What is Shell Scripting?

A shell script is a script written for a command-line shell, which is a user interface for access to an operating system's services. Shell scripts allow users to automate tasks, execute sequences of commands, and perform various operations within the shell environment.

types of shells.

Bourne shell

C shell

Bash shell

Korn shell

Z shell

Why do we need shell scripts?

1. **To avoid repetitive work and automation**

2. System admins use shell scripting for routine backups.

3. System monitoring

4. Adding new functionality to the shell etc.

Examples:

How to send email alerts ?

script checks the available free RAM on the system, and if it's less than or equal to 700 MB, it sends a warning email to the specified email address.

```
#!/bin/bash
to="xyz@gmail.com"
var=$(free -mt | grep Total | awk '{print $4}')
if [ "$var" -le 700 ];
then
 echo "Sending mail because free size is less than 700"
 echo "Warning: RAM size is low" | sendmail "$to"
fi
~
~
```

When you run this script, it will check the available RAM and send an email if the free size is less than 700 MB. Make sure that your system has the sendmail command configured correctly to send emails.

```
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~# free -mt | grep Total | awk '{print $4}'
157
root@ip-172-31-86-128:~# ./hello.sh
Sending mail because free size is less than 700
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~#
```

Automate Server Inventory using Shell Script

This script collects information about the server, such as the server name, IP address, uptime, and OS type, and then stores this information in a CSV file

```
aws  Services  Search  [Alt+S]
#!/bin/bash

# Assign server name to a variable
server_name=$(uname -n)

# Assign IP address to a variable using ip command and filter it
ip=$( ip address | grep inet | awk 'NR==3{print $2}')

# Assign uptime to a variable using uptime command and extract the third field
uptime=$(uptime | awk '{print $3}')

# Assign OS type to a variable using uname command
os_type=$(uname)

# Create a new CSV file named csv_file and write header
echo "server_name,ip,uptime,os_typ, all information store in csv file " > csv_file

# Append server information to the CSV file
echo "$server_name,$ip,$uptime,$os_typ," >> csv_file
~
~
~
```

When you run this script, it will collect the server information and store it in the csv_file in CSV format.

output:

```
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~# ls
hello.sh  new_directory  servers.sh  snap  sv.sh  temp.txt  variable.sh  wh.txt
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~# ./hello.sh
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~# ls
csv_file  hello.sh  new_directory  servers.sh  snap  sv.sh  temp.txt  variable.sh  wh.txt
root@ip-172-31-86-128:~#
root@ip-172-31-86-128:~# cat csv_file
server_name,ip,uptime,os_typ, all information store in csv file
ip-172-31-86-128,172.31.86.128/20,10,,
root@ip-172-31-86-128:~#
```

Working with PS1 Prompt Environment Variable

Display username, hostname and current working directory in the prompt

- **\u – Username**
- **\h – Hostname**
- **\w – Full path of the current working directory**

This command will display the current value of the PS1 variable, which represents your Bash prompt configuration.

echo "$PS1"

output:

```
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$ echo "$PS1"
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$
ubuntu@ip-172-31-86-128:~$
```

custom prompt for your shell

PS1="[\t ==>\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$ "

output:

[\t ==>: This part includes the current time (\t) followed by the string ==>

PS1 to this value, your shell prompt will include the current time, the string ==>, the username, hostname, working directory, and whether the user is running as root or not.

```
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$ PS1="[\t ==>\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$ "
[08:58:52 ==>ubuntu@ip-172-31-86-128:~$
[08:58:55 ==>ubuntu@ip-172-31-86-128:~$
[08:58:55 ==>ubuntu@ip-172-31-86-128:~$
[08:58:56 ==>ubuntu@ip-172-31-86-128:~$ ls
input_command
[08:59:00 ==>ubuntu@ip-172-31-86-128:~$
```

```
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$ PS1="[\t ==>\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$ "
[08:58:52 ==>ubuntu@ip-172-31-86-128:~$
[08:58:55 ==>ubuntu@ip-172-31-86-128:~$
[08:58:55 ==>ubuntu@ip-172-31-86-128:~$
[08:58:56 ==>ubuntu@ip-172-31-86-128:~$ ls
input_command
[08:59:00 ==>ubuntu@ip-172-31-86-128:~$
[08:59:23 ==>ubuntu@ip-172-31-86-128:~$
[08:59:24 ==>ubuntu@ip-172-31-86-128:~$
[08:59:25 ==>ubuntu@ip-172-31-86-128:~$ PS1="\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$ "
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
```

This setup will display the prompt with a red color for the time, a red color for the window title, and the username, hostname, and working directory in default color.

