Linux Basics & Shell Scripting

Linux Introduction:

Linux is Open Source Operating system developed by a Finnish student LinuS Torvalds in 1991.

It is free to download and use.

Linux is more reliable

Linux is compatible on many h/w like Macs, Mainframes, super computers, cell phones etc.

Very resistant to malware such as viruses, spyware etc.

What is Operating System:

Operating system is the software which acts as the interface between the hardware and the software we want to run on the hardware

Ex: Microsoft office is the software

Microsoft windows is operating system

Laptop/desktop is hardware

Ex:

Mobile is hardware

Examples of mobile hardware:

Symbiosis

Ios

android is OS

Application is the software

Unix was developed in 1970 by Denis retchi at Bell labs

Linux was developed by Linux torwalds in the year 1990.

Unix vs Linux

Unix is called mother of OS which laid foundation to Linux

Unix is designed mainly for mainframes and is in enterprises

Linux is for computer users, developers, servers

Linux is free, Unix is not

Unix runs on specific hardware only (AIX on IBM boxes)

Linux runs on many h/w

Linux Architecture

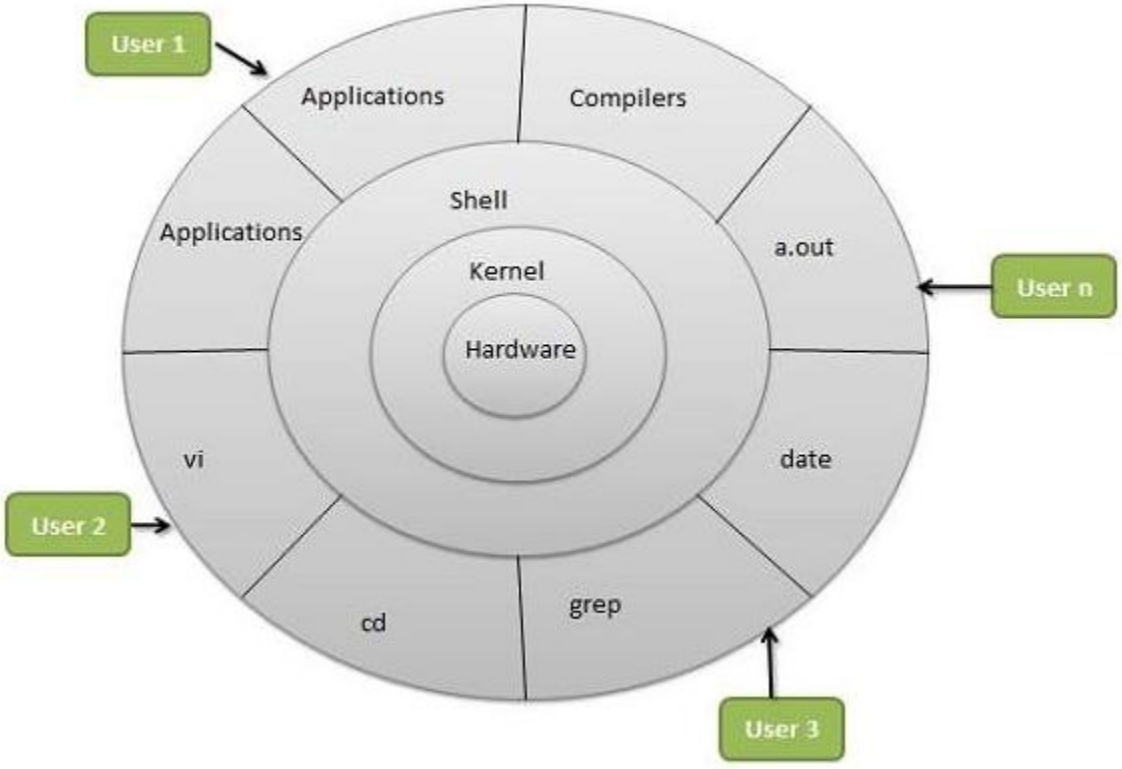
Linux has

H/w includes HDD RAM CPU

Kernel Heart of OS, talks to H/w, provides low level services to upper layer components

Shell interface between Kernel and user

Applications/Utilities – provide most of functions to users.



Shell:

Shell is the screen that user use to interact with operating system

Windows shell is the GUI – click on icons – it is called GUI shell

LUI – Line User interface – looks like DOS (Disk Operating system)prompt

Type a command and hit enter

Line user interface much more powerful

With LUI we get prompt to work on .

Bash shell is the default shell in Linux

ROOT:

Like Administrator for windows machine

Highest level of user/anything

As root anything can be performed

In Linux users will have home folders

Home directory – is the highest level of any user folder

root user – highest user

root directory – highest folder, home directory

Capitazalation:

Upper case and lower case letters are different

Home/home/hOMe- all are same in windows

All are different in case of Linux

Because each letter is ASCII character in Linux

Same for username

Windows is case insensitive, 2 files cannot be created wth names File, file.

Linux is case sensitive, 2 files can be created with names File, file

Server vs Desktop:

Server version is stripped down version of Linux – no GUI, lot of other tools are not installed automatically, it has what a server needs.

Desktop version: gives GUI version of Linux – looks like Windows/Mac.

