

UNIX SHELL SCRIPTING

A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finish executing, it displays that program's output.

Shell Types:

- 1.The Bourne Shell....default prompt \$
- 2.The C Shell. Default prompt is the % character

Bourne Shell:

- 1.Bourne Shell (.sh)
- 2.Korn Shell (.ksh)
- 3.Bourne Again Shell (.bash)
- 4.POSIX shell(sh)

C-Types

- 1.C Shell (csh)
- 2.TENEX/TOPS C Shell(tcsh)

VI EDITOR COMMANDS

I(insert)
CLTRL+C (save the file)
ESC (save the file)
: WQ! (save and exit from vi editor without saving and exit)
: q! ()
Vi filename

Script/code: list of commands/functions which are coded in order of execution.

test.sh
#!/bin/sh

UNIX VARIABLES:

Variable is a character String to which we will assign a value. Value could be Number,Text,Filename,device or any other type of data.

******A variable name could contain any alphabet (a-z, A-Z), any digits (0-9), and an underscore (_).

****** However, a variable name must start with an alphabet or underscore. It can never start with a number.

a to z or A to Z --- and 0 to 9, _

Name1="name_of_user"

UPPERCASE.

EX:

```
NAME="TEST_USER"
ALLOWED TYPE OF NAMING CONVENTIONS
NAME
NAME_1
VAR_1
NAME_A
NAME_2
_AV_3
AV232
```

NOT ALLOWED

```
2_NAME
-NAME
VAR1-VAR2
VAR_A!
```

ASSIGNING A VALUE TO A Variable

```
VAR="Variable Data"
```

read only VAR --- which marks the variable read only, so that we can not re assign a value to it.

****** If you want to delete / unsetting the variable use below command.

unset -- command tells the shell to remove variable from the list of variables that it tracks.

EX:

```
VAR_1="Un setting VARIABLE"
unset VAR
echo $VAR_1 --displays empty data.
```

UNIX -- Special Variables:

\$0 ---- The filename of the current script

\$n --- \$1,\$2 --- These variables correspond to the arguments with which the script was invoked.

\$# --- The number of arguments supplied to a script

\$* --- \$1 \$2 \$3--- "All the arguments are double quoted"

\$@ -- All the arguments are individually quoted

\$? --- The exit status of the last command executed. --IF IT IS SUCCESS STATUS VALUE IS 0 and IF IT IS FAILS STATUS VALUE IS 1

\$\$ -- The process ID of the current shell.

EX: `echo $0`
`echo $1`
`echo $2`
`echo $@`
`echo $?`
`echo $$`

UNIX --- USING ARRAYS

A shell variable is capable of holding a single value --- This type of variables are called as SCALAR VARIABLES.

EX:

```
          0  1  2  3  4
FLOWER=[ROJA,LILLY,JASMINE,TULIP,LOTUS] - SIZE - 1 = INDEX
FLOWER1=ROJA
FLOWER2=LILLY
FLOWER3=JASMINE
FLOWER4=TULIP
FLOWER5=LOTUS
```

Array variable is a special type of variable that can hold multiple values at the same time. Arrays provide a method of grouping a set of variables. instead of creating new name for each variable, you can use a single array variable that stores all the other variables.
BASED ON INDEX VALUE -- INDEX STARTS FROM 0

SYNTAX

`array_name[index]=value`

```
FLOWER[0]="ROJA"
FLOWER[1]="LILLY"
FLOWER[2]="JASMINE"
FLOWER[3]="TULIP"
FLOWER[4]="LOTUS"
```

KSH SHELL

`set -A FLOWER ROJA LILLY JASMINE TULIP LOTUS`

BASH SHELL

`FLOWER=(ROJA LILLY JASMINE TULIP LOTUS)`

DESCISION MAKING STATEMENTS

IF FI STATEMENT

If the resulting value is true, given statement(s) are executed. If the expression is false then no statement would be executed.