PS1="[\[\e[0;31m\]\t ==> \[\e]0;31m\]\u@\h: \w\a]\u@\h:\w\$ "

output:

```
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$ ]PS1="[\e[0;31m \t ==>^C
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$ echo "$PS1"
\[\e]0:31m; \t \u@\h: \w\a\]\u@\h:\w$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$
ubuntu@ip-172-31-86-128:~$ PS1="[\e[0;31m \t ==>\[\e]0:31m; \t \u@\h: \w\a\]\u@\h:\w$ "
[ 09:05:04 ==>ubuntu@ip-172-31-86-128:~$
[ 09:05:10 ==>ubuntu@ip-172-31-86-128:~$
[ 09:05:10 ==>ubuntu@ip-172-31-86-128:~$
[ 09:05:10 ==>ubuntu@ip-172-31-86-128:~$
[ 09:05:10 ==>ubuntu@ip-172-31-86-128:~$
[ 09:05:11 ==>ubuntu@ip-172-31-86-128:~$
[ 09:05:11 ==>ubuntu@ip-172-31-86-128:~$
```

options:

- **\a an ASCII bell character (07)**
- **\d the date in "Weekday Month Date" format (e.g., "Tue May 26″)**
- **\D{format} – the format is passed to sift time (3) and the result is inserted into the prompt string; an empty format results in a locale-specific time representation. The braces are required**
- **\e an ASCII escape character (033)**
- **\h the hostname up to the first part**

- \H the hostname
- \j the number of jobs currently managed by the shell
- \l the base name of the shell's terminal device name
- \n newline
- \r carriage return
- \s the name of the shell, the base name of $0 (the portion following the final slash)
- \t the current time in 24-hour HH:MM:SS format
- \T the current time in 12-hour HH:MM:SS format
- \@ the current time in 12-hour am/pm format
- \A the current time in 24-hour HH:MM format
- \u the username of the current user
- \v the version of bash (e.g., 2.00)
- \V the release of bash, version + patch level (e.g., 2.00.0)
- \w the current working directory, with $HOME abbreviated with a tilde
- \W the base name of the current working directory, with $HOME abbreviated with a tilde
- \! the history number of this command
- \# the command number of this command
- $ if the effective UID is 0, a #, otherwise a $
- \nnn the character corresponding to the octal number nnn
- \ a backslash
- \[begin a sequence of non-printing characters, which could be used to embed a terminal control sequence into the prompt
- \e[– Indicates the beginning of color prompt
- x;ym – Indicates color code. Use the color code values mentioned below.
- \e[m – indicates the end of color prompt

how to write and execute shell script ? simple shell script to install tomcat

```
#!/bin/bash

wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.85/bin/apache-tomcat-9.0.85.tar.gz

tar -xvf  apache-tomcat-9.0.85.tar.gz

rm  apache-tomcat-9.0.85.tar.gz

mv apache-tomcat-9.0.85/ tomcat9
```

output:

```
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ./tomcat.sh
--2024-02-18 09:56:31--  https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.85/bin/apache-tomcat-9.0.85.tar.gz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11809177 (11M) [application/x-gzip]
Saving to: 'apache-tomcat-9.0.85.tar.gz'

apache-tomcat-9.0.85.tar.gz          100%[===================================================================================>]  11.26M  --.-KB/s    in 0.08s

2024-02-18 09:56:31 (148 MB/s) - 'apache-tomcat-9.0.85.tar.gz' saved [11809177/11809177]

apache-tomcat-9.0.85/conf/
apache-tomcat-9.0.85/conf/catalina.policy
apache-tomcat-9.0.85/conf/catalina.properties
apache-tomcat-9.0.85/conf/context.xml
apache-tomcat-9.0.85/conf/jaspic-providers.xml
apache-tomcat-9.0.85/conf/jaspic-providers.xsd
apache-tomcat-9.0.85/conf/logging.properties
apache-tomcat-9.0.85/conf/server.xml
apache-tomcat-9.0.85/conf/tomcat-users.xml
apache-tomcat-9.0.85/conf/tomcat-users.xsd
```

==========

how to run user defined shell script as a system command ? (free ram size in MB)

root@ip:~# vimfree.sh

#!/bin/bash

free -m | awk 'NR==2{print $4 "MB" }'

root@ip~# chmod +xfree.sh

root@ip:~# ./free.sh

183MB

OR

root@ip:~# pwd

/root

root@ip:~# /root/free.sh

183MB

```
aws   Services   Search                                   [Alt+S]
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ./free.sh
183MB
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# /root/free.sh
183MB
root@ip-172-31-89-149:~#
```

