LINUX NOTES FOR DEVOPS NEWBIES

Crafted by Parmeshwar Gudadhe

Absolute and Relative Paths

Absolute Path:

We start the path from the root directory "/"

```
[param@mac ~]$ cd /home/param/notes/java
[param@mac java]$
```

Relative Path (Short Path):

We do not start the path from the root directory "/"

```
[param@mac notes]$ cd java/
[param@mac java]$
```

Command -Options and Arguments

```
[param@mac ~]$ ls -l /var
```

Where,
ls = command
-l = option/s
/var = arguments

Linux “ls” Command

Introduction:

ls command is one of the most frequently used commands in Linux. I believe **ls** command is the first command you may use when you get into the command prompt of Linux Box**.**

We use **ls** commands daily and frequently even though we may not be aware and never use all the **ls** options available. In this article, we’ll be discussing basic **ls** commands where we have tried to cover as many parameters as possible**.**

Let us see a high-level view of some of the new features that are introduced in RHEL 8.

ls Command with options**.**

1. **List files using ls with no option**

ls with no option list files and directories in a bare format where we won’t be able to view details like file types, size, modified date and time, permission and links etc**.**

```
ls
```

2. **List Files with option -l**

Here, **ls -l** (-l is a character not one) shows file or directory, size, modified date and time, file or folder name and owner of the file and its permission.

```
ls -l
```

3. **View Hidden Files**

List all files including hidden files starting with “ls -a”.

```
ls -a
```

4. **List Files with Human Readable Format with option -lh**

With a combination of **-lh** option, it shows sizes in a human-readable format.

```
ls -lh
```

5. **List Files and Directories with ‘/’ Character at the end**

Using **-F** option with **ls** command will add the ‘/’ Character at the end of each directory.

```
ls -F
```

6. **List Files in Reverse Order**

The following command with **ls -r** option displays files and directories in reverse order.

```
ls -r
```

7. **Recursively list Subdirectories**

ls -R option will list very long listing directory trees. See an example of the output of the command.

```
ls -R
```

8. **Reverse Output Order**

With the combination of **-ltr** it will show the latest modification file or directory date as last.

```
ls -ltr
ls -ltra
```

9. Sort Files by File Size

With the combination of **-lS** displays file size in order, it will display big at first.

```
ls -lS
ls -lSa
```

10. The display Inode number of File or Directory

We can see some numbers printed before the file/directory name. With **-i** options list file/directory with an inode number.

```
ls -ia
```

11. Shows version of ls command

Check the version of ls command

```
ls –version
```

12. Show Help Page

List the help page of ls command with their option.

13. List Directory Information

With **ls -l** command list files under directory **/tmp.** Wherein with **-ld** parameters display information of **/tmp** directory.

```
ls -l /tmp
ls -ld /tmp
```

14. Display UID and GID of Files

To display **UID** and **GID** of files and directories. Use option **-n** with **ls** command

```
ls -n
```

15. Some Extra ls commands

```
ls -l dir1/ <long listing format>
ls -l -a <long listing format>
ls -l -all <long natation>
ls -l -s <allocated size of each file>
ls --size <allocated size of each file using long notation>
ls -lsa <also  print the size of the file in long listing format>
ls  -lsh <size of the files in human readable format>
ls --human-readable -s <long notation>
```

```
ls -lr <reverse the order of sorting>
ls -lX <search based on alphabets>
ls -l -R <list sub directories recursively>
ls -l -t <search by modification time>
ls -l -x <list by lines>
ls -u -l -t <list by last access time shows newest first>
```

Linux “cd” Command

Introduction:

In linux **‘cd’ (Change Directory)** command is one of the most important and most widely used commands for newbies as well as system administrators. For admins on a headless server, ‘cd’ is the only way to navigate to a directory to check the log, execute a program/application/script and for every other task. For the newbie it is among those initial commands they make their hands dirty with.

cd command with options

1. Change from the current directory to /usr/local.

```
cd /usr/local
```

2. Change from the current directory to /usr/local/lib using an absolute path.

```
cd /usr/local/lib
pwd
```

3. Change from the current working directory to /usr/local/lib using a relative path.

```
cd ..
cd lib
pwd
```

4. Switch back to the previous directory where you worked earlier.

```
cd -
```

5. Change Current directory to parent directory

```
cd ..
pwd
```

6. Move two directories up from where you are now.

```
cd ../../
```

7. Move to the user's home directory from anywhere

```
cd ~ OR
cd - OR
cd
```

8. Change the working directory to the current working directory (seems no use in General).

```
cd .
pwd
```

9. Change from the current working directory to /home/student/music/ without typing in full using TAB.

```
cd /h<TAB_key>/s<TAB_key>/m<TAB_key>.
pwd
```

Linux "date" Command

Introduction:

Date command is helpful to display the date in several formats. It also allows you to set systems date and time.

This article explains a few examples of how to use date commands with practical examples.

When you execute the date command without any option, it will display the current date and time as shown below.

```
date
```

Date command with options

1. **Display Date from a String Value using -date Option**

If you have a static date or time value in a string, you can use -d or -date option to convert the input string into date format as shown below.

Please note that this doesn't use the current date and time value. Instead it uses the date and time value that you pass as a string.

The following examples take an input date only string and display the output in date format. If you don't specify the time, it uses 00:00:00 for time.

The following example takes an input date and time string and displays the output in the date format.

```
date --date="22/02/2003"
date --date="Feb 22 2003 10:10:10"
```

2. **Read Date Patterns from a file using -file option**

This is similar to the -d or -date option that we discussed above. But, you can do it for multiple date strings. If you have a file that contains various static date strings, you can use -f or -file option as shown below.

In this example, we can see that datefile contained 2 date strings. Each line of datefile is parsed by date command and date is outputted for each time.

```
echo 12/2/2020 >> datefile
echo Feb 7 2020 10:10:10 >> datefile
date –file=datefile
```

3. **Get Relative Date Using -date option**

For example, the following examples get the date of next Monday.

```
date
date --date="next mon"
```

It displays the date in which 5 seconds are elapsed since epoch 1970-01-01 UTC:

```
date --date=@5
```

It displays the date in which 10 seconds are elapsed since epoch 1970-01-01 UTC:

```
date --date=@10
```

It displays the date in which 1 minute (i.e 60 seconds) is elapsed since epoch 1970-01-01 UTC:

```
date --date=@60
```

4. **Display Past Date**

You can display a past date using the -date command. Few possibilities are shown below.

```
date
date --date='30 seconds ago'
date --date='1 day ago'
date --date='yesterday'
date --date='1 month ago'
date --date='2 years ago'
```

5. **Set Date and Time using -s or -set option**

You can set the date and time of your system using the **-s** or **-set** option as shown below.

In this example, initially, it displayed the time as 20:09:31. We then used the date command to change it to 21:00:00.

```
date
sudo date -s 'Mon May 1 00:00:00 EDT 2020'
date
```

6. **Display Universal Time using -u option**

You can display the date in UTC format using -u, or -utc, or -universal option as shown below.

```
date -u
```

7. **Display Last Modification Time using -r option**

```
date
touch datefile
ls -l datefile
date -r datefile
```

Various Date Command Formats

$ date +%<format-option>

Format options	Purpose of Option	Output
date +%a	Display Weekly name in short (like Mon, Tue, Wed)	Thu
date +%A	Display Weekly name in full short (like Monday, Tuesday)	Thursday
date +%b	Displays Month name in short (like Jan, Feb, Mar)	Feb
date +%B	Displays Month name in full short (like January, February)	February
date +%d	Display Day of the month (e.g. 01)	01
date +%D	Display Current Date; shown in MM/DD/YY	22/2/22

date +%F	Display Date; shown in YYYY-MM-DD	2022-01-22
date +%H	Display hour in (23) format	23
date +%I	Display hour in (01..12) format	11
date +%j	Display day of the year (001..366) 038	038
date +%m	Display month (01..12)	02
date +%M	Display minute (00..59)	44
date +%S	Display second (00..60)	17
date +%N	Display nanoseconds (0000000..999999)	573587606
date +%T	Display time; shown as HH:MM:SS	23:44:17
date +%u	Display day of the week (1..7); 1 is Monday	4
date +%U	Display week number of year, with Sunday as the first day of the week (00..53)	05
date +%Y	Display full year i.e. YYYY	2022
date +%Z	Alphabetic time zone abbreviation (e.g. EDT)	IST

Linux “cat” Command

Introduction:

The cat (short for “concatenate”) command is one of the most frequently used commands in Linux/Unix like operating systems. **cat** command allows us to create single or multiple files, view the contents of the file, concatenate files and redirect output in terminal or files. In this article, we are going to find out the handy use of **cat** commands with their example in Linux.

General Syntax

```
cat [OPTION] [FILE]…
```

cat command with options

1. Display Contents of File

In the below example, it will show contents of **/etc/passwd** file.

```
cat /etc/passwd
```

2. View Contents of Multiple Files in terminal

In the below example, it will display the contents of the **test** and **test1** file in the terminal.

```
echo this is test file > test
echo this is test1 file > test1
```

3. Create a File with Cat Command

We will create a file called **test2** file with the below command.

```
cat > test2
```

4. Use Cat Command with More & Less Options

If a file has a large amount of content that won't fit in the output terminal and the screen scrolls up very fast, we can use parameters more and less with **cat** commands as shown above.

```
cat /etc/passwd | more
cat /etc/passwd | less
```

5. Display Line Number in File

With **-n** option you could see the line numbers of a file /etc/passwd in the output terminal.

```
cat -n /etc/passwd
```

6. Display Multiple Files at Once

In the below example we have three files **test1**, **test2** and **test3** and are able to view the contents of those files as shown below. We need to separate each file with ; (semicolon).

```
cat test1 test2 test3
```

7. Use Standard Output with Redirection Operator

We can redirect the standard output of a file into a new file with an existing file with '>' (greater than) symbol. Careful, existing contents of **test1** will be overwritten by the content of the **test2** file.

```
cat test1 > test2
```

8. Appending Standard Output with Redirection Operator

Appends in the existing file with '**>>**' (double greater than) symbol. Here the contents of the **test1** file will be appended at the end of the **test2** file.

```
cat test1 >> test2
```

9. Redirecting Standard Input with Redirection Operator.

When you use the redirect with standard input '**<**' (less than) symbol. It uses file name test2 as input for command and output will be shown in a terminal.

```
cat < test2
```

10. Sorting Contents of Multiple FIles in a Single File.

This will create a file **test4** and the output of **cat** command is piped to sort and the result will be redirected in a newly created file.

```
cat test1 test2 test3 | sort > test4
```

Linux “touch” Command

Introduction:

In **Linux**, every single file is associated with timestamps and every file stores the information of last access time, last modification time and last change time. So, whenever we create a new file, access or modify an existing file, the timestamps of that file automatically update.

In this document, we will cover some useful practical examples of the Linux **touch command**. The **touch command** is a standard program for **Unix/Linux** operating systems that is used to create, change and modify timestamps of a file. Before heading up for touch command examples, please check out the following options.

Touch Command Options

- **-a**, change the access time only.
- **-c**, if the file does not exist, do not create it.
- **-d**, update the access and modification times.
- **-m**, change the modification time only.
- **-r**, use the access and modification times of the file.
- **-t**, creates a file using a specified time.

touch command with options

1. How to Create an Empty File

The following touch command creates an empty (zero-byte) new file called **Param**.

```
touch Param
```

2. How to Create Multiple Files

By using touch command, you can also create more than one single file. For example, the following command will create 3 files named, file1, file2 and file3.

```
touch file1 file2 file3
```

3. How to Avoid Creating New File

Using -c option with touch command avoids creating new files. For example, the following command will not create a file called **file4** if it doesn't exist.

```
touch -c new_file4
```

4. How to Change File Modification Time

If you like to change the only modification time of a file called **file**, then use the -m option with touch command. Please note it will only update the last modification times (not the access times) of the file.

```
ls -l file
touch -m file
ls -l file
```

5. Explicitly Set the Access and Modification times

You can explicitly set the time using the **-c** and **-t** option with touch command. The format would be as follows.

```
ls -l file
touch -c -t 05050808 file
```

6. How to Use the Timestamp of another File.

The following touch command with **-r** option, will update the timestamp of file **file1** with the timestamp of **file2** file. So, both the files hold the same time stamp.

```
ls -l file1 file2
touch -r file1 file2
```

7. Create a File using a specified time

If you would like to create a file with a specified time other than the current time, then the format should be.

```
touch -t 05050808 file
ls -l file
```

Linux “mkdir” Command

Introduction:

The **mkdir** command in UNIX allows users to create directories or folders as they are referred to in some operating systems. The mkdir command can create multiple directories at once and also set permissions when creating the directory. The user running the command must have appropriate permissions on the parent directory to create a directory or will receive a permission denied error.

mkdir command with options

1. How to create a directory

To create a directory in UNIX or Linux using the mkdir command, pass the name of the directory to the mkdir command.

```
mkdir mydirectory
```

2. How to create multiple directories

To create multiple directories in UNIX or Linux using the mkdir command pass the names of directories to be created to the mkdir command. The names of directories should be separated by spaces.

```
mkdir foo bar baz
```

3. How to create parent directories

To create parent directories using the mkdir command pass the -p option. Suppose that the directory path foo/bar/baz to be created. This can be created with mkdir as follows.

```
cd foo/
mkdir bar
cd bar/
mkdir baz
cd ../..
tree foo/
```

This may also be achieved in a single command with the -p flag.

```
mkdir -p foo1/bar1/baz1
tree foo1
```

4. How to set permissions when creating a directory

To set permissions when creating a directory pass the -m option. This accepts a number value to set the file mode. If no options are passed to mkdir the directory will be created with reading, write and execute permissions for the user (755). In the following example, the directory is created to be world-readable.

```
mkdir -m 777 foo2
ls -ld foo2
```

Linux “head and tail” Command

head Command

The head command reads the first ten lines of any given file name. The basic syntax of the head command is:

```
head [options] [file(s)]
```

1. A Basic Example of head Command

For example, the following command will display the first ten lines of the file named '**/etc/passwd**'

```
head /etc/passwd
```

2. Read multiple files using head command

If more than one file is given, the head will show the first ten lines of each file separately. For example, the following command will show ten lines of each line.

```
head /etc/passwd /etc/yum.repos.d/amzn2-extras.repo
```

3. Read more or fewer lines than 10.

If it is desired to retrieve more lines than the default ten, then **-n** option is used along with an integer telling the number of lines to be retrieved. For example, the following command will display the first 5 lines from the file '/etc/passwd' file.

```
head -n5 /etc/passwd
```

In fact, there is no need to use the **-n** option. Just the hyphen and specify the integer without spaces to get the same result as the above command.

```
head -5 /etc/passwd
```

4. Read the desired number of bytes using **'-c**'

The head command can also display any desired number of bytes using the '**-c**' option followed by the number of bytes to be displayed. For example, the following command will display the first 32 bytes of the given file.

```
head -c32 /etc/passwd
```

tail Command

The tail command allows you to display the last ten lines of any text file. Similar to the head command above, the tail command also supports options 'n' number of lines and 'n' number of characters.

The basic syntax of the tail command is:

```
tail [options] [filenames]
```

1. A Basic Example of tail command

For example, the following command will print the last ten lines of a lines called '**/etc/passwd**'

```
tail /etc/passwd
```

2. Read multiple files using tail command

If more than one file is provided, the tail will print the last ten lines of each file as shown below.

```
tail /etc/passwd /etc/yum.repos.d/amzn2-extras.repo
```

3. Read more or fewer lines than 10.

Similarly, you can also print the last few lines using the '**-n**' option as shown below.

```
tail -n5 /etc/passwd
```

4. Read the desired number of bytes using '-c'

You can also print the number of characters using the '-c' argument as shown below.

```
tail -c23 /etc/passwd
```

Linux "wc" Command

The wc (word count) command in Unix/Linux operating systems is used to find out the number of newline count, word count, byte and characters count in a file specified by the file arguments. The syntax of wc command as shown below.

```
wc [options] filenames
```

The following are the options and usage provided by the command.

wc -l: Prints the number of lines in a file.
wc -w: Prints the number of words in a file.
wc -c: Displays the count of bytes in a file.
wc -m: Prints the count of characters from a file.
wc -L: Prints only the length of the longest line in a file.