SYNTAX: if [expression]
 then
 Statement(s) to be executed if expression is true
 fi

EX: a=10
 b=20

```
if [ $a == $b ]  
then  
    echo "a is equal to b"  
fi
```

```
if [ $a != $b ]  
then  
    echo "a is not equal to b"  
fi
```

O/P: a is not equal to b

IF....ELSE...FI

SYNTAX: if [expression]
 then
 Statement(s) to be executed if expression is true
 else
 Statement(s) to be executed if expression is not true
 fi

EX: a=10
 b=20

```
if [ $a == $b ]  
then  
    echo "a is equal to b"  
else  
    echo "a is not equal to b"  
fi
```

O/P: a is not equal to b

IF... ELIF..FI

SYNTAX:

if [expression 1]

then

Statement(s) to be executed if expression 1 is true

elif [expression 2]

then

Statement(s) to be executed if expression 2 is true

elif [expression 3]

then

Statement(s) to be executed if expression 3 is true

else

Statement(s) to be executed if no expression is true

fi

EX:

```
a=10
b=20

if [ $a == $b ]
then
    echo "a is equal to b"
elif [ $a -gt $b ]
then
    echo "a is greater than b"
elif [ $a -lt $b ]
then
    echo "a is less than b"
else
    echo "None of the condition met"
fi
```

O/P: a is less than b

LOOPING STATEMENTS:

WHILE LOOP

SYNTAX: while command

do

Statement(s) to be executed if command is true

done

EX:

```
a=0

while [ $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
```

O/P: 0

1
2
3
4
5
6
7
8
9

FOR LOOP

SYNTAX: for var in word1 word2 ... wordN

do

Statement(s) to be executed for every word.

done

EX: `for var in 0 1 2 3 4 5 6 7 8 9`
`do`
`echo $var`
`done`

O/P: 0 1 2 3 4 5 6 7 8 9

UNIX -- BASIC OPERATORS:

1.Arithmetic operators

with the help of external programs like awk or expr, it evaluate arithmetic operation

+ -- Addition --- ``expr $a + $b``

- -- Subtraction --- ``expr $a - $b``

* -- Multiplication --- ``expr $a * $b``

/ -- Division --- ``expr $a / $b``

% -- Modulus --- ``expr $a % $b``

= -- Assignment -- `a=$b`

== -- Equality -- `[$a == $b]`

!= -- No Equality -- `[$a != $b]`

Ex: `a=10`

`b=20`

`val=`expr $a + $b``
`echo "a + b : $val"`

`val=`expr $a - $b``
`echo "a - b : $val"`

```
val=`expr $a \* $b`  
echo "a * b : $val"
```

```
val=`expr $b / $a`  
echo "b / a : $val"
```

```
val=`expr $b % $a`  
echo "b % a : $val"
```

O/P: a + b : 30

a - b : -10

a * b : 200

b / a : 2

b % a : 0

Different ways to compute arithmetic operations in a bash

1.expr command with backticks

Arithmetic expansion could be done using backticks and expr.

SYNTAX: `expr item1 operator item2`

Ex:

```
a=10  
b=3
```

```
# there must be spaces before/after the operator
```

```
sum=`expr $a + $b`  
echo $sum
```

```
sub=`expr $a - $b`  
echo $sub
```

```
mul=`expr $a \* $b`  
echo $mul
```

```
div=`expr $a / $b`  
echo $div
```

o/p: 13

7

30

3

2.Double Parenthesis

SYNTAX: \$((expression))

This could be used for arithmetic expansion. Let's see an example to see the use of double parenthesis.

Ex: a=10

b=3

```
echo $((a+b))
```

```
echo $((a+b)) #this is also valid
```

```
echo $((a-b))
```

```
echo $((a-b)) #this is also valid
```

O/P:13

13

7

7

3. Using let command

Let command is also used to perform arithmetic operations

Ex: x=10

y=3

```
let "z = $(( x * y ))" # multiplication
```

```
echo $z
```

```
let z=$((x*y)) #this is also valid
```

```
echo $z
```

```
let "z = $(( x / y ))" # division
```

```
echo $z
```

```
let z=$((x/y)) #this is also valid
```

```
echo $z
```

o/p: 30

30

3

3

2. RELATIONAL OPERATORS

10,20,"10","20"...relational operators work on this

but not on "TEN" and "TWENTY"

-eq -- checks if the value of two operators are equal or not, if yes then condition becomes TRUE. -- [\$a -eq \$b]

-ne -- checks if the value of two operators are equal or not, if values are not equal then condition becomes TRUE -- [\$a -ne \$b]

-gt -- Checks if the value of left operand is greater than the value of right operand, if yes the condition becomes TRUE. -- [\$a -gt \$b]

