Operating Systems Lab

Lab # 13

Shell Scripting

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL2006

Department of Computer Science,

National University of Computer and Emerging Sciences FAST

Peshawar Campus

Table of Contents

Shell Scripting

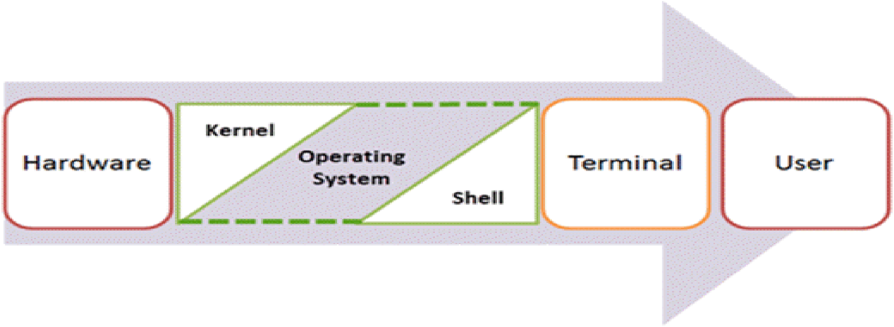
Shell programming is a group of commands grouped together under single filename. IntroductionAfter logging onto the system a prompt for input appears which is generated by a Command String Interpreter program called the shell. The shell interprets the input, takes appropriate action, and finally prompts for more input. The shell can be used either interactively - enter commands at the command prompt, or as an interpreter to execute a shell script. Shell scripts are dynamically interpreted, NOT compiled.

What is Shell?

Shell is a UNIX term for an interface between a user and an operating system service. Shell provides users with an interface and accepts human-readable commands into the system and executes those commands which can run automatically and give the program's output in a shell script.

An Operating is made of many components, but its two prime components are –

- Kernel
- Shell



Components of Shell Program

A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.

A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**.

Why Shell scripts?

Since the user cannot interact with the kernel directly, Shell programming skills are a must to be able to exploit the power of UNIX to the fullest extent. A shell script can be used for variety of tasks and some of them are listed below.

Uses of Shell scripts

1. Customizing your work environment. For Example Every time you login, if you want to see the current date, a welcome message, and the list of users who have logged in you can write a shell script for the same.
2. Automating your daily tasks. For example, to back up all the programs at the end of the day.
3. Automating repetitive tasks.
4. Executing important system procedures, like shutting down the system, formatting a disk, creating a file system etc.
5. Performing some operations on many files.

Shell Keywords

echo, read, if fi, else, case, esac, for , while , do , done, until , set, unset, readonly, shift, export, break, continue, exit, return, trap , wait, eval ,exec, ulimit , umask.

General things in Shell

The shebang line	The "shbang" line is the very first line of the script and lets the kernel know what shell will be interpreting the lines in the script. The shbang line consists of a #! followed by the full pathname to the shell, and can be followed by options to control the behavior of the shell. **EXAMPLE** #!/bin/sh
Comments	Comments are descriptive material preceded by a # sign. They are in effect until the end of a line and can be started anywhere on the line. **EXAMPLE**

	# this text is not # interpreted by the shell
Wildcards	There are some characters that are evaluated by the shell in a special way. They are called shell metacharacters or "wildcards." These characters are neither numbers nor letters. For example, the *, ?, and [ ] are used for filename expansion. The <, >, 2>, >>, and | symbols are used for standard I/O redirection and pipes. To prevent these characters from being interpreted by the shell they must be quoted. **EXAMPLE** Filename expansion: rm *; ls ??; cat file[1-3]; Quotes protect metacharacter: echo "How are you?"

Types of Shell

There are two main shells in Linux:

1. The **Bourne Shell**: The prompt for this shell is $ and its derivatives are listed below:

- POSIX shell also is known as sh
- Korn Shell also knew as sh
- **B**ourne **A**gain **SH**ell also knew as bash (most popular)

2. The C shell: The prompt for this shell is %, and its subcategories are:

- C shell also is known as csh
- Tops C shell also is known as tcsh

How to Write Shell Script in Linux/Unix

Shell Scripts are written using text editors. On your Linux system, open a text editor program, open a new file to begin typing a shell script or shell programming, then give the shell permission to execute your shell script and put your script at the location from where the shell can find it.

