SHELL SCRIPTING

SHELL:- shell is command line interpreter. It takes commands form user and executes them. It is an interface between user and kernel.

The three most widely used UNIX shells are Bourne shell, korn shell and c shell.

Shell	**Developed by**	**Sell prompt**	**Execution command**
Bourne shell	Steve bourne	$	Sh
Korn shell	David korn	$	Ksh
C shell	Bill joy, California university student	%	Csh

Each shell has merits and demerits of its own. Moreover the shell scripts written for one shell may not work with the other shell. This is because different shells use different mechanisms to execute the commands in the shell script. Bourne shell since it is one of the most widely used UNIX sells in existence today.

Almost all UNIX implementations offer the Bourne sell as part of their standard configuration. It is smaller than the other two sell s and therefore more efficient for most shell processing. However, it lacks features offered by the c and the korn shell.

All shell programs written for the Bourne shell are likely to work with the korn shell. The reverse however may not be true, this is so since the facilities like arrays, command aliasing and history mechanism available in the korn shell are not supported by the Bourne shell.

The c shell programming language resembles the c language and is quite different from the language of the Bourne shell. only the very basic shell scripts will run under both the c and Bourne shell; a vast majority will not shell keeps track of commands as you enter them (history) and allows you to go back and execute them again without typing the command. Or, if you want to, you can recall them, make modifications, and then execute the new command.

Shell program:-

A shell program is nothing but a series of such commands. Instead of specifying one job at a time, we give the shell a to –do list – a program – that carries our entire procedure. Such programs are known as “shell scripts”.

When to use shell scripts:

1. Customizing your work environments. For example, every time you log in 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 scripts for the same.

2. Automating your daily tasks. For example, you may want to back up all your programs at the end of day. This can be done using a shell script.

3. Automating repetitive tasks. For example, the repetitive task of compiling a c program, linking it with some libraries and executing the executable code can be assigned to a shell script.

4. Executing important system procedures like shutting down the system, formatting a disk, creating a file system, mounting the file system, letting the users the floppy and finally un mounting the disk.

5. Performing same operation on many files. For example, you may want to replace a string prontf with a sting by printf in all the c programs present in a directory.

Shell variables:-

Variable is a data name and it is used to store value. Variable value can change during execution of the program.

Variables in Unix are two types.

Unix-defined variables or system variables.
User defined variables.

Unix –defined variables:

These are standard variables which are always accessible the shell provides the values for these variables these variable are usually used by the system itself and govern the environment we work under. If we so desire we can change the values of these variables as per our preferences and customize the system environment.

The list of all system variables and their values can be displayed by saying at the $prompt,

```
$set
HOME=/usr/crimson
HZ=100
IFS=
LOGNAME=crimson
MAILCHECK=600
OPTIND=1
PATH=/bin:/usr/bin:/usr/crimson:/bin:.
PS1=$
PS2=>
SHELL=/bin/sh
TERM=vt100
TZ=IST-5:30
```

Variable	**Meaning**
PS1	Primary shell prompt
PS2	The system prompt 2, default value is “>”
PATH	Defines the path which the shell must search in order to execute any command for file
HOME	Stores the default working directory of the user.
LOGNAME	Stores the login name of the user
MAIL	Defines the file where the mail of he user is stored
MAILCHECK	Defines the duration after which the shell checks whether the user has received any mail. By default its value is 600(seconds)
IFS	Defines the internal field separator, which is a space, a tab or a new line
SHELL	Defines the name of you default working shell
TERM	Defines the name of the terminal on which you are working
TZ	Defines the name of the time Zone in which we are working.

User Defined Variables

These are defined by user and are used most extensively in shell programming.

Rules for crating User Defined Shell Variables:-

1. The first character of a variable name should be alphabet or underscore
2. No commas or blanks are allowed within a variable name
3. Variables names should of any reasonable length
4. Variable names area case sensitive. That is name, Name, nAme, name are all different variable names
5. Variable name shouldn't be a reserve word

Shell keywords:-

Keywords are the words whose meaning has already been explained to the shell. The key words are also called as "Reserve Words".

The lists of keywords available in Bourne shell are

Echo	Until
If	Trap
Read	Case
Else	Wait
Set	Esac
Fi	Eval
Unset	Break
While	Exec
Read only	Continue
Do	Ulimit
Shift	Exit
Done	Umask
Export	Return
for	

1. **Echo**

Echo command is used to display the messages on the screen and is used to display the value stored in a shell variable.