==========

check the path of file or command

root@ip:~# which ls

/usr/bin/ls

root@ip:~# which cp

/usr/bin/cp

root@ip:~# echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin

How to change file path

Move the file to/snap/bin/

root@ip:~#mvfree.sh/snap/bin/

without using (./) run file name

root@ip:~# free.sh

374MB

without using (.sh)

root@ip:~# cd /snap/bin/

root@ip:/snap/bin# ls

free.sh

root@ip:/snap/bin# mv free.sh freem

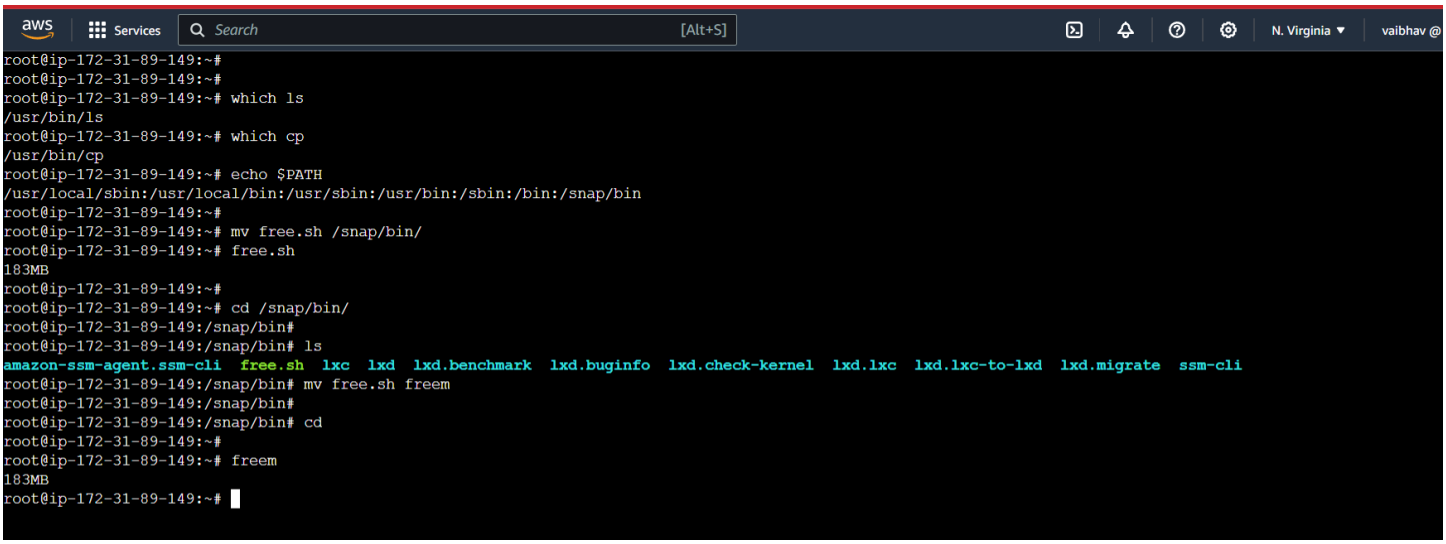
return to home location

root@ip:/snap/bin# cd

run only file name

root@ip:~# freem

374MB



==========

how to change the temporary execute PATH

root@ip:~# mkdir new_directory

root@ip:~# ls

new_directory

move the file

root@ip:~# mv /snap/bin/freem new_directory/

root@ip:~# cd new_directory/

root@ip:~/new_directory# ls

freem

check the file PATH

root@ip:~/new_directory# pwd

/root/new_director

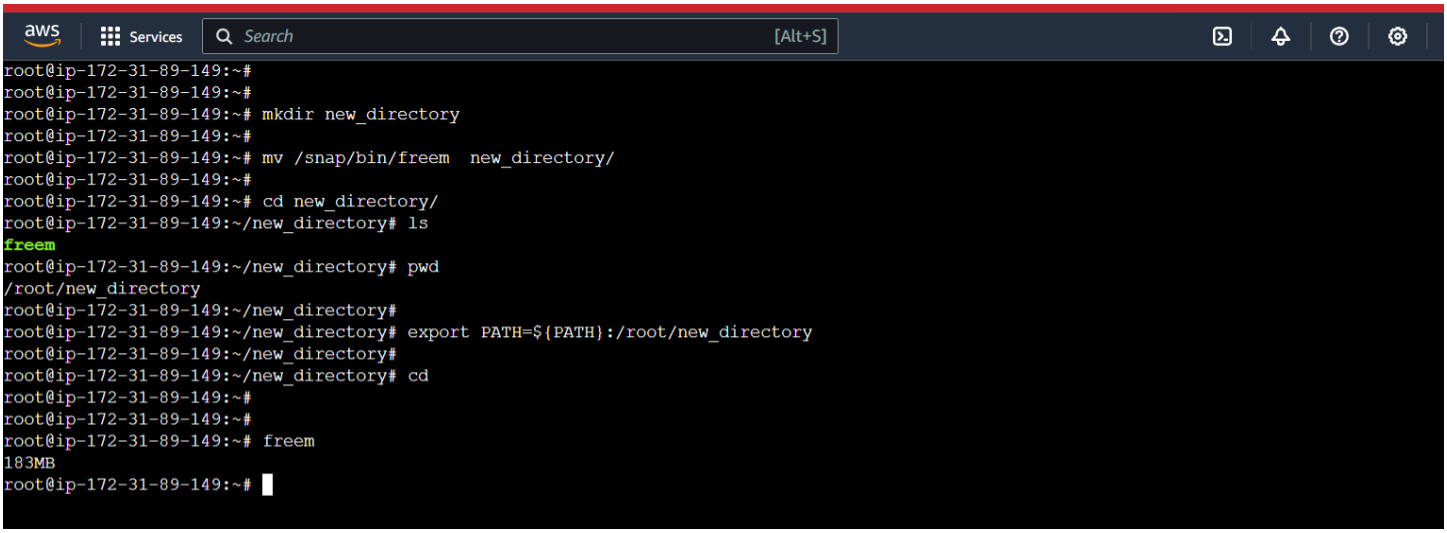
change the temporary execute PATH

export PATH=${PATH}:/root/new_directory

run from any location in your terminal without specifying the full path to the file.

root@ip:~# freem

183MB



==========

how to change the permanent execute PATH

