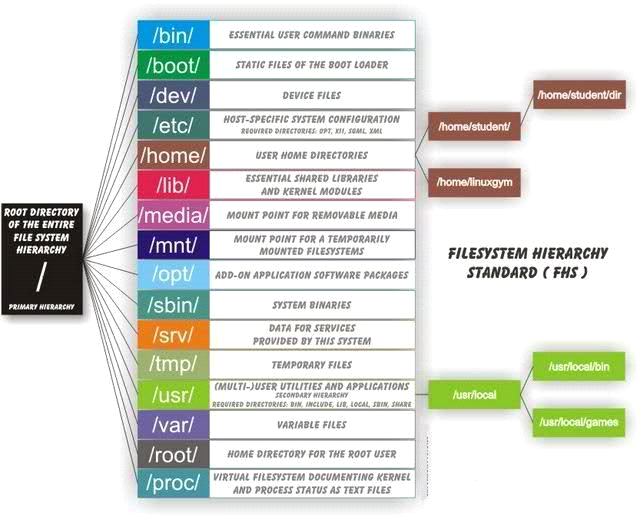
***Linux File System***

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



Each of the above directory (which is a file, at 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.

For everything other then home directory we require admin privileges that is we cant edit anything anywhere else other then home.

*In Linux files are hidden by placing a . in front of the filename. In order to see those files just press Ctrl+H.*

*With the help of bash\_history in /etc i can see all commands typed on that user or else i can go to the terminal and type history*

***/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.

***/boot :***

Holds important files during boot-up process, including Linux Kernel.

***/dev :***

Contains device files for all the hardware devices on the machine e.g., cdrom, cpu, etc

***/etc :***

Contains Application’s configuration files, startup, shutdown, start, stop script for every individual program.This is where bashrc files is stored which we need to configure during hadoop set up.

The /etc hierarchy contains confguration files. A "configuration file" is a local file used to control the operation of a program; it must be static and cannot be an executable binary.

***/home*** *:*

Home directory of the users. Every time a new user is created, a directory in the name of user is created within home directory which contains other directories like Desktop, Downloads, Documents, etc.

***/lib :***

The Lib directory contains kernel modules and shared library images required to boot the system and run commands in root file system. This contains files like .dll in windows.

***/lost+found :***

This Directory is installed during installation of Linux, useful for recovering files which may be broken due to unexpected shut-down.

***/media :***

Temporary mount directory is created for removable devices viz., media/cdrom.

***/mnt :***

Temporary mount directory for mounting file system.

***/opt :***

Optional is abbreviated as opt. Contains third party application software. Viz., Java, etc.

Third party application packages which does not conform to the standard Linux file hierarchy can be installed here.

***/proc :***

A virtual and pseudo file-system which contains information about running process with a particular Process-id aka pid.

***/root :***

This is the home directory of root user and should never be confused with ‘/‘. This is root user directory its different from /(root).

***/run :***

This directory is the only clean solution for early-runtime-dir problem.

***/sbin :***

Contains binary executable programs, required by System Administrator, for Maintenance. Viz., iptables, fdisk, ifconfig, swapon, reboot, etc.

***/srv :***

Service is abbreviated as ‘srv‘. This directory contains server specific and service related files.

***/sys :***

Modern Linux distributions include a /sys directory as a virtual filesystem, which stores and allows modification of the devices connected to the system.

***/tmp :***

System’s Temporary Directory, Accessible by users and root. Stores temporary files for user and system, till next boot.

***/usr*** *:*

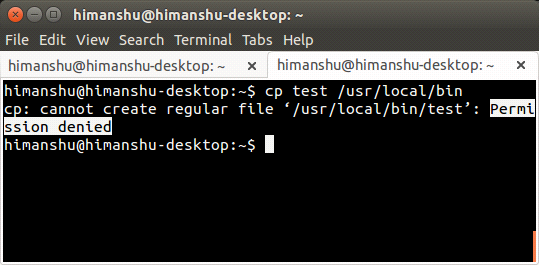
Contains executable binaries, documentation, source code, libraries for second level program. "usr" stands for universal system resources

***/var :***

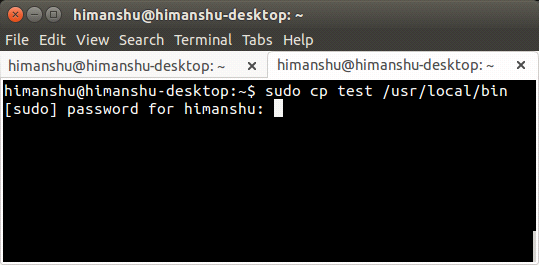
Stands for variable. The contents of this file is expected to grow. This directory contains log, lock, spool, mail and temp files.