So, let's see how we can use the 'wc' command with their few available arguments and examples in this article. We have used the 'passwd' file for testing the commands. Let's find out the output of the file using cat command as shown below.

```
tail -10 /etc/passwd > passwd
cat > passwd
```

1. A Basic example of WC command

The ‘wc’ command without passing any parameter will display a basic result of the ‘passwd’ file. The three numbers shown below are 10 (number of lines), 21 (number of words), and 573 (number of bytes) of the file.

```
wc passwd
```

2. Count Number of Lines

To count the number of newlines in a file use the option ‘**-l**’, which prints the number of lines from a given file. Say, the following command will display the count of newlines in a file. In the output, the first field is assigned as count and the second field is the name of the file.

```
wc -l passwd
```

3. Display Number of Words

Using ‘**-w**’ argument with ‘wc’ command prints the number of words in a file. Type the following command to count the words in a file.

```
wc -w passwd
```

4. Count Number of Bytes and Characters

When using options ‘**-c**’ and ‘**-m**’ with ‘**wc**’ command will print the total number of bytes and **characters** respectively in a file.

```
wc -c passwd
wc -m passwd
```

5. Display Length of Longest Line

The ‘**wc**’ command allows an argument ‘**-L**’, it can be used to print out the length of the longest (number of characters) line in a file. So, we have the longest character line (‘Scientific Linux’) in a file.

```
wc -L passwd
```

6. Check More WC Options

For more information and help on the wc command, simply run the ‘wc -help’ and ‘man wc’ from the command line.

```
wc --help
```

Linux “File” Command

What is the file command in UNIX?

The file command determines the file type of a file. It reports the file type in human-readable format (e.g. ‘ASCII text’) or MIME type (e.g. ‘text/plain; charset=us-ascii’). As filenames in UNIX can be entirely independent of file type, the file can be a useful command to determine how to view or work with a file.

How to determine the file type of a file

To determine the file type of a file, pass the name of a file to the file command. The file name along with the file type will be printed to standard output.

```
file /etc/passwd
```

To show just the file type pass the -b option.

```
file -b /etc/passwd
```

How to determine the file type of multiple files

The file command can also operate on multiple files and will output a separate line to standard output for each file.

```
file /etc/passwd /etc/yum.repos.d/amzn2-extras.repo
```

How to view the mime type of a file.

To view the mine type of a file rather than the human-readable format pass the -i option.

```
file -i /etcpasswd/
```

This can be combined with the -b option to just show the mime type.

```
file -i -b /etcpasswd/
```

```
file -i -b /dev/tty
```

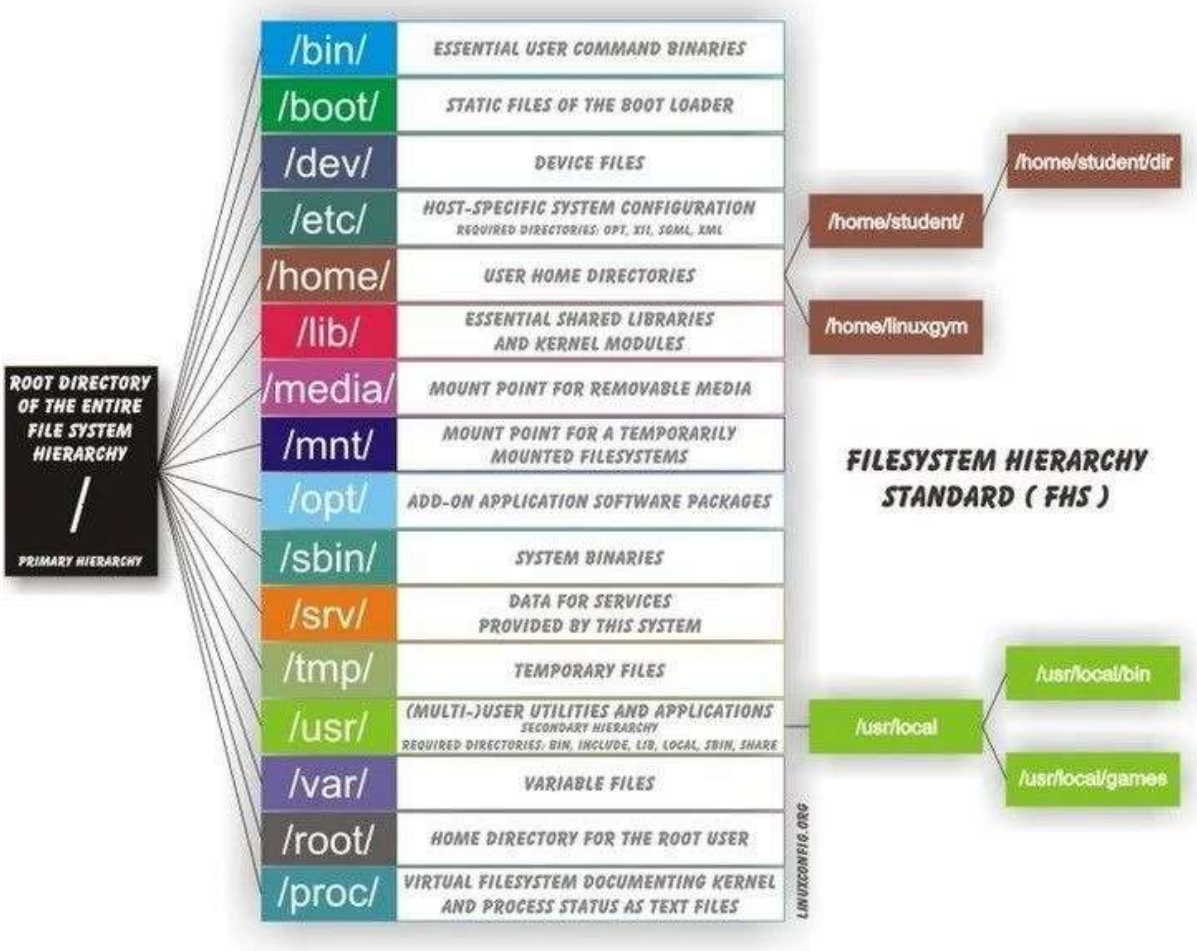
Linux File System Hierarchy

For any person, who does not have a sound knowledge of Linux Operating System and Linux File System, dealing with the files and their location, their use may be horrible and a newbie may really mess up.

This article is aimed to provide information about the Linux File System, some of the important files, their usability and location.

Linux Directory Structure Diagram

A standard Linux distribution follows the directory structure as provided below with Diagram and Explanation.



Each of the above directories (which is a file, in the first place) contains important information, required for booting to device drivers, configuration files etc. Describing briefly the purpose of each directory, we are starting hierarchically.

1. **/bin**: All the executable binary programs (file) required during booting, repairing, files required to run into single-user-mode and other important, basic commands viz, cat, du, df, tar, rpm, wc, history, etc.

2. **/boot**: Holds important files during the boot-up process, including **Linux-Kernel**.
3. **/dev**: Contains device files for all the hardware devices on the machine e.g**. cdrom**, **cpu**, etc**.**
4. **/etc**: Contains Application's configuration files, **startup, shutdown, start, stop** script for every individual program**.**
5. **/home**: Home directory of the users**.** Every time a new user is created, a directory in the name of the user is created within the home directory which contains other directories like **Desktop**, **Downloads**, **Documents**, etc**.**
6. **/lib**: The Lib directory contains **kernel modules** and **shared library** images required to boot the system and run commands in the root file system**.**
7. **/lost+found**: This Directory is installed during installation of **Linux**, useful for recovering files that may be broken due to unexpected shut-down**.**
8. **/media**: Temporary mount directory is created for removable devices viz**.**, **media/cdrom.**
9. **/mnt**: Temporary mount directory for mounting file system**.**
10. **/opt**: Optional is abbreviated as opt. It contains third party application software**.** Viz**.**, Java, etc**.**
11. **/proc**: A virtual and pseudo-file-system which contains information about the running process with a particular **Process-id** aka **PID.**
12. **/root**: This is the home directory of root user and should never be confused with '/'
13. **/run**: This directory is the only clean solution for **the early-runtime-dir** problem**.**
14. **/sbin**: Contains binary executable programs, required by **System Administrator,** for **Maintenance.** Viz**.**, iptables, fdisk, ifconfig, swapon, reboot, etc**.**
15. **/srv**: Service is abbreviated as '**srv**'**.** This directory contains server-specific and service-related files**.**
16. **/sys**: Modern Linux distributions include a **/sys** directory as a **virtual filesystem**, which stores and allows modification of the devices connected to the system**.**
17. **/tmp**: System's Temporary Directory, Accessible by users and root**.** Stores temporary files for **user** and **system**, till next boot**.**
18. **/usr**: Contains executable **binaries, documentation, source code, libraries** for the second-level program**.**
19. **/var**: Stands for the variable**.** The contents of this file are expected to grow**.** This directory contains **log**, **lock**, **spool**, **mail** and **temp** files**.**

Exploring Important file, their location and their Usability

Linux is a complex system that requires a more complex and efficient way to **start**, **stop**, **maintain** and **reboot** a system unlike **Windows.** There is a well-defined configuration **file**, **binaries**, **man pages**, **info files**, etc**.** for every **process** in **Linux.**

1. **/boot/vmlinuz**: The Linux Kernel file**.**
2. **/dev/hda**: Device file for the first **IDE HDD (Hard Disk Drive)**
3. **/dev/hdc**: Device file for the **IDE CDROM**, commonly**.**
4. **/dev/null**: A pseudo-device that doesn't exist**.** Sometimes garbage output is redirected to **/dev/null**, so that it gets lost, forever**.**
5. **/etc/bashrc**: Contains system **defaults** and **aliases** used by bash shell.
6. **/etc/crontab**: A shell script to run specific commands on a predefined time Interval.
7. **/etc/exports**: Information on the file system available on **the network.**

8. **/etc/fstab**: Information of **Disk Drive** and their mount point.
9. **/etc/group**: Information of **Security Group**.
10. **/etc/grub.conf**: grub **bootloader** configuration file.
11. **/etc/init.d**: Service **startup** Script.
12. **/etc/lilo.conf**: lilo **bootloader** configuration file.
13. **/etc/hosts**: Information of **IP addresses** and corresponding **hostnames**.
14. **/etc/hosts.allow**: List of **hosts allowed** to access services on the local machine.
15. **/etc/host.deny**: List of **hosts denied** to access services on the local machine.
16. **/etc/inittab**: INIT process and their interaction at the various **run levels**.
17. **/etc/issue**: Allows editing the **pre-login** messages.
18. **/etc/modules.conf**: Configuration files for **system modules**.
19. **/etc/motd**: motd stands for **Message Of The Day**, The Message users get upon login.
20. **/etc/mtab**: Currently mounted **blocks** information.
21. **/etc/passwd**: Contains **password** of system **users** in a shadow file, a security implementation.
22. **/etc/printcap**: **Printer** Information.
23. **/etc/profile**: Bash shell **defaults**.
24. **/etc/profile.d**: Application script, executed after **login**.
25. **/etc/rc.d**: Information about **run level** specific script.
26. **/etc/rc.d/init.d**: Run level **Initialization** Script.
27. **/etc/resolv.conf**: Domain Name Servers (**DNS**) being used by the system.
28. **/etc/securetty**: Terminal list, where **root** login is possible.
29. **/etc/skel**: Script that populates new user **home** directory.
30. **/etc/termcap**: An ASCII file that defines the behaviour of **Terminal**, **console** and **printers**.
31. **/etc/X11**: Configuration files of **X-Windows** system.
32. **/usr/bin**: Normal user **executable** commands.
33. **/usr/bin/X11**: Binaries of **X-Windows** System.
34. **/usr/include**: Contains include files used by '**c**' program.
35. **/usr/share**: Shared directories of **man files**, **info files**, etc.
36. **/usr/lib**: Library files that are required during program **compilation**.
37. **/usr/sbin**: Commands for **Super User**, for System Administration.
38. **/proc/cpuinfo**: **CPU** Information.
39. **/proc/filesystems**: File system **Information** being used currently.
40. **/proc/interrupts**: Information about the current **interrupts** being utilised currently.
41. **/proc/ioports**: Contains all the **Input/Output** addresses used by devices on the server.
42. **/proc/meminfo**: **Memory Usages** Information.
43. **/proc/modules**: Currently using the Kernel module.
44. **/proc/mount**: Mounted **File system** Information.
45. **/proc/stat**: Detailed **Statistics** of the current system.
46. **/proc/swaps**: **Swaps** File Information.
47. **/version**: Linux **Version** Information.
48. **/var/log/lastlog**: log of last **boot** process.
49. **/var/log/messages**: log of messages produced by **Syslog** daemon at boot.
50. **/var/log/wtmp**: list login **time** and **duration** of each user on the system currently.

Absolute and Relative Pathnames in UNIX

A path is a unique location to a file or a folder in a file system of an OS. A path to a file is a combination of / and alpha-numeric characters.

Absolute Path-name

An absolute path is defined as specifying the location of a file or directory from the root directory(/).

To write an absolute path-name:

Start at the root directory (/) and work down.
Write a slash (/) after every directory name (last one is optional)

For example:

```
cat abc.sql
```

Will work only if the file “abc.sql” exists in your current directory. However, if the file is not present in your working directory and is present somewhere else say in /home/kt, then this command will work only if you will use it like shown below:

```
cat /home/kt/abc.sql
```

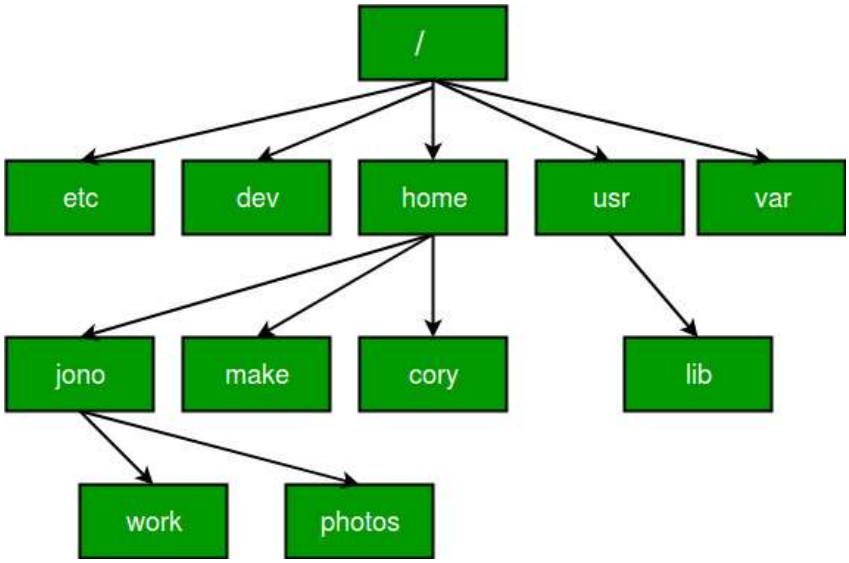
In the above example, if the first character of a pathname is /, the file’s location must be determined with respect to root. When you have more than one / in a pathname, for each such /, you have to descend one level in the file system like in the above kt is one level below the home, and thus two levels below the root.

An absolute path is defined as specifying the location of a file or directory from the root directory /. In other words, we can say that an absolute path is a complete path from the start of the actual file system from / directory.

Relative Path

The relative path is defined as the path related to the present working directory (pwd). It starts at your current directory and **never starts with a** /.

To be more specific let’s take a look on the below figure in which if we are looking for photos then the absolute path for it will be provided as /home/jono/photos **but assuming that we are already present in** *jono* **directory then the relative path for the same can be written as simple** *photos*.