E.g.1: $echo “crimson is a training institute”
Crimson is a training institute
Note: double quotes are option in echo statement

E.g.2: $echo “today date is:’date’ “
Today date is sun July 12 11:20:15 IST 2009
Note: the unix command should be in back quotes in echo statement otherwise it treat as text

E.g.3 $echo “my file has ‘wc –l file1’ lines
My file has 10 lines

E.g.4: $echo “my log name is: ‘log name’ “
My log name is crimson

E.g.5: $echo “my present working directory is: ‘pwd’ “
My present working directory is: /usr/crimson/abc

Shell variables

E.g.1: $a=10

Note: ther are no predefined data types in unix. Each and every thing it treat as character. Each character occupies 1 byte of memory.

E.g.2:$b=2000
In above example a occupies 2 bytes in Unix**.** Each and everything it treats as character. Each character occupies 1 byte of memory.

$is the operator to read variable value
E.g.1:$n=100
$echo $n
100
E.g.2:$name=”Crimson”
$echo $name
Crimson
$echo welcome to $name
Welcome to crimson

E.g.3:$now=date
$now
Sun July 12 09:35:21 IST 2009

E.g.4: $mypath=/usr/crimson/abc/a1/a2
$cd $mypath

Now it changes to a2 directory, to check say at $ promt pwd command
$pwd
/usr/crimson/abc/a1/a2

Null variables

A variable which has been defined but has not been given any value is known as a null variable. A null variable can be created in any of the following ways.
1.$n=””
2.$n=’ ‘
3.$n=

$echo n
On echoing a null variable, only a blank line appears on the screen.

Constant:

Constant is a fixed value. It doesn’t change during execution of the program
$a=20

$readonly a

When the variables are made read only, the shell does not allow us to change their values. So a value can read but can’t change.

Note: if we want the shell to forget about a variable altogether, we use the unset command.

$unset a
On issuing the above command the variable a and with it the value assigned to it are erased from the shell’s memory.

SI.No	Escape Character	Meaning	Example
1	“\07”	Bell Sound	$echo “hello\07”
2	“\n”	New line	$echo “hello \n crimson” Hello Crimson
3	“\t”	Tab space	$echo “hello \t crimson” hello crimson
4	“\b”	Backspace	$echo “hello\bcrimson Hellcrimson
5	“\r”	Carriage return i.e it places cursor beginning of the current line	$echo “welcome to \r crimson “ Crimsono
6	“\c”	It places cursor at end of the current statement	$echo “hello ….\c” Hello ….
7	\”	Double quote	$echo “\”hello\”” “Hello”
8	\’	Single quote	$echo “\’hello\”’ ‘Hello’
9	\\	Back slash	$echo \\hello\\ \Hello\
10	”\033[om”	Normal characters	$echo”\003[om Crimson” Crimson
11	”\033[1m”	Bold characters	$echo”\003[1m Crimson” Crimson
12	”\033[5m”	Blinkin charcters	$echo”\003[5m Crimson”
13	”\033[7m”	Reverse video charcters	$echo”\003[7m Crimson”

Shell Programs

1. write a program to display list of files, the current working user's list and present working directory

vi sp1

ls –x
who
pwd

:wq(save and quit)

Execution of shell program:

sh is the command to execute bourne shelll programs

eg: $sh sp1
or
$chmod 744 sp1
$sp1

2. write program to display address

vi sp2

echo "Crimson Solutions"
echo "No:109, 1stfloor,"
echo "Annapurna block,"
echo "Aditya Enclave,"
echo "Hyderabad"
echo "ph.no:5583966 / 9440327911"

:wq(save and quit)

3) write a program count no of users are currently logged into the system

vi sp3

echo “Three are ‘who | wc –l ‘ users”

:wq(save and quit)

Note: - read is command to read variable value form the user.read command reads value form keyboard upto space or enter key.
Eg 1: read a
Eg 2: read a b c

4) write program read name and display

vi sp4

echo “what is your name?”
read name
echo “hello $name”

:wq(save and quit)

Note:- read is a command to read variable value form the user. read command reads value form keyboard up to space or enter key.
Eg 1: read a
Eg 2: read a b c

5) write program read 2 numbers and display

vi sp5

echo “enter 2 numbers:\c”
read a b
echo “your numbers are $a and $b

:wq(save and quit)

