Part 2: Getting Started with Linux Shell Programming

Shell Programming

Topics:

- What is a Kernel?
- What is a Linux Shell?
- What is a Shell Script?
- Why to write a Shell Script?
- How to write a Shell Script?
- How to execute a Shell Script?
- Variables in Shell
- Actual Shell Scripting

Shell Programming

- **What is a KERNEL?**
- Kernel is the heart of Linux OS.
- It manages resource of Linux OS.
- Resources means facilities available in Linux.

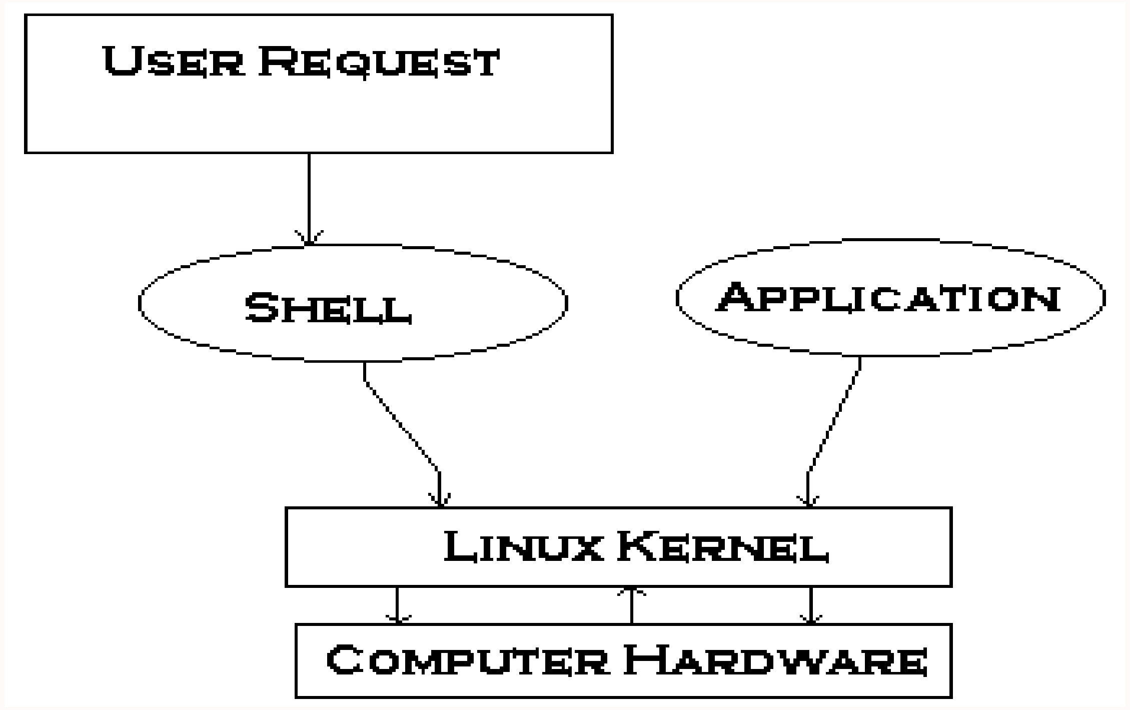
Examples: Facility to store data, print data on printer, memory, file management etc .

- Kernel decides who will use this resource, for how long and when. It runs your programs (or set up to execute binary files).

Shell Programming

- **What is a KERNEL?**

•The kernel acts as an intermediary between the computer hardware and various programs / application/shell.



Shell Programming

- **What is a KERNEL?**

- It is Memory resident portion of Linux. It performs the following tasks :
    - I/O management
    - Process management
    - Device management
    - File management
    - Memory management
    -

Shell Programming

- **What is a LINUX SHELL?**

- Shell is a user program or its environment provided for user interaction.

- Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

- Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

Shell Programming

- **What is a LINUX SHELL?**

Several shell available with Linux including:

Shell Name	Developed by	Where	Remark
BASH (Bourne-Again SHell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	
TCSH	See the man page. Type $ man tcsh		TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

Shell Programming

- **What is a SHELL SCRIPT?**

Shell Script is **series of commands** written **in plain text file**. Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file."