Every distro has desktop and server editions ex: readhat/Ubuntu etc

As a desktop OS, Linux is not best choice.

Linux is good for Server configurations.

For setting up for website, does need to reboot often.

Linux Distributions:

People started developing their own code out of kernel developed by Linus

So there are many versions of linux which is called distributions/flavours

Example:

Redhat

Ubuntu

Google android

Fedora

Centos

Every distribution has a different purpose

Trustix linux – most secure

Laptop desktop – Ubuntu

Entreprise (On servers)– Redhat linux - customer center to support

DSL - very small distro – 53 MB

Open source:

All open source software is not free (usage is free, support is not free)

Everyone is allowed to see the source code

Ex: mysql database

They will give free software, but support they will charge

They get money from support

Open source software can be used at home/personal use/ at test lab

But cannot be used on production server, we need license to use them.

Linux Boot process

What happens when a power button is pressed on the server/laptop

BIOS – Basic Input Output System – does POST Poweron Self Test – checks whether all i/o devices are working fine (mouse monitor keyboard RAM HDD etc), searches loads and executes boot loader program

MBR – Master boot record :

Located in the first sector of bootable disk. ex: /dev/sda or /dev/hda

first program to be executed in Linux,of 512 bytes,

it has 3 components

1)primary bootloader of 446 bytes

2)partition table info in next 64 bytes

3) MBR validation check in last 2 bytes

So it loads and executes GRUB into memory

The Master Boot Record (MBR) is the information in the first sector of any hard disk or diskette that identifies how and where an operating system is located so that it can be boot (loaded) into the computer's main storage or random access memory.

The Master Boot Record is also sometimes called the "partition sector" or the "master partition table" because it includes a table that locates each partition that the hard disk has been formatted into. In addition to this table, the MBR also includes a program that reads the boot

sector record of the partition containing the operating system to be booted into RAM. In turn, that record contains a program that loads the rest of the operating system into RAM.

GRUB – Grand Unified Boot loader - responsible for selecting OS, loads kernel into memory

if multiple OS images are present, one image can be chooses using GRUB

GRUB displays a splash screen, waits for sometime, if user does not choose anything, it loads default kernel

So it loads and executes Kernel

Kernel –

executes init program

PID 1

init

Looks at /etc/inittab file to decide run level

Runlevel programs

Basic runlevels are:

0 – halt

1 – Single user mode

2 – Multiuser, without NFS

3 – Full multiuser mode CLI no graphics with network

4 – unused

5 – X11 – graphics, multiuser mode with network.

6 – reboot

Run levels:

Run level 0 – /etc/rc.d/rc0.d/

Run level 1 – /etc/rc.d/rc1.d/

Run level 2 – /etc/rc.d/rc2.d/

Run level 3 – /etc/rc.d/rc3.d/

Run level 4 – /etc/rc.d/rc4.d/

Run level 5 – /etc/rc.d/rc5.d/

Run level 6 – /etc/rc.d/rc6.d/

Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.

Programs starts with S are used during startup. S for startup.

Programs starts with K are used during shutdown. K for kill.

There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.

Linux Installation

Linux can be installed using CD/USD/using iso image/using network installation like kickstart

Or single machine can be installed using iso (which can be downloaded from net)

Linux can be installed using iso

iso – international standard organization

example:

redhat-7.3-x86_64.ga.iso

Directory structure:

/

- /root
- /tmp
- /dev/ /dev/sda, /dev/sdb
- /bin/
- /lib
- /usr
- /var
- /etc
- /home
- /boot
- /opt

iso;

RHEL-7.2-x86_64.iso → RHEL OS on x86_64 hardware

SUSE-12.SP1-ppc64le.iso → SLES on ppc64le hardware

Ubuntu-16.10-x86_64.iso → Ubuntu on x86_64 hardware

FQDN – Fully qualified Domain Name –

ctx1p25 – shortname

ctx1p25.in.xyz.com – FQDN

Basic Commands

ls; ls –lrt

ls list files in a directory

ls –lrt → lists files in long list format with time stamp

root@ctx2p02:~# ls -lrt

total 44

-rwxrwxrwx 1 root root 179 Nov 24 06:31 3.sh

-rwxrwxrwx 1 root root 277 Nov 24 07:07 fib.sh

-rwxrwxrwx 1 root root 103 Nov 24 07:19 1.sh

-rwxrwxrwx 1 root root 424 Nov 24 09:21 2.sh

-rwxr--r-- 1 root root 22 Nov 24 10:34 file

-rwxrwxrwx 1 root root 171 Nov 24 20:45 7.sh

-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh

-rw-r--r-- 1 root root 82 Nov 24 21:59 8.sh

-rwxrwxrwx 1 root root 936 Nov 25 05:49 10.sh

-rwxrwxrwx 1 root root 73 Nov 25 08:42 until.sh

-rw-r--r-- 1 root root   9 Nov 25 09:14 file90

root@ctx2p02:~#

cd; cd - ; cd ../../ abs vs rel path, cd ~

cd changes directory

cd .. → goes one directory up

cd - → takes to previous directory

cd ~ → take to user home directory

mkdir , mkdir –p

mkdir → create directory

mkdir –p → creates path

rm , rm –rf

rm → remove a directory

rm –rf → removes directory recursively forcibly

rmdir → remove a directory

pwd → shows present working directory

umask → decides the default permissions with which a file/directory will be created by a user.

