Automation with Shell Scripting

Shell Scripting is a great way to Automate repetitive tasks in our Linux/Unix Environment.

1) **What is a Shell Script ?**

   Shell Script is a sequence of system commands pasted in a text file.

   Example:
   Send some XYZ Report to compliance team @ Every Friday

   Report: Versions Info of Docker and Nginx and OS Uname

2) **We can enhance the shell scripts by using the below concepts:**
   Variables
   Filters like grep, cut, awk and sed Commands
   Conditional Statements
   Loops
   Functions
   Job Scheduling and Many more

Demo:-

Example:

```
#!/bin/bash
DockerVersion=$(docker -v | cut -d " " -f 3 | tr "," " ")
GitVersion=$(git -v | cut -d " " -f 3 )
OsVersion=$(uname)

cat <<EOF mail -s "$(date): report on $(hostname)" "manojkrishnappa7@gmail.com"
Docker version: $DockerVersion
Git Version: $GitVersion
Os Uname: $OsVersion
EOF
```

After that assign to crontab

Crontab -e

```
* * * * * <script path>
```

1. SIMPLE USAGE OF ECHO COMMAND

- echo command is used to display a string/message or variable value or command result.
- Simple echo command syntax to display a message:

```
echo string/message
echo 'string/message'
echo "string/messgae"
```

2. VARIABLES

Variables are used to store data in shell scripts and later we can use them when required

Simple variable x=4
Default value of a variable is Empty/Nothing

In Linux shell scripting there are two types of variables:

System variables :
Created and maintained by Operating system itself
This type of variables are defined in CAPITAL letters
We can see them by using the set command.

User defined Variables:
Created and maintained by the user
Generally defined in lower case letters but we can use combination of upper and lower case as well

Rules to define user defined variables
Variable Name should contain only a-z, A-Z, 0-9 and _ characters.
Variable names are case sensitive. Means x and X are different.
Don't provide spaces on either side of equal symbol while defining variables.
No need to declare variable type, Linux will automatically take care while executing commands or scripts.
Use double quotes for the data if data consist of spaces.

```
[ec2-user@shell-script practice]$ x=4

[ec2-user@shell-script practice]$ echo '$x' -- will treat the content inside single quotes as string
$x
```

```
[ec2-user@shell-script practice]$ echo $x
4

[ec2-user@shell-script practice]$ echo "$x"
4
```

We can store the output of a command into a variable as follows.

```
anyVariable=$(command)
anyVariable=`command`
```

We can assign one variable value/data into another using:

```
Name="Shell Scripting"
NewName=$Name
NewName=${Name}
```

3. ADVANCED USAGE OF ECHO COMMAND :-

----

echo command is used to display string/message or variable value or command result.

echo -e "Message or variable"

Escape characters :-

```
\n New line
\t Horizontal tab
\v vertical tab
\b backspace -- cursor will go one step back
\r carriage return -- cursor will come to starting position
\ Escape character
```

We can also display the message in colors.

```
[ec2-user@shell-script ~]$ echo "$(whoami)"
ec2-user

[ec2-user@shell-script ~]$ echo "The currently logged in user is : $(whoami)"
The currently logged in user is : ec2-user

[ec2-user@shell-script ~]$ echo "This is first line\nThis is second line"
This is first line\nThis is second line

[ec2-user@shell-script ~]$ echo -e "This is first line\nThis is second line"
This is first line
This is second line
```

[ec2-user@shell-script ~]$ echo -e "\033[0;31mThis is some default color" -- This message will be displayed in red color but the terminal also will become red color

```
This is some default color
```

```
[ec2-user@shell-script ~]$ echo -e "\033[0;31mThis is some default color\033[0m" --
We can turn off the color like this
This is some default color

[ec2-user@shell-script ~]$ cat advanced_echo.sh
#!/bin/bash

echo -n "This i the first line"
echo "This is the second line"
[ec2-user@shell-script ~]$ ./advanced_echo.sh
This i the first lineThis is the second line
```

echo -n won't make the cursor to go the next line

4. Multi line block :-

```
[ec2-user@shell-script ~]$ cat usageOfEchoCommand.sh
#!/bin/bash
cat << EOF
The user is $USER
The home directory of the user $UER id $HOME
EOF

[ec2-user@shell-script ~]$ ./usageOfEchoCommand.sh
The user is ec2-user
The home directory of the user  id /home/ec2-user
```

EOF can be other string as well.

```
[ec2-user@shell-script ~]$ cat usageOfEchoCommand.sh
#!/bin/bash
cat << Manoj
The user is $USER
The home directory of the user $UER id $HOME
Manoj
[ec2-user@shell-script ~]$ ./usageOfEchoCommand.sh
The user is ec2-user
The home directory of the user  id /home/ec2-user
```

