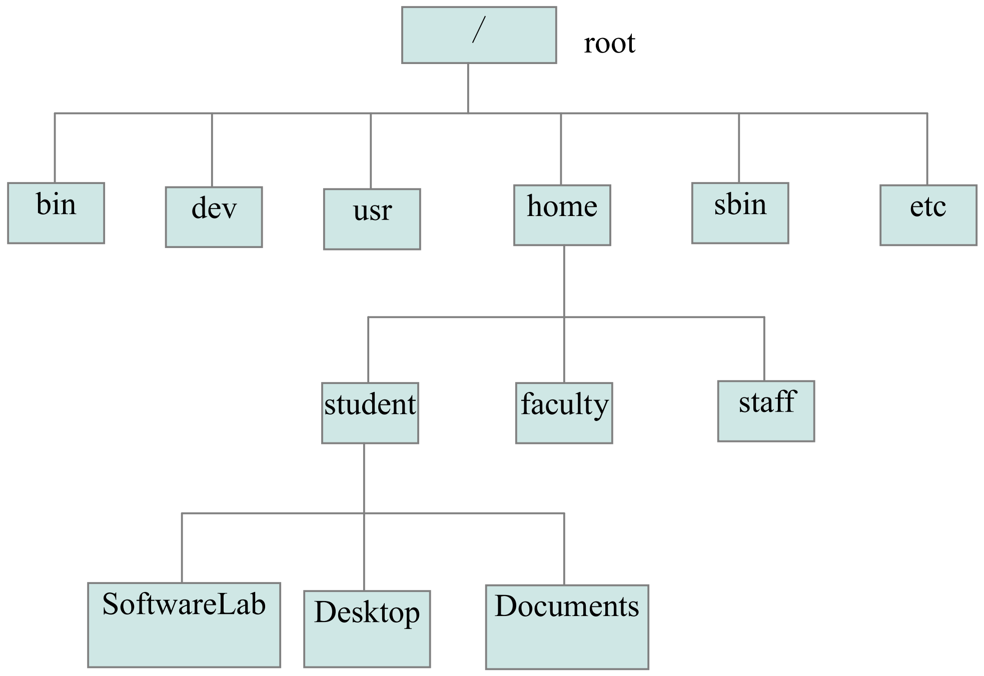
Linux Tutorial

Linux Directory Structure:



Users of the Linux operating system will be given user name and password to login and access the facilities of operating system. One of the most important users of the operating system is having the user name as root and he will have all the privileges and permissions to access/modify all the functionalities/settings of the operating system. Other users are normal users and may have only limited access permissions.

Linux Directory Structure is a tree like hierarchical structure which is composed of many directories and files. This tree like structure starts with the root (/) directory at the top. The root directory may contain many sub directories like bin, dev, etc, home, sbin, etc and so on. Each of these sub directories may contain sub directories and so on. The purpose of these sub directories is specified as below.

/boot - contains the files and information needed to boot the Linux operating system.
/usr – is the directory that contains the different user applications
/bin, /usr/bin – contains the important programs (binary files) of the applications like shells and binaries related to the Linux commands and etc.
/sbin, /usr/sbin – contains the system admin programs which can only be run by the root user

/home – is the directory contains the home directories of different users
/root – home directory of the root user
/dev – contains the devices that are available to the system
/etc – contains the configuration files of the system.

For More information about the directory structure refer to the following link.
http://www.codepuppet.com/directory-structure-in-linux/

Every file and directory of the Linux operating system can uniquely addressed using the complete hierarchical path (**absolute path**) of the file or directory starting from the root.

- For Example absolute path of the SoftwareLab directory is /home/student/SoftwareLab

Similarly every file or directory can be uniquely identified relative to some other directory using the **relative path** of the directory.
For Example

- Relative path of the /dev directory from SoftwareLab directory is ../../../dev (where ../ stands for reference to the parent directory)
- Relative path of the SoftwareLab directory from /home directory is student/SoftwareLab

Linux Commands:
Every Linux command has three parts.
<COMMAND>   [OPTIONS]   [ARGUEMENTS]
Each command takes list of options (optional) followed by the list of arguments (optional). Command, options and arguments are always separated by space character otherwise command will not be executed by the shell.
Linux is a **case-sensitive**. Commands, options, arguments, file/directory names should be spelled with correct case.
who – displays the details of the users who are all logged into the machine. (Since Linux is a multi-user operating system more than one user can login to the machine at the same time)
whoami or who am i – displays the current user details
pwd – print working directory. This command prints the absolute path of the current working directory of the user
man – manual command displays the manual (help) document of the corresponding Linux command.

Usage: man <COMMAND>

- For example **man pwd** will display the manual related to the command pwd.

ls – Lists the contents of a directory

Usage: ls [OPTIONS]   [ARGUEMENTS]

Some of the OPTIONS:

- **-l** – long listing
- **-t** – lists by modification time
- **-s** – lists by the size of the files/directories
- **-a** – lists all the files including hidden files
- **-1** – lists one file per line
- **-m** – comma separated list of entries
- **-R** – list sub directories recursively
- **-r** – lists in reverse order

Examples:

- **ls** – displays the list of files and directories of the current working directory
- **ls /home/student/SoftwareLab** – displays the list of files and directories of the directory /home/student/SoftwareLab
- **ls -l** - displays the long list of the files and directories. Long list includes:
  - ❖ 1st character specifies whether the file is a file or a directory d for directory and - (dash) for file
  - ❖ next 9 characters specifies the file permissions (r – read, w – write, x – execute) for the owner, group and others respectively
  - ❖ next numeric value indicates the number of links or directories inside that directory
  - ❖ next arguments corresponds to owner, group, size of the file in bytes, last modification date and time and the name of the file respectively.
- **ls -a** – displays all the files including hidden files of the current working directory
- **ls -a /home/student** – displays the all the files including hidden files of the directory **/home/student**
- **ls -R** – displays the contents of the directory and sub directories of the current directory recursively.
- **ls -la** – displays all the contents (including hidden files) of the current working directory