OPERATORS:

1. Arithmetic operators
2. Relational operators
   a. Numeric comparison operators
   b. String comparison operators
3. Logical operators

1. Arithmetic operators

Operator	**Meaning**
+	Addition
-	Subtraction
*****	Multiplication
/	Division
%	Modulus division

2. Numeric Comparison operators

Operator	**Meaning**
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal to
-eq	Equal to
-ne	Not equal to

3. String Comparison operators

Operator	**Meaning**
>	Greater than
<	Less than
=	Equal to
!=	Not equal to

```
c=’echo $a - $b | bc’
echo “a-b=$c”
c=’echo $a \* $b | bc’
echo “a*b=$a”
```

c=’echo $a / $b | bc’
echo “a/b=$c”

:wq(save and quit)

CONTROL STATEMENTS:

There are four types of control instructions in shell. They are:

1. Sequence control instruction
2. Selection or Decision Control Instructions
3. Repetition or loop Control Instruction
4. Case Control Instruction

The sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program. Decision and case control instructions allow the computer to take a decision as to which instruction is to be executed next. The loop control instruction is to be executed next. The loop control instruction helps computer to execute a group of statements repeatedly.

Decision Control statement

if -then-fi statement
if-then-else-if statement
if-then-else-fi statement
case-esac statement g

1. if-then-fi statement
Syntax:

```
if control command
   ----------------------
  ----------------------
   ----------------------
fi
```

The if statement of UNIX is concerned with the exit status of a command. The exit status indicates whether the command was executed successfully or not.

The exit status of a command is 0 if it has been executed successfully, 1 otherwise.

8) write a program to change directory

vi sp8

```
echo enter directory name
read dname
if cd $dname
then
echo “changed to $dname”
pwd
fi
```

:wq(save and quit)

2.if-then-else-fi statement

Syntax

```
if condition
----------
----------
else
----------
----------
fi
```

The exit status of the control is 0 then it executes then statements otherwise it executes else statements.

9) Write a program to copy a file

vi sp9

```
echo enter source, filename and target file name
read src trg
if cp $src $trg
then
echo file copied successfully
else
```

```
echo failed to copy the file
fi
```

:wq(save and quit)

10) Write a program to search string in a file

vi sp1o

```
echo “enter a file name”
read fname
echo “enter to string to search”
read str
if grep $str $fname
then
echo “$str is found in the $file”
else
echo “str is not found in $fname”
fi
```

:wq(save and quit)

11) write a program find greatest number of 2 numbers

vi sp11

```
echo enter two numbers
read a b
if [ $a –gt $b ]
then
echo $a is the greatest value
else
echo $b is the greatest value
fi
```

:wq(save and quit)

12) Write a program to check given no is even or odd

vi sp12

```
echo enter a number
read n
if [ ‘expr $n % 2’ –eq  0 ]
then
echo $n is even number
else
echo $n is odd number
fi
```

:wq(save and quit)

The test Command

if constructs depends upon whether or not the condition results into true or not.

If constructs are generally used in conjunction with the test command.

13) write a program to find student result

vi sp13

```
echo “enter three subjects marks:”
read m1 m2 m3
if [ $m1 –ge 40 ]
then
    if [ $m2 –gt40 ]
    then
        if [ $m2 –gt 40 ]
        then
            echo “PASS”
        else
            echo “FAIL”
        fi
    else
        echo “FAIL”
    fi
else
```

```
    echo “FAIL”
fi
:wq(save and quit)
```

14) Write a program to print greeting
vi sp16

```
hour=’date | cut –c 12,13’

if[$hour–ge 0 –a $hour –le 11 ]
then
echo “Good Morning”
else
    if [ $hour –ge 12 –a $hour –le 17 ]
    then
    echo “Good Afternoon”
    else
    echo “Good Evening”
    fi
fi
:wq(save and quit)
```

File Test Commands

The test commands have several options for checking the status of a file.