chmod → change permissions of a user

chown → changes user

mv → move a file or a rename

cp, cp –r

cp → copies a file

cp –r → copies all files in a directory.

scp → secure copy protocol

syntax:

to transfer a file file from machine with ip 192.168.100.100 to 192.168.100.101

to remote directory /tmp from /var

scp /var/file username@192.168.100.101:/tmp

to transfer from 192.168.100.101 to 100

scp username@192.168.100.101:/tmp/file .

ftp – file transfer protocol

syntax: ftp ip or ftp FQDN

mget tp get all files with pattern matching

ssh – secure shell – to connect to a linux machine, listens on port 22

telnet - to connect to linux box, listens on port 23

ssh is preferred as the password Is sent encrypted over network

mount

to attach a file system from remote server to local machine.

for this NFS server has to be configured

syntax:

mount –o nfsvers=3 192.168.100.101:/mount /mnt

mounts file system from /mount from 192.168.100.101 to local machine in the mount point /mnt

umask to decide the default permissions with which a file/directory will be created by a user.

Default

Umask 022

For a directory:

Read permission – to list out the files

Write - create rename delete files within directory.

Execute – change the directory

redirection operator > >>

> this operator creates a new file , overwrites if any file exists

>> this operator creates a new file, appends to file, if the file exists

pipe → to pass output of one command to other command as input

cat → to display content of file on terminal

Shell properties

control U → removes everything on the line

control A → to go to beginning of line

control E → to go to end of line

control R (reverse search) → search the history based on a word

control w → remove word

tab → for command completion

Editors:

vi vim editor

insert mode – press i to go to insert mode

command mode - press ESC to go to command mode

yy to copy a line

p to paste

dd to cut

2yy – copies 2 lines

:1 → take to 1st line in file

:$ → takes to last line

:set nu → sets numbers

:se ic → case insensitive

:se nonum → removes numbers

:se noic → removes case insensitive

:w → save file

:wq save and quit

:q quit

:q! quit without saving

working on multiple file

vim → Improved version of vi editor

can open multiple files

split files

vim -o file1 file2 → to open 2 files

vim -O file1 file2

copy few lines from file1 to file2

to switch between files control+w

nano

control +x + enter to save file.

Editors usage:

Nano is for normal users. Emacs and Vim are for programmers

Vim/emacs – shows the loops /functions in different colour forms, easy to understand syntax

Vi/nano – shows in plain text without colours

grep

awk

sed

umask normal user 002

root 022

head → displays first n lines

syntax: head -10 file

tail → displays last n lines

syntax: tail –10 file

nohup

- ➔ to execute a command in background in no hangup mode
- ➔ nohup command &
- ➔ logs output to nohup.out

to display contents of a file

less

more

sed - stream editor

grep - global regular expression print

awk

package management commands

redhat:

rpm – redhat package manager

rpm –ivh → to install

rpm –qa → to search

rpm –e → to remove

Ubuntu

dpkg

dpkg –l → to list installed packages

dpkg –I → to install

dpkg –r → to remove

dpkg –purge → to remove completely

To install packages to handle dependencies automatically

for redhat/centos/fedora

yum

yum install <>

to configure:

/etc/yum.repos.d/iso.repo → to install from iso

/etc/yum.repos.d/redhat.repo → from net

yast →on SUSE linux

AIX commands:

lslpp -l → to list installed packages

installp → to install

grep command:

grep = global regular expression print

to search a pattern in a file

grep -i → case insensitive

grep –w → to search for a word

grep –n → to display number of pattern match

grep -q

grep -A3

grep -v → -ve search

egrep '|'

fgrep old

egrep

extended grep

grep uses basic regular expressions where the plus is treated literally, any line with a plus in it is returned.

grep "+" myfile.txt

egrep on the other hand treats the "+" as a meta character and returns every line because plus is interpreted as "one or more times".

egrep "+" myfile.txt

fgrep

fixed grep

will not take meta characters

grep "." myfile.txt

The above command returns every line of myfile.txt. Do this instead:

fgrep "." myfile.txt

Then only the lines that have a literal '.' in them are returned. fgrep helps us not bother escaping our meta characters.

? one

* 0 or more

[]

telnet

ssh

ping

traceroute

nslookup

tar

untar

zip

yum

yast

apt-get

rpm

dpkg

top

pstree

vmstat

sar

iostat

stat

cpio

dd

fdisk –l

lsblk

partitions – primary

extended

DNS

LDAP

reverse DNS

ftp protocol flow

configuring yum – how it works internally
file sytem types: ext2 3 4 xfs zfs

find

find . –name xx

find . – mtime 5

find . –name xx –exec rm

LVM – creating

creating vg, lv, fs

RAIDs

software raid

hardware raid

how to configure raid on Linux

Linux Package installation

For installing a single package:

RHEL /CentOS/Fedora/SLES - rpm -ivh <xyz.rpm>

Ubuntu – dpkg -I <xyz.deb>

For installing all dependent packages:

Configure the repository

Use the tools

RHEL / CentOS/Fedora – yum

Ubuntu – apt-get

SLES – yast

Yum can be configured in many ways:

1) using iso
2) Using internet
3) Using ftp sites

Using iso:

1) mount iso to local directory

mount -o loop RHEL-7.3-20161019.0-Server-x86_64-dvd1.iso /mnt

2) update /etc/yum.repos.d/iso.repo

[iso]

name=iso

baseurl=file:/mnt/

enabled=1

gpgcheck=0

3) install/remove packages using yum

yum install <>

yum remove <>

Shell scripting

The first line of shell script is called She-Bang

she bang line:#!

that instructs the shell to execute this script using the interpreter

Comment:# → that command won't be executed

command line arguments and return status

$? → gives Return status of a command

$# - no. of args

$@ - all args

$* - all args

$0 – file name

$1 – first argument

$n – nth arg

$$ - PID

echo – to print

sh –vx to debug

1.cleanup script

```
 # Cleanup
# Run as root, of course.

cd /var/log
cat /dev/null > messages
cat /dev/null > wtmp
echo "Log files cleaned up."
```

2. a proper cleanup script:

```
#!/bin/bash
# Proper header for a Bash script.
# Cleanup, version 2
# Run as root, of course.
# Insert code here to print error message and exit if not root.

LOG_DIR=/var/log
# Variables are better than hard-coded values.
cd $LOG_DIR

cat /dev/null > messages
cat /dev/null > wtmp
```

```
echo "Logs cleaned up."


exit #  The right and proper method of "exiting" from a script.

    #  A bare "exit" (no parameter) returns the exit status

    #+ of the preceding command.
```

The sha-bang (#!)

[1] at the head of a script tells your system that this file is a set of commands to be fed to the command interpreter indicated. The #! is actually a two-byte

[2] magic number, a special marker that designates a file type, or in this case an executable shell script (type man magic for more details on this fascinating topic).

Immediately following the sha-bang is a path name. This is the path to the program that interprets the commands in the script, whether it be a shell, a programming language, or a utility. This command interpreter then executes the commands in the script, starting at the top (the line following the sha-bang line), and ignoring comments. [3]

```
#!/bin/sh

#!/bin/bash

#!/usr/bin/perl

#!/usr/bin/tcl

#!/bin/sed -f

#!/bin/awk -f
```

Each of the above script header lines calls a different command interpreter

More commonly seen in the literature as she-bang or sh-bang. This derives from the concatenation of the tokens sharp (#) and bang (!)

Case statement:

```
case "$variable" in
```

```
  abc)  echo "\$variable = abc" ;;
  xyz)  echo "\$variable = xyz" ;;
          esac
```

Hello world program

```
#!/bin/bash
        STR="Hello World!"
        echo $STR
```

Difference between $@ and $*:

both are same when used without quotes with echo

when used with quotes, with IFS – filed separator

$@ prints does not print any thing

$* prints first character of IFS in between args

```
for i in "$@"
do
```

```
   echo $i  # loop $# times
done


for i in "$*"
do
   echo $i  # loop 1 times
done
```

It's safer to use "$@" instead of $*. When you use multiword strings as arguments to a shell script, it's only "$@" that interprets each quoted argument as a separate argument.

As the output above suggests, if you use $*, the shell makes a wrong count of the arguments.

```
#!/bin/bash
#A scrip to explain command line args
echo "filename $0";
echo $1
echo $2
echo $3
echo $4
echo $5
echo $6
echo $7
```

```
echo $8
echo $9



[root@reviewb ~]# vi xyz.sh
#!/bin/bash
# take 2 numbers as input from user and add them
#echo "Enter a";
#read a;
#echo "Enter b";
#read b;
#sum=0;

a=$1;
b=$2;

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



[root@r8r3m2kvm tmp]# cat 2.sh
#!/bin/bash
#A scrip to explain command line args
echo "filename $0";
echo $1
```

```
echo $2
echo $3
echo $4
echo $5
echo $6
echo $7
echo $8
echo $9
echo $10
echo $11
echo $12
echo $13
echo $14
[root@r8r3m2kvm tmp]# ./2.sh 1 2 3 4 5 6 7 8 9 10 11 12 13
filename ./2.sh
1
2
3
4
5
6
7
8
9
10
11
```

12

13

```
[root@r8r3m2kvm tmp]# cat 2.sh
#!/bin/bash
#A scrip to explain command line args
echo "filename $0";
echo "Bfore shift";
echo $1
echo $2
echo $3
shift;
echo "After shift";
echo $1
echo $2
echo $3
[root@r8r3m2kvm tmp]# ./2.sh 1 2 3
filename ./2.sh
Bfore shift
1
2
3
After shift
2
3
```

```
[root@r8r3m2kvm tmp]#
```

```
[root@r8r3m2kvm tmp]# cat 3.sh
#!/bin/bash
echo "enter a variable";
read n;
echo "You have entred $n";
echo "enter a variable";
read m;
echo "You have entred $m";
l=`expr $n + $m`;
echo "The sum is $l";

[root@r8r3m2kvm tmp]# ./3.sh
enter a variable
1
You have entred 1
enter a variable
2
You have entred 2
The sum is 3
[root@r8r3m2kvm tmp]#
```