This command displays all files and directories, including hidden

root@ip:~# ls -a

. .. .bash_history .bashrc .profile .ssh

root@ip:~# vim .profile

export PATH="$PATH:/root/new_directory"

:wq! —-save file

source a shell configuration file like .profile to apply changes made to them,

root@ip:~# source .profile

root@ip:~# freem

374MB

root@ip:~# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin:/root/new_directory

==========

echo command :

echo command to display command output

advanced syntax of echo command : echo [option] string /$variable/$(command)

option : \n ----line , \t ---tab space , \c ---cut next word ,

Examples:

```
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# echo -e "ganesh\nvaibhav"
ganesh
vaibhav
root@ip-172-31-89-149:~# echo -e "ganesh\tvaibhav"
ganesh  vaibhav
root@ip-172-31-89-149:~# echo -e "ganesh\cvaibhav"
ganeshroot@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# echo -e "ganesh\avaibhav"
ganeshvaibhav
root@ip-172-31-89-149:~# echo -e "ganesh\bvaibhav"
ganesvaibhav
root@ip-172-31-89-149:~#
```

****

colors	FG	BG
black	30	40
red	31	41
green	32	42
yellow	33	43
blue	34	44
Magenta	35	45
cyan	36	46
wight	37	47

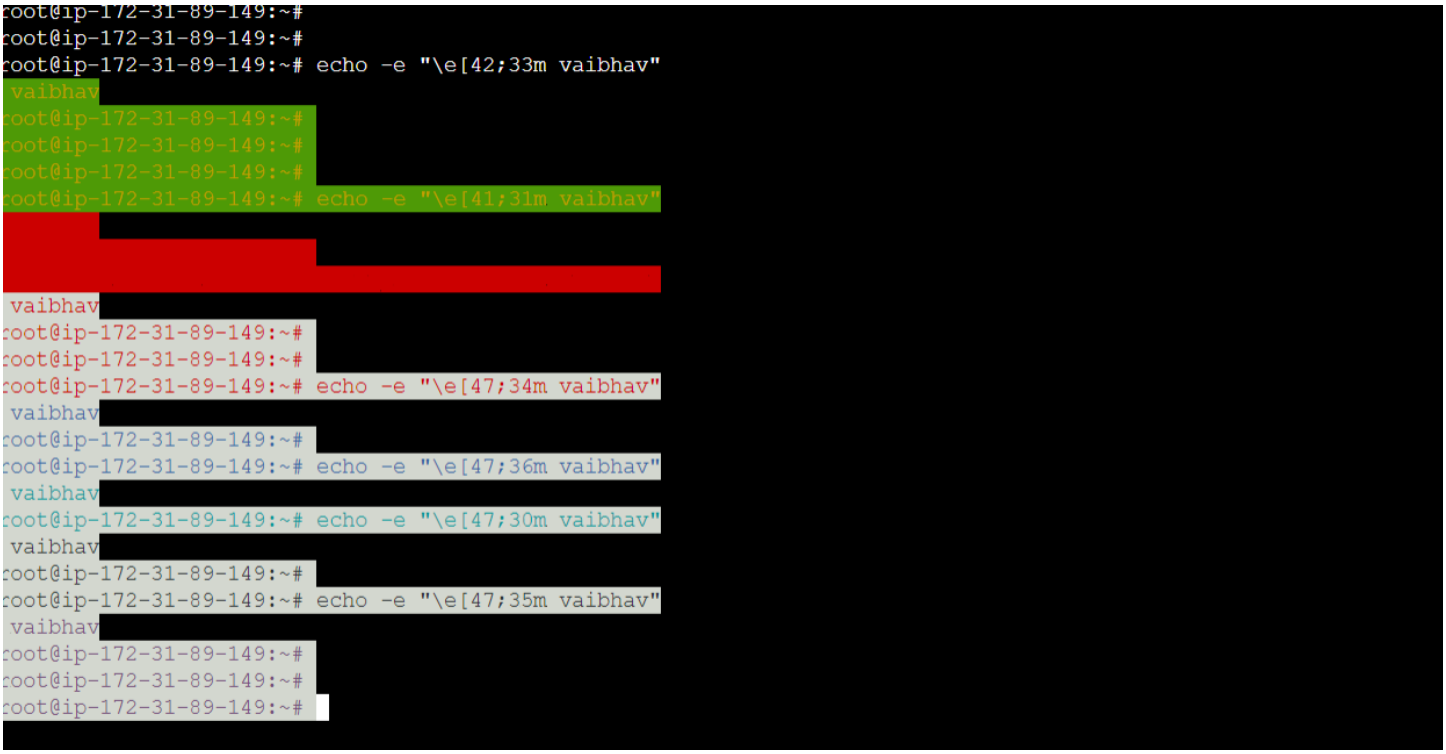
syntax :