We can redirect the output to a file as well like this.

```
[ec2-user@shell-script ~]$ cat << EOF > demo.txt
> This is first line
> This is second line
> EOF
```

```
[ec2-user@shell-script ~]$ cat demo.txt
This is first line
This is second line

[ec2-user@shell-script ~]$ echo "This is first line" | tr [a-z] [A-Z]
THIS IS FIRST LINE

[ec2-user@shell-script ~]$ tr [a-z] [A-Z] <<< "This is first line" -- Instead of echo we
can pass strings like this
THIS IS FIRST LINE

[ec2-user@shell-script ~]$ name="BASH SHELL SCRIPTING"
[ec2-user@shell-script ~]$ tr [A-Z] [a-z] <<< $name -- we can variables like this
bash shell scripting

[ec2-user@shell-script ~]$ tr [a-z] [A-Z] <<< $(whoami) -- we can pass commands
output as well like this
ec2-user
```

5. COMMENTS FOR BASH SHELL SCRIPTING :-

A comment is a human readable explanation that is written in shell script.
Adding comment to your bash scripts will save you a lot of time and effort when you look at your code in the future.
Comments are used to explain the code
The comments also help other developers and system administrators who may need to maintain the script to understand your code and its purpose

We have single line comments and multi line comments.
Note : Comments won't run while running or executing the script.

```
[ec2-user@shell-script ~]$ cat inventory_comments_usage.sh
#!/bin/bash

#Author : Manoj

<< Manoj
This is a multi line comment.
We can declare it like this
Manoj

: '
This is also a multi line comment
We can declare it like this as well
'

echo "This is an inventory script"

[ec2-user@shell-script ~]$ ./inventory_comments_usage.sh
This is an inventory script
```

The above script will help us in understanding how to define comments in shell scripting

6. MAKE BASH SHELL SCRIPTS AS PORTABLE WITH UNIX/LINUX SYSTEMS:-
----

```
[ec2-user@shell-script ~]$ cat demo.sh
#!/usr/bin/env bash
echo "This is about portability"

[ec2-user@shell-script ~]$ ./demo.sh
This is about portability
```

Instead of using shebang line #!/bin/bash we can use #/usr/bin/env bash which will make our script run on all the different flavors of linux OS because we can't guarantee the path of bash shell in all the linux OS to be same.

Debugging bash shell scripts :-

Debugging is determining the cause which fails the script.
Syntax Error and Runtime error
Syntax error stop script execution and runtime error don't stop script.

Actually we don't have good debugging procedures with shell scripting, but we can try with some commands.
There are different debugging commands and we will work with set command.
Our bash is an interpreter meaning it will execute the script the line by line.

We can use set -n option on the beginning of the script to look for syntax error without executing the script.

```
[ec2-user@shell-script ~]$ cat demo.sh
#!/usr/bin/env bash
set -n
pwd
user=$(whoami
echo "This is next line"

[ec2-user@shell-script ~]$ ./demo.sh
./demo.sh: line 4: unexpected EOF while looking for matching `)'
./demo.sh: line 6: syntax error: unexpected end of file
```

without executing the script it will let us know the syntax errors.

set -x prints the command before execution

```
[ec2-user@shell-script ~]$ cat demo.sh
#!/usr/bin/env bash
set -x
pwd
user=$(whoami
echo "This is next line"

[ec2-user@shell-script ~]$ ./demo.sh
+ pwd
/home/ec2-user
./demo.sh: line 4: unexpected EOF while looking for matching `)'
./demo.sh: line 6: syntax error: unexpected end of file
```

set -e exit the script if any command fails

```
[ec2-user@shell-script ~]$ cat demo.sh
#!/usr/bin/env bash
set -e
pwd
asjahdh
user=$(whoami
echo "This is next line"

[ec2-user@shell-script ~]$ ./demo.sh
/home/ec2-user
./demo.sh: line 4: asjahdh: command not found
[ec2-user@shell-script ~]$
```

set -v more verbose way of running the script.

```
[ec2-user@shell-script ~]$ cat demo.sh
#!/usr/bin/env bash
set -v
pwd
asjahdh
user=$(whoami
echo "This is next line"

[ec2-user@shell-script ~]$ ./demo.sh
pwd
/home/ec2-user
asjahdh
./demo.sh: line 4: asjahdh: command not found
user=$(whoami
echo "This is next line"
./demo.sh: line 5: unexpected EOF while looking for matching `)'
./demo.sh: line 7: syntax error: unexpected end of file
```

Instead of writing set command in the script we can execute the script a bash -option.

```
[ec2-user@shell-script ~]$ cat demo.sh
#!/usr/bin/env bash
pwd
asjahdh
user=$(whoami
echo "This is next line"
[ec2-user@shell-script ~]$ bash -x demo.sh
+ pwd
/home/ec2-user
+ asjahdh
demo.sh: line 3: asjahdh: command not found
demo.sh: line 4: unexpected EOF while looking for matching `)'
demo.sh: line 6: syntax error: unexpected end of file
```

Another way of debugging the script

```
[ec2-user@shell-script ~]$ cat demo.sh
#!/bin/bash -x
pwd
asjahdh
user=$(whoami
echo "This is next line"

[ec2-user@shell-script ~]$ ./demo.sh
+ pwd
/home/ec2-user
+ asjahdh
./demo.sh: line 3: asjahdh: command not found
./demo.sh: line 4: unexpected EOF while looking for matching `)'
./demo.sh: line 6: syntax error: unexpected end of file
```

7. Exit status of a command :-

Each linux command returns a status when it is executed.
Status 0 command got execute successfully
and non zero status means either command failed or returned error.

