All shell and Unix commands takes input from the standard input (STDIN), write oupt to the terminal as standard output (STDOUT) and write error output to the standard error (STDERR).

In this, STDIN is connected to the keyboard, mouse.

STDOUT and STDERR is connected to the terminal.

Basically we can do the following task Shu:

1) redirect stdout to a file

2) redirect stderr to a file

3) redirect stdout to a stderr

4) redirect stderr to a stdout

5) redirect stderr and stdout to a file

6) redirect stderr and stdout to stdout

7) redirect stderr and stdout to stderr

So, in Linux uses file descriptors number for the STDIN, STDOUT and STDERR as 0,1,2.

Some of the examples below redirection rules:

---------------------------------------------

1) > --> Redirect standard output.

2) 2> --> Redirect standard error.

3) 2>&1 --> Redirect standard error to standard output.

4) < --> Redirect standard input.

5) | --> Pipe, which pass the output of previous command to next command.

6) >> --> Append to standard output.

7) 2>&1| --> Pipe standard error and standard output to next command.

How to set proper binary path in the script file?

==

root@linuxtestserver:~/bashscript# ./test

-bash: ./test: /usr/bin/bash: bad interpreter: No such file or directory

root@linuxtestserver:~/bashscript# cat test

#!/usr/bin/bash --> Replace the value with the path of the bach binary file (/bin/bash)

echo $SHELL

echo "This is test script"

==

Variables:

----------

A variable in bash can contain a number, a character, a string of characters.

1) Global variables or environment variables are available in all shells. The environment or print environment commands can be used to display environment variables.

Example: use environment or printenv command to view the predefined global variables or system variables which will be represented using capital letter.

echo $HOME, echo $SHELL

2) User defined variables, which are created by the user to store the value.

Example:

name = visual path

echo $name

Conditional Execution:-

----------------------

1) There are two operations && (AND operation) and || (OR Operation)

Example:

$ rm myf && echo "File is removed successfully" || echo "File is not removed"

If file (myf) is removed successful (exist status is zero) then "echo File is removed successfully" statement is executed, otherwise "echo File is not removed" statement is executed (since exist status is non-zero)

Conditions let you to decide whether to perform an action or not.

IF statement:

-------------

It has many forms, if expression then statement. Here statement will be executed only when the expression is true.

Even it has another form, if expression then statement1 else statement2.

Syntax:

-------

if [expression]; then

statement

fi

-------

if [expression]; then

statement1

else

statement2

fi

--------

Case statement:

---------------

The case statement can execute commands based on pattern matching decision.

Syntax:

case <COND> in

<PATTERN1>)

statement

;;

<PATTERN2>)

statement

;;

esac

Exit status:

++++++++++++

Any program executed in linux/unix system will return a status back to the system, this status is a number that usually indicates whether the program successfully ran which states as exit status.

If the program/condition is successfull then the exit status will be ZERO(0) else it will be ono-zero(1-255).

File types and attributes:

--------------------------

- --> regular file

l --> symbolic link

c --> Character special

b --> Block special

p --> Named pipe

s --> socket file

d --> Directory file

Options in echo command:

------------------------

-n --> No new line (A line feed is automatically added after the string is displaed, it can be supressed)

-e --> enable escape sequence

Input/Output (Redirectors):

---------------------------

> --> Redirect standard output

2> --> Redirect standard error

2>&1 --> Redirect standard error to standard output

< --> Redirect standard input

| --> Pipe standard

>> --> Append

2>&1| --> Pipe standard output and standard error to standard output

Exit status:

------------

0 --> for success

1 --> general input error

130 --> script termination error

command substitution:

---------------------

VAR=`command`

VAR=$(command)

Arithematic substitution:

-------------------------

syntax:

VAR=$((expression))

ARRAY Variables:

----------------

VAR[x]=value

where:

VAR is variable name

x is the index value

#MYARRAY=("first" "second" "third")

echo $MYARRAY

Example:

#!/bin/bash

SERVERLIST=("webserver01" "webserver02" "webserver03")

COUNT=0

for INDEX in ${SEVERLIST[\*]}; do

echo "Processing Server: ${SERVERLIST[COUNT]}"

COUNT=`expr $COUNT + 1`

done

unset variables:

----------------

unset varaiablename

Readonly variable:

------------------

Read command:

--------------

syntax: read -p "Enter your name: " name

read -p -s "Enter Password: " password

or

echo -n "Enter your name: " name

read name

Regular Expressions:

--------------------

. (dot) - a single character.

? - the preceding character matches 0 or 1 times only.

\* - the preceding character matches 0 or more times.

+ - the preceding character matches 1 or more times.

{n} - the preceding character matches exactly n times.

{n,m} - the preceding character matches at least n times and not more than m times.

[agd] - the character is one of those included within the square brackets.

[^agd] - the character is not one of those included within the square brackets.

[c-f] - the dash within the square brackets operates as a range. In this case it means either the letters c, d, e or f.

() - allows us to group several characters to behave as one.

| (pipe symbol) - the logical OR operation.

^ - matches the beginning of the line.

$ - matches the end of the line.

Task:

-----

1) Find the total number of empty lines in the files. (use any command and regular expressions)

2)

Conditions:

-----------

Two flow control mechines are available in the shell

1) case statement and 2) if statement

case statement:

---------------

syntax:

case <WORD> in

<PATTERN1>)

LIST 1

;;

<PATTERN2>)

LIST 2

;;

esac

;; can be ignored in the last pattern before esac which is not madetory and If you sepcify \* in pattern then if the word is not matched with any of the pattern then the LIST will be executed.