Using . and .. in Relative Pathnames

Linux offers a shortcut in the relative pathname that uses either the current or parent directory as a reference and specifies the path relative to it. A relative pathname uses one of these cryptic symbols.

. (a single dot) represents the current directory.
.. (two dots) represents the parent directory.

Now, what this actually means is that if we are currently in directory /home/kt/abc and now you can use .. as an argument to cd to move to the parent directory /home/kt as:

```
pwd
/home/kr/abc
cd.. **move one level up**
pwd
/home/kt
```

NOTE: Now / when used with .. has a different meaning, instead of moving down a level, it moves one level up.

```
pwd
/home/kr/abc
cd../.. **move two level up**
pwd
/home
```

Example of Absolute and Relative Path

Suppose you are currently located in home/kt and you want to change your directory to home/kt/abc. Lets see both the absolute and relative path concepts to do this:

Changing directory with relative path concept:

```
pwd
/home/kt
cd abc
pwd
/home/kt/abc
```

Managing Files using Command line tool:

Activity	**Single Source**	**Multiple Source**
Copy file	cp file1 file2	cp file1 file2 file3 dir (5)
Move file	mv file1 file2 (1)	mv file1 file2 file3 dir (4)
Remove file	rm file1	rm -f file1 file2 file3 (5)
Create directory	mkdir dir	mkdir -p par1/par2/dir (6)
Copy directory	cp -r dir1 dir2 (2)	cp -r dir1 dir2 dir3 dir4 (4)
Move directory	mv dir1 dir2 (3)	mv dir1 dir2 dir3 dir4 (4)
Remove directory	rm -r dir1	rm -rf dir1 dir2 dir3 (5)

Note:

(1) The result is a rename.
(2) The **recursive** option is required to process a source directory.
(3) If dir2 exists, the result is a move. If dir2 doesn't exist, the result is a rename.
(4) The last argument must be a directory.
(5) Use caution with **force** option, you will not be prompted to confirm your action.
(6) Use caution with **create parent** option, typing errors are not caught.

Get help in Red Hat Enterprise Linux

- Reading Documentation using man Command
- Reading Documentation using pinfo Command
- Reading Documentation in /usr/share/doc

Introduction of man command

man is the system's manual pager.
man command in Linux is used to display the user manual of any command that we run on the terminal.
Each page argument given to man is normally the name of a program, utility or function.
Syntax: man [OPTION]...[COMMAND NAME]...

Section of the Linux manual

Section	Content type
1	User commands (both executable and shell programs)
2	System calls (kernel routines invoked from user space)

3	Library functions (provided by program libraries)
4	Special files (such as device files)
5	File formats (for many configuration files and structures)
6	Games (historical section for amusing programs)
7	Conventions, standards and miscellaneous (protocols, file systems)
8	System administration and privileged commands (maintenance tasks)
9	Linux kernel API (internal kernel calls)

Navigating man pages

Command	Result
Spacebar	Scroll forward (down) one screen
PageDown	Scroll forward (down) one screen
PageUp	Scroll backward (up) one screen
DownArrow	Scroll forward (down) one line
UpArrow	Scroll back (up) one line
d	Scroll forward (down) one half-screen
u	Scroll background (up) one half-screen
/*string*	Search forward (down) for *string* in the man page
n	Repeat previous search forward (down) in the man page
N	Repeat previous search background (up) in the man page
g	Go to the start of the man page.
G	Go to the end of the man page.
q	Exit man and return to the command shell prompt

Options in man

- **-f option**: One may not be able to remember the sections in which a command is present. So this option gives the section in which the given command is present. man -f ls
- **-a option**: This option helps us to display all the available intro manual pages in succession. man -a intro

- **-k option**: This option searches the given command as a regular expression in all the commands and it returns the manual pages with the section number in which it is found. man -k passwd
- **-w option**: This option returns the location in which the manual page of a given command is present. man -w ls

Introduction of info & pinfo

- This is a program for viewing info files.
- You specify which page you want to read by passing it an infopage argument. This argument contains the name of an info page (i.e. “bash”)
- **Configuration file**: /etc/pinforc
- **Commands**: **(/usr/share/info)** ls, tar, pwd, who, cat, pinfo
- **Read from man**: man, cd, intro, passwd, httpd
- **pinfo -m pinfo**: read pinfo pages as man

Reading Documentation in /usr/share/doc

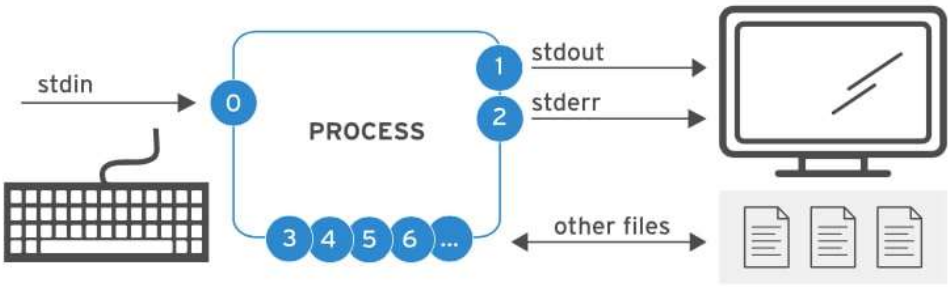
- In addition to man and pinfo, we can also choose to include documentation in their application’s RPM distribution package.
- We can access this using CLI or GUI
- CLI: cd/usr/share/doc
- GUI: firefox file: ///usr/share/doc

Creating, Viewing and Editing Text Files

- Redirecting Output to a File or Program
- Editing Text Files from the Shell Prompt
- Editing Text Files with a Graphical Editor

Standard Input, Output and Error

- A process structure is constructed with numbered channels (file descriptors) to manage open files.
- Processes connect to files to reach data content or devices these files represent.
- Processes are created with default connections for channels 0, 1 and 2, known as standard input, standard output and standard error.



Channels (File Descriptors)

Number	Channel name	Description	Default Connection	Usage
0	stdin	Standard input	Keyboard	Read only
1	stdout	Standard output	Terminal	Write only
2	strderr	Standard error	Terminal	Write only
3+	filename	Other files	none	Read and/or Write

Redirecting Output to a file

- Channel redirection replaces default channel destinations with file names representing either output file or devices.
- Using redirection, process output and error messages can be captured as file contents, sent to a device, or discarded.
- The special file /dev/null quietly discards channel output redirected to it.

Manage local users and groups:

'useradd' command:

We all are aware of the most popular command called 'useradd' or 'adduser' in Linux. There are times when a Linux System Administrator asked to create user accounts on Linux with some specific properties, limitations or commands.

In Linux, a 'useradd' command is a low-level utility that is used for adding/creating user accounts in Linux and other Unix-like operating systems. The 'adduser' is much similar to the useradd command because it is just a symbolic link to it.

In some other Linux distributions, useradd command may come with slightly different versions. I suggest you read your documentation, before using our instructions to create new user accounts in Linux.

When we run 'useradd' command in Linux terminal, it performs the following major things:

1. It edits /etc/passwd, /etc/shadow, /etc/group and /etc/gshadow files for the newly created User account.
2. Creates and populates a home directory for the new user.
3. Sets permissions and ownerships to the home directory.

The basic syntax of the command is:

useradd [options] username

Take help

```
usedadd –help
```

1. How to Add a New User in Linux

To add/create a new user, all you've to do is follow the command 'useradd' or 'adduser' with 'username'. The 'username' is a user login name that is used by the user to login to the system.

Only one user can be added and that username must be unique (different form another username already exists on the system).

For example, to add a new user named 'user1', use the following command.

When we add a new user in Linux with 'useradd' command it gets created in a locked state and to unlock that user account, we need to see a password for that account with 'passwd' command.

```
useradd user1
id user1
passwd user1
```

Once a new user is created, its entry is automatically added to the '/etc/passwd' file. The file is used to store the user's information and the entry should be.

```
id user1
tail -n 3 /etc/passwd
ls /home/
```

The above entry contains a set of seven colon-separated fields, each field has its own meaning. Let's see what these fields are:

1. Username: User login name to login to the system, It should be between 1 to 32 characters long.
2. Password: User password (or x character) stored in /etc/shadow file in encrypted format.
3. User ID (UID): Every user must have a User ID (UID) User Identification Number. By default, UID 0 is reserved for the root user and UID ranging from 1-99 are reserved for other predefined accounts. Further UID's ranging from 100-199 are reserved for system accounts and groups.
4. Group ID (GID): The primary Group ID (GID) Group Identification Number stored in /etc/group file.
5. User info: This field is optional and allows you to define extra information about the user. For example, user full name. This field is filled with a 'finger' command.
6. Home directory: The absolute location of the user's home directory.
7. Shell: The absolute location of a user's shell i.e. /bin/bash.

2. Create a User with Different Home Directory

By default '**useradd**' command creates a user's home directory under **/home** directory with a username**.** Thus, for example, we've seen above that the default home directory for the user '**user1**' is '**/home/user1**'**.**

However, this action can be changed by using '**-d**' option along with the location of the new home directory (i**.**e**. /user2**)**.** For example, the following command will create a user 'user2' with a home directory '**/user2home**'.

```
useradd -d /user2home user2
tail -n 3 /etc/passwd
ls -ld /user2/
```

3. Create a User with Specific User ID

In Linux, every user has its own **UID (Unique Identification Number)**. By default, whenever we create a new user account in **Linux**, it assigns user id **1000**, **1001**, **1002** and so on..

But, we can create users with a custom user id with '**-u**' option**.** For example, the following command will create a user '**user3**' with custom user id '**2000**'.

```
useradd -u 2000 user3
tail -n 1 /etc/passwd
id user3
```

4. Create a User with Specific Group ID

Similarly, every user has its own GID (Group Identification Number)**.** We can create users with specific group ID as well with the -g option**.**

Here in this example, we will add a user 'user4' with a specific UID and GID simultaneously with the help of '**-u**' and '**-g**' options.

```
useradd -g 10 user4
id user4
```

5. Add a User to Multiple Groups

The '**-G**' option is used to add a user to additional groups**.** Each group name is separated by a comma, with no intervening spaces**.**

Here in this example, we are adding a user 'user5' into multiple groups like **bin**, **wheel** and **games**.

```
useradd -G 1, 10, 20 user5
id user5
useradd -G bin, wheel, games user5
```

```
useradd -G bin, wheel, games user5
id user5
```

6. Create a User with Account Expiry Date

By default, when we add the user with '**useradd**' command the user account never expires i.e. their expiry date is set to 0 (means never expired).

However, we can set the expiry date using the '-e' option, which sets a date in **YYYY-MM-DD** format. This is helpful for creating temporary accounts for a specific period of time.

Here in this example, we create a user '**user6**' with account expiry date i.e. **27th Nov 2020** in **YYYY-MM-DD** format.

```
useradd -e 2022-02-05 user6
chage -l user6
```

7. Add a User with Custom Comments

The '-c' option allows you to add custom comments, such as the user's full name, phone number, etc to the /etc/passwd file. The comment can be added as a single line without any spaces.

For example, the following command will add a user 'user7' and insert that user's full name, user7 for Linux, into the comment field.

```
useradd -c 'user7 for Linux' user7
tail -n 1 /etc/passwd
```

8. Change User Login Shell:

Sometimes, we add users who have nothing to do with the login shell or sometimes we require to assign different shells to our users. We can assign different login shells to each user with '-s' option.

Here in this example, we will add a user '**user8**' without login shell i.e. '/sbin/nolgin' shell.

```
useradd -s /sbin/nologin user8
tail -n l /etc/passwd
```

'groupadd' command

Groups in Linux refer to the user groups. In Linux, there can be many users of a single system, (normal users can take uid from 1000 to 60000 and one root user (uid 0) and 999 system users (uid 1 to 999). In a scenario where there are many users, there might be some privileges that some users have and some don't and it becomes difficult to manage all the permissions at the individual user level. So using groups, we can group together a number of

users and set privileges and permissions for the entire group. groupadd command is used to create a new user group.

Syntax:

```
groupadd [option] group_name
groupadd –help
```

1. Creating a Group in Linux

To create a new group type groupadd followed by the new group name.

```
groupadd group1
tail -n 1 /etc/group
```

2. Creating a Group with Specific GID

In Linux and Unix-like operating systems, groups are identified by their name and a unique GID (a positive integer).

By default, when a new group is created, the system assigns the next available GID from the range of range IDs specified in the login.defs file.

```
groupadd -g 3000 group2
tail -n l /etc/group
```

3. Creating a System Group

There is no real technical difference between the system and regular (normal) groups. Usually, system groups are used for some special system operation purposes, like creating backups or doing system maintenance.

System groups GIDs are chosen from the range of system group UDs specified in the login.defs file, which is different from the range used for regular groups.

Use the -r (--system) option to create a system group. For example, to create a new system group named **group3** you would run:

```
groupadd -r group3
tail -n 1 /etc/group
```

4. Creating a System Group with Password

Adding a password to a group has no practical use and may cause a security problem since more than one user will need to know the password.

The -p (--password) option followed by password allows you to set a password for the new group:

```
groupadd -p grouppassword group4
tail -n l /etc/gshadow
tail -n l /etc/group
```

‘usermod’ command

In Unix/Linux distributions, the command ‘**usermod**’ is used to modify or change any attributes of an already created user account via the command line. The command ‘**usermod**’ is similar to that of ‘**useradd**’ or ‘**adduser**’ but the login is granted to an existing user.

After creating user accounts, in some scenarios where we change the attributes of an existing user such as, change user’s home directory, login name, login shell, password expiry date, etc, where in such case ‘usermod’ command is used.

When we execute the ‘usermod’ command in the terminal, the following files are used and affected.

1. /etc/passwd - User account information.
2. /etc/shadow - Secure account information.
3. /etc/group - Group account information.
4. /etc/gshadow - Secure group account information.
5. /etc/login.defs - Shadow password suite configuration.

Basic syntax of command is:

```
usermod [option] username
```

Requirements

1. We must have existing user accounts to execute usermod commands.
2. Only superuser (root) is allowed to execute usermod commands.
3. The usermod command can be executed on any Linux distribution.
4. Must have basic knowledge of usermod commands with options.

Options of Usermod

The ‘**usermod**’ command is simple to use with lots of options to make changes to an existing user. Let us see how to use the usermod command by modifying some existing users in the Linux box with the help of the following options.

- -c = We can add a comment field for the user account.
- -d = To modify it directly for any existing user account.
- -e = Using this option we can make the account expire in a specific period.

- -g = Change the primary group for a User.
- -G = To add a supplementary group.
- -a = To add anyone of the group to a secondary group.
- -l = To change the login name from user2 to user2_admin.
- -L = To lock the user account. This will lock the password so we can't use the account.
- -m = moving the contents of the home directory from existing home dir to new dir.
- -p = To use an unencrypted password for a new password. (NOT Secured).
- -s = Create a Specified shell for new accounts.
- -u = Used to Assign UID for the user account between 0 to 999.
- -U = To unlock the user accounts. This will remove the password lock and allow us to use the user account.

1. Adding Information to User Account

The '-c' option is used to set a brief comment (information) about the user account. For example, let's add information on 'user2' user, using the following command.

```
usermod –c "this is user1" user1
cat /etc/passwd | grep user1
```

2. Change User Home Directory

In the above step we can see that our home directory is under /home/user1/, If we need to change it to some other directory we can change it using -d option with usermod command.

```
cat /etc/passwd | grep user1
usermod -d /user1 user1
cat /etc/passwd | grep user1
```

3. Change User Primary Group

To set or change a user primary group, we use option '-g' with usermod command. Before changing the user primary group, first make sure to check the current group for the user user1 then, set the group1 group as a primary group to user user1 and confirm the changes.

```
id user1
usermod -g group1 user1
id user1
```

4. Adding Group to an Existing User

If you want to add a new group called 'group2' to 'user1' user, you can use option '-G' with usermod command as shown below.

```
usermod -G group2 user1
id user1
cat /etc/group | grep group2
```

Note: Be careful, while adding a new group to an existing user with '**-G**' option alone, will remove all existing groups that user belongs to. So, always add the '**-a**' (append) with '**-G**' option to add or append new groups.