Define a variable using

var=10

vehicle=bus

Rules:

don’t put spaces a = 10 wrong

a=10 right

variables are case sensitive

variable can be printed using

echo $ var

Pipes

ls| grep hello

filters

bc-linux calculator

if- take 2numbers and compare

#!/bin/bash

#program to showif

echo “entera”

read a

echo “enterb”

read b

if [a==b]

then

print “aandbare equal”

else

print “a andb arenotequal”

fi

test:tocheck a conditionis true or not

if [test $1 –gt 0]

then

echo “$1 is +ve”

fi

if conditions:

-s file- empty

-f file exists

-d directory – r read

-w write

-x execute

```
if [ ]
then
do this
fi
```

```
if []
then
dothis
else
dothis
fi
```

```
if []
then
dothis
elif []
then
do this
else
dothis
fi
```

```
        if  []
        then
        dothis
        else
         if []
        then
        fi
        fi
        fi
```

```
        1)
if  [ -f file ]
then
echo “file exists”;
fi
```

```
if[  -f file ]
then
echo “fileexists”
else
echo”filedoes notexists”;
fi
```

```
$n=100
if [$n –eq 100  ]
then
echo “n is 100 “;
elif [ $n –gt 100 ]
then
 echo “nisgtthatn100”;
else
echo”countislessthan 100”;
fi
```

test-

```
#!/usr/bin/bash
echo "Enter file name";
read filename
if [ -f $filename ]
then
echo "$filename exists";
else
echo "$filename does not exists, creating file";
touch $filename;
      if [ $? -eq 0 ]
```

```
        then
        echo "file is created successfully";
        else
        echo "file is not created";
        fi
fi
~
```

When to use double equal to vs when to use -eq

When working on numbers its recommended to use clike syntax

```
#!/usr/bin/bash
echo "enter a";
read a;
echo "enter b";
read b
if (( $a >= $b ))
then
echo "a is greater"
else
echo "b is greater"
fi
```

```
[root@rscthydnet1 ~]# cat while.sh
#!/usr/bin/bash
```

```
# while loop
i=100;
while (( $i >= 0 ))
do
echo $i;
i=`expr $i - 1`;
done
[root@rscthydnet1 ~]#
```

When working on strings use -eq,

Compare 2 decimal numbers

```
#!/usr/bin/bash

echo "enter a";
read a;
echo "enter b";
read b
if [ "$a" == "$b" ]
then
echo "equal"
else
echo "not equal"
fi
```

for decimal comparisons c-like syntax or -eq does not work

While loop :

the below script prints numbers from 100 to 1

operators

-eq equal to

-gt greater than

-lt lesser than

-le less than or equal to

-ge greater or equal to

[root@r8r3m2kvm ~]# cat while.sh

#!/usr/bin/bash

i=100;

```
while [ $i -ge 0 ]
do
echo $i;
i=`expr $i - 1`;
done
```

the below script will print numbers from 1 to 10

```
#!/usr/bin/bash
i=0;
while [ $i -le 10 ]
do
echo $i;
i=`expr $i + 1`;
done
```

expr is the command to perform arithmetic operations

For loop:

Below script prints 1 2 3 4 5 on the screen

```
#!/usr/bin/bash
for i in 1 2 3 4 5
do
```

```
echo $i
done
```

```
#!/usr/bin/bash
for ((i=0; i <= 5; i++ ))
do
for ((j=0; j <=5; j++ ))
do
echo -n $i;
done
echo " ";
done
```

the above scripts o/p is:

```
[root@r8r3m2kvm ~]# ./for2.sh
000000
111111
222222
333333
444444
555555
```

Case statement:

```
[root@r8r3m2kvm ~]# cat case.sh
#!/usr/bin/bash
echo "Enter number"
read n;
case $n in
"1") echo "you have entered 1";;
"2") echo "you have entered 2";;
"3") echo "you have entered 3";;
"4") echo "you have entered 4";;
*) echo "you have entered something else";;
esac
[root@r8r3m2kvm ~]#
```

Case statement is to select one of the options

```
#!/usr/bin/bash
echo "Enter your vehicle type";
echo "Enter 1 for bus, 2 for car, 3 for bike, 4 for van";
read n;
```

```
case $n in
"1") echo "the fee for bus is 100";;
"2") echo "the fee for car is 50";;
"3") echo "the fee for bike is 20";;
"4") echo "the fee for van is 80";;
*) echo "you have entered something else, please choose the right option";;
esac
```

Fibonacci series:

```
#!/bin/bash
 echo "How many number of terms to be generated ?"
 read n
 x=0
 y=1
 i=2
 echo "Fibonacci Series up to $n terms :"
 echo "$x"
 echo "$y"
 while [ $i -lt $n ]
 do
    i=`expr $i + 1 `
    z=`expr $x + $y `
    echo "$z"
    x=$y
    y=$z
 done
```

~

```
#!/bin/bash
 echo "How many number of terms to be generated ?"
 read n
 x=0
 y=1
 i=2
 echo "Fibonacci Series up to $n terms :"
 echo "$x"
 echo "$y"
 while [ $i -lt $n ]
 do
    i=`expr $i + 1 `
    z=`expr $x + $y `
    echo "$z"
    x=$y
    y=$z
 done
```

Functions:

Saves lot of time.