Example:

--------

#!/bin/bash

read -p 'Enter the OS name: ' name

case $name in

linux)

cat /etc/issue

;;

unix)

uname -a ;;

esac

Note: if you want to pass multiple patterns case statement as mentioned below, then each of the pattern should be delimited with the pipe symbol.

syntax:

case $name in

linux|ubuntu|debian)

LIST1 ;;

other)

LIST2 ;;

esac

String comparisons:

-------------------

1) checking whether a string is empty or not

2) checking whether a string two string are equal or not.

-z string --> True, if the string has zero length

-n string or ! -z string --> True, if the string has non zero length.

string1 = string2 --> True, if the strings are equal.

string1 != string2 --> True, if the strings are not equal.

Bash File Testing:

------------------

-b --> Block special file

-c --> Special character file

-d --> Check for directory existence

-e --> Check for file existence

-f --> Check for regular file existence not a directory

-G --> Check if file exists and is owned by effective group ID.

-g --> true if file exists and is set-group-id.

-k --> Sticky bit

-L --> Symbolic link

-O --> True if file exists and is owned by the effective user id.

-r --> Check if file is a readable

-S --> Check if file is socket

-s --> Check if file is nonzero size

-u --> Check if file set-ser-id bit is set

-w --> Check if file is writable

-x --> Check if file is executable

Numerix comparision:

--------------------

1) int1 -eq init2 --> True, if int1 is equal to int2

2) int1 -ne init2 --> True, if int1 is not qual to int2

3) int1 -lt int2 --> True, if int1 is less than int2

4) int1 -le int2 --> True, if int1 is less than or qual to int2

5) int1 -gt int2 --> True, if int1 is greater than int2

6) int1 -ge int2 --> True, if int1 is greater than equal to int2

IF conditions:

--------------

if .. then... fi statement --> simple if

if .. then .. else .. fi statement --> if else

if .. elif .. else .. fi statement --> else if ladder

if .. then .. else .. if .. then .. fi .. fi statement --> Nested if

simple if:

----------

1)

if [ condition expression ]; then

statement

...

fi

If else:

--------

if [ condition expression ]; then

statement1

..

else

statement2

fi

Else if ladder:

---------------

if [ condition expression ]; then

statement1

statement2

elif [ condition expression ]; then

statement3

statement4

else

statement5

fi

Next class:

----------

test command which is used to check the file type.

Compund expressions:

--------------------

test options string or test [ option string ]

!expr --> True, if expr is false.

expr1 -a expr2 or expr1 && expr2 --> True, if both the expressions are true.

expr1 -o expr2 or expr1 || expr2 --> True, if either of expression is true.

Example:

-------

#!/bin/bash

read -p 'Enter your name: ' name

read -p 'Enter your course: ' course

if [ -z "$name" -o -z "$course" ]; then

echo "Invalid data!!. Please enter again"

exit

fi

echo "Welcome to Visual path Technologies!!!!"

Executional Operators:

----------------------

&& --> AND condition, which means that both the conditions need to be satisfied.

|| --> OR condition, which means that any one of the statement need to be true

Example:

#!/bin/bash

echo "Enter value between 1 and 5: "

read VALUE

if [ "$VALUE" -eq 1 ] || [ "$VALUE" -eq "3" ] || [ "$VALUE" -eq "5" ]; then

echo "You entered a ODD value of $VALUE"

else

echo "You entered a value of $VALUE"

fi

LOOPS:

------

Looping is repeatedly executing a section of your commands based on the condition.

1) While loop

2) For loop

While syntax:

-------------

while [ condition ]

do

statement1

statement2

..

..

done

Example:

--------

#!/bin/bash

x=1

while [ $x -le 5 ]

do

echo "Welcome $x times"

x=$(( $x + 1 ))

done

untile loop is opposite of the while loop.

#!/bin/bash

x=1

untile [ $x -gt 5 ]

do

echo "Sucessful"

x=$(( $x + 1 ))

done

For loop:

---------

syntax:

for varname in list

do

commands

done

example:

--------

#!/bin/bash

for i in 1 2 3 4 5

do

echo $i

done

It can also be writen in format which is similar to C language.

for (( i=0; i <=5; i++ ))

do

echo $i

done

select loop:

------------

syntax:

-------

select varname in list

do

commands

done

Example:

--------

#!/bin/bash

select var in linux unix

do

echo "You selected: $var"

done

loop control:

-------------

The break and continue are loop control commands in shell to have better control of loops in some cases of infinite loop or even in regular loops.

break --> will terminates the loop.

continue --> will jump to the next iteration of the loop.

Functions:

----------

example:

--------

#!/bin/bash

myfunc() {

echo "This is function"

}

myfunc

retrun status in function:

--------------------------

#!/bin/bash

myfunc() {

echo "Executing command"

return $?

}

if [ $? -eq 0]; then

echo "Command in function executed successfully"

else

echo "command not executed with error"

fi

export function

readonly function

basename <file>

parcing values in functions:

myfun () {

echo '$1 is a function:' $1

echo '$2 is a function:' $2

echo '$\* is a function:' $8

echo '$# is a function:' $#

}

myfunc 20 30

Global and Local variables:

---------------------------

Global varaibles values can be used at any point of time in the script

Local variables will be used in the function and this variable cant be used out side the function.

GETOPTS:

--------

getopts always includes with while loop in order to read all the options in the loop manner and also most of the times getopts has to include in order to read the arguments and the values parsed to arguments and to store in a contant variable.

OPTARG: which stores the value of the option argument found by getopts.

OPTIND: which contains the index of the next argument to be processed.

#!/bin/bash

while getopts a:b: var

do

case $var in

a) a=$OPTARG;;

b) b=$OPTARG;;

\?) echo "Invalid option: "; exit ;;

esac

done

echo -e "a=$a\tb=$b"