Note: More than one options can be combined together as we did in the previous example combining -l and -a options we can use -la instead of -l -a

mkdir – making a directory

Usage: mkdir [OPTIONS] <DIRECTORY_NAME>

OPTIONS:

- **-p** – creates directories hierarchically

Examples:

- **mkdir DIR1** – creates directory with the name DIR1 in the current directory
- **mkdir -p D1/D2/D3** – creates directory structure D1/D2/D3 in the current directory

cd – change directory

Usage: cd <DIRECTORY_PATH>

Examples:

- **cd ~** - change the directory to the home directory of the user from where ever you are
- **cd -** - changes the directory to the previous working directory from the current working directory
- **cd ..** - changes the directory to the parent directory of the current directory (moves one directory up in the directory structure)
- **cd ../../../** - moves three directories up in the directory structure
- **cd .** - . (dot) represents the current directory
- **cd /home/student/SoftwareLab** – changes the directory to /home/student/SoftwareLab from the current directory

Note: If the directory does not exist (or the name of the directory is incorrectly spelled) Linux shell will give an error message and change of directory will not be successful.

Text Editors:

Vi Editor:

Vi editor is a simple shell based text editor. This will be available in any Linux/Unix OS by default.

Usage: vi <filename> - To create a text file with your own file name

Vi editor has two modes of operation:

1. Insert Mode – To edit/create the text (Pressing ESC key changes the mode from Insert mode to Command Mode)
    - In order to create a text file (enter text into a file) you should first in Insert mode

- Pressing INSERT key enables Insert Mode.
- Once you are in Insert mode you can type/remove/modify the text contents of the file.

2. Command Mode – To execute some commands on the text.
    - When you are in command mode every character typed is a command to the text editor.
    - **:x <ENTER>** – To save and close the text file
    - **:wq<ENTER> -** To save and close the text file
    - **:w<ENTER> -** To save the text file
    - **:q<ENTER> -** To exit the text editor.
    - **:q!<ENTER> -** To close the text editor (ignoring the changes that are not saved)
    - **u** – Undo the recent changes
    - Pressing INSERT key changes mode from Command mode to Insert Mode.

For More information Refer http://www.cs.colostate.edu/helpdocs/vi.html

gedit is a GUI based text editor.

Usage: gedit [FILE_NAME]

Examples:

- **gedit** – invokes the text editor with unnamed document (you can type in this document and later you give some name while you are saving the file)
- **gedit Hello.c** - opens the text file Hello.c which is present in the current directory. If Hello.c file does not exist in the current directory editor opens a empty document with the name Hello.c which you can add some text and save
- **gedit <FILE_NAME> &** - invoking the text editor in the background

Note: Any Linux command can be run in back ground by appending & symbol at the end of the command.

cat – concatenate files

Usage: cat <List of space separated file names>

Examples:

- **cat Hello.c** – displays contents of the file Hello.c which is located in the current directory. If the file is not exist or wrongly spelled the shell will produce an error message saying the file does not exist.
- **cat Hello.c Sample.c Test.c** – displays the contents of the files Hello.c followed by

Sample.c followed by Test.c

Note: Whenever we are using file name or directory name as an argument, we can use an absolute path or a relative path of the file or directory

tac – reverse concatenations

Usage: tac <List of space seperated file names>

Very similar to cat command but displays the contents of the file in reverse order of the lines

more – displays the contents of the files allowing scrolling downwards using the Enter key

Usage: more <FILE_NMAE>

less - displays the contents of the files allowing scrolling upward and downwards using the up and down arrow keys

Usage: less <FILE_NMAE>

head – displays the top few lines of the text file

Usage: head -n <FILE_NMAE>

Example:

- **head -10 Hello.c** – displays the top 10 lines of the file Hello.c

tail – displays the bottom few lines of the text file

Usage: tail -n <FILE_NMAE>

Example:

- tail -20 Test.c – displays the bottom 20 lines of the file Test.c

cp – used to create copies of files and directory

Usage: cp [OPTIONS] <SOURCE_FILE> <DESTINATION_FILE>

OPTIONS:

- **-i** – interactive copy, prompts the user before over writing a file
- **-f** – force copy, no prompt while over writing a file
- **-r** – recursively copies the directories.

Examples:

- **cp Hello.c Test.c** – creates a copy of the file Hello.c in the current directory with the name Test.c
- **cp Hello.c /home/student/SoftwareLab/** - creates a copy of the file Hello.c in the directory /home/student/SoftwareLab/ with the same name
- **cp -rf SoftwareLab SoftwareLabCopy** – creates a copy of the SoftwareLab directory (with entire files sub directories) with the name SoftwareLabCopy and the copy is force

(no prompt will be given if there is any over write of the file)

Note: If the destination directory is already present then copy will be created inside the destination directory.

mv – move files from one place to another (or to rename files or directories)

Usage: mv [OPTIONS] <SOURCE_FILE> <DESTINATION_FILE>

OPTIONS:

- **-i** – interactive move, prompts the user before over writing a file
- **-f** – force move, no prompt while over writing a file

Examples:

- **mv Hello.c Test.c** – renames the file Hello.c to Test.c
- **mv Hello.c /home/student/SoftwareLab/** - moves the file Hello.c from the current directory