We can display the exit status of a command using $?

```
[ec2-user@shell-script ~]$ pwd
/home/ec2-user
[ec2-user@shell-script ~]$ echo $?
0

[ec2-user@shell-script ~]$ adjah
-bash: adjah: command not found
[ec2-user@shell-script ~]$ echo $?
127

[ec2-user@shell-script ~]$ adjaha
-bash: adjaha: command not found
```

```
[ec2-user@shell-script ~]$ cmd_rc=$?
[ec2-user@shell-script ~]$ echo $cmd_rc
127

[ec2-user@shell-script ~]$ cat demo.sh | grep "echo"
echo "This is next line"
[ec2-user@shell-script ~]$ echo $?
0
[ec2-user@shell-script ~]$ cat demo.sh | grep "Manoj"
[ec2-user@shell-script ~]$ echo $?
1
```

A non zero command (1-255 values) exit means command was a failure.

Example

```
        127 -- command not found
          1 -- command failed during execution
          2 -- Incorrect command usage for example ls -xyz
```

8. SIMPLE IF AND IF-ELSE CONDITIONAL STATEMENT :-

```
[ec2-user@shell-script ~]$ cat simpleif.sh
#!/usr/bin/env bash

if which java
then
  echo "Java is installed"
  echo "You can write some java programs"
fi
[ec2-user@shell-script ~]$ ./simpleif.sh
/usr/bin/java
Java is installed
You can write some java programs
```

To nullify the output of which java command we can write the script like this.

```
[ec2-user@shell-script ~]$ cat simpleif.sh
#!/usr/bin/env bash

if which java 2> /dev/null 1> /dev/null
then
  echo "Java is installed"
  echo "You can write some java programs"
fi
```

```
[ec2-user@shell-script ~]$ ./simpleif.sh
Java is installed
You can write some java programs
```

No output will be returned if the condition in the if statement doesn't return any value.

```
[ec2-user@shell-script ~]$ cat simpleif.sh
#!/usr/bin/env bash

if which apache2 2> /dev/null 1> /dev/null
then
  echo "Java is installed"
  echo "You can write some java programs"
fi
[ec2-user@shell-script ~]$ ./simpleif.sh
[ec2-user@shell-script ~]$
```

We can use exit status of a command as well in the if condition.

```
[ec2-user@shell-script ~]$ cat simpleif.sh
#!/usr/bin/env bash

which java
if [[ $? -eq 0 ]]
then
  echo "Java is installed"
fi

which docker 2>/dev/null
if [[ $? -eq 0 ]]
then
  echo "Docker is installed"
fi
[ec2-user@shell-script ~]$ ./simpleif.sh
/usr/bin/java
Java is installed
```

9. SIMPLE IF ELSE STATEMENT :-

```
[ec2-user@shell-script ~]$ cat simpleif.sh
#!/usr/bin/env bash

which java 2>/dev/null 1>/dev/null
if [[ $? -eq 0 ]]
then
  echo "Java is installed"
fi

which docker 2>/dev/null
if [[ $? -eq 0 ]]
then
  echo "Docker is installed"
else
  echo "Docker is not installed"
fi
[ec2-user@shell-script ~]$ ./simpleif.sh
Java is installed
Docker is not installed
```

10. LOGICAL OPERATORS :-

Logical AND operator && or -a -- will compare two inputs and if both are true it will return true else false
Logical OR operator || or -o -- will check two conditions will return true if any of them is true and return false if both of them are false
Logical NOT operator ! -- will return true when the condition is false and return false if the condition is true

11. FOR LOOP SCRIPTS:

Simple for loop output

```
#!/bin/bash
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

Simple for loop output

```
#!/bin/bash
for i in eat run jump play
do
echo See Imran $i
done
```

for loop to create 5 files named 1-5

```
#!/bin/bash
for i in {1..5}
do
touch $i done
```

for loop to delete 5 files named 1-5

```
#!/bin/bash
for i in {1..5}
do
rm $i done
```

Specify days in for loop

```
#!/bin/bash
i=1

for day in Mon Tue Wed Thu Fri
do
 echo "Weekday $((i++)) : $day"
done
```

List all users one by one from /etc/passwd file

```
#!/bin/bash
i=1
for username in `awk -F: '{print $1}' /etc/passwd` do
 echo "Username $((i++)) : $username"
done
```

12. DO-WHILE SCRIPT

Script to run for a number of times

```
#!/bin/bash
c=1
while [ $c -le 5 ]
do
    echo "Welcone $c times"
    (( c++ ))
done
```

Script to run for a number of seconds

```
#!/bin/bash
count=0
num=10
while [ $count -lt 10 ]
do
    echo
    echo $num seconds left to stop this process $1
    echo
    sleep 1
num=`expr $num - 1`
count=`expr $count + 1`
done
echo
echo $1 process is stopped!!!
echo
```