Let us understand the steps in creating a Shell Script:

1. **Create a file using** a **vi** editor(or any other editor). Name script file with **extension .sh**
2. **Start** the script with **#! /bin/sh**
3. Write some code.
4. Save the script file as filename.sh
5. For **executing** the script type **bash filename.sh**

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#! /bin/sh" the script gets directed to the bourne-shell.

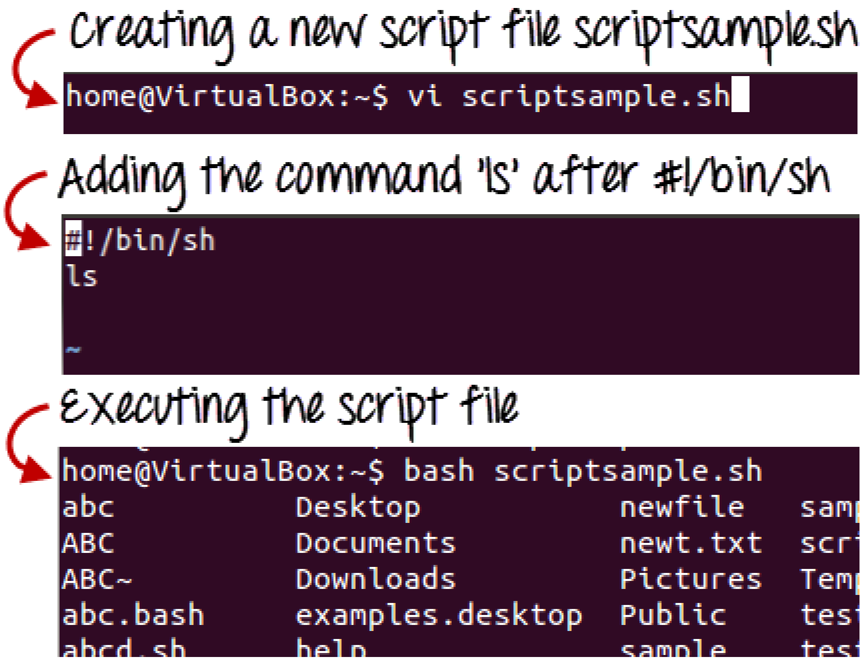
Let's create a small script –

#!/bin/sh

ls

- Basic Shell Scripting Commands in Linux: cat, more, less, head, tail, mkdir, cp, mv, rm, touch, grep, sort, wc, cut and, more.

Let's see the steps to create Shell Script Programs in Linux/Unix -



Creating a new script file scriptsample.sh

```
home@VirtualBox:~$ vi scriptsample.sh
```

Adding the command 'ls' after #!/bin/sh

```
#!/bin/sh
ls

~
```

Executing the script file

```
home@VirtualBox:~$ bash scriptsample.sh
abc            Desktop            newfile    samp
ABC            Documents          newt.txt   scr
ABC~           Downloads          Pictures   Temp
abc.bash       examples.desktop   Public     tes
abcd.sh        help               sample     tes
```

What are Shell Variables?

As discussed earlier, Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can by the shell only.

For example, the following creates a shell variable and then prints it:

```
variable ="Hello"

echo $variable
```

To read standard input into a shell script use the read command.

Below is a small script which will use a variable.

```
#!/bin/bash

echo "what is your name?"

read name

echo "How do you do, $name?"

read remark

echo "I am $remark too!"
```

Arithmetic in SHELL script

Various forms for performing computations on shell variables using expr command are:

```
expr val_1 op val_2 (Where op is operator)

expr $val_1 op $val_2

val_3=’expr $val_1 op $val_2`
```

Examples:

```
expr 5 + 7                  # Gives 12

expr 6 – 3                  # Gives 3

expr 3 \* 4                 # Gives 12

expr 24 / 3                 # Gives 8

sum=’expr 5 + 6’

echo $sum                   # Gives 11
```

Example:

```
#!/bin/bash
a=12
b=90
echo sum is $a + $b                # Will display sum is 12 + 90
echo sum is `expr $a + $b`         # Gives sum is 102
```

Example 1

Write a shell program to perform arithmetic operations

Steps

1: get the input

2: perform the arithmetic calculation.

3: print the result.

4: stop the execution.