Shell Programming

- **Why to write a SHELL SCRIPT?**

Shell script can take input from user, file and output them on screen.

- Useful to create our own commands.
- Save lots of time.
- To automate some task of day today life.
- System Administration part can be also automated.

Shell Programming

□ Practical examples where shell scripting actively used

□Monitoring your Linux system.

□● Data backup and creating snapshots.

□● Dumping Oracle or MySQL database for backup.

□● Creating email based alert system.

□● Find out what processes are eating up your system resources.

□● Find out available and free memory.

□● Find out all logged in users and what they are doing.

□● Find out if all necessary network services are running or not. For example if web server failed then send an alert to system administrator via a pager or an email.

□● Find out all failed login attempt, if login attempt are continue repeatedly from same network IP automatically block all those IPs accessing your network/service via firewall.

□● User administration as per your own security policies.

□● Find out information about local or remote servers.

Shell Programming

- **How to write a SHELL SCRIPT?**

Following steps are required to write shell script:

- Use any editor like vi or mcedit to write shell script.
- After writing shell script set execute permission for your script as follows:

Syntax: **chmod permission your-script-name**

Examples:

```
$ chmod +x your-script-name
$ chmod 755 your-script-name
```

Note: This will set read write execute(7) permission for owner, for group and other permission is read and execute only(5).

Shell Programming

- **How to execute a SHELL SCRIPT?**

Syntax:

```
bash your-script-name
sh your-script-name
./your-script-name
```

Examples:

```
bash bar
sh bar
./bar
```

NOTE: In the last syntax ./ means current directory, But only . (dot) means execute given command file in current shell without starting the new copy of shell. The syntax for . (dot) command is as follows:

Shell Programming

- **Sample Coding of SHELL SCRIPT?**

Now you are ready to write first shell script that will print **"Knowledge is Power“** on screen.

Syntax:	**$ vi first**
Type:	**# My first shell script**
	clear
	echo "Knowledge is Power“
To Save, press	**Alt+Shift : wq**

Shell Programming

- **Sample Coding of SHELL SCRIPT**

After saving the above script, you can run the script as follows:

Syntax: **./first**

This will not run script since we have not set execute permission for our script *first*; to do this type command

Syntax:

```
chmod 755 first
./first
```

First screen will be clear, then **Knowledge is Power** is printed on screen.

Shell Programming

- **Sample Coding of SHELL SCRIPT**

Script Command(s)	**Meaning**
$ vi first	Start vi editor
# # My first shell script #	# followed by any text is considered as comment. Comment gives more information about script, logical explanation about shell script. *Syntax:* # comment-text
Clear	clear the screen
echo "Knowledge is Power"	To print message or value of variables on screen, we use echo command, general form of echo command is as follows *syntax:* echo "Message"

Shell Programming

- **Variables in SHELL**

•To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data.

•Programmer can give a unique name to this memory location/address called **memory variable or variable** (It is a named storage location that may take different values, but only one at a time).

Shell Programming

- **Variables in SHELL**

  In Linux (Shell), there are two types of variable:

- **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.

- **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

-

Shell Programming

- **SHELL Arithmetic**
  Use to perform arithmetic operations.

Syntax: **expr op1 math-operator op2**

Examples:

```
expr 1 + 3
expr 2 - 1
expr 10 / 2
expr 20 % 3
expr 10 \* 3
echo `expr 6 + 3`
```

Note:
expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2.
expr 10 * 3 - Multiplication use * and not * since its wild card.

Shell Programming

- **Variables in SHELL**

•How to create a Shell Script using User Defined Variables?

Syntax: **vi sum**

Type:

```
echo “Enter a number: “
read x
echo “Enter another number: “
read y
echo “The sum of $x  and $y is”
`expr $x + $y`
```

To Save & Exit: ***Alt+Shift: wq***

To Execute: **./sum**

Shell Programming

- **Variables in SHELL**

•**How to assign expression in a User Defined Variable?**

Syntax:

```
sum=`expr $x + $y`
```

Shell Programming

- **Condition Statements**

•**if condition**

•**if condition** is used for decision making in shell script, if given condition is true then command1 is executed.

