq 0 ]

then

echo "$n is even"

else

echo "$n is odd"

fi

**OUTPUT:**

[students@localhost ~]$ sh JK2.sh

Enter a number

6

6 is even

[students@localhost ~]$ sh JK2.sh

Enter a number

7

7 is odd

**Program 5: Write a shell program to find largest of three numbers.**

**SOURCE CODE:**

echo -n "Enter first number"

read a

echo -n "Enter second number"

read b

echo -n "Enter third number"

read c

if [ $a -gt $b ] && [ $a -gt $c ]

then

echo -n "$a is greater"

elif [ $b -gt $a ] && [ $b -gt $c ]

then

echo -n "$b is greater"

elif [ $c -gt $a ] && [ $c -gt $b ]

then

echo -n "$c is greater"

fi

**OUTPUT:**

[students@localhost ~]$ sh JK.sh

Enter first number10

Enter second number20

Enter third number30

30 is greater

**Program 6: Write a shell program to concatenate two strings and display the resultant string along with its string length.**

**SOURCE CODE:**

echo "Enter string 1"

read name1

echo "Enter string 2"

read name2

name3=$name1$name2

echo "Concatenated string is $name3 "

len=`expr length $name3`

echo "Length=$len"

**OUTPUT:**

[students@localhost ~]$ sh JK4.sh

Enter string 1

Jawwad

Enter string 2

Kazi

Concatenated string is JawwadKazi

Length=10

**Program 7: Write a program to check given string is palindrome or not.**

**SOURCE CODE:**

echo "Enter a string"

read name

name1=$( echo $name | rev )

if [ $name = $name1 ]

then

echo "$name is palindrome"

else

echo "$name is not palindrome"

fi

**OUTPUT:**

[students@localhost ~]$ sh JK3.sh

Enter a string

MADAM

MADAM is palindrome

[students@localhost ~]$ sh JK3.sh

Enter a string

JAWWAD

JAWWAD is not palindrome

**Program 8: Write a shell program to input a number and check it is a Armstrong number.**

**SOURCE CODE:**

echo "Enter a number"

read n

d=0

sum=0

r=0

m=$n

while [ $n -gt 0 ]

do

d=$(($n % 10))

r=`expr $d\*$d\*$d`

n=$(($n / 10))

sum=$(($sum + $r))

done

if [ $m -eq $sum ]

then

echo "$m is Armstrong"

else

echo "$m is not Armstrong"

fi

**OUTPUT:**

[students@localhost ~]$ vi p8.sh

[students@localhost ~]$ sh p8.sh

Enter a number

153

153 is Armstrong

[students@localhost ~]$ sh p8.sh

Enter a number

100

100 is not Armstrong

**Program 9: Write a shell script program to enter a number and find its reverse. Also find it is Palindrome or not.**

**SOURCE CODE:**

echo "Enter a number"

read n

d=0

r=0

m=$n

while [ $n -gt 0 ]

do

d=$(($n % 10))

r=$(($r\*10 + $d))

n=$(($n / 10))

done

echo "Reverse of $m = $r "

if [ $m -eq $r ]

then

echo "$m is Palindrome"

else

echo "$m is not Palindrome"

fi

**OUTPUT:**

[students@localhost ~]$ vi p9.sh

[students@localhost ~]$ sh p9.sh

Enter a number

1111

Reverse of 1111 = 1111

1111 is Palindrome

[students@localhost ~]$ sh p9.sh

Enter a number

1234

Reverse of 1234 = 4321

1234 is not Palindrome

**Program 10: Write a shell script program to print sum of first nth numbers.**

**SOURCE CODE:**

echo "Enter nth term"

read n

s=0

for ((i=1;i<=n;i++))

do

s=`expr $s + $i`

done

echo "Sum of first $n term=$s"

**OUTPUT:**   
[students@localhost ~]$ vi p10.sh

[students@localhost ~]$ sh p10.sh

Enter nth term

10

Sum of first 10 term=55

**Learning Outcomes Achieved :**

Learned shell script and C programming in UNIX editor environment.

**Conclusion:**

Hence, we have performed some programs in shell script as we have performed in C programming.

**13. Expriment/Assignment Evaluation**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SR** | **Parameters** | **Weight** | **Excellent** | **Good** | **Average** | **Poor** | **Not as per requirement** |
| **Scale Factor ->** | 5 | 4 | 3 | 2 | 0 |
| 1 | Technical Understanding | 25 |  |  |  |  |  |
| 2 | Performance / Execution | 25 |  |  |  |  |  |
| 3 | Question Answers | 20 |  |  |  |  |  |
| 4 | Punctuality | 20 |  |  |  |  |  |
| 5 | Presentation | 10 |  |  |  |  |  |
|  | Total out of 100 -->  #(to be converted as per term-work evaluation applicable to the subject) | | **∑ (Weight \* Scale Factor)/5 = \_\_\_\_\_\_\_\_** | | | | |

**References**:

[1] Unix, concepts and applications by Sumitabha Das, McGraw-Hill

[2] Mastering Shell Scripting, Randal. K. Michael, Second Edition, Wiley Publication

**Viva Questions**

* What is shell scripting?
* How can we print any statement in shell script?
* How can we read any value from user?