#\e: This is the escape character.

#42: Sets the background color to green.

#33: Sets the text color to yellow.

echo -e "\e[42;33m vaibhav"



small script

root@ip:~# vim hello.sh

#!/bin/bash

#\e[0m: This ANSI escape sequence resets the text attributes to their default values.

echo -e "\e[1;34mWelcome to my blog!\e[0m"

root@ip:~# chmod +xhello.sh

root@ip:~# ./hello.sh

Welcome to my blog!

==========

quotes:

using single quotes output :

root@ip-:~# echo '$(ls)'

$(ls)

using double quotes output :

root@ip-:~# echo "$(ls)"

hello.sh

new_directory

using non quotes output :

root@ip-:~# echo $(ls)

hello.sh new_directory snap

==========

comments :

***use single line comment* : #**

***use multi line comment* : << myname**

myname

<<myname

\e: This is the escape character.

42: Sets the background color to green.

33: Sets the text color to yellow.

myname

echo -e "\e[42;33m vaibhav"

==========

variables 😀

A shell variable is a character string in a shell that stores some value. It could be an integer, filename, string, or some shell command itself.

****Rule :*the name of a variable can only contain letters (*a to z or A to Z*) , number (*0 to 9*) , and the underscore character*(_)**

#!/bin/bash

echo "variables"

x=10

y=20

z=30

echo "value of x: $x"

echo "value of y: $y"

echo "value of z: $z"

OR

```
#!/bin/bash
echo "variables"
x=10
y=20
z=30
echo -e "value of x: $x \n value of y: $y \n value of z: $z"
~
~
~
~
```

Output:

```
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory# freem
variables
value of x: 10
 value of y: 20
 value of z: 30
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory#
```

Different types of variables like:

user defined variables:

root@ip:~# x=10

root@ip:~# echo "$x"

10

system defined variables:

root@ip-:~# echo "$SHELL”

/bin/bash

root@ip-:~# env #---------show all system defined variables

special variables:

root@ip-:~# echo $?

0

==========

input & output command

*****echo :******output command***

*****read :******input command***

simple way 😀:

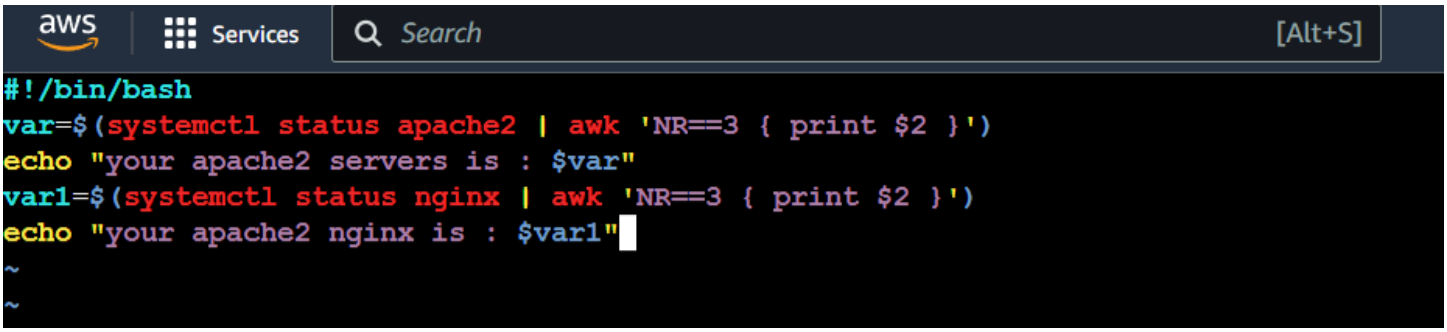
#!/bin/bash

var=$(systemctl status apache2 | awk 'NR==3 { print $2 }')

echo "your apache2 servers is : $var"

var1=$(systemctl status nginx | awk 'NR==3 { print $2 }')

echo "your apache2 nginx is : $var1"