Syntax:

```
if condition
then
command1 if condition is true or if exit status
of condition is 0 (zero)
...
...
fi
```

Shell Programming

• Condition Statements

•Condition is defined as:

"Condition is nothing but comparison between two values."

•Expression is defined as:

"An expression is nothing but combination of values, relational operator **(such as >,<, <> etc)** and mathematical operators **(such as +, -, / etc).**"

Shell Programming

- **Condition Statements**

•**Relational Operators**

Shell Programming

- **Condition Statements**

•String Comparison and Logical Operators

Shell Programming

- **Condition Statements**

•**test command or [expr]**

test command or [expr] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.

Syntax:

test expression OR [expression]

Shell Programming

- **Condition Statements**

•**test command or [expr]**

Example: **(Using test command)**

•Following script determine whether given argument number is positive.

```
$ cat > pos
# Script to see whether argument is positive
if test $1 –gt 0
then
echo "$1 number is positive“
else
echo "$1 number is negative“
fi
```

Shell Programming

- **Condition Statements**

•**test command or [expr]**

Example: **(Using [expr])**

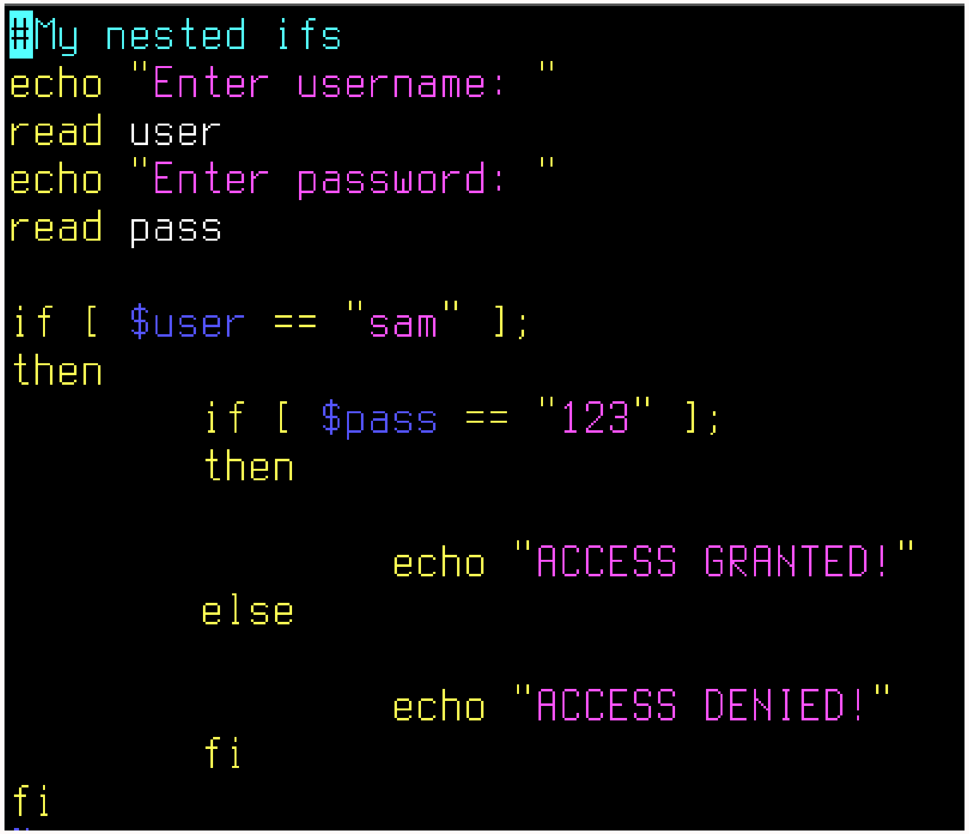
```
$ cat > pos
Echo “Enter a number: “
Read x
if [ $x –gt 0 ]
then
echo "$x number is positive“
else
echo "$x number is negative“
fi
```

Shell Programming

- **Condition Statements**

•**Nested (if…else…fi)**

Example: **$ vi nestedif.sh**



```
#My nested ifs
echo "Enter username: "
read user
echo "Enter password: "
read pass

if [ $user == "sam" ];
then
        if [ $pass == "123" ];
        then

                echo "ACCESS GRANTED!"
        else

                echo "ACCESS DENIED!"
        fi
fi
```