***sudo in linux***

Ever got a 'Permission denied' error while working on the Linux command line? Chances are that you were trying to perform an operation that requires root permissions. For example, the following screenshot shows the error being thrown when I was trying to copy a binary file to one of the system directories:



So what's the solution to this problem? Simple, use the sudo command.



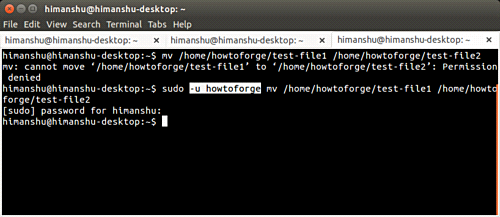
The user who is running the command will be prompted for their login password. Once the correct password is entered, the operation will be performed successfully.

While sudo is no doubt a must-know command for any and everyone who works on the command line in Linux, there are several other related (and in-depth) details that you should know in order to use the command more responsibly and effectively. And that's exactly what we'll be discussing here in this article.

The sudo command, as most of you might already know, is used to execute a command with elevated privileges (usually as root). An example of this we've already discussed in the introduction section above. However, if you want, you can use sudo to execute command as some other (non-root) user.

This is achieved through the -u command line option the tool provides. For example, in the example shown below, I (himanshu) tried renaming a file in some other user's (howtoforge) home directory, but got a 'permission denied' error. And then I tried the same 'mv' command with 'sudo -u howtoforge,' the command was successful:

*Here in sudo -u , "-u" stands for user. So this statements is like for the user howtoforge i need extra privileges to move file from other user location to howtoforge.*

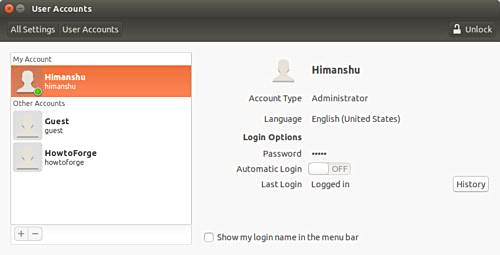


***Can anyone use sudo??***

No. For a user to be able to use sudo, an entry corresponding to that user should be in the /etc/sudoers file. The following paragraph - taken from Ubuntu's website - should make it more clear:

*"The /etc/sudoers file controls who can run what commands as what users on what machines and can also control special things such as whether you need a password for particular commands. The file is composed of aliases (basically variables) and user specifications (which control who can run what)."*

If you are using Ubuntu, it's easy to make sure that a user can run the sudo command: all you have to do is to make that user account type 'administrator'. This can be done by heading to System Settings... -> User Accounts.



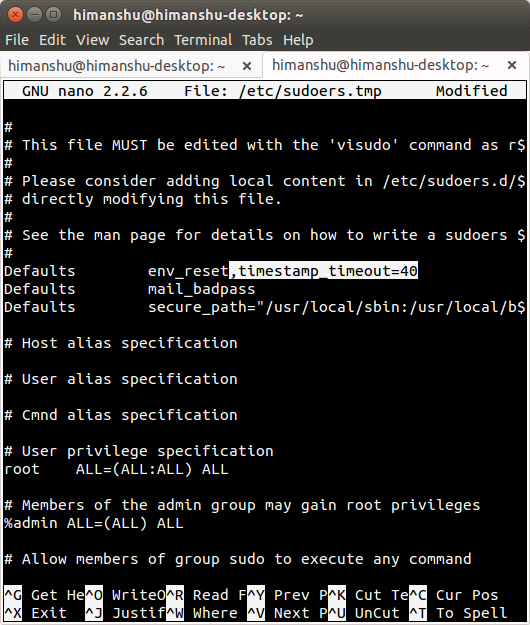
Then selecting the user whose account type you want to change, and then changing the type to 'administrator'