5. Adding Supplementary and Primary Group to User

If you need to add a user to any one of the supplementary groups, you can use the options '**-a**' and '**-G**'. For example, here we are going to add a user account user1 with the wheel user.

```
usermod -aG group3 user1
usermod -aG group4 user1
id user1
tail -n 3 /etc/group
```

6. Change User Login Name

To change any existing user login name, we can use the '**-l**' (new login) option. In the example below, we are changing the name **user1** to **user1_admin.** So the username user1 has been renamed with the new name **user1_admin**.

```
usermod -l user1_admin user1
id user1_admin
cat /etc/passwd | grep user1_admin
```

7. Lock User Account

To lock any system user account, we can use the '**-L**' (lock) option. After the account is locked we can't login by using the password and you will see a ! added before the encrypted password in /etc/shadow file, meaning password disabled.

```
usermod -L user2
cat /etc/shadow | grep user2
```

8. Unlock User Account

The '**-U**' option is used to unlock any locked user, this will remove the ! before the encrypted password.

```
usermod -U user2
cat /etc/shadow | grep user2
```

9. Change User Shell

The user login shell can be changed or defined during user creation with **useradd** command or changed with **usermod** command using option '**-s**' (shell). For example, the user '**user2**' has the **/bin/bash** shell by default, now I want to change it to **/bin/sh.**

```
usermod -s /bin/sh user2
cat /etc/passwd | grep user2
```

10. Change User ID (UID)

In the example below, you can see that user account '**user2**' holds the UID of **1002**, now I want to change it to **2222** as my UID.

```
usermod -u 2222 user2
id user2
cat /etc/passwd | grep user2
```

'groupmod' command

The groupmod command in Linux is used to modify or change the existing group on the Linux system. It can be handled by a superuser or root user. Basically, it modifies a group definition on the system by modifying the right entry in the database of the group.

Syntax:

```
groupmod [option] GROUP
```

Options: There are the following options available in groupmod command.

- **-g, gid GID:** The group ID of the given GROUP will be changed to GID.
- **-n, -new-name NEW_GROUP:** The name of the group will change into a newname.
- **-h, help:** This option displays a help message and exists.
- **-o, -non-unique:** This option is used with the -g option that allows changing the group GID to a non-unique value.
- **-p, -password PASSWORD:** This gives the encrypted password.
- **-R, -root CHROOT_DIR:** Apply changes in the CHROOT_DIR directory and use the configuration files from the CHROOT_DIR directory.

```
groupmod –help
```

Example:

Below command will change the **group1** to **admin_group1** using the **-n** option.

'userdel' command

userdel command in the linux system is used to delete a user account and related files. This command basically modifies the system account files, deleting all the entries which refer to the username LOGIN. It is a low-level utility for removing the users.

Syntax:

```
userdel [options] LOGIN
```

With option userdel -f:

This option forces the removal of the specified user account. It doesn't matter that the user is still logged in. It also forces the userdel to remove the user's home directory and mail spool, even if another user is using the same home directory or even if the mail spool is not owned by the specified user in ubuntu Operating System. In RHEL 8, it will not delete the home directory and mail spool.

```
userdel -f user8
ls /home/
```

With option userdel -r:

Whenever we are deleting a user using this option then the files in the user's home directory will be removed along with the home directory itself and the user's mail spool. All the files located in other file systems will have to be searched for and deleted manually.

```
ls /home/
userdel -r user7
ls /home/
```

'groupdel' command

The groupdel command is used to delete an existing group. It will delete all entries that refer to the group, modify the system account files, and it is handled by superuser or root user.

Syntax:

```
groupdel –help
groupdel [options] GROUP
```

Deleting the group

```
cat /etc/group | grep group4
groupdel group4
```

Linux Chage Command to Set Password Ageing for User

The command name 'chage' is an acronym for 'change age'. This command is used to change the user's password's ageing/expiry information.

As a system administrator, it's your task to enforce password changing policies so that after a certain period of time, users will be compelled to reset their passwords.

No other unauthorised users can view the passwords ageing/expiry information. As the root user, you can execute this command to modify the ageing information.

Syntax:

```
chage [-m mindays] [-M maxdays] [-d lastday] [-I inactive] [-E expirydate] [-W warndays] user
```

Actually, you can also force the user to change their password periodically via **/etc/login.defs** file.

But /etc/login.defs will affect every user that registered in the system. If you want to set up different rules to a different user, then chage is the right tool for you.

1. List the password ageing information of a user

To view the password expiry details of a user, run the command below

```
chage -l user2
```

As you can see, password information for this user.

2. Setup password policy of a user

In most cases, as an administrator, you need to set a password policy for all users for the purpose of better security.

```
chage user2
```

or

```
chage -m 0 -M 90 -W 5 -I 90 -E 2020-08-30 user2
chage -l user2
```

Ch_07. Files Permission in Linux with Example

Introduction:

Linux is a clone of UNIX, the **multi-user operating system** which can be accessed by many users simultaneously. Linux can also be used in mainframes and servers without any modifications. But this raises security concerns as an unsolicited or **malign user** can **corrupt, change or remove crucial data.** For effective security, Linux divides authorization into 2 levels.

1. Ownership
2. Permission

Ownership of Linux files

Every file and directory on your Unix/Linux system is assigned 3 types of the owner, given below.

User

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

Group

A user-group can contain multiple users. All users belonging to a group will have the same access permission to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

Other

Any other user who has access to a file. This person has neither created the file, nor belongs to a user group who could own the file. Practically, it means everyday else. Hence, when you set permission for others, it is also referred to as set permissions for the world.

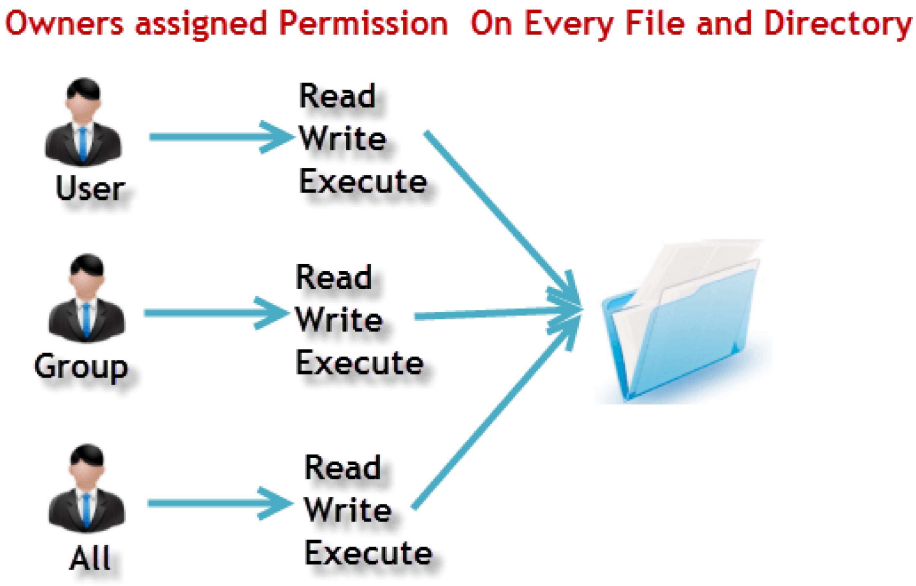
Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's vital information/data. It is like you do not want your colleague, who works on your linux computer, to view your images. This is where **Permissions** set in and they define **user behaviour**.

Let us understand the **Permission system** on Linux.

Permissions

Every file and directory in your UNIX/Linux system has the following 3 permissions defined for all the 3 owners discussed above.

Read: This permission gives you the authority to open and read a file. Read permission on a directory gives you the ability to list its content.
Write: The write permission gives you the authority to modify the contents of a file. The write permission on a directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
Execute: In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code (provided read & write permissions are set), but not run it.



ls -l on terminal gives

-rw-rw-r-- 1 ec2-user ec2-user 0 Feb 19 05:28 test1
drwxrwxr-x 2 ec2-user ec2-user 6 Feb 19 05:31 param

Here, we have highlighted 'rw-rw-r–' and this weird looking code is the one that tells us about the permissions given to the owner, user group, and the world.

Here, the first '-' (-rw-rw-r–) indicate that we have selected a file.p

Else, if it were a directory, d would have been shown.

Where, (drwxrwxr-x) represents the directory.

The characters are pretty easy to remember.

r = read permission
w = write permission
x = execute permission
- = no permission

Let us look at it this way.

The first part of the code is '**rw-**'. This suggests that the owner 'Home' can:

-rw-rw-r– (no execute permission)

- Read the file
- Write or edit the file
- He cannot execute the file since the execute bit is set to '-'.

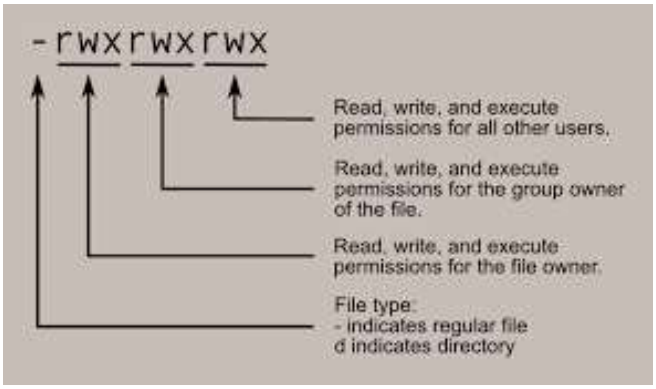
By design, many Linux distributions like Fedora, CentOS, Ubuntu, etc. will add users to a group of the same group name as the user name. Thus, a user 'tom' is added to a group named 'tom'.

The second part is '**rw-**'. It for the user group 'Home' and group-members can:

- Read the file
- Write or edit the file

The third part is for the world which means any user. It says '**r--**'. This means the user can only:

- Read the file



Changing file/directory permissions with 'chmod' command

Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions.

We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.

Syntax:

```
chmod permissions filename
```

There are 2 ways to use the command:

1. Absolute mode
2. Symbolic mode

Absolute (Numeric) Mode

In this mode, file permissions are not represented as characters but a three-digit octal number.

The table below gives numbers for all permissions types.

Number	Permission Type	Symbol
0	No Permission	- - -
1	Execute	- - x
2	Write	- w -
3	Execute + Write	- wx
4	Read	r - -
5	Read + Execute	r - x
6	Read + Write	rw -
7	Read + Write + Execute	rwx

Let's see the chmod command in action.

Checking current file permissions

```
ls -l sample
-rw-rw-r-- 1 ec2-user ec2-user 0 Feb 19 06:17 sample
```

chmod 764 and checking permissions again

```
chmod 764 sample
ls -l sample
-rwxrw-r-- 1 ec2-user ec2-user 0 Feb 19 06:17 sample
```

In the above-given terminal window, we have changed the permissions of the file 'sample' to '764'.

Where,

- '7' - Read + Write + Execute (rwx)
- '6' - Read (r--)
- '4' - Read + Write (rw-)

'764' absolute code says the following:

- Owner can read, write and execute
- Usergroup can read and write
- The world can only read

This is shown as '-rwxrw-r-'

This is how you can change the permissions on file by assigning an absolute number.

Symbolic Mode

In the Absolute mode, you can change permissions for all 3 owners. In the symbolic mode, you can modify the permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

Operator	Description
+	Adds permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as:

User Denotations	
u	user
g	group
o	other
a	all

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example.

Current file permissions
```
ls -l sample
-rw-rw-r-- 1 ec2-user ec2-user 0 Feb 19 08:07 sample
```
Setting permissions to the 'other' users
```
chmod o=rwx sample
ls -l sample
-rw-rw-rwx 1 ec2-user ec2-user 0 Feb 19 08:07 sample
```
Adding 'execute' permission to the usergroup
```
chmod g+x sample
ls -l sample
-rw-rwxrwx 1 ec2-user ec2-user 0 Feb 19 08:07 sample
```
Removing 'read' permission for 'user'
```
chmod u-r sample
ls -l sample
--w-rwxrwx 1 ec2-user ec2-user 0 Feb 19 08:07 sample
```

Changing Ownership and Group

Syntax:

```
chown user.group filename
```

Let's see this in action

Check the current file ownership using `ls -l`

```
--w-rwxrwx 1 ec2-user ec2-user 0 Feb 19 08:07 sample
```

Change the file owner to

```
sudo chown n100 sample
```

Ownership changed to n100

```
--w-rwxrwx 1 n100     ec2-user 0 Feb 19 08:07 sample
```

Changing user and group to root 'chown user:group file

```
sudo chown root:root sample
```

User and Group ownership changed to root

```
--w-rwxrwx 1 root root 0 Feb 19 08:07 sample
```

In case you want to change group-owner only, use the command

```
chgrp group_name filename
```

'chgrp' stands for change group.

Check the current file ownership using is `-dl`

```
ls -dl test1
-rw-rw-r-- 1 ec2-user ec2-user 0 Feb 19 08:24 test1
```

Change the file owner to root . you will need sudo

```
sudo chgrp root test1
ls -dl test1
-rw-rw-r-- 1 ec2-user root 0 Feb 19 08:24 test1
```

Special File Permissions (setid, setgid and Sticky bit)

Three special types of permissions are available for executable files and public directories. When these permissions are set, any user who runs that executable file assumes the user ID of the owner (or group) of the executable file.

You must be extremely careful when you set special permissions, because special permissions constitute a security risk. For example, a user can gain superuser privileges by executing a program that sets the user ID (UID) to root. Also, all users can set special permissions for files they own, which constitutes another security concern.

setuid Permission

When set-user identification (setuid) permission is set on an executable file, a process that runs this file is granted access based on the owner of the file (usually root), rather than the user who is running the executable file. This special permission allows a user to access files and directories that are normally only available to the owner. For example, the setuid permission on the password command makes it possible for a user to change passwords, assuming the permissions of the root ID:

```
ls -l /bin/passwd
ls -l /bin/sudo
```

This special permission presents a security risk, because some determined users can find a way to maintain the permissions that are granted to them by the setuid process even after the process has finished executing.

Note: The use of setuid permissions with the reserved UIDs (0-100) from a program might not set the effective UID correctly. Use a shell script instead or avoid using the reserved UIDs with setuid permissions.

setgid Permission

The set-group identification (setgid) permissions is similar to setuid, except that the process's effective group D (GID) is changed to the group owner of the file, and a user is granted access based on permissions granted to that group. The /usr/bin/mail command has setgid permissions:

```
mkdir /testdir
chmod g+s /testdir
ls -ld /testdir
```

When setgid permission is applied to a directory, files that were created in this directory belong to the group to which the directory belongs, not the group to which the creating process belongs. Any user who has write and execute permissions in the directory can create a file there. However, the file belongs to the group that owns the directory, not to the user's group ownership.

Sticky Bit

The sticky bit is a permission bit that protects the files within a directory. If the directory has the sticky bit set, a file can be deleted only by the owner of the file, the owner of the directory, or by root. This special permission prevents a user from deleting other users files from public directories such as /tmp:

```
ls -ld /tmp
```

Ch_08 Monitoring and Managing Linux Processes

What is a process?

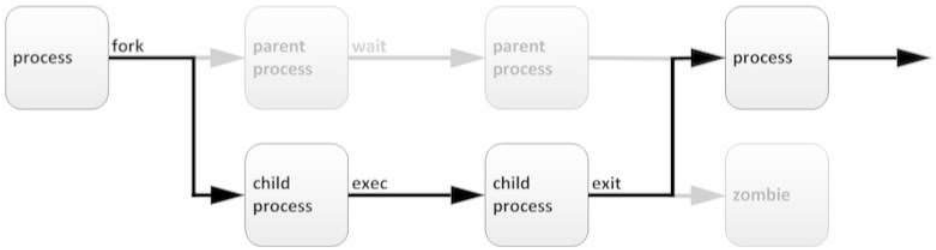
A process is a running instance of a launched, executable program. A process consists of:

- An address space of allocated memory.
- Security properties including ownership credentials and privileges.
- One or more execution threads of program code and
- The process state

The environment of a process includes:

- Local and global variables.
- A current scheduling context and
- Allocated system resources, such as file descriptors and network ports.