Shell Programming

- **Condition Statements**

•**Multilevel (if…then…else)**

Syntax:

```
 if condition
then
        condition is zero (true - 0)
        execute all commands up to elif statement
elif condition1
then
        condition1 is zero (true - 0)
        execute all commands up to elif statement
elif condition2
then
        condition2 is zero (true - 0)
        execute all commands up to elif statement
else
        None of the above condtion,condtion1,condtion2 are true (i.e.
        all of the above nonzero or false)
        execute all commands up to fi
fi
```

Shell Programming

- **Condition Statements**

•Multilevel (if…then…else)

Example:

```
$ cat > elf
echo “Enter a number: “
read x
        if [ $x -gt 0 ]; then
                echo "$1x is positive"
        elif [ $x -lt 0 ]
        then
                 echo "$x is negative"
        elif [ $x -eq 0 ]
        then
                echo "$x is zero"
        else
                 echo "Opps! $x cannot be determined!"
        fi
```

Shell Programming

• Case Statements

The case statement is good alternative to Multilevel if-then-else-fi statement. It enables you to match several values against one variable. Its easier to read and write.

Syntax:

```
case  $variable-name  in
  pattern1)   command
                 ...
                 command;;
  pattern2)   command
                                ...
                                command;;
patternN)   command
               ...
                                command;;
*)             command
                ...
                                command;;
                esac
```

Shell Programming

- **Case Statements**

Example:

```
$ vi switch.sh
```

Shell Programming

- **Looping Statements**

Loop is defined as "Computer can repeat particular instruction again and again, until particular condition satisfies. A group of instruction that is executed repeatedly is called a **loop.**"

Bash supports:

•for loop
•while loop

Note that in each and every loop,
1.First, the variable used in loop condition must be initialized, then execution of the loop begins.
2.A test (condition) is made at the beginning of each iteration.
3.The body of loop ends with a statement that modifies the value of the test (condition) variable.

Shell Programming

- **Looping Statements**

Using *for* Loop

Syntax 1:

for *{ variable name } in { list }*

do

execute one for each item in the list until the list is not finished (And repeat all statement between do and done)

done

Shell Programming

- **Looping Statements**

Using *for* Loop

Syntax 2:

```
for (( expr1; expr2; expr3 ))
do
      .....
      ...
      repeat all statements between do and
      done until expr2 is TRUE
done
```

Shell Programming

- **Looping Statements**

  Example:

```
$ cat > testfor
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

Shell Programming

- **Looping Statements**

Using *for* Loop

Syntax:

```
$ cat > for2
for ((  i = 0 ;  i <= 5;  i++  ))
do
echo "Welcome $i times"
done
```

Shell Programming

- **Looping Statements**

Example:

```
$ cat > mutlitable
echo “Enter a number: “
read x
n=$x
for i in 1 2 3 4 5 6 7 8 9 10
do
echo “$n * $i = `expr $i \* $n`”
done
```

Shell Programming

- **Looping Statements**

  Nested for loops

  Example:

```
$ vi nestedfor.sh
for (( i = 1; i <= 5; i++ ))      ### Outer for loop ###
do

          for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###
          do
          echo -n "$i "
          done

          echo "" #### print the new line ###

done
```

Shell Programming

- **Looping Statements**

  Nested for loops

  Output:

```
$ chmod +x nestedfor.sh
$ ./nestefor.sh
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Shell Programming

- **Looping Statements**

While loop

Syntax:

```
while [ condition ]
 do
        command1
        command2
        command3
        ..
        ....
        done
```

Shell Programming

- **Looping Statements**

While loop

Example:

```
while [ condition ]
 do
        command1
        command2
        command3
        ..
        ....
        done
```

Shell Programming

- **Others**

•**shutdown script**

Example:

•Following script will shutdown your computer

$ cat > power

`shutdown –h now`

Shell Programming

- **Others**

•**shutdown script**

Example:

# This script displays the date, time, username and # current directory. echo "Date and time is:" date echo echo "Your username is: `whoami` \n" echo "Your current directory is: \c" pwd

Shell Programming

End of Part 2

Thank you.