Avoids rewriting of same code again and again

Program is easier to write.

Program maintains is very easy.

```
[root@r8r3m2kvm ~]# cat fun.sh
#!/usr/bin/bash
sayhello()
{
echo "Hello, I am inside the function";
}
sayhello;

[root@r8r3m2kvm ~]#
```

[root@r8r3m2kvm ~]# cat fun.sh

#!/usr/bin/bash

sayhello()

{

echo "Hello, I am inside the function";

}

sayhello;

The above function prints hello when executed.

function has 2 parts

declaration /definition

calling function

The below script prints words,

uses function output in a for loop

#!/usr/bin/bash

generate_list ()

{

```
  echo "one two three"
}
for word in $(generate_list)
do
  echo "$word"
done
```

the below is example for

function with arguments

function within a function

```
root@ctx2p02:~# ./1.sh
Hello abc
Hi
root@ctx2p02:~# cat 1.sh
#!/bin/bash
sayhello()
{
echo "Hello $1";
sayhi;
return 100;
}

sayhi()
{
echo "Hi";
}
```

sayhello abc;

function in command line

root@ctx2p02:~# . 1.sh

Hello abc

Hi

root@ctx2p02:~# sayhello

Hello

Hi

root@ctx2p02:~# sayhi

Hi

factorial of a number:

root@ctx2p02:~# cat 3.sh

```
#!/bin/bash
factorial()
{

if [ "$1" -gt "1" ]
then
i=`expr $1 - 1`;
j=`factorial $i`;
k=`expr $1 \* $j`;
echo $k;
else
echo 1
fi
}

echo "Enter a number";
read x;
factorial $x;
```

getopts:

to take inputs from command line

example:

```
#!/bin/bash
# Usage: ani -n -a -s -w -d
#
#
# help_ani() To print help
#
help_ani()
{
  echo "Usage: $0 -n -a -s -w -d"
  echo "Options: These are optional argument"
  echo " -n name of animal"
  echo " -a age of animal"
  echo " -s sex of animal "
  echo " -w weight of animal"
  echo " -d demo values (if any of the above options are used "
  echo " their values are not taken)"
  exit 1
}
#
```

```
isdef=0
na=Moti
age="2 Months"
sex=Male
weight=3Kg
#
#if no argument
#
if [ $# -lt 1 ]; then
  help_ani
fi
while getopts n:a:s:w:d opt
do
  case "$opt" in
    n) na="$OPTARG";;
    a) age="$OPTARG";;
    s) sex="$OPTARG";;
    w) weight="$OPTARG";;
    d) isdef=1;;
    \?) help_ani;;
  esac
done
if [ $isdef -eq 0 ]
then
```

```
  echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (user define mode)"

else

  na="Pluto Dog"

  age=3

  sex=Male

  weight=20kg

  echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (demo mode)"

fi
```

Regular expressions:

^ –Caret/Power symbol to match a starting at the beginning of line.

$ –To match end of the line

* –0 or more occurrence of previous character.

. –To match any single character

? – matches one or no chanracters (The preceding item is optional and will be matched, at most, once.)

[] –Range of character

[^char] –negate of occurrence of a character set

<word> –Actual word finding

–Escape character

```
root@ctx2p02:/var/tmp# ls -l | grep ^d

drwx------ 3 root root 4096 Oct  7 19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
```

drwx------ 3 root root 4096 Nov 20 11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# grep '^#' 1.sh

#!/bin/bash

root@ctx2p02:/var/tmp# ls -lrt | grep sh$

-rwxrwxrwx 1 root root 179 Nov 24 06:31 3.sh

-rwxrwxrwx 1 root root 277 Nov 24 07:07 fib.sh

-rwxrwxrwx 1 root root 103 Nov 24 07:19 1.sh

-rwxrwxrwx 1 root root 424 Nov 24 09:21 2.sh

-rwxrwxrwx 1 root root 171 Nov 24 20:45 7.sh

-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh

-rw-r--r-- 1 root root   82 Nov 24 21:59 8.sh

-rwxrwxrwx 1 root root 936 Nov 25 05:49 10.sh

-rwxrwxrwx 1 root root   73 Nov 25 08:42 until.sh

root@ctx2p02:/var/tmp#

finding empty lines in a file

root@ctx2p02:/var/tmp# grep '^$' *

1.sh:

1.sh:

2.sh:

2.sh:

2.sh:

2.sh:

2.sh:

3.sh:

3.sh:

3.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

8.sh:

fact.sh:

fact.sh:

fact.sh:

file:

grep: systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ: Is a directory

grep: systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex: Is a directory

grep -E 'word1|word2' filename

root@ctx2p02:/var/tmp# ls -lrt | grep 'syste*d*'

drwx------ 3 root root 4096 Oct 7 19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

drwx------ 3 root root 4096 Nov 20 11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

root@ctx2p02:/var/tmp# ls -lrt | grep 'un*l*'

-rwxrwxrwx 1 root root 73 Nov 25 08:42 until.sh

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# ls -lrt | grep '[1-9]'.sh

-rwxrwxrwx 1 root root 179 Nov 24 06:31 3.sh

-rwxrwxrwx 1 root root 103 Nov 24 07:19 1.sh

-rwxrwxrwx 1 root root 424 Nov 24 09:21 2.sh

-rwxrwxrwx 1 root root 171 Nov 24 20:45 7.sh

-rw-r--r-- 1 root root 82 Nov 24 21:59 8.sh

-rw-r--r-- 1 root root 0 Nov 28 05:21 9.sh

root@ctx2p02:/var/tmp#

[a-z] –Match's any single char between a to z.