-lt -- Checks if the value of left operand if less than the value of right operand, if yes then the condition becomes TRUE. -- [\$a -lt \$b]

-ge -- Checks if the value of left operand if greater than or equal to the value of right operand. if yes then the condition becomes TRUE. -- [\$a -ge \$b]

-le -- Checks if the value of left operand if less than or equal to the value of right operand. If yes then the condition becomes TRUE. -- [\$a -le \$b]

Ex:

```
a=10
b=20
```

```
if [ $a -eq $b ]
then
    echo "$a -eq $b : a is equal to b"
else
    echo "$a -eq $b: a is not equal to b"
fi
```

```
if [ $a -ne $b ]
then
    echo "$a -ne $b: a is not equal to b"
else
    echo "$a -ne $b : a is equal to b"
fi
```

```
if [ $a -gt $b ]
then
    echo "$a -gt $b: a is greater than b"
else
    echo "$a -gt $b: a is not greater than b"
fi
```

```
if [ $a -lt $b ]
then
    echo "$a -lt $b: a is less than b"
else
    echo "$a -lt $b: a is not less than b"
fi
```

```
if [ $a -ge $b ]
then
    echo "$a -ge $b: a is greater or equal to b"
else
    echo "$a -ge $b: a is not greater or equal to b"
fi
```

```
if [ $a -le $b ]
then
```

```
    echo "$a -le $b: a is less or equal to b"
else
    echo "$a -le $b: a is not less or equal to b"
fi
```

OUTPUT:

10 -eq 20: a is not equal to b
10 -ne 20: a is not equal to b
10 -gt 20: a is not greater than b
10 -lt 20: a is less than b
10 -ge 20: a is not greater or equal to b
10 -le 20: a is less or equal to b

EX 2:

```
a=30
b=40
```

```
if(( $a -eq $b ))
then
    echo a is equal to b.
else
    echo a is not equal to b.
fi
```

```
if(( $a! -ne $b ))
then
    echo a is not equal to b.
else
    echo a is equal to b.
fi
```

```
if(( $a -lt $b ))
then
    echo a is less than b.
else
    echo a is not less than b.
fi
```

```
if(( $a -le $b ))
then
    echo a is less than or equal to b.
else
    echo a is not less than or equal to b.
fi
```

```
if(( $a -gt $b ))
then
    echo a is greater than b.
else
    echo a is not greater than b.
fi
```

```
if(( $a -ge $b ))
then
    echo a is greater than or equal to b.
else
    echo a is not greater than or equal to b.
```

```
fi
```

3.BOOLEAN OPERATORS

FALSE

TRUE

! -- This is logical negation. This inverts the true condition in to false and vice versa

-o -- This is logical OR. If one of the operands is true then the condition would be TRUE.

-a -- This is logical AND . If both the operands are true then the condition would be TRUE otherwise it would be false.

Ex:

```
#!/bin/sh
```

```
a=10
```

```
b=20
```

```
if [ $a != $b ]
then
    echo "$a != $b : a is not equal to b"
else
    echo "$a != $b: a is equal to b"
fi
```

```
if [ $a -lt 100 -a $b -gt 15 ]
then
    echo "$a -lt 100 -a $b -gt 15 : returns true"
else
    echo "$a -lt 100 -a $b -gt 15 : returns false"
fi
```

```
if [ $a -lt 100 -o $b -gt 100 ]
then
    echo "$a -lt 100 -o $b -gt 100 : returns true"
else
    echo "$a -lt 100 -o $b -gt 100 : returns false"
fi
```

```
if [ $a -lt 5 -o $b -gt 100 ]
then
    echo "$a -lt 100 -o $b -gt 100 : returns true"
else
    echo "$a -lt 100 -o $b -gt 100 : returns false"
fi
```

OUTPUT:

10 != 20 : a is not equal to b

10 -lt 100 -a 20 -gt 15 : returns true

10 -lt 100 -o 20 -gt 100 : returns true

10 -lt 5 -o 20 -gt 100 : returns false

4. STRING OPERATORS

String -- collection of characters

1.EQUAL OPERATOR (=)