-s file	True if the file exists and has a size greater than 0
-f file	True if the file exists and is not a directory
-d file	True if the exists and is a directory file
-c file	True if the file exists and is character special file
-b file	True if the file exists and is a block special file
-r file	True if the file exists and you have a read permission to it
-d file	True if the file exists and you have a write permission to it
-x file	True if the file exists and you have a execute permission to it

15) Write a program to check for ordinary file and display It contents

vi sp15

```
echo enter a file name
read fname
if test –f $fname
then
cat $fname
else
echo “given file is not ordinary file”
fi
```

:wq(save and quit)

16) write a program to check read permission

vi sp16

```
echo “enter a file name”
read fname
if [ -r $fname ]
cat $fname
else
chmod u+r $fname
cat $fname
fi
```

:wq(save and quit)

17) Write a program to append data to the file

vi sp17

```
echo  “enter a file name”
read $fname
if  [ -f $fname ]
then
      if [ -w $fname ]
      Then
```

```
                echo “enter data to file to stop press ctrl+d…”
                cat>>$fname
else
        chmod u+w $fname
        echo “enter data to file to stop press ctrl+d…”
        cat>$fname
fi
else
        echo “enter data to file to stop press ctrl+d...”
        cat>$fname
fi
```

:wq(save and quit)

String test commands

Condition	Meaning
String1 = string2	True if the strings are same
String1 != string2	True if the strings are different
-n string1	True if the length of string is greater than 0
-z string	True if the length of the string is zero

18) Write a program to compare two strings

vi sp21

```
echo “enter first string:”
read str1
echo “enter second string:”
read str2
if test $str1 = $str2
then
echo “both strings are equal”
else
echo “strings are not equal”
fi
```

:wq(save and quit)

19) Write a program check given string is empty or not
vi sp22

```
echo enter a string
read str
if [ -z $str ]
then
echo “string is empty”
else
echo “given string is not empty”
 fi
```

:wq(save and quit)

Case Control Statement

Syntax

```
Case value in
Choice1)
    ----------
    ----------
    ;;
Choice2)
    ----------
    ----------
    ;;
Choice3)
    ----------
    ----------
    ;;
:
:
Choicen)
   ----------
   ----------
     ;;
*)
   ----------
   ----------
     ;;
esac
```

Firstly, the expression following the case keyword is evaluated. The value that it yields is then matched, one by one against the potential choices(choice1, choice2 and choice3 in the above form). When a match is found, the shell executes all commands in that case up to;; . this pair of semicolons placed at the end of each choice are necessary.

20) Sample program for case

vi sp20

```
echo “enter a number between 1 to 4 “\c”
read num
case $num in
1)echo “you entered 1”
  ;;
2)echo “you entered 2”
  ;;
3) echo “you entered 3”
  ;;
4)echo “you entered 4”
  ;;
*)echo “invalid number. enter number between 1 to 4 only”
  ;;
```

:wq(save and quit)

21)write a program to check given is upper case alphabet or lower case alphabet or digit or special character

vi sp21

```
echo “enter a single character”
read ch
case $ch in
[a-z])echo “you entered a small case alphabet”
        ;;
[A-Z])echo “you entered a upper case alphabet”
        ;;
[0-9])echo “you entered a digit”
```

```
        ;;
?)echo “you entered  a special character”
        ;;
*)echo “you entered more than one character”
        ;;
esac
```

:wq(save and quit)

22) write a program to display file contents or write on to file or execute based on user choice

vi sp22

```
echo “enter a file name:\c”
read fname
echo “Main Menu”
echo “--------------“
echo “r.read mode”
echo “w.write mode”
echo “x. execute mode”
echo “enter mode:\c”
read mode
case $mode in
r)
if [ -f $fname –a –r $fname ]
then
cat $fname
fi
;;
w)
if [-f $fname –a –w $fname ]
then
echo “enter dat to file at end press ctrl+d:”
cat>$fname
fi
;;
x)
if [ -f $fname –a –x $fname ]
then
```

```
chmod u+x $fname
$fname
fi
;;
*)echo “you entered invalid mode…”
  ;;
esac
```

:wq (save and quit)

23) Write a menu driven program which has following options:
1) contents of current directory
2) list of users who have currently logged in
3) present working directory
4) calendar
5) exit

vi sp23

```
echo “Main Menu”
echo “----------“
echo “1.list of files”
echo “2.list of users”
echo “3.present working directory”
echo “4.display calendar”
echo “5.exit”

echo “enter your choice:\c”
read choice
case $choice in
1) ls -x
  ;;
2)who
  ;;
3)pwd
  ;;
4)echo “enter month and year”
 read m y
cal $m $y
  ;;
```

```
5)banner “thank you”
Sleep 1
Exit 0
;;
*)echo “choice is wrong try again”
;;
esac
```

:wq(save and quit)

The Exit Statement

To terminate execution of shell program.