output:

```
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory# freem
your apache2 servers is : active
your apache2 nginx is : failed
root@ip-172-31-89-149:~/new_directory# systemctl status nginx
× nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: failed (Result: exit-code) since Sun 2024-02-18 10:27:46 UTC; 31s ago
       Docs: man:nginx(8)
    Process: 16463 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 16464 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=1/FAILURE)
        CPU: 9ms

Feb 18 10:27:45 ip-172-31-89-149 nginx[16464]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Unknown error)
Feb 18 10:27:45 ip-172-31-89-149 nginx[16464]: nginx: [emerg] bind() to [::]:80 failed (98: Unknown error)
Feb 18 10:27:45 ip-172-31-89-149 nginx[16464]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Unknown error)
Feb 18 10:27:45 ip-172-31-89-149 nginx[16464]: nginx: [emerg] bind() to [::]:80 failed (98: Unknown error)
Feb 18 10:27:46 ip-172-31-89-149 nginx[16464]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Unknown error)
Feb 18 10:27:46 ip-172-31-89-149 nginx[16464]: nginx: [emerg] bind() to [::]:80 failed (98: Unknown error)
Feb 18 10:27:46 ip-172-31-89-149 nginx[16464]: nginx: [emerg] still could not bind()
Feb 18 10:27:46 ip-172-31-89-149 systemd[1]: nginx.service: Control process exited, code=exited, status=1/FAILURE
Feb 18 10:27:46 ip-172-31-89-149 systemd[1]: nginx.service: Failed with result 'exit-code'.
Feb 18 10:27:46 ip-172-31-89-149 systemd[1]: Failed to start A high performance web server and a reverse proxy server.
```

==========

read command :

```bash
#!/bin/bash

# Prompt the user to enter the value of 'a' and 'b'
read -p "Enter value of a: " a
read -p "Enter value of b: " b

# Calculate the sum of 'a' and 'b'
result=$(expr $a + $b)

# Display the result
echo "a + b = $result"
```

output:

```
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory#
root@ip-172-31-89-149:~/new_directory# freem
Enter value of a: 5
Enter value of b: 6
a + b = 11
root@ip-172-31-89-149:~/new_directory# freem
Enter value of a: 45
Enter value of b: 22
a + b = 67
root@ip-172-31-89-149:~/new_directory#
```

==========

Advance:

```
#!/bin/bash

# Prompt the user to enter the server name

read -p "enter your server name " web_sv


# Use systemctl status to get the status of the specified service and extract the status using awk

var1=$(systemctl status $web_sv | awk 'NR==3 { print $2 }')


# Print the status of the service with formatting

echo -e "your $web_sv is :\e[31m$var1\e[0m "
```

==========

how to providing input command line arguments

```
#!/bin/bash

read -p "first value v1:" v1

read -p "second value v2 :" v2
```

```
result= $(expr $v1 + $v2)

echo "addition of $v1 and $v2 is : $result"
```

output:

first value v1: 1

second value v2 : 2

addition of 1 and 2 : 3

```
#!/bin/bash

v1=$1

v2=$2

result= $(expr $v1 + $v2)

echo "addition of $v1 and $v2 is : $result"
```

output:

ubuntu@ip-172-31-86-128:~$ ./input_command 5 6

addition of 5 and 6 : 11

==========

Decision Control statements of Shell Scripting

By using conditional statements like if statement, elif statement and else statement,

Syntax :

```
if [ [condition ] ]

then

statement

fi
```

Example:

```
#!/bin/bash
a=$#
 #checks if the value of the variable a is equal to 3.
if [ $a -eq 3 ]; then
 v1=$1
 v2=$2
 v3=$3
 result=$(expr $v1 + $v2 + $v3)
 echo "Addition of $v1 and $v2 & $v3 is: $result"
fi
```

output:

ubuntu@ip-172-31-86-128:~$ ./ if_condition 3 4 4

Addition of 3 and 4 & 4 is: 11

Example:

Your script seems to provide options to start, stop, and check the status of the Apache2 service based on user input

This script now correctly handles the user input for starting, stopping, and checking the status of the Apache2 service, and it displays appropriate messages.

```
#!/bin/bash
read -p "select service action start and stop : " action
if [ "$action" == "start" ]
then
 sudo systemctl start apache2
 echo "please wait....."
```

```
 echo "apache2 service start"
fi
if [ "$action" == "stop" ]
then
 sudo systemctl stop apache2
 echo "please wait....."
 echo "apache2 service stop"
fi
if [ "$action" == "status" ]
then
 sudo systemctl status apache2
 echo "please wait....."
 echo "apache2 service status show "
fi
```

```
aws  Services  Search  [Alt+S]
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# systemctl status apache2
○ apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Sun 2024-02-18 10:48:14 UTC; 21s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 16721 ExecStop=/usr/sbin/apachectl graceful-stop (code=exited, status=0/SUCCESS)
   Main PID: 15751 (code=exited, status=0/SUCCESS)
        CPU: 119ms

Feb 18 10:27:00 ip-172-31-89-149 systemd[1]: Starting The Apache HTTP Server...
Feb 18 10:27:01 ip-172-31-89-149 systemd[1]: Started The Apache HTTP Server.
Feb 18 10:48:14 ip-172-31-89-149 systemd[1]: Stopping The Apache HTTP Server...
Feb 18 10:48:14 ip-172-31-89-149 systemd[1]: apache2.service: Deactivated successfully.
Feb 18 10:48:14 ip-172-31-89-149 systemd[1]: Stopped The Apache HTTP Server.
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ./demo.sh
select service action start and stop : start
please wait.....
apache2 service start
root@ip-172-31-89-149:~# systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2024-02-18 10:48:58 UTC; 5s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 16734 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 16738 (apache2)
      Tasks: 55 (limit: 1121)
     Memory: 4.7M
```