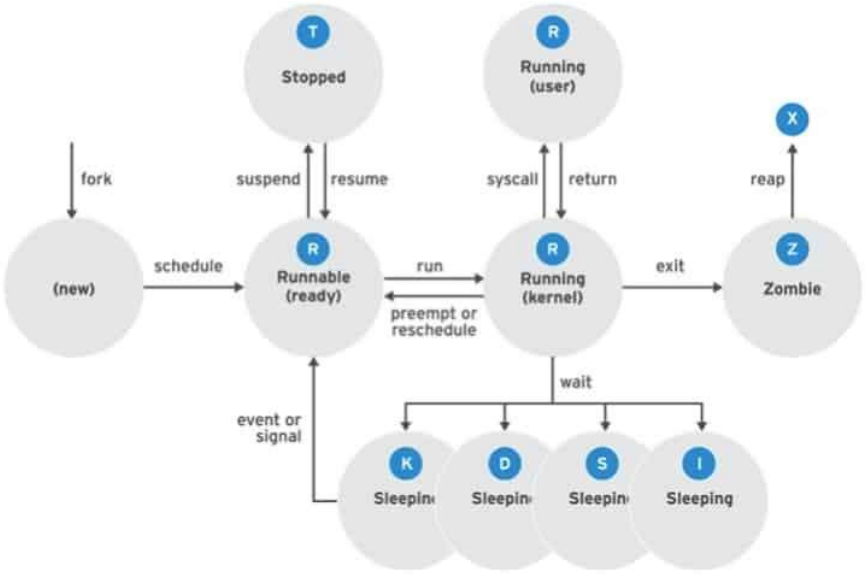
An existing (parent) process duplicates its own address space (fork) to create a new (child) process structure. Every new process is assigned a unique process ID (PID) for tracking and security. The PID and the parent's process ID (PPID) are elements of the new process environment. Any process may create a child process. All processes are descendants of the first system process, which is systemd(1) on a Red Hat Enterprise Linux 7 system.



Through the fork routine, a child process inherits security identities, previous and current file descriptors, port and resource privileges ,environment variables, and program code. A child process may then exec its own program code. Normally, a parent process sleeps while the child process runs, setting a request (wait) to be signalled when the child completes. Upon exit, the child process has already closed or discarded its resources and environment; the remainder is referred to as a zombie. The parent, signalled awake when the child exited, cleaned the remaining structure, then continued with its own program code execution.

Process States

In a multitasking operating system, each CPU (or CPU core) can be working on one process at a single point in time. As a process runs, its immediate requirements for CPU time and resource allocation change. Processes are assigned a state, which changes as circumstances require.



The Linux process states are illustrated in the previous diagram and described in the following table.

NAME	FLAG	KERNEL-DEFINED STATE NAME AND DESCRIPTION
Running	R	TASK_RUNNING: The process is either executing on a CPU or waiting to run. Processes can be executing user routines or kernel routines (system calls), or be queued and ready when in the Running (or Runnable) state.
Sleeping	S	TASK_INTERRUPTABLE: The process is waiting for some condition: a hardware request, system resource access, or signal. When an event or signal satisfies the condition, the process returns to Running
	D	TASK_UNINTERRUPTABLE: This process is also Sleeping, but unlike S state, it will not respond to delivered signals. Used only under specific conditions in which process interruption may cause an unpredictable device state.
	K	TASK_KILLABLE: Identical to the uninterruptible D state, but modified to allow the waiting task to respond to a signal to be killed (exited completely). Utilities frequently display Killable processes as D states.
Stopped	T	TASK_STOPPED: The process has been Stopped (suspended), usually by being signalled by a user or another process. The process can continue (resumed) by another signal to return to Running.
	T	TASK_TRACED: A process that is being debugged is also temporarily Stopped and shared the same T state flag.

Zombie	Z	EXIT_ZOMBIE: A child process signals its parent as it exists. All resources except for the process identity (PID) are released.
	X	EXIT_DEAD: When the parent cleans up (reaps) the remaining child process structure, the process is now released completely. This state will never be observed in process-listing utilities.

Listing Processes

The **ps** command is used for listing current processes. The command can provide a detailed process information, including:

- The user identification (UID) which determines process privileges.
- The unique process identification (PID)
- The CPU and real-time already expended
- How much memory the process has allocated in various locations
- The location of process STDOUT, known as the controlling terminal, and
- The current process state.

A common display listing (options **aux**) displays all processes, with columns in which users will be interested, and includes processes without a controlling terminal. A long listing (options lax) provides more technical details but may display faster by avoiding the username lookup. The similar UNIX syntax uses the options **-ef** to display all processes.

```
man ps
ps
```

Jobs and Sessions

Job control is a feature of the shell which allows a single shell instance to run and manage multiple commands.

A job is associated with each pipeline entered at a shell prompt. All processes in that pipeline are part of the job and are members of the same process group. (If only one command is entered at a shell prompt, that can be considered to be a minimal “pipeline” of one command. That command would be the only member of that job.)

Only one job can read input and keyboard-generated signals from a particular terminal window at a time. Processes that are part of that job are foreground processes of that controlling terminal.

A background process of that controlling terminal is a member of any other job associated with that terminal. Background processes of a terminal can not read input or receive keyboard-generated interrupts from the terminal but may be able to write to the terminal. A

job in the background may be stopped (suspended) or it may be running. If a running background job tries to read from the terminal, it will be automatically suspended.

Each terminal is its own session and can have a foreground process and independent background processes. A job is part of exactly one session, the one belonging to its controlling terminal.

The ps command will show the device name of the controlling terminal of a process in the TTY column. Some processes, such as system daemons, are started by the system and not from a shell prompt. These processes do not have a controlling terminal, are not members of a job, and can not be brought to the foreground. The ps command will display a question mark (?) in the TTY column for these processes.

Running Jobs in the background:

Any command or pipeline can be started in the background by appending an ampersand (&) to the end of the command line. The bash shell displays a job number (unique to the session) and the PID of the new child process. The shell does not wait for the child process and redisplays the shell prompt.

```
sleep 1000 &
jobs
```

A background job can be brought to the foreground by using the fg command with its job ID (%job number).

In the preceding example, the sleep command is now running in the foreground on the controlling terminal. The shell itself is again asleep, waiting for this process to exit.

To send a foreground process to the background, first press the keyboard-generated suspend request (CTRL+Z) on the terminal.

```
jobs
fg %1
jobs
```

To start the suspended process running in the background, use the bg command with same job ID.

```
jobs
bg %1
jobs
```

Killing Processes

Process Control Using Signals

A signal is a software interrupt delivered to a process. Signals report events to an executing program. Events that generate a signal can be an error, external event (e.g., I/O request or expired timer), or by explicit request (e.g., use of a signal-sending command or by keyboard sequence). The following table lists the fundamental signals used by system administrators for routine process management. Refer to signals by either their short (HUP) or proper (SIGHUP) name.

Fundamental process management signals

Signal Number	Short Name	Definition	Purpose
1	HUP	Hangup	Used to report the termination of the controlling process of a terminal.Also used to request process reinitialization (configuration reload) without termination.
2	INT	Keyboard interrupt	Causes program termination. It can be blocked or handled. Sent by pressing the INTR key combination (CTRL+C).
3	QUIT	Keyboard Quit	Similar to SIGINT, but also produces a process dump at termination. Sent by pressing the QUIT key combination (CTRL+\).
9	KILL	Kill, unblockable	It causes abrupt program termination, It cannot be blocked, ignored or handled; always fatal.
15 default	TERM	Terminate	Causes program termination. Unlike SIGKILL can be blocked, ignored or handled. The polite way to ask a program to terminate; allows self-cleanup.
18	CONT	Continue	Sent to a process to resume if stopped. It cannot be blocked. Even if handled, it always resumes the process.
19	STOP	Stop unblockable	Suspends the process. It cannot be blocked or handled.
20	TSTP	Keyboard stop	Unlike SIGSTOP, it can be blocked, ignored, or handled. Sent by pressing the SUSP key combination (CTRL+Z).

```
kill -l
jobs
kill -9 %1
kill -KILL %2

sleep 1100 &
sleep 1200 &
sleep 1000

sleep 1100 &
sleep 1200 &
kill -9 3867
kill -TERM 3868
```

Monitoring Process Activity

Load Average

- The Linux kernel calculates a load average metric as an exponential moving average of the load number, a cumulative CPU count of active system resource requests.
- Active requests are counted from per-CPU queues for running threads and threads waiting for I/O, as the kernel tracks process resource activity and corresponding process state changes.
- A load number is a calculation routine run every five seconds by default, which accumulates and averages the active requests into a single number for all CPUs.
- The exponential moving average is a mathematical formula to smooth out trending data highs and lows, increase current activity significance and decrease ageing data quality.
- The load average is the load number calculation routine result. Collectively, it refers to the three displayed values of system data averaged for the last 1,5 and 15 minutes.

```
uptime
```

Real-Time Process Monitoring

The top program is a dynamic view of the system's processes, displaying a summary header followed by a process or thread list similar to ps command information. Unlike the static ps output, top continuously refreshes at a configurable interval and provides capabilities for column reordering, sorting and highlighting. User configurations can be saved and made persistent.

Default output columns are recognizable from other resource tools:

- The process ID (PID).
- User name (USER) is the process owner.
- Virtual memory (VIRT) is all memory the process is using, including the resident set, shared libraries and may mapped or swapped memory pages. (Labelled VSZ in the ps command.)

- Resident memory (RES) is the physical memory used by the process, including any
- A resident shared objects. (Labelled RSS in the ps command.)
- Process state (S) displays as:
    1. D = Uninterruptible Sleeping
    2. R = Running or Runnable
    3. S = Sleeping
    4. T = Stopped or Traced
    5. Z = Zombie

CPU time (TIME) is the total processing time since the process started. Maybe toggled to include the cumulative time of all previous children.

Fundamental keystrokes in top

Key	Purpose
? or h	Help for interactive keystrokes.
l, t, m	Toggled for load, threads and memory header lines.
1	Toggle showing individual CPUs or a summary for all CPUs in the header.
s	Change the refresh (screen) rate, in decimal seconds (e.g. 0.5, 1, 5).
b	Toggle reverse highlighting for Running processes; default is bold only.
B	Enables use of bold in the display, in the header and for Running processes.
H	Toggle threads; show process summary or individual threads.
u, U	Filter for any user name (effective, real).
M	Sorts process listing by memory usage, in descending order.
P	Sorts process listing by processor utilisation, in descending order.
k	Kill a process. When prompted, enter PID, then nice_value.
r	Renice a process. When prompted, enter PID, then nice_value.
W	Write (save) the current display configuration for use at the next top restart.
q	Quit.

top

Ch_09 Controlling Services and Daemons

Identifying Automatically Started System Process

Introduction to 'systemd'

System startup and server processes are managed by the systemd System and Service Manager. This program provides a method for activating system resources, server daemons and other processes both at boot time and on a running system.

Daemons are processes that wait or run in the background performing various tasks. Generally, daemons start automatically at boot time and continue to run until shutdown or until they are manually stopped. By convention, the names of many demon programs end in the letter 'd'.

To listen for connections, a daemon uses a socket. This is the primary communication channel with local or remote clients. Sockets may be created by daemons or may be separated from the daemon and be created by another process, such as systemd. The socket is passed to the daemon when a connection is established by the client.

A service often refers to one or more daemons, but starting or stopping a service may instead make a one-time change to the state of the system, which does not involve leaving a daemon process running afterward (called one shot).

Service states

The status of a service can be viewed with systemctl status name.type. If the unit type is not provided, systemctl will show the status of a service unit, if one exists.

```
systemctl status sshd.service
```

Serval keywords indicating the state of the service can be found in the status output:

Keyword	Description
loaded	The unit configuration file has been processed.
active (running)	Running with one or more continuing processes.
active (excited)	Successfully completed a one-time configuration.
active (waiting)	Running but waiting for an event.
inactive	Not running.
enabled	It will start at boot time.
disabled	It will not be started at boot time.

static	Can not be enabled, but may be started by an enabled unit automatically.

Listing Unit Files with systemctl

1. Query the state of all units to verify a system setup.

```
systemctl
```

2. Query the state of only the service units.

```
systemctl --type=service
```

3. Investigate any unit which is in a failed or maintenance state. Optionally, add the -l option to show the full output.

```
systemctl status sshd -l
```

4. The status argument may also be used to determine if a particular unit is active and show if the unit is enabled to start at boot time. Alternate commands can also easily show the active and enabled states:

```
systemctl is-active sshd
systemctl is-enabled sshd
```

5. List the active state of all loaded units. Optionally, limit the type of unit. The –all option will add inactive units.

```
systemctl list-units –type=service
systemctl list-units –type=service –all
```

6. View the enabled and disabled settings for all units. Optionally, limit the type of unit.

```
systemctl list-unit-files –type=service
```

Controlling System Services

Starting and Stopping System Daemons on A Running System

Changes to a configuration file or other updates to a service may require that the service be restarted. A service that is no longer used may be stopped before removing the software. A service that is not frequently used may be manually started by an administrator only when it is needed.

In this example, please follow along with the next steps while your instructor demonstrates managing services on a running system.

1. View the status of a service

```
systemctl status crond.service
```

2. Verify that the process is running

```
ps -up 1107
```

3. Stop the service and verify the status

```
stop crond.service
status crond.service
```

4. Start the service and view the status. The process ID has changed.

```
start crond.service
status crond.service
```

5. Stop, then start, the service in a single command.

```
systemctl restart crond.service
systemctl status crond.service
```

6. Issue instructions for a service to read and reload its configuration file without a complete stop and start. The process ID will not change.

```
systemctl reload crond.service
systemctl status crond.service
```

Masking services

At times, a system may have conflicting services installed. For example, there are multiple methods to manage networks (networks and NetworkManger) and firewalls (iptables and firewalld). To prevent an administrator from accidentally starting a service, that service may be masked. Masking will create a link in the configuration directories so that if the service is started, nothing will happen.

```
systemctl mask crond.service
systemctl unmask crond.service
```

Enabling System Daemons to Start or Stop at Boot

Starting a service on a running system does not guarantee that the service will be started when the system reboots. Similarly, stopping a service on a running system will not keep it from starting again when the system reboots. Services are started at boot time when links are created in the appropriate systemd configuration directories. These links are created and removed with systemctl commands.

In this example, please follow along with the next steps while your instructor demonstrates enabling and disabling services.

1. View the status of a service.

```
systemctl status crond.service
```

2. Disable the service and verify the status. Note that disabling a service does not stop the service.

```
systemctl disable crond.service
```

3. Enable the service and verify the status.

```
systemctl enable crond.service
```

Summary of systemctl commands

Services can be started and stopped on a running system and enabled or disabled for automatic-start at boot time.

Task	Command
View detailed information about a unit state.	systemctl status UNIT
Stop service on a running system.	systemctl stop UNIT
Start a service on a running system.	systemctl stop UNIT
Restart a service on a running system.	systemctl restart UNIT
Reload configuration file of a running service.	systemctl reload UNIT
Completely disable a service from being started, both manually and at boot.	systemctl mask UNIT
Make a masked service available.	systemctl unmask UNIT
Configure a service to start at boot time.	systemctl enable UNIT

Disable a service from starting at boot time.	systemctl disable UNIT
List units which are required and wanted by the specified unit.	systemctl list-dependencies UNIT

Ch_10 Configuring and Securing OpenSSH Service

Accessing The Remote Command Line with SSH

What is the OpenSSH Secure Shell (SSH)?