Code

```
#!/bin/bash
echo "enter the a value"
read a
echo "enter b value"
read b
c=`expr $a + $b`
echo "sum:"$c
c=`expr $a - $b`
echo "sub:"$c
c=`expr $a \* $b`
echo "mul:"$c
c=`expr $a / $b`
echo "div:"$c
```

Operators on Numeric Variables

-eq : equal to

-ne : not equals to

-gt : grater than

-lt : less than

-ge : greater than or equal to

-le : less than equal to

Logical Operators

Combining more than one condition is done through the logical AND, OR and NOT operators.

-a : logical AND

-o : logical OR

! : logical NOT

Control Structures

if..else Statement

The Bash `if..else` statement takes the following form:

```
if TEST-COMMAND
then
 STATEMENTS1
else
 STATEMENTS2
fi
```

Example 2

```
#!/bin/bash

echo -n "Enter a number: "
read VAR

if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
else
  echo "The variable is equal or less than 10."
fi
```

if..elif..else Statement

The Bash if..elif..else statement takes the following form:

```
if TEST-COMMAND1

then

  STATEMENTS1

elif TEST-COMMAND2

then

  STATEMENTS2

else

  STATEMENTS3

fi
```

Example 3

```
#!/bin/bash

echo -n "Enter a number: "
read VAR
```

```
if [[ $VAR -gt 10 ]]
then
  echo "The variable is greater than 10."
elif [[ $VAR -eq 10 ]]
then
  echo "The variable is equal to 10."
else
  echo "The variable is less than 10."
fi
```

FOR Loop

```
for item in [LIST]

do

 [COMMANDS]

done
```

Example 4

```
#!/bin/bash
for element in Hydrogen Helium Lithium Beryllium
do
  echo "Element: $element"
done
```

Example 5: Write a sell program to check whether a number is even or odd.

Code

```
#!/bin/bash
num="1 2 3 4 5 6 7 8"
for n in $num
do
```

```
q=`expr $n % 2`
if [ $q -eq 0 ]
then
echo "even no"
continue
fi
echo "odd no"
done
```

Example 6:

Table of a given number.

Code

```
#!/bin/bash
echo " which table you want"
read n
for i in 1 2 3 4 5 6 7 8 9 10
do
echo $n "*" $i "=" `expr $i \* $n`
done
```

WHILE Loop

Example 7:

```
#!/bin/bash
a=1
while [ $a -lt 11 ]
# -ge -gt -lt -le -eq -ne
#[ $a -ne 11 -a $a -ne 12 ]
#[ $a -ne 11 -o $a -ne 12 ]
do
echo "$a"
a=`expr $a + 1`
done
```

Interpret the output of the above program.

IF Statement

Example 8:

```
#!/bin/bash
for var1 in 1 2 3
do
for var2 in 0 5
do
if [ $var1 -eq 2 -a $var2 -eq 0
]
then
continue
else
echo "$var1 $var2"
fi
done
done
```

Interpret the output of the above program.

ELSE-IF Statement

Example 9:

```
#!/bin/bash
for var1 in 1 2 3
do
for var2 in 0 5
do
if [ $var1 -eq 2 -a $var2 -eq 0
]
then
continue
else if [ $var1 -eq 2 -a $var2
-eq 5 ]
then
echo "$var1"
else
echo "$var1 $var2"
fi
fi
done
done
```

Functions

Example 10:

```
#!/bin/bash
add()
{
c=`expr $1 + $2`
echo "addition = $c"
}
add 5 10
```

Switch Statement

```
case EXPRESSION in
 PATTERN_1)
   STATEMENTS
   ;;
 PATTERN_2)
   STATEMENTS
   ;;
 PATTERN_N)
   STATEMENTS
   ;;
 *)
   STATEMENTS
   ;;
esac
```

Example 11:

```
#!/bin/bash
ch='y'
while [ $ch = 'y ' ]
do
echo "enter your choice"
echo "1 no of user loged on"
echo "2 print calender"
echo "3 print date"
read d
case $d in
1) who;;
2) cal 2022;;
3) date;;
*) break;;
esac
echo "do you wish to continue
(y/n)"
read ch
done
```

Exercise

Write a shell script to create a file with extension .c. Copy contents(c code) from another file to this file. Now use if statement to ask user if user enter 1 just compile it. If user enters 2 compile it and run it. If user enters 3 just print the contents of the original file. Otherwise print the contents of the current directory. Perform the same task using switch inside a function.

References

https://www.guru99.com/introduction-to-shell-scripting.html

https://linuxize.com/post/bash-if-else-statement/