[A-Z] –Match's any single char between A to Z.

[0-9] –Match's any single char between 0 to 9.

[a-zA-Z0-9] – Match's any single character either a to z or A to Z or 0 to 9

[!@#$%^] — Match's any ! or @ or # or $ or % or ^ character.

root@ctx2p02:/var/tmp# ls | grep '[abc]'

a

aa

aaa

fact.sh

fib.sh

systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

root@ctx2p02:/var/tmp#

[root@reviewb abc]# ls -lrt /etc/*release

-rw-r--r--. 1 root root 52 Sep 27 15:12 /etc/redhat-release

-rw-r--r--. 1 root root 495 Sep 27 15:12 /etc/os-release

lrwxrwxrwx. 1 root root 14 Nov 17 15:15 /etc/system-release -> redhat-release

[root@reviewb abc]# ls -lrt /etc/*release*

-rw-r--r--. 1 root root 45 Sep 27 15:12 /etc/system-release-cpe

-rw-r--r--. 1 root root 52 Sep 27 15:12 /etc/redhat-release

-rw-r--r--. 1 root root 495 Sep 27 15:12 /etc/os-release

lrwxrwxrwx. 1 root root 14 Nov 17 15:15 /etc/system-release -> redhat-release

/etc/lsb-release.d:

total 0

-rw-r--r--. 1 root root 0 Sep 3 2014 desktop-4.1-noarch

-rw-r--r--. 1 root root 0 Sep 3 2014 desktop-4.1-amd64

-rw-r--r--. 1 root root 0 Sep 3 2014 core-4.1-noarch

-rw-r--r--. 1 root root 0 Sep 3 2014 core-4.1-amd64

[root@reviewb abc]#

root@ctx2p02:/var/tmp# ls | grep '[^abc]'

10.sh

1.sh

2.sh

3.sh

7.sh

8.sh

9.sh

fact.sh

fib.sh

file

file90

systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

until.sh

root@ctx2p02:/var/tmp#

```
root@ctx2p02:/var/tmp# grep '\[' *
10.sh:if [ $# -lt 1 ]; then
10.sh:if [ $isdef -eq 0 ]
2.sh:if [ $? -eq 0 ]
3.sh:if [ "$1" -gt "1" ]
7.sh:while [ $k -lt $n ]
fact.sh:  if [ "$1" -gt "1" ]; then
fib.sh:  while [ $i -lt $n ]
grep: systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ: Is a directory
grep: systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex: Is a directory
until.sh:until [ $a -lt 10 ]
root@ctx2p02:/var/tmp#
```

{n} –n occurrence of previous character

{n,m} – n to m times occurrence of previous character

{m, } –m or more occurrence of previous character.

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'a{2}'
-rw-r--r-- 1 root root    0 Nov 28 05:28 aa
```