The term OpenSSH refers to the software implementation of the Secure Shell Software used in the system. The OpenSSH Secure Shell, ssh, is used to securely run a shell on a remote system. If you have a user account on a remote Linux system providing SSH services, ssh is the command normally used to remotely log into that system. The ssh command can also be used to run an individual command on a remote system.

Secure Shell Examples:

- **Start Server and Client Machine and find out IP add**

```
ip add
```

- **Create a remote interactive shell as the current user, then return to your previous shell when done with the exit command.**

```
ssh 192.168.84.227
```

- **Connect to a remote shell as a different user (remote user) on a selected host (remote host):**

```
ssh root@192.168.84.227
```

- **Execute a single command (hostname) on a remote host (remote host) and as a remote user (remote user) in a way that returns the output to the local display.**

```
ssh root@192.168.84.227 hostname
```

- **The w command displays a list of users currently logged into the computer. This is especially useful to show which users are logged in using ssh from which remote locations and what they are doing.**

```
w -f
```

SSH Host keys

SSH secures communication through public-key encryption. When an ssh client connects to an SSH server, before the client logs in the server sends it a copy of its public key. This is used to set up the secure encryption for the communication channel and to authenticate the server to the client.

The first time a user uses ssh to connect to a particular server, the ssh command stores the server's public key in the user's **~/.ssh/known_hosts** file. Every time the user connects after that, the client makes sure it gets the same public key from the server by comparing the server's entry in the **~/.ssh/known_hosts** file to the public key the server sent. If the keys do not match, the client assumes that the network traffic is being hijacked or that the server has been compromised and breaks the connection.

This means that if a server's public key is changed (because the key was lost due to hard drive failure or replaced for some legitimate reason), users will need to update their **~/.ssh/known_hosts** files and remove the old entry in order to log in.

- Host IDs are stored in **~/.ssh/known_hosts** on your local client system:

```
cat ~/.ssh/known_hosts
```

- Host key are stored in **/etc/ssh/ssh_host_key*** on the SSH server.

```
ls /etc/ssh/ssh
```

Configuring SSH key-based authentication

SSH key-based authentication

Users can authenticate ssh logins without a password by using public-key authentication. SSH allows users to authenticate using a **private-public** key scheme. This means that two keys are generated, a private key and a public key. The private key file is used as the authentication credential and as a password, it must be kept secret and secure. The public key is copied to systems the user wants to log into and is used to verify the private key. The public key does not need to be secret. An SSH server that has the public key can issue a challenge that can only be answered by a system holding your private key. As a result, you can authenticate using the presence of your key. This allows you to access systems in a way that doesn't require typing password every time but is still secure.

The key generation is done using the ssh-keygen command. This generates the private key **~/.ssh/id_rsa** and the public key **~/.ssh/id_rsa.pub**.

- Use ssh-keygen to create a public-private key pair.

```
ssh-keygen
```

- Use ssh-copy-id copy the public key to the correct location on a remote system. For example:

```
ssh-copy-id student@192.168.84.227
```

Customising SSH service configuration

The OpenSSH server configuration file

While OpenSSH server configuration usually does not require modification, additional security measures are available. Various aspects of the OpenSSH server can be modified in the configuration file **/etc/ssh/sshd_config.**

Prohibit the root user from logging in using

SSH from a security standpoint, it is advisable to prohibit the root user from directly logging into the system with ssh.

- The username root exists on every Linux system by default, so a potential attacker only has to guess the password, instead of a valid username and password combination.
- The root user has unrestricted privileges.

The OpenSSH server has an internal configuration file setting to prohibit a system login as user root, which is commented out by default in the **/etc/ssh/sshd_config** file:

```
#PermitRootLogin yes
```

By enabling the previous option in the **/etc/ssh/sshd_config** file as follows, the root user will be unable to log into the system using the ssh command after the sshd service has been restarted:

```
#PermitRootLogin no
```

- The sshd service has to be restarted to put the changes into effect:

```
sudo systemctl restart sshd
```

- Test connection using the root user

```
ssh root@192.168.84.227
```

Prohibit password authentication using SSH

Only allowing key-based logins to the remote command line has various advantages:

- SSH keys are longer than the average password, which adds security.
- Less effort to initiate remote shell access after the initial setup.

There is an option in the /etc/ssh/sshd_config configuration file which turns on password authentication by default:

```
PasswordAuthentication yes
```

To prevent password authentication, the PasswordAuthentication option has to be set to no and the sshd service needs to be restarted:

```
PasswordAuthentication no
```

- Test password-based connection using new ‘abc’ user

```
sudo systemctl restart sshd
```

Access VM’s from Physical Machine using Git Bash

Step 1: Download and Install Git Bash

https://git-scm.com/

Step 2: Open Git Bash and Access VM’s

```
ssh student@192.168.84.169
```

Ch_11 Analysing and Storing logs

System Log Architecture

System Logging

Processes and the operating system kernel need to be able to record a log of events that happen. These logs can be useful for auditing the system and troubling problems. By convention, the **/var/log** directory is where these logs are persistently stored.

A standard logging system based on the Syslog protocol is built into Red Hat Enterprise Linux. Many programs use this system to record events and organise them into log files. In Red Hat Enterprise Linux 7, syslog messages are handled by two services, **systemd-journald** and **rsyslog**.

The **systemd-journald** daemon provides an improved log management service that collects messages from the kernel, the early stages of the boot process, standard output and error of daemons as they start up and run and syslog. It writes these messages to a structured journal of events that, by default, does not persist between reboots. This allows syslog messages and events which are missed by syslog to be collected in one central database. The syslog messages are also forwarded by **systemd-journald** to **rsyslog** for further processing.

The **rsyslog** service then sorts the syslog messages by type (or facility) and priority and writes them to persistent files in the **/var/log** directory.

The **/var/log** directory holds various system- and service-specific log files maintained by rsyslog:

Overview of system log files

Log File	Purpose
/var/log/messages	Most syslog messages are logged here. The exceptions are messages related to authentication and email processing that periodically run jobs and those which are purely debugging related.
/var/log/secure	The log file for security and authentication-related messages and errors.
/var/log/maillog	The log file with mail server-related messages.
/var/log/cron	The log file relates to periodically executing tasks.
/var/log/boot.log	Messages related to system startup are logged here.

Reviewing Syslog Files

Syslog Files

Many programs use the syslog protocol to log events to the system. Each log message is categorised by a facility (the type of message) and a priority (the severity of the message). The facilities which are available are documented by the rsyslog.conf(5) man page.

The eight priorities are also standardised and ranked as follows:

Overview of syslog priorities:

Code	Priority	Severity
0	emerg	System is unusable
1	alert	Action must be taken immediately.
2	crit	Critical condition.
3	err	Non-critical error condition.
4	warning	Warning condition.
5	notice	Normal but significant event.
6	info	Information event.
7	debug	Debugging-level message.

Sample Rules Section of rsyslog.conf

The **rsyslogd** service uses the facility and priority of log messages to determine how to handle them. This is configured by the file **/etc/rsyslog.conf** and by ***.conf** files in **/etc/rsyslog.d**. Programs and administrators can change **rsyslog.conf** and **by *.conf** files in **/etc/rsyslog.d**. Programs and administrators can change **rsyslogd** configuration in a way that will not be overwritten by updates to rsyslog by putting customised files with a.conf suffix in the **/etc/rsyslog.d** directory.

Analyse a syslog entry with a tail

The system logs written by rsyslog start with the oldest message on top and the newest message at the end of the log file. All log entries in log files managed by rsyslog are recorded in a standard format. The following example will explain the anatomy of a log file message in the **/var/log/secure** log file.

```
sudo tail -n 2 /var/log/secure
```

1. The timestamp when the log entry was recorded.
2. The host from which the log message was sent.
3. The program or process that sent the log message.
4. The actual message sent.

Send a syslog message with logger

The **logger** command can send messages to the **rsyslog** service. By default, it sends the messages to the facility user with severity notice (user.notice) unless specified otherwise with the -p option. It is especially useful to test changes to the rsyslog configuration. To send a message to rsyslogd that gets recorded in the **/var/log/boot.log** log file, execute:

Finding events with journalctl

The **systemd** journal stores logging data in a structured, indexed binary file. This data includes extra information about the log event. For syslog events, this can include the facility and priority of the original message, for example. The journalctl command shows the full system journal, starting with the oldest log entry when run as the root user:

```
journalctl
```

The journalctl command highlights in bold text messages of priority notice or warning, and messages of priority error and higher are highlighted in red.

The key to successfully using the journal for troubleshooting and auditing is to limit the journal searches to only show relevant output. In the following paragraphs, various different strategies to reduce the output of journal queries will be introduced.

By default, journalctl -n shows the last 10 log entries. It takes an optional parameter for how many of the last log entries should be displayed. To display the last 5 log entries, run:

```
journalctl -n 5
```

When troubleshooting problems, it is useful to filter the output of the journal by priority of the journal entries. The journalctl -p takes either the name or the number of the known priority levels and shows the given levels and all higher-level entries. The priority levels known to journalctl are debug, info, notice, warning, err, crit, alert and emerg.

To filter the output of the journalctl command to only list any log entry of priority err or above run:

```
journalctl -p err
journalctl -p debug
```

When looking for specific events, it is useful to limit the output to a specific time frame. The **journalctl** command has two options to limit the output to a specific time range, the **–since** and **–until** options. Both options take a time parameter in the format **YYYY-MM-DD hh:mm:ss**. If the date is omitted, the command assumes the date is today, and if the time part is omitted, the whole day starting 00:00:00 is assumed. Both options take **yesterday, today** and **tomorrow** as valid parameters in addition to the date and time field.

Output all journal entries that got recorded today:

```
journalctl –since today
```

Output the journal entries from **10th May 2020 20:30:00** to **11th May 2020 12:00:00**

```
journalctl –since “2020-05-20 20:30:00” –until “2020-05-11 12:00:00”
```

In addition to the visible content of the journal, there are fields attached to the log entries that can only be seen when verbose output is turned on. All of the displayed extra fields can be used to filter the output of a journal query. This is useful to reduce the output of complex searches for certain events in the journal.

```
journalctl -o verbose
```

Among the more useful options to search for lines relevant to a particular process or event are:

_COMM The name of the command
_EXE The path to the executable for the process
_PID The PID of the process
_UID The UID of the user running the process
_SYSTEMD_UNIT The systemd unit that started the process

Finding Events with JOURNALCTL

1. Output only systemd journal messages that originate from the systemd process that always runs with process id 1

```
journalctl _PID-1
```

2. Display all systemd journal messages that originate from a system service started with User-id 1000

```
journalctl _UID=1000
```

3. Output the journal messages with priority warning and above

```
journalctl -p warning
```

4. Create a journal query to all log events recorded in the previous 10 minutes

```
journalctl –since 14:40:00 –until 14:50:00
```

5. Display only the events originating from the sshd service with the system unit file sshd.service recorded since 14:30:00

```
journalctl –since 14:30:00
```

Maintaining Accurate Time

Set Local Clocks and Time Zone

Correct synchronised system time is very important for log file analysis across multiple systems. The Network Time Protocol (NTP) is a standard way for machines to provide and obtain correct time information on the Internet. A machine may get accurate time information from public NTP services on the Internet such as the NTP Pool Project. A high-quality hardware clock to serve accurate time to local clients is another option.

The timedatectl command shows an overview of the current time-related system settings, including current time, time zone and NTP synchronisation settings of the system.

```
timectl
```

A database with known time zones is available and can be listed with:

```
timectl list-timezones
```

The system setting for the current time zone can be adjusted as user root:

```
timectl set-time Asia/Kolkata
```

To change the current time and date settings with the timedatectl command, the set-time option is available. The time is specified in the “YYYY:MM:DD hh:mm:ss” format, where either date or time can be omitted. To change the time to 09:00:00, run:

```
timedatectl set-time 9:00:00
timedatectl
```

The set-ntp option enables or disables NTP synchronisation for automatic time adjustment. The option requires either a true or false argument to turn it on or off. To turn on NTP synchronisation, run:

```
timedatectl set-ntp true
timedatectl
```

Ch_11 Managing Red Hat Enterprise Linux Networking

Networking Concepts

IPV4 Networking

TCP/IP standards follow a four-layer network model specified in RFC1122.

Application: Each application has specifications for communication so that clients and servers may communicate across platforms. Common protocols include SSH (remote login), HTTPS (secure web), NFS or CIFS (file sharing) and SMTP (electronic mail delivery).
Transport: Transport protocols are TCP and UDP. TCP is a reliable connection-oriented communication, while UDP is a connectionless datagram protocol. Application protocols use TCP or UDP ports. A list of well-known and registered ports can be found in the /etc/services file. When a packet is sent on the network, the combination of the service port and IP address forms a socket. Each packet has a source and a destination socket. This information can be used when monitoring and filtering.
Internet: The internet or network layer, carries data from the source host to the destination host. Each host has an IP address and a prefix used to determine network addresses. Routers are used to connect networks. ICMP is a control protocol at this layer. Instead of ports, it has types. The ping utility is an example of using ICMP packets to test connectivity. Ping sends an ICMP ECHO_REQUEST packet. A successful ping may receive ICMP error messages such as “destination unreachable” or may not receive any response.
Link: The link or media access layer provides the connection to physical media. The most common types of networks are wired Ethernet (802.3) and wireless WLAN (802.11). Each physical device has a hardware address (MAC) which is used to identify the destination of packets on the local network segment.

IPv4 Addresses

An IPv4 address is a 32-bit number, normally expressed in decimal as four octets ranging in value from 0 to 255, separated by dots. The address is divided into two parts: the network part and the host part. All hosts on the same subnet, which can talk to each other directly without a router, have the same network part; the network part identifies the subnet. No two hosts on the same subnet can have the same host part; the host part identifies a particular host on a subnet.

In the modern Internet, the size of an IPv4 subnet is variable. To know which part of an IPv4 address is the network part and which is the host part, an administrator must know the

network which is assigned to the subnet. The netmask indicates how many bits of the IPv4 address belong to the subnet. The more bits that are available for the host part, the more hosts can be on the subnet.

The lowest possible address on a subnet (host part is all zeros in binary) is sometimes called the network address. The highest possible address on a subnet (host part is all ones in binary is used for broadcast messages in IPv4 and is called the broadcast address.

Network masks are expressed in two forms. The older syntax for a network which uses 24 bits for the network part would read 255.255.255.0. A newer syntax, called notation, would specify a network prefix of /24. Both forms convey the same information; namely, how many leading bits in the IP address contribute to its network address.

IP Addresses Range Table

Class	IP Address Ranges
A	1.0.0.1 to 126.255.255.254
B	128.1.0.1 to 191.155.255.254
C	192.0.1.1 to 223.255.254.254
D	224.0.0.0 to 239.255.255.255
E	240.0.0.0 to 254.255.255.254

	Private Address Range	
Class	**Start Address**	**Finish Address**
A	10.0.0.0	10.255.255.255
B	172.16.0.0	172.31.255.255
C	192.168.0.0	192.168.255.255

	Public Address Range	
Class	**Start Address**	**Finish Address**
A	10.0.0.0	126.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255

D	224.0.0.0	239.255.255.255
E	240.0.0.0	254.255.255.255

DHCP or static network configuration

Many systems are configured to obtain network settings automatically at boot time. The local configuration files indicate that DHCP should be used and a separate client service queries the network for a server and obtains a lease for network settings.

If a DHCP server is not available, the system must use a static configuration where the network settings are read from a local configuration file. The correct network settings are obtained from the network administrator or architecture team to ensure there are no conflicts with other systems.

Since DHCP uses the hardware address to track assignments, only one address may be assigned per interface with DHCP. Multiple static addresses may be assigned to a single interface. This practice is common in systems hosting services for multiple clients, such as HTTP IP-based hosting. Red Hat Enterprise Linux interfaces typically have an IPv4 address and an IPv4 local-link address but may have more addresses assigned.