However, if you aren't on Ubuntu, or your distribution doesn't provide this feature, you can manually edit the /etc/sudoers file to make the change. You'll be required to add the following line in that file:

*[user] ALL=(ALL:ALL) ALL*

Needless to say, [user] should be replaced by the user-name of the account you're granting the sudo privilege. An important thing worth mentioning here is that the officially suggested method of editing this file is through the visudo command - all you have to do is to run the following command:

*sudo visudo*



*The -s option*

There might be times when you work requires you to run a bucketload of commands that need root privileges, and you don't want to enter the sudo password every now and then. Also, you don't want to tweak the sudo session timeout limit by making changes to the /etc/sudoers file.

In that case, you may want to use the -s command line option of the sudo command.

*su root*

We can use this command to login to root.

*Important note*

For security purposes only, the root user or members of the sudo group may transfer ownership of a file.

***Home directory vs root directory in linux***

*Root directory which is referred to as / (a slash) is the topmost level of the system drive while Home directory which is /Users/<short username> (also referred to as ~)* is the directory where the folders like Documents, Music, Pictures reside.

In Linux which also follows a similar convention has / as the root and */home/<username> (referred to as ~) is home.*

While C: is the Windows-counterpart to root and C:\Users\<username> is the Home or User folder.

***~/ (tilde slash)***

The tilde (~) is a Linux "shortcut" to denote a user's home directory. Thus tilde slash (~/) is the beginning of a path to a file or directory below the user's home directory.

For example, for user01, file /home/user01/test.file can also be denoted by ~/test.file.

That construct can be used in Linux commands, for example: ls -al ~/test.file.

Similary ./ will denote current working directory instead of home directory.

***$ and # symbol***

"$" - is just a sign of the shell prompt, means that shell is ready to accept commands, you can understand it as a separator after which, you can interact with a shell. Can also be "#" which shows that root is the user who's session is going on.

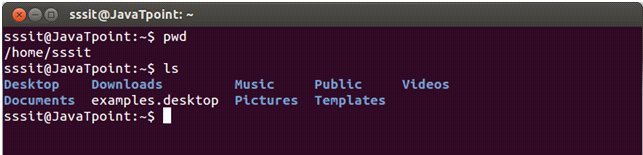
***Listing Directories and Files***

All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.

You can use the ls command to list out all the files or directories available in a directory. Following is the example of using ls command with -l option.

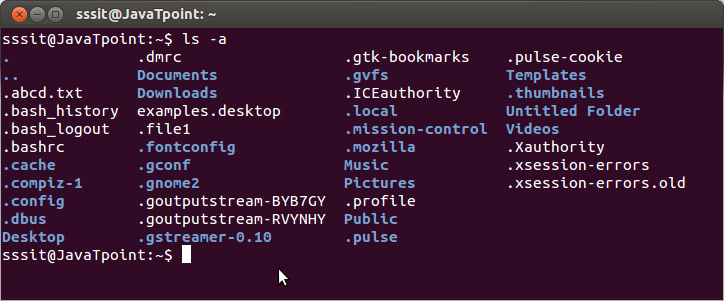
*$ ls*

It will list all the files and directories in current working directory.



*$ls -a*

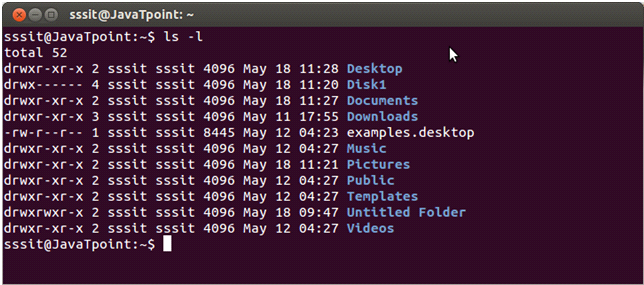
It will give you the whole list of a directory including the hidden files also. In Linux, hidden files start with a dot (.) and can't be seen in regular directory.