==========

what is a Loop ?

some time we need to execute the same code several time

following types

The while loop

The for loop

The until loop

The select loop

for statement in Shell Script in Linux

The for loop operates on lists of items. It repeats a set of commands

Syntax:

```
#!/bin/bash

for var in $(seq 1 10)

do

echo "welcome to shell scripting "

echo " i am vaibhav "

done
```

==========

Shell Scripting Generate or Print Range of Numbers (Sequence of Numbers for Loop)

The syntax is as follows:

1. ***seq LAST***

2. ***seq FIRST LAST***
3. ***seq FIRST INCREMENT LAST***

Examples:

```
root@ip-172-31-89-149:~# seq 5
1
2
3
4
5
root@ip-172-31-89-149:~# seq 1 2 10
1
3
5
7
9
root@ip-172-31-89-149:~# seq 0 2 10
0
2
4
6
8
10
root@ip-172-31-89-149:~#
```

Examples:

Your script is a simple Bash script that prints "Welcome" followed by the iteration number, for 5 iterations.

#!/bin/bash

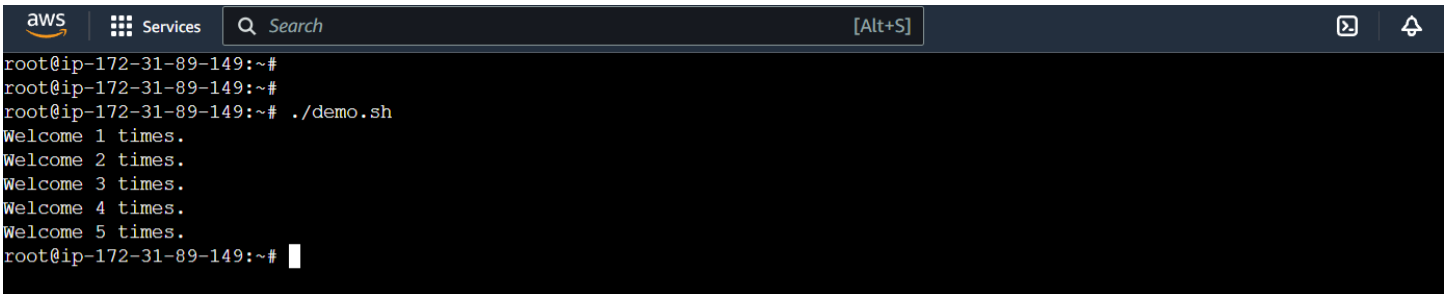
for i in $(seq 5)

do

echo "Welcome $i times"

done

output:



Using bash brace expansion in shell scripting to generate sequence of numbers

root@ip-:~# echo {1..3}

***output*:**

1 2 3

Examples:

When you run this script, it will produce the same output as before: "Welcome 1 times." through "Welcome 5 times." each on a separate line.

#!/bin/bash

for ((a=1; a <= 5 ; a++))

do

echo "Welcome $a times."

done

for ((a=1; a <= 5 ; a++)): This line sets up a C-style for loop. It initializes the variable a to 1, specifies the condition a <= 5 for continuing the loop, and increments a by 1 after each iteration.

output :

```
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ./demo.sh
Welcome 1 times.
Welcome 2 times.
Welcome 3 times.
Welcome 4 times.
Welcome 5 times.
root@ip-172-31-89-149:~#
```

==========

while loop

while loop is used to execute a set of commands repeatedly as long as a specified condition is true.

syntax:

while [condition]

do

 # Commands to execute

done

Examples:

while loop to print the value of a counter variable from 1 to 5. Here's a breakdown of each part:

while [$counter -le 5]: This line starts a while loop that continues as long as the value of the counter variable is less than or equal to 5.