Validating Network Configuration

Displaying IP Addresses

```
ip addr show eth0
```

1. An active interface has the status of UP.
2. The link line specifies the hardware (MAC) address of the device.
3. The inet line shows the IPv4 address and prefix.
4. The broadcast address, scope and device name are also on this line.
5. The inet6 line shows IPv6 information.

The UP command may also be used to show statistics about network performance. The received (RX) and transmitted (TX) packets, errors and dropped counters can be used to identify network issues caused by congestion, low memory and overruns.

```
ip -s link show eth0
```

Troubleshooting Routing

All packets destined for the 10.0.0.0/8 network will be sent directly to the destination through the device eth1. All packets destined for the 172.25.X.0/24 network will be sent directly to the destination through the device eth0. All other packets will be sent to the default router located at 172.25.X.254 and also through device eth0.

The ping command is used to test connectivity. The command will continue to run until CTRL+C is pressed unless options are given to limit the number of packets sent.

```
ping -c3 172.25.X.254
```

To trace the path to a remote host, use either traceroute or tracepath. Both commands can be used to trace a path with UDP packets; however, many networks block UDP and ICMP traffic. The traceroute command has options to trace the path with UDP (default), ICMP (-I) or TCP (-T) packets, but may not be installed by default.

```
tracepath access.redhat.com
```

Each line in the output of tracepath represents a router or hop that the packet passes through between the source and the final destination. Additional information is provided as available, including the round trip timing (RTI) and any changes in the maximum transmission unit (MTU) size.

Troubleshooting Ports and Services

```
ss -ta
```

1. The port used for SSH is listening on all IPv4 addresses. The "*" is used to represent "all" when referencing IPv4 addresses or ports.
2. The port used for SMTP is listening on the 127.0.0.1 IPv4 loopback interface.
3. The established SSH connection is on the 172.25.X.10 interface and originates from a system with an address of 172.25.254.254.
4. The port used for SSH is listening on all IPv4 addresses. The "::" syntax is used to represent all IPv6 interfaces.

Options for ss and netstat

Option	Description
-n	Show numbers instead of names for interfaces and ports.
-t	Show TCP sockets.
-u	Show UDP sockets.
-l	Show only listening sockets.
-a	Show all (listening and established) sockets.
-p	Show the process using the sockets.

Configuration Networking with nmcli

NetworkManager

NetworkManager is a daemon that monitors and manages network settings. In addition to the daemon, there is a GNOME Notification Area applet that provides network status information. Command-line and graphical tools talk to NetworkManager and save configuration files in the /etc/sysconfig/network-scripts directory.
A display a list of all connections, use nmcli con show. To list only the active connections, add the –active option.

```
nmcli con show
nmcli con show –active
```

Specify a connection ID (name) to see the details of that connection.

```
nmcli con show “ens160”
```

The nmcli command can also be used to show device status and details.

```
nmcli device status
nmcli dev show
nmcli dev show en160
```

Creating Network Connections with nmcli

Examples of creating new connections

Follow along with the next steps while your instructor discusses nmcli syntax.

1. Define a new connection named “default” which will autoconnect as an Ethernet connection on the eth0 device using DHCP.

```
nmcli con add con-name default type ethernet ifname ens224
```

2. Create a new connection named “static” and specify the IP address and gateway. Do not autoconnect.

```
nmcli con add con-name default type ethernet ifname ens224 autoconnect no type ethernet ipv4.addresses 192.168.10.11/24 gw4 192.168.10.1
```

```
nmcli con show
```

3. The system will autoconnect with the DHCP connection at boot. Change to the static connection.

```
nmcli con up static
nmcli con show
```

```
ip add | grep ens224
```

4. Change back to the DHCP connection.

```
nmcli con down static
nmcli con up default
nmcli con show
ip add | grep ens224
```

Modifying Network Interfaces with nmcli

An existing connection may be modified with **nmcli con and mod arguments.** The arguments are sets of key/value pairs. The key includes a setting name and a property name. Use nmcli con show “<ID>” to see a list of current values for a connection. The nm-settings(5) man page documents the setting and property names and usage.

```
nmcli con show static
```

Example of connection modifications

Follow along with the next steps while your instructor discusses nmcli syntax.

1. Turn on autoconnect

```
nmcli con modify static autoconnect yes
nmcli con show static | grep autoconnect:
```

2. Specify a DNS server

```
nmcli con show static | grep ipv4.dns
```

3. Some configuration arguments may have values added or removed. Add a +/- symbol in front of the argument. Add a secondary DNS server.

```
nmcli con modify static +ipv4.addresses 192.168.10.100/24
nmcli con show static | grep ipv4.addresses:
```

4. Add a secondary IP address without a gateway.

```
nmcli con modify static +ipv4.addresses 192.168.10.100/24
nmcli con show static | grep ipv4.addresses:
```

Deleting Network Interfaces with nmcli

```
nmcli con delete default
nmcli con show
```

Summary of nmcli Commands

Command	Use
nmcli dev static	List all devices.
nmcli con show	List all connections.
nmcli con up “<ID>”	Activate a connection
nmcli con down “<ID>”	Deactivate a connection. The connection will restart if autoconnect is yes.
nmcli dev dis <DEV>	Bring down an interface and temporarily disable autoconnect.
nmcli net off	Disable all managed interfaces.
nmcli con add …	Add a new connection.
nmcli con mod “<ID>” …	Modify a connection.
nmcli con del “<ID>”	Delete a connection.

Editing Network Configuration Files

Modifying Network Configuration

It is also possible to configure the network by editing interface configuration files.

Interface configuration files control the software interfaces for individual network devices. These files are usually named /etc/sysconfig/network-scripts/ifcfg-<name>, where <name> refers to the name of the device or connection that the configuration file controls. The following are standard variables found in the file used for static or dynamic configuration.

```
pwd
ls
cat ifcfg-static
```

In the static settings, variables for IP address, prefix and gateway have a number at the end. This allows multiple sets of values to be assigned to the interface. The DNS variable also has a number which is used to specify the order of lookup when multiple servers are specified.

Modified file

```
cat ifcfg-static
```

After modifying the configuration files, run nmcli con reload to make NetworkManager read the configuration changes. The interface still needs to be restarted for changes to take effect.

Configuring Host Names and Name Resolution

Changing The System Host Name

The hostname command displays or temporarily modifies the system's fully qualified host name.

```
hostname
```

A static host name may be specified in the /etc/hostname file. The hostnamectl command is used to modify this file and may be used to view the status of the system's fully qualified host name. If this file does not exist, the host name is set by a reverse DNS query once the interface has an IP address assigned.

```
sudo hostnamectl set-hostname master
hostnamectl
cat /etc/hostname
```

Configuring Name Resolution

The stub resolver is used to convert host names to IP addresses or the reverse. The contents of the file /etc/hosts are checked first.

```
cat /etc/hosts
```

Ch_13 Archiving and Copying Files Between Systems

Managing Compressed TAR Archives

What is TAR?

Archiving and Compressing files are useful when creating backups and transferring data across a network. One of the oldest and most common commands for creating and working with backup archives is the tar command.

With **tar**, users can gather large sets of files into a single file (archive). The archive can be compressed using **gzip**, **bzip2** or **xz** compression.

The tar command can list the contents of archives or extract their files to the current system. Examples of how to use the **tar** command are included in this section.

To use the **tar** command, one of the three following actions is required.

- c (create an archive)
- t (list the contents of an archive)
- x (extract an archive)

Commonly used options are:

- f file name (file name of the archive to operate on)
- v (verbosity; useful to see which files get added to or extracted from the archive)

Archive Files and Directories with tar

Before creating a tar archive, verify that there is no other archive in the directory with the same name as the new archive to be created. The tar command will overwrite an existing archive without any feedback.

The first option to use when creating a new archive is the c option, followed by the f option, then single space, then the file name of the archive to be created and finally the list of files and directories that should get added to the archive. The archive is created in the current directory unless specified otherwise.

In the following example, an archive named backup.tar is created with the contents of **file1**, **file2** and **file3** in the user's home directory.

```
touch file{1..3}
ls file*
tar cfv backup.tar file1 file2 file3
ls backup.tar
```

For tar to be able to archive the selected files, it is mandatory that the user executing the **tar** command is able to read the file(s). For example, creating an archive of the **/etc** folder and all of its content requires root privileges, because only root is allowed to read all of the files there. An unprivileged user could create an archive of the **/etc** folder, but the archive would omit files which do not include read permission for the user and it would omit directories which do not include both read and execute permission for the user.

Create the tar archive **/root/etc.tar** with the **/etc** directory as content as user root:

```
tar cf etc.tar /etc/
ls etc.tar
```

List contents of a tar archive

To list the content of an archive, the t and f options, accomplished by the archive to operate, are required.

List the content of the archive **/root/etc.tar**:

```
tar tf etc.tar
```

Extract An Archive Created with tar

A tar archive should normally be extracted in an empty directory to ensure it does not overwrite any existing files. If files are extracted by root, tar attempts to preserve the original user and group ownership of the files. If a regular user extracts files using tar, the extracted files are owned by that user.

Extract the archive **/root/etc.tar** to the **/root/etcbackup** directory:

```
mkdir etcbackup
cd etcbackup/
tar xf ../etc.tar
ls -l
```

Create A Compressed tar Archive

There are three different compression methods supported by the tar command. The gzip compression is the fastest and oldest one and is most widely available. The bzip2 compression usually leads to smaller archive files compared to gzip and is less widely available than gzip, while the xz compression method is relatively new, but usually offers the best compression ratio of the methods available.

It is good practice to use a single top-level directory, which can contain other directories and files, to simplify the extraction of the files in an organised way.

To create a compressed tar archive, one of the following tar options can be specified:

z for gzip compression (filename.tar.gz or filename.tgz)
j for bzip2 compression (filename.tar.bz2)
J for xz compression (filename.tar.xz)

Create (c option) a gzip-compressed (z option) tar archive **/root/etcbackup.tar.gz** of the /etc directory on server:

```
tar czf etcbackup.tar.gz /etc/
ll etc*
```

Create (c option) a bzip2-compressed (j option) tar archive **/root/logbackup.tar.bz2** of the **/var/log** directory on server:

```
tar cjf logbackup.tar.bz2 /var/log/
ls -l logbackup.tar.bz2
```

Create (c option) a xz-compressed (J option) tar archive **/root/ssconfig.tar.bz2** of the **/etc/ssh** directory on serverX:

```
tar cJf /root/sshconfig.tar.xz /etc/ssh
ls -l sshconfig.tar.xz
```

The first step when extracting a compressed tar archive is to determine where the archived files should be extracted to, then create and change to the target directory. To successfully

extract the archive, it is usually not necessary to use the same compression option used when creating the archive, as the **tar** command will determine which compression was used. It is valid to add the decompression method to the **tar** options as follows:

Extract (x option) the contents of a gzip-compressed (z option) tar archive named **/root/etcbackup.tar.gz** to the directory **/tmp/etcbackup**:

```
mkdir /tmp/etcbackup
cd /tmp/etcbackup
tar xzf /root/etcbackup.tar.gz
ls -l
```

Extract (x option) the contents of a bzip2-compressed (j option) tar archive named **/root/logbackup.tar.bz2** to the directory **/tmp/logbackup**:

```
mkdir /tmp/logbackup
cd /tmp/logbackup
tar xjf /root/logbackup.tar.bz2
ls -l
```

Extract (x option) the contents of a xz-compressed (J option) tar archive named **/root/sshbackup.tar.xz** to the directory **/root/sshbackup**:

Overview of tar options:

Option	Meaning
c	Create a new archive.
x	Extract from an existing archive.
t	List the contents of an archive.
v	Verbose; shows which files get archived or extracted.
f	File name; this option needs to be followed by the file name of the archive to use/create.
p	Preserve the permissions of files and directories when extracting an archive, without subtracting the umask.
z	Use gzip compression (.tar.gz).
j	Use bzip2 compression (.tar.bz2). bzip2 typically archives a better compression ratio than gzip.
J	Use xz compression (.tar.xz). Xz typically achieves a better compression ratio than bzip2.

Copying Files Between Systems Security

Copy Files to or from a remote location location with scp

The **ssh** command is useful for securely running shell commands on remote systems. It can also be used to securely copy files from one machine to another. The **scp** command transfers files from a remote host to the local system or from the local system to a remote host. It utilises the SSH server for authentication and encrypted data transfer.

Remote file system locations are always specified in the format **[user@]host:/path** for either the source or target location of the files to be transferred. The **user@** portion is optional and if it is missing the current local user that invokes the **scp** command is used. Before the transfer is initiated, the user must authenticate with the SSH server by password or SSH keys.

Copy file from local to the remote machine

```
scp backup.tar student@192.168.84.227:home/student
```

Copy file from remote to the local machine

```
mkdir remote
cd remote/
scp student@192.168.84.227:/home/student/backup.tar
ls -l
```

Copy files recursively

```
scp -r root@192.168.84.227:/var/log .
```

Transfer Files Remotely with sftp

If an interactive tool is preferred when uploading or downloading files to SSH server, the **sftp** command can be used. A session with **sftp** is similar to a classic FTP session, but uses the secure authentication mechanism and encrypted data transfer of the SSH server.

To initiate a **sftp** session, the **sftp** expects a remote location in the format **[user@]host**, where the user@ portion is option and if it is missing the user invoking the **sftp** command is used. To establish the **sftp** session, authenticating with any of the methods the SSH server accepts is necessary.

```
sftp student@192.168.84.227
```

The **sftp** session accepts various commands that work the same way on the remote file system as they do in the local file system, such as ls, cd, mkdir, rmdir and pwd. In addition, there are the **put** and **get** commands for uploading and downloading files. The exit command exits the **sftp** session.

Upload file into the remote machine

```
sftp> ls
```

```
sftp> put /etc/passwd
sftp> ls
```

Download files from remote to local machine

```
sftp> get /etc/yum/repos.d/redhat.repo
sftp> lls
```

Synchronise Files and Folders with rsync

The rsync tool is another way to securely copy files from one system to another. It differs from **scp** in that if two files or directories are similar between systems, rsync only needs to copy the differences between the systems, while **scp** would need to copy everything.

One of the advantages of **rsync** is that it can copy files between a local system and a remote system securely and efficiently. While the initial synchronisation only requires the differences to be copied over the network.

One of the most important options of rsync is the -n option to perform a dry run. A dry run is a simulation of what happens when the command really gets executed. It will display the changes it will perform when the command is executed without the dry run option. It is recommended to perform a dry run of any **rsync** operation to ensure no important files get overwritten or deleted.

The two most common options when synchronising files and folders with **rsync** are the **-a** and **-v** options. While the -v option adds verbosity to the output as the synchronisation proceeds, the -a options stands for “archive mode” and enables the following options all in one:

- -r: synchronise recursively the whole directory tree
- -l: synchronise symbolic links
- -p: preserve permissions
- -t: preserve timestamps
- -g: preserve group ownership
- -o: preserve the owner of the files
- -D: synchronise device files

While the **-a** already synchronises symbolic links, there are additional options necessary to preserve hard links, as they are treated as separate files instead. The -H option enables the handling of hard links, so the **rsync** command will identify the hard links present in the source folder and link the files accordingly in the destination folder instead of just copying them as separate files.

Sync with local machine’s directories

```
mkdir dir1 dir2
touch dir1/file{1..3}.txt
rsync -av dir1/dir2/
```

Sync with remote machine's directories

```
rsync -av dir1 192.168.84.227:/home/student/dir2
```

Ch_14 Installing and Updating Software Packages

Examining RPM Package Files

Examining Downloaded Packages with RPM

The RPM utility is a low-level tool that can get information about the contents of package files and installed packages. It gets its information from a local database or the package files themselves.