= -- checks if the value of two operands are equal or not, if yes then the condition becomes TRUE -- [\$A = \$B]

```
Ex: str1="GeeksforGeeks";
str2="geeks";
if [ $str1 = $str2 ]
then
    echo "Both string are same";
else
    echo "Both string are not same";
fi
```

o/p: Both string are not same

2.NOT EQUAL OPERATOR (!=)

!= -- checks if the value of two operands are equal or not,if values are not equal then the condition becomes TRUE -- [\$A != \$B]

```
EX: str1="GeeksforGeeks";
str2="geeks";
if [ $str1 != $str2 ]
then
    echo "Both string are not same";
else
    echo "Both string are same";
fi
```

OUTPUT: both string is not same

CHECK STRING LENGTH EQUAL TO 0:

This operator is used to check if the string is empty.

-z --Checks if the given string operand size is zero, if it is zero length the it return TRUE. -- [-z \$A]

Ex:

```
str="Nishanth"
if [ -z $str ]
then
    echo "String is empty";
else
    echo "String is not empty";
fi
```

o/p: string is not empty

Ex:

```
str=""
if [ -z $str ]
then
    echo "String is empty";
else
    echo "String is not empty";
fi
```

O/P: String is empty

CHECK STRING LENGTH GREATER THAN 0:

This operator is used to check the string is not empty.

-n -- Checks if the given string operand size is non zero. If it is non zero length then it returns TRUE -- [-n \$A]

EX:

```
#!/bin/sh

str="GeeksforGeeks";
if [ -n $str ]
then
    echo "String is not empty";
else
    echo "String is empty";
fi
```

O/P: String is not empty

STR

Str -- Check if str is not the empty String. if it is empty then it returns FALSE. -- [\$A]

Ex:

```
a="Nishanth"
if [ $a ]
then
    echo "$a : string is not empty"
else
    echo "$a : string is empty"
fi
```

Ex:2

```
a="abc"
b="efg"
```

```

if [ $a = $b ]
then
    echo "$a = $b : a is equal to b"
else
    echo "$a = $b: a is not equal to b"
fi

if [ $a != $b ]
then
    echo "$a != $b : a is not equal to b"
else
    echo "$a != $b: a is equal to b"
fi

if [ -z $a ]
then
    echo "-z $a : string length is zero"
else
    echo "-z $a : string length is not zero"
fi

if [ -n $a ]
then
    echo "-n $a : string length is not zero"
else
    echo "-n $a : string length is zero"
fi

if [ $a ]
then
    echo "$a : string is not empty"
else
    echo "$a : string is empty"
fi

```

OUTPUT:

```

abc = efg: a is not equal to b
abc != efg : a is not equal to b
-z abc : string length is not zero
-n abc : string length is not zero
abc : string is not empty

```

5. FILE TEST OPERATORS

- d file -- Check if the file is a directory, if yes then the condition becomes TRUE. -- [-d \$file]
- r file -- Check if the file is readable, if yes then condition becomes TRUE -- [-r \$file]
- w file -- Check if the file is Writable, if yes then condition becomes TRUE -- [-w \$file]
- x file -- Check if the file is Executable, if yes then condition becomes TRUE -- [-x \$file]
- s file -- Check if the file has greater than 0 size if yes then the condition becomes TRUE -- [-s \$file]
- e file -- Check if the file exists. Is TRUE even if file is a directory but exists. -- [-e \$file]
- f file -- Check if the file is an ordinary file , if yes then the condition becomes TRUE -- [-f \$file]

/home/anushabutharaju157222/class_work/string_operators.sh

EX:

```
#!/bin/sh

file="/home/anushabutharaju157222/class_work/string_operators.sh"

if [ -r $file ]
then
    echo "File has read access"
else
    echo "File does not have read access"
fi

if [ -w $file ]
then
    echo "File has write permission"
else
    echo "File does not have write permission"
fi

if [ -x $file ]
then
    echo "File has execute permission"
else
    echo "File does not have execute permission"
fi

if [ -f $file ]
then
    echo "File is an ordinary file"
else
    echo "This is sepcial file"
fi

if [ -d $file ]
then
    echo "File is a directory"
else
    echo "This is not a directory"
fi

if [ -s $file ]
then
    echo "File size is not zero"
else
    echo "File size is zero"
fi

if [ -e $file ]
then
    echo "File exists"
else
    echo "File does not exist"
fi
```