((counter++)): This line increments the value of the counter variable by 1

```
#!/bin/bash

counter=1

while [ $counter -le 5 ]

do

 echo "Counter is: $counter"

 ((counter++)) # Increment the counter
```

done

output :

```
root@ip-172-31-89-149:~#  cat demo.sh
#!/bin/bash
counter=1
while [ $counter -le 5 ]
do
 echo "Counter is: $counter"
 ((counter++)) # Increment the counter
done
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ./demo.sh
Counter is: 1
Counter is: 2
Counter is: 3
Counter is: 4
Counter is: 5
root@ip-172-31-89-149:~#
```

Reading a file with a while loop

root@ip-:~# echo "my name is vaibhav " > temp.txt

root@ip-:~# vim while.sh

#!/usr/bin/bash

This line assigns the filename "temp.txt" to the variable file.

file="temp.txt"

while loop that reads each line from the file specified by the variable file

while read -r line;

do

echo $line: This line echoes (prints) the contents of the variable line

echo $line

< "$file" instructs Bash to redirect the contents of the file specified by the variable $file as

#input to the while loop.

done < "$file"

When you run this script, it will read each line from the "temp.txt" file and print it to the console. If "temp.txt" contains:

output :

```
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# cat temp.txt
 hii my name is vaibhav
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ls
demo.sh  snap  temp.txt
root@ip-172-31-89-149:~# ./demo.sh
hii my name is vaibhav
root@ip-172-31-89-149:~#
```

***Examples*:**

Writing to a file using a while loop

script prompts the user to enter content into a file named "wh.txt" and then continuously reads input from the user until they press Ctrl+D (indicating end-of-file). Each line of input is then appended to the file. Here's a breakdown of the script:

```
#! /bin/bash

file=wh.txt

echo “Enter the content into the file $file”

while read line

do

#This line appends the content of each line read from the user to the file specified by the variable $file

echo $line >> $file

done
```

output :

```
root@ip-172-31-89-149:~# ./demo.sh
"Enter the content into the file wh.txt"
hiii

bye

root@ip-172-31-89-149:~#
logout
ubuntu@ip-172-31-89-149:~$ sudo -i
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ls
demo.sh  snap  temp.txt  wh.txt
root@ip-172-31-89-149:~# cat wh.txt
hiii

bye

root@ip-172-31-89-149:~#
```

==========

Infinite while loop

```
#!/bin/bash

while :

do

echo "An Infinite loop"

done

# We can press Ctrl + C to exit the script
```

==========

select loop

select loop in Bash is used to create simple text-based menus in scripts. It allows users to choose from a list of options presented to them.

Example:

script provides a menu for selecting a department (CS, IT, ECE, EE)

```
#!/bin/bash

select department in CS IT ECE EE Quit

do
```

```
case $department in

CS)

echo "I am from CS department."

;;

IT)

echo "I am from IT department."

;;

ECE)

echo "I am from ECE department."

;;

EE)

echo "I am from EE department."

;;

Quit)

echo "Exiting..."

break

;;

*) echo "Invalid selection"

;;

esac

done
```

- **select department in CS IT ECE EE Quit: This line presents a menu with options: CS, IT, ECE, EE, and Quit. The user can select one of these options.**
- **case $department in ... esac: This construct checks the value of the variable $department and executes the corresponding block of code based on the selected department.**

- CS) ... IT) ... ECE) ... EE): These are the cases for each department. They echo a message indicating the selected department.
- Quit): This case handles the option to quit. It echoes a message and then exits the loop using break.
- *): This is the default case, executed when the user selects an option that doesn't match any of the defined cases. It echoes an "Invalid selection" message.
- esac: This keyword marks the end of the case construct.

output:

root@ip-172-31-86-128:~# ./hello.sh

1) CS

2) IT

3) ECE

4) EE

5) Quit

#? 1

I am from CS department.

#? 4

I am from EE department.

#? 2

I am from IT department.

#? 7

Invalid selection

#? 5

Exiting..

Example:

script is a simple calculator program that allows users to perform addition, subtraction, and multiplication operations on two numbers. Here's a

breakdown of how it works:

```
#!/bin/bash
select department in add sub multi Quit
do
case $department in
add)
read -p "enter first value:" v1
read -p "enter secand value:" v2
echo "addition of two numbers: $((v1+v2))"
;;
sub)
read -p "enter first value:"v1
read -p "enter secand value:" v2
echo "addition of two numbers: $((v1-v2))"
;;
multi)
read -p "enter first value:"v1
read -p "enter secand value:" v2
echo "addition of two numbers: $((v1*v2))"
;;
Quit)
echo "Exiting..."
break
;;
*) echo "Invalid selection"
;;
esac
done
~
~
```

output:

```
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~#
root@ip-172-31-89-149:~# ./demo.sh
1) add
2) sub
3) multi
4) Quit
#? 1
enter first value:5
enter secand value:6
addition of two numbers: 11
#? 3
enter first value:v15
enter secand value:2
addition of two numbers: 10
#? 2
enter first value:v18
enter secand value:2
addition of two numbers: 3
#? 4
Exiting...
root@ip-172-31-89-149:~#
```

==========

Until Loop :

The Until loop is used to iterate over a block of commands until the required condition is false.

Example:

#!/bin/bash

i=0

```
until [[ $i -eq 5 ]]
do
 echo "$i"
 ((i++))
done
```