Task	Command
Display information about a package	rpm -q -i NAME
List all files included in a package	rpm -q -l NAME
List configuration files included in a package	rpm -q -c NAME
List documentation files included in a package	rpm -q -d NAME
Show a short summary of the reason for a new package release	rpm -q -changelog NAME
Display the shell scripts included in a page	rpm -q –scripts NAME

Working with RPM Package Files

1. Switch into the /mnt/disc/AppStream/packages and list all the packages and Choice any package.

```
cd /mnt/disc/AppStream
ls Packages/
```

```
rpm -q -a | wc
```

2. Check the package is installed or not

```
rpm -q firewalld
rpm -q httpd
```

3. Display information about a package

```
rpm -q -i firewalld
```

4. List all files included in a package

```
rpm -q -l firewalld
```

5. List configuration files included in a package

```
rpm -q -c firewalld
```

6. List documentation files included in a package

```
rpm -q -d firewalld
```

7. Display the shell scripts included in a package

```
rpm -q –scripts firewalld
```

8. Check all above info with the available package

```
rpm -q -i -p httpd-2.4.37-10.module+el8+2764+7127e69e.x86_64.rpm
```

9. Install the package using RPM command

```
rpm vsftpd-3.0.0-28.el8.x86_64.rpm
```

Managing Software with yum

Working with yum

yum is a powerful command-line tool that can be used to more flexibly manage (install, update, remove and query) software packages. Official Red Hat packages are normally downloaded from Red Hat's content distribution network. Registering a system to the subscription management service automatically configures access to software repositories based on the attached subscriptions.

Finding software with yum

- **yum help** will display usage information.
- **yum list** displays installed and available packages.

Task	Command
List installed and available packages by name	yum list [NAME-PATTERN]
List installed and available groups	yum grouplist
Search for a package by keyword	yum search KEYWORD

Show details of a package	yum info PACKAGE-NAME
Install a package	yum install PACKAGE-NAME
Install a package group	yum groupinstall “GROUP-NAME”
Update all packages	yum update
Remove a package	yum remove PACKAGE-NAME
Display transaction history	yum history

1. **List installed and available packages by name**

```
yum list | wc
yum list installed | wc
yum list available | wc
```

2. List installed and available groups

```
yum grouplist | wc
yum grouplist | wc
```

3. Search for a package by keyword

```
yum search httpd
```

4. Show details of a package

```
yum info firewalld
```

5. Install a package

```
yum install httpd
```

6. Install a package group

```
yum groupinfo “Security Tools”
yum groupinstall “Security Tools”
yum grouplist
```

7. Update all packages

```
yum update
```

8. Remove a package

```
yum remove httpd
```

9. Display transaction history

```
yum history
```

Managing Software with dnf
DNF is a software package manager that instals, updates and removes packages on RedHat 8 and is the successor to YUM (Yello-Dog Updater Modified). DNF makes it easy to maintain packages by automatically checking for dependencies and determining the actions required to install packages. This method eliminates the need to manually install or update the package and its dependencies using the rpm command. DNF is now the default software package management tool in RedHat 8. **We can use all yum commands with the dnf command.**

DNF aims at improving the bottlenecks of YUM viz., Performance, Memory Usages, Dependency Resolution, Speed and lots of other factors. DNF does Package Management using RPM, libsolv and Hawkey library. Though it does not come pre-installed in CentOS and RHEL 7 you can yum, dnf and use it alongside the yum.

```
ls -l /bin/yum
ls -l /bin/dnf
```

Why is Yum Replaced by DNF?

- Yum Package Manager has been replaced by DNF Package Manager since many long-standing issues in Yum remain unresolved.
- These problems include poor performance, excessive memory usage, slowdown for dependency resolution.
- DNF uses “libsolv” for dependency resolution, developed and maintained by SUSE to improve performance.
- It was written mostly in python and it has its own way of coping with dependency resolution.
- Its API is not fully documented and its extension system only allows Python plugins.
- Yum is a front-end tool for rpm that manages dependencies and repositories and then uses RPM to install, download and remove packages.
- Why would they want to build a new tool instead of fixing existing problems?
- Ales Kozamblak explained that the fixing was not technically feasible and that the yum team was not ready to accept the changes immediately.
- Also, the big challenge is that there are 56K lines for yum but only 29K lines for DNF.
- So there is no way to fix it, except the fork.

The Difference Between DNF and YUM

S. No	DNF (Dandified YUM)	YUM (Yellowdog Updater, Modified)
1	DNF uses libsolv for dependency resolution, developed and maintained by SUSE.	YUM uses the public API for dependency resolution

2	API is fully documented	API is not fully documented
3	It is written in C, C++, Python	It is written only in Python
4	DNF is currently used in Fedora, Red Hat Enterprise Linux 8 (RHEL), CentOS 8, OEL 8 and Mageia 6/7.	YUM is currently used in Red Hat Enterprise Linux 6/7 (RHEL), CentOS 6/7, OEL 6/7.
5	DNF supports various extensions	Yum supports only Python based extension
6	The API is well documented so it's easy to create new features.	It is very difficult to create new features because the API is not properly documented.
7	The DNF uses less memory when synchronising the metadata of the repositories.	The YUM uses excessive memory when synchronising the metadata of the repositories.
8	DNF uses a satisfiability algorithm to solve dependency resolution (It's using a dictionary approach to store and retrieve package and dependency information).	Yum dependency resolution gets sluggish due to the public API.
9	All performance is good in terms of memory usage and dependency resolution of repository metadata.	Overall performance is poor in terms of many factors.
10	DNF Update: If a package contains irrelevant dependencies during a DNF update process, the package will not be updated.	YUM will update a package without verifying this.
11	If the enabled repository does not respond, dnf will skip it and continue the transaction with the available repositories.	If a repository is not available, YUM will stop immediately.
12	dnf update and dnf upgrade equal.	It's different in yum
13	The dependencies on package installation are not updated.	Yum offered an option for this behaviour
14	Clean-Up Package Removal: When removing a package, dnf automatically removes any dependency packages not explicitly installed by the user.	Yum didn't do this
15	Repo Cache Update Schedule: By default, ten minutes after the system boots, updates to configured repositories are checked by dnf hourly. This action is controlled by the system timer unit named	Yum do this too.

	"/usr/lib/systemd/system/dnf-makecache.timer".	
16	Kernel packages are not protected by dnf. Unlike Yum, you can delete all kernel packages, including one that runs.	Yum will not allow you to remove the running kernell
17	libsolv: for solving packages and reading repositories. hawkey: hawkey, library providing simplified C and Python API to libsolv. librepo: library providing C and Python (libcURL like) API for downloading linux repository metadata and packages. libcomps: Limcomps is an alternative for yum.comps library. It's written in pure C as library and there's binding for Python2 and Python3	Yum does not use separate libraries to perform this function.
18	DNF contains 29k lines of code	Yum contains 56k line of code
19	DNF was developed by Ales Kozumplik	YUM was developed by Zdenek Pavlas, Jan Silhan and team members

Ch_15 Accessing Linux File Systems

Identifying File Systems and Devices

Storage Management Concepts

A file system is an organised structure of data-holding files and directories residing on a storage device, such a physical disk or partition. The file system hierarchy discussed earlier assembles all the file systems into one tree of directories with a single root, the /directory. The advantage here is that the existing hierarchy can be extended at any time by adding a new disk or partition containing a supported file system to add disk space anywhere in the system tree. The process of adding a new file system to the existing directory tree is called **mounting**. The directory where the new file system is mounted is referred to as a **mount point**. This is a fundamentally different concept than that used on a Microsoft Windows system, where a new file system is represented by a separate drive letter.

Hard disks and storage devices are normally divided up into smaller chunks called partitions. A partition is a way to compartmentalise a disk. Different parts of it can be formatted with different file systems or used for different purposes. For example, one partition could contain user home directories while another could contain system data and logs. If a user fills up the home directory partition with data, the system partition may still have space available.

Placing data in two separate file systems on two separate partitions helps in planning data storage.

Storage devices are represented by a special file type called block device. The block device is stored in the /dev directory. In Red Hat Enterprise Linux, the first SCSI, PATA/SATA, or USB hard drive detected is **/dev/sda**, the second is **/dev/sdb** and so on. This name represents the whole drive. The first primary partition on **/dev/sda** is **/dev/sda1**, the second partition is **/dev/sda2** and so on. But in RedHat 8 system partition structure is changed.

```
ls /dev/nv*
df -h
```

Another way of organising disks and partitions is with **logical volume management (LVM).** With LVM, one or more block devices can be aggregated into a storage pool called a volume group. Disk space is made available with one or more logical volumes. A logical volume is the equivalent of a partition residing on a physical disk. Both the volume group and the logical volume have names assigned upon creation. For the volume group, a directory with the same name as the volume group exists in the /dev directory. Below that directory, a symbolic link with the same name as the logical volume has been created. For example, the device file representing the mylv logical volume in the myvg volume group is **/dev/myvg/mylv.**

Making Links Between Files

Creating Hard Links

A hard link is a new directory entry with a reference to an existing file on the file system. Every file in a file system has one hard link by default. To save space, instead of copying a new hard link can be created to reference the same file. A new hard link either needs to have a different file name, if it is created in the same directory as the existing hard link, or it needs to reside in a different directory. All hard links pointing to the same file have the same permissions, link count, user/group ownerships, timestamps and file content. Hard links pointing to the same file content need to be on the same file system.

The **ls -l** shows the hard link count after the permissions and before the owner of a file.

```
echo “Hello World” > newfile.txt
ls -l newfile.txt
```

The command **ln** creates new hard links to existing files. The command expects an existing file as the first argument, followed by one or more additional hard links. The hard links can reside anywhere as long as they are on the same file system as the existing file. After a new hard link is created, there is no way to tell which of the existing hard links is the original one.

Create a hard link **newfile-link2.txt** for the existing file **newfile.txt**

```
ln newfile.txt newfile-hlink2.txt
ls -l newfile.txt newfile-hlink2.txt
```

Check inode number of both the files

```
ls -li newfile.txt newfile-hlink2.txt
```

Even if the original file gets deleted, the content of the file is still available as long as at least one hard link exists.

```
rm -f newfile.txt
ls -li newfile-hlink2.txt
```

Create soft links

The **ln -s** command creates a soft link, which is also called a “symbolic link”. A soft link is not a regular file, but a special type of file that points to an existing file or directory. Unlike hard links, soft links can point to a directory and the target to which a soft link points can be on a different file system.

```
ln -s newfile-hlink2.txt newfile-symlink.txt
ls -li newfile-hlink2.txt newfile-symlink.txt
```

When the original file gets deleted, the soft link is still pointing to the file but the target is gone. A soft link pointing to a missing file is called a “dangling soft link.”

```
rm -f newfile-hlink2.txt
ls -li newfile-symlink.txt
```

A soft link can point to a directory. The soft link then acts like a directory. Changing to the soft link directory with cd works as expected.

```
ln -s /etc configfiles/
cd configfiles/
```

Locating Files on the System

Tools for finding files

A system administrator needs tools for searching files matching certain criteria on the file system. This section discusses two commands that can search for files in the file system. The locate command searches a pre-generated database for file names or file paths and returns the results instantly. The find command searches the file system in real-time by crawling through the file system.

Locating Files by Name with locate

The **locate** command returns search results based on file name or path from the locate database. The database stores filename and path information.

When searching entries as a regular user, results are returned only for where the user invoking the locate search has read permissions on the directory tree that contains the matching element.

Search for files with “passwd” in the name or path in directory trees readable by user students on the server.

```
locate passwd
```

Results are returned even when the filename or path is only a partial match to the search query.

The -i option performs a case-insensitive search. With this option, all possible combinations of upper- and lowercase letters match the search.

```
locate -i lMage | head
```

The -n option limits the number of returned search results by location. The following example limits the search results returned by locating the first five matches.

```
locate -n 10 passwd
```

Searching For Files with Find

The find command performs a real-time search in the local file systems to find files that match the criteria of the command-line arguments. The **find** command is looking at files in the file system as your user account. The user invoking the find command must have read and execute permission on a directory to examine its contents.

The first argument to the **find** command is the directory to search. If the directory argument is omitted, **find** will start the search in the current directory and look for matches in any of the subdirectories.

To search the home directory of user student, five **find** a starting directory of **/home/student**. To search the entire system provide a starting directory of **/**.

With the name option

The -name option followed by a file name loops up files matching the given filename and returns all exact matches. To search for files named sshd_config in the / directory and all subdirectories on the server, run:

```
find / -name sshd_config
```

Wild cards are available to search for a file name and return all results that are a partial match. When using wildcards, it is important to quote the file name to look for to prevent the terminal from interpreting the wild card.

The following example searches for files in the / directory on the server that end in .txt:

```
find / -name '.txt' | head
```

To search for files in /etc/ that contain pass anywhere in their names on the server, run:

```
find / -name '*pass*' | head
```

To perform a case-insensitive search for a given file name, use the -iname option, followed by the file name to search. To search case-insensitively for files that have messages in their names in the / directory on the server, run:

find / iname '*message*' | head

With the uid and gid option

find can search for files based on their ownership or permissions. Useful options when searching by owner are **-user** and **-group**, which search by name and **-uid** and **-gid**, which search by ID.

Search for files owned by the user student in the **/home/student** directory on the server.

```
cd /home/student/
find -user student | head
```

Search for files owned by the group student in the **/home/student** directory on the server.

```
find -group student | head
```

Search for files owned by user ID 1000 in the **/home/student** directory on the server.

```
find -uid 1000 | head
```

Search for files owned by group ID 1000 in the **/home/student** directory on the server.

```
find -gid 1000 | head
```

Search for files owned by user root and group mail on the server machine.

```
find / -user root -group mail | head
```

.

With the Permission option

The **-perm** option is used to look for files with a particular set of permissions. Permissions can be described as octal values, with some combination of 4, 2 and 1 for **read, write** and **execute**. Permissions can be preceded by a / or - sign.

Numeric permission preceded by / will match files that have at least one bit of user, group or other for that permission set. A file with permissions **r–r–r–** does not match **/222**, but one with **rw–r–r** does. A - sign before permission means that all three instances of that bit must be on, so neither of the previous examples would match, but something like **rw-rw-rw-** would.
To use a more complex example, the following command would match any file for which the user has read, write and execute permissions, members of the group have read and write permissions and others have read–only access:

```
find /home -perm 664 | head
```

664 is exact permission

To match files for which the **user has at least write and execute permissions** and the **group has at least write permissions** and **others have at least read** access:

```
find /home -perm -334 | head
```

334 At least u=3, g=2, o=4

To match files for which the **use has read permissions or the group has at least read permissions or others have at least write access:**

```
find /home -perm /442 | head
```

442 At least u=4, g=4, o=2

When used with / or -, a value of **0** works like a wild card, since it means “permission of at least nothing.”

To match any file in the **/home/student** directory for which others have at least read access on the server, run:

```
find /home -perm -004 | head
```

With the Size Option

The find command can lookup files that match a size specified with the **-size** option, followed by a numerical value and the unit.

Units to be used with the -size options are:

- k, for kilobyte
- M, for megabyte
- G, for megabyte

Search for files with a size of exactly 1 megabyte.

```
find /home/ -size lM | head
```

Find files with a size of more that 2 MB

```
find /home/ -size +2M | head
```

List all files with a size less than 10 kilobytes.

```
find /home/ -size -10k | head
```

With the Minutes option

The -mmin option, followed by the time in minutes, searches for all files that had their content changed at exactly the given time in the past.

To find all files that had their file content changed exactly 120 minutes ago on the server, run:

```
find /etc/ -mmin +200 | head
```

The -modifier changes the search to look for all files in the **/etc** directory which have been changed less than 150 minutes ago.

```
find /etc/ -mmin -150 | head
```

With the Type Option

The -type option limits the search scope to a given file type, such as:

- f, for regular file
- d, for the directory
- l, for a soft link
- b, for block device

Search for all directories in the /etc folder on the server.

```
find /etc -type d | head
```

Search for all soft links on the server system.

```
find /etc -type l | head
```

Generate a list of all block devices in the **/dev** directory on the server.

```
find /dev -type b | head
```