LINUX
SHELL SCRIPTING

------SAGAR MUNGSE------

Shell Scripting

- Text files that contain sequences of UNIX commands, created by a text editor
- No compiler required to run a shell script, because the UNIX shell acts as an interpreter when reading script files
- After you create a shell script, you simply tell the OS that the file is a program that can be executed, by using the chmod command to change the mode to be executable

A few global (env) variables

SHELL	Current shell
DISPLAY	Used by X-Windows system to identify the display
HOME	Fully qualified name of your login directory
PATH	Search path for commands
MANPATH	Search path for <man> pages
PS1 & PS2	Primary and Secondary prompt strings
USER	Your login name
TERM	terminal type
PWD	Current working directory

Positional Parameters

A shell script is invoked with a set of command line parameters each of these parameters are copied into

- $0 This variable that contains the name of the script
- $1, $2, ….. $n 1st, 2nd 3rd command line parameter
- $# Number of command line parameters
- $$ process ID of the shell
- $@ same as $* but as a list one at a time
- $? Return code ‘exit code’ of the last command

Example:

```
        sh ./ positinalparam_example.sh one two
```

Positional Parameters Example

```
$ sh ./positinalparam_example.sh
```

Content of positionalparam_example.sh

```
#!/bin/bash
echo "File Name: $0"
echo "First Parameter : $1"
echo "First Parameter : $2"
echo "Quoted Values: $@"
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#"
echo "Process Number : $$"
echo "Exit Status : $?"
```

read command

- The read command allows you to prompt for input and store it in a variable.
- Example (read.sh)

  -
    ```
    #!/bin/bash
    echo -n “Enter name of file to delete: ”
    read file
    echo “Type 'y' to remove it, 'n' to change your mind ... ”
    rm -i $file
    echo "That was YOUR decision!"
    ```
- Line 3 creates a variable called file and assigns the input from keyboard to it. Then the value of this variable is retrieved by putting the '$' in at its beginning.

crontab

- crontab can schedule to run a command or a script once or periodically like minutely, hourly, daily, weekly, monthly, yearly.

```
cronatb -l          lists the jobs of the user
crontab -e          allows to edit the jobs
```

Format

```
*    *    *    *    *
|    |    |    |    |
|    |    |    |    +--- day of week (0 - 6) (Sunday=0)
|    |    |    +-------- month (1 - 12)
|    |    +------------- day of month (1 - 31)
|    +------------------ hour (0 - 23)
+----------------------- min (0 - 59)
```

Crontab examples

```
# every 0th min of 0th hour (12am) script will
run
0 0 * * * /bin/sh /home/santoshk/bd/sc
# every min
* * * * * /bin/sh /home/santoshk/bd/sc

# once in every 30 minutes the script will run
*/30 * * * * /bin/sh  home/santoshk/ping.sh
>/dev/null

# every wednesday at 2.30 a.m. the script will
run
30 2 * * 3 /bin/sh  home/santoshk/ping.sh
>/dev/null
```

Arithmetic Comparison

[n1 -eq n2]	(true if n1 same as n2, else false)
[n1 -ge n2]	(true if n1 >= n2, else false)
[n1 -le n2]	(true if n1 <= equal to n2, else false)
[n1 -ne n2]	(true if n1 is not same as n2, else false)
[n1 -gt n2]	(true if n1 > n2, else false)
[n1 -lt n2]	(true if n1 < n2, else false)

String Comparison

- “$string1” = “$string2”   True if equal
- “$string1” == “$string2”   True if equal
- “$string1” != “$string2”   True if *not* equal
- -n “$string”   True if length of string is greater then 0
- -z “$string”   True if length string is zero

Examples

[$1 = $2]   (true if s1 same as s2, else false)

[$1 != $2]   (true if s1 not same as s2, else false)

[$1]   (true if s1 is not empty, else false)

[-n $1]   (true if s1 has a length greater then 0, else false)

[-z $2]   (true if s2 has a length of 0, otherwise false)

File Conditions

-d file	True if file a directory
-f file	True if the file exits and is not directory
-s file	True if the file exist and greater than 0
-e file	True if the file exist
-c file	True if the file is character special file
-b file	True if the file is block special file
-r file	True if file exists and you have read permissions
-w file	True if file exists and you have write permissions
-x file	True if file exists and you have excute permissions
-k file	True if file exists and its sticky bit set

Logical Conditions

!	negate (NOT) a logical expression
-a	logically AND two logical expressions
&&	logically AND two logical expressions
-o	logically OR two logical expressions
||	logically OR two logical expressions

/,*,%	-first priority
+,-	-second priority

In Logical

!	not
-lt,-gt,-le,-ge,-eq,-ne	relational
-a	and
-o	or

Conditional Statements (if)

```
if command executes successfully
then
        execute command
elif this command executes successfully
then
        execute this command
        and execute this command
else
        execute default command
fi
```

However- elif and/or else clause can be omitted.

#You can use below statement in nested conditions.

break: The break statement is used to jump out of loop.

continue: Using continue we can go to the next iteration in loop.

exit: it is used to exit the execution of program.(exit is function not a statement)

Example

```
#! /bin/sh
#  number is +ve, zero or -ve
echo –e "enter a number:\c"
read number

if [ “$number” -lt 0 ]
then
                    echo “Input is negative"
elif [ “$number” -eq 0 ]
then
                    echo “ Input is zero”
else
          echo “Input is positive”
fi
```

Loops

For Loop example:

# To check only file name from directory

```
for  i in  `ls -1`
do
    echo $i
 done
```

While Loop Example:

#To get the value of first field from file inputfile.csv

```
while read line
          do
            ID=echo $line | cut -f 1 -d “,”
             echo $ID
          done < inputfile.csv
```

Switch Case

simplifies matching when you have a list of choices

```
echo -n "Enter the name of vehicle for rent. e.g. car, van, jeep:"
read rental
case $rental in
        "car") echo "For $rental Rs.20 per k/m";;
        "van") echo "For $rental Rs.10 per k/m";;
        "jeep") echo "For $rental Rs.5 per k/m";;
        "bicycle") echo "For $rental 20 paisa per k/m";;
        *) echo "Sorry, I can not get a $rental for you";;
esac
```

Function example

Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual task when it is needed.

```
$ sh ./function.sh
```

Contents of function.sh

```
SayHello()
{   echo “Hello $LOGNAME, Have nice computing”
    }
SayHello
```

Output:

```
Hello opr, Have nice computing
```

Debugging shell scripts

- **There may be times where a shell script does something unexpected (due to user error).**
- **It may be helpful to see exactly what commands the shell is currently executing.**
- **This can be done in several ways**
  - **Call your script with /bin/bash –x myscript.sh**
  - **Insert the line set –vx near the top of the script**

This is useful to monitor your script line by line.

References

- Unix shell programming -by Yashwant Kanetkar
- Unix Concepts and Applications –by Sumitabha Das
- http://www.grymoire.com/Unix/Sed.html
- http://www.grymoire.com/Unix/Awk.html
- http://www.grymoire.com/Unix/Quote.html
- http://www.grymoire.com/Unix/Find.html

THANK YOU