Looping Control statements

A loop involves repeating some portion of the program either a specified no of the program either a specified no of times or until a particular condition is being satisfied. There are three methods by way of which we can repeat a part of a program. They are

1) Using a **while** statement
2) Using a **until** statement
3) Using a **for** statement

1) While Statement
Syntax

```
While [condition]
do
---------------
---------------
---------------
done
```

The statements within the while loop would keep on getting executed till the condition is true. When the condition is false, the control transfers to after done statement.

24) write a program display numbers 1 to 10

vi sp24

```
echo “the number form 1 to 10 are:”
i=1
while [$i –le 10]
do
echo $i
i=’expr $i + 1’
done
```

:wq(save and quit)

25)write a program copy file from root to user directory

vi sp25

```
flag=1
while [ $flag –eq 1 ]
do
echo “enter a file name:\c”
read fname
cp $fname /usr/crimson/$fname
echo “$fname copied…….
echo “do u wish to continue [1-yes/0-no]:”
read flag
done
```

:wq(save and quit)

The Break Statement

When the keyword break is encountered inside any loop, control automatically passes to the first statement after the loop.

The Continue Statement

When the keyword continue is encountered inside any loop, control Automatically passes to the beginning of the loop.

26)write a program display file contents if file existing

vi sp26

```
x=0
while test $x=0
do
echo “enter a file name:\c”
read fname
if test ! –f $fname
then
echo “$fname is not found....”
continue
else
break
fi
done
cat $fname | more
```

:wq(save and quit)

The Until Loop

Syntax

```
Until [condition ]
do
----------
----------
----------
done
```

The statements within the until loop keep on getting executed till the condition is false. When the condition is true the control transfers to after done statement.

27)write a program print numbers 1to 10

vi sp27

```
i=1 until [$i –gt 10 ]
do
echo $i
i=’expr 4i +1’
done
```

:wq(save and quit)

The true and false command
To execute the loop an infinite no of times.

28)write a sample program for true command

vi sp28

```
while true do clear
banner “hello”
sleep 1
clear
banner “crimson”
sleep 1
done
```
:wq(save and quit)

Note: the above program executes continous to stop execution press ctrl + break.

29)write a sample program for false command

vi sp29

```
Until false
do
clear
banner “hello”
Sleep 1
clear
Banner “crimson”
sleep 1
Done
```

:wq(save and quit)

The sleep command

The sleep command stops the execution of the program the specified no of seconds.

The for loop

Syntax

```
For variable in value1 value2 value3……value
do
----------
----------
----------
Done
```

The for allows us to specify a list of values which the variable in the lop can take. The loop is then executed for each value mentioned in the list.

30) write a program to demonstrate for loop

vi sp30
For I in 12345
Do echo 4i
Done

:wq(save and quit)

31) write a program to demonstrate for loop

vi sp31
For i in crimson soft is a training institute.
So echo $i
Done

:wq

32) write a program to display all files in current directory

vi sp32
for I in *
do
if test –f 4i –a –r 4i
then
cat $i | more
sleep 1
clear
fi
done

:wq(save and quit)

33) write a program to display all sub-directories in the current directory

vi sp33

```
for I in *
do
if [ -d $i ]
then
echo $i
fi
done
```

:wq(save and quit)

Positional parameters

When the arguments are passed with the command line, shell puts each word on the command line into special variables. For this, the shell uses something called as “positional parameters”. These can be thought of as variables defined by the shell. They are nine in number, named $1
Through $9.

Consider the following statement, where sp37 is any executable shell scriptfile and the remaining are the arguments.
$sp37 crimson solutions is a computer training and development instituteon entering such command, each word is automatically stored serially in the positional parameters. Thus, $0 would be assigned sp37, $1 would be assigned “crimson”, $2 “solutions”, $3 is “is” and so on, till “institute”, which is assigned to $9

34)write a program to copy a file using positional parameters

vi sp34

```
if cp $1 $2
then echo ‘file copied successfully’
else
echo “failed to copy”
fi
```

:wq(save and quit)

$chmod 7f44 sp37

$sp37 a1 a2
In above command it passes a1 to $1 and a2 into $2, then it copies a1 contents to a2
S* -contains entire string of arguments
S# -contains the no for arguments specified in the command