*$ls -l*

The ls command will only display the files. But if you want your files to displayed in a long list format then you be can use ls -l command.

we can combine ls -al together to get all files in long list format including the hidden files.



*ls -lh*

This command will show you the file sizes in human readable format. Size of the file is very difficult to read when displayed in terms of byte. The (ls -lh)command will give you the data in terms of Mb, Gb, Tb, etc.

*ls -lhS*

If you want to display your files in descending order (highest at the top) according to their size, then you can use (ls -lhS) command.

*ls -R*

It will display the content of the sub-directories also

*ls ~*

It wildisplay the content of the home directory.l

***~/.bashrc file***

The .bashrc file is a shell script which is run every time a user opens a new shell.

Every time you open a terminal window the bashrc file is performed.

The .bashrc file is a good place therefore to run commands that you want to run every single time you open a shell.

As an example open the .bashrc file using nano as follows:

nano ~/.bashrc

At the end of the file enter the following command:

echo "Hello $USER"

Save the file by pressing CTRL and O and then exit nano by pressing CTRL and X.

Within the terminal window run the following command:

bash

The word "Hello" should be displayed along with the username you are logged in as.

The .bashrc file is commonly used to set aliases to commonly used commands so that you don't have to remember long commands.

Some people consider this a bad thing because you could forget how to use the real command when placed on a machine where your own particular .bashrc file doesn't exist.

***What if ./bashrc file is deleted.***

If you've completely erased your ~/.bashrc, you can replace it by copying /etc/skel/.bashrc (This is the template that the user creation script used to create your bashrc).

***source ~/.bashrc***

We need to throw this command after editing the bashrc in order to save the changes we have made.

***Linux chown: change owner***

Command chown is used to change the owner of the file.

Syntax:

chown <newOwner> <fileName>

Example:

***chown jtp list***

Command chown can also be used to change both user owner and group.

Syntax:

chown <newOwner:newGroup> <fileName>

Example:

***chown jtp:php msg.txt***

Chaning ownership recrusively that is ownership of all contents of folders will also get changed.

Syntax:

chown -R <newOwner:newGroup> <fileName>

Example:

***chown -R jtp:php msg.txt***

***Linux File Permissions***

Ownership of Linux files

Every file and directory on your Unix/Linux system is assigned 3 types of 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 permissions 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 he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred 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 behavior.

Let us understand the Permission system on Linux.

***Permissions***

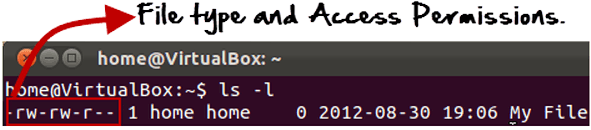
Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

***Read:*** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.

***Write:*** The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the 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.

*Run: ls -l*



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 '-' implies that we have selected a file.p>



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

Image

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:

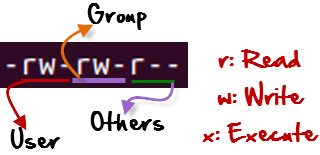
*Read the file  
Write or edit the file  
He cannot execute the file since the execute bit is set to '-'*

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 -

*Absolute mode  
Symbolic mode*

***Absolute mode***

The table below gives numbers for all for permissions types.

*Number Permission Type Symbol*

*0 No Permission ---*

*1 Execute --x*

*2 Write -w-*

*3 Execute + Write -wx*

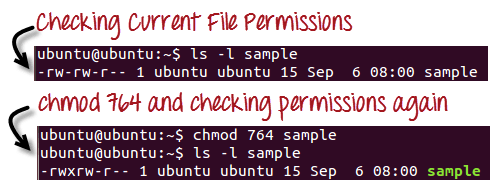
*4 Read r--*

*5 Read + Execute r-x*

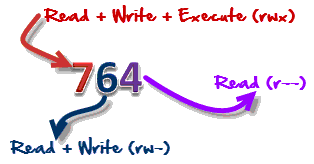
*6 Read +Write rw-*

*7 Read + Write +Execute rwx*

Let's see the chmod command in action.



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



'764' absolute code says the following:

Owner can read, write and execute  
Usergroup can read and write  
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 change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

*Operator Description*

*+ Adds a 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/owner*

*g group*

*o other*

*a all*