```
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'a{1,3}'
total 56
-rw-r--r-- 1 root root    8 Nov 28 05:24 a
-rw-r--r-- 1 root root    0 Nov 28 05:28 aa
-rw-r--r-- 1 root root    0 Nov 28 05:29 aaa
-rwxrwxrwx 1 root root  206 Nov 24 21:58 fact.sh
drwx------ 3 root root 4096 Oct  7 19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
drwx------ 3 root root 4096 Nov 20 11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'u+'
-rwxrwxrwx 1 root root   73 Nov 25 08:42 until.sh
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'a|b'
total 56
-rw-r--r-- 1 root root    8 Nov 28 05:24 a
-rw-r--r-- 1 root root    0 Nov 28 05:28 aa
```

-rw-r--r-- 1 root root   0 Nov 28 05:29 aaa

-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh

-rwxrwxrwx 1 root root 277 Nov 24 07:07 fib.sh

drwx------ 3 root root 4096 Oct 7 19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

drwx------ 3 root root 4096 Nov 20 11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

root@ctx2p02:/var/tmp#

AWK

root@ctx2p02:/var/tmp# ls -lrt | awk '{print $9}'

3.sh

fib.sh

1.sh

2.sh

7.sh

fact.sh

8.sh

10.sh

until.sh

file90

9.sh

a

file

aa

aaa

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# ls -lrt | awk '{print $8 "\t" $9}'

19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

06:31 3.sh

07:07 fib.sh

07:19 1.sh

09:21 2.sh

20:45 7.sh

21:58 fact.sh

21:59 8.sh

05:49 10.sh

08:42 until.sh

09:14 file90

05:21 9.sh

05:24 a

05:27 file

05:28 aa

05:29 aaa

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# ls -lrt | awk '{print $8 "," $9}'

,

19:42,systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

11:15,systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

06:31,3.sh

07:07,fib.sh

07:19,1.sh

09:21,2.sh

20:45,7.sh

21:58,fact.sh

21:59,8.sh

05:49,10.sh

08:42,until.sh

09:14,file90

05:21,9.sh

05:24,a

05:27,file

05:28,aa

05:29,aaa

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# ls -lrt| awk '/a/ {print $0}'

total 56

drwx------ 3 root root 4096 Oct 7 19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

drwx------ 3 root root 4096 Nov 20 11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex

-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh

-rw-r--r-- 1 root root   8 Nov 28 05:24 a

-rw-r--r-- 1 root root   0 Nov 28 05:28 aa

-rw-r--r-- 1 root root   0 Nov 28 05:29 aaa

-rw-r--r-- 1 root root   0 Nov 28 05:47 ba

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# ls -lrt | awk '/a/{++cnt} END {print "Count = ", cnt}'

Count = 8

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# ls -lrt | awk 'FNR == 2 {print}'

drwx------ 3 root root 4096 Oct 7 19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ

root@ctx2p02:/var/tmp# ls -lrt | grep ^- | awk 'FNR == 5 {print $9}'

7.sh

root@ctx2p02:/var/tmp#root@ctx2p02:/var/tmp#

to print the j'th field of the i'th line

awk -v i=5 -v j=3 'FNR == i {print $j}'

root@ctx2p02:/var/tmp# ls -lrt | grep ^- | awk -v i=5 -v j=9 'FNR == i{print $j}'

7.sh

root@ctx2p02:/var/tmp#

Syntax:

BEGIN { …. initialization awk commands …}

{ …. awk commands for each line of the file…} END

{ …. finalization awk commands …}

root@ctx2p02:/var/tmp# ls -l | awk 'BEGIN {sum=0} {sum=sum+$5} END {print sum}'

10693

root@ctx2p02:/var/tmp#

SED

By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace the second, third...occurrence in the line.

sed 's/unix/linux/' file.txt → replaces word unix with linux

sed 's/unix/linux/2' file.txt → replaces 2nd occurance of word

to replace all the occurrences of file

sed 's/unix/linux/g' file.txt

Replacing from nth occurrence to all occurrences in a line.

sed 's/unix/linux/3g' file.txt - The following sed command replaces the third, fourth, fifth... "unix" word with "linux" word in a line.

There might be cases where you want to search for the pattern and replace that pattern by adding some extra characters to it. In such cases & comes in handy. The & represents the matched string.

sed 's/unix/{&}/' file.txt

root@ctx2p02:/var/tmp# cat file.txt | sed 's/unix/{&}/g '

{unix} {unix}

linux

{unix}

root@ctx2p02:/var/tmp#

The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.

root@ctx2p02:/var/tmp# sed 's/unix/linux/p' file.txt

linux unix

linux unix

linux

linux

linux

root@ctx2p02:/var/tmp# cat file.txt

unix unix

linux

unix

How to replace a specific word only using sed

[root@reviewb ~]# cat /file

abc 123 1234

xyz

XYZ

abc

ABC

Def

Replace only 123 and not 1234

[root@reviewb ~]# ***cat /file | sed 's/\<123\>/456/g'***

abc 456 1234

xyz

XYZ

abc

ABC

def

Replacing string on a specific line number.

You can restrict the sed command to replace the string on a specific line number. An example is

root@ctx2p02:/var/tmp# sed '3 s/unix/linux/' file.txt

unix unix

linux

linux

root@ctx2p02:/var/tmp# sed '1,3 s/unix/linux/' file.txt

linux unix

linux

linux

root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# sed '2,$ s/unix/linux/' file.txt

unix unix

linux

linux

Here $ indicates the last line in the file. So the sed command replaces the text from second line to last line in the file.

sed '2 d' file.txt

deletes the 2nd line from file

sed '5,$ d' file.txt

Sed as grep command

sed command can be used as grep

>grep 'unix' file.txt

>sed -n '/unix/ p' file.txt

Here the sed command looks for the pattern "unix" in each line of a file and prints those lines that has the pattern.

You can also make the sed command to work as grep -v, just by using the reversing the sed with NOT (!).

>grep -v 'unix' file.txt

>sed -n '/unix/ !p' file.txt

after:

root@ctx2p02:/var/tmp# sed '/unix/ a "Add a new line"' file.txt

unix unix

"Add a new line"

linux

unix

"Add a new line"

before :

root@ctx2p02:/var/tmp# sed '/unix/ i "Add a new line"' file.txt

"Add a new line"

unix unix

linux

"Add a new line"

unix

lower case to upper case

sed 'y/ul/UL/' file.txt

it can be used to replace an entire line with a new line. The "c" command to sed tells it to change the line.

>sed '/unix/ c "Change line"' file.txt

root@ctx2p02:/var/tmp# sed '/unix/ c "Change line"' file.txt

"Change line"

linux

"Change line"

to make the change permanent

root@ctx2p02:/var/tmp# sed -i '/unix/ c "Change line"' file.txt

root@ctx2p02:/var/tmp# cat file.txt

"Change line"

linux

"Change line"

root@ctx2p02:/var/tmp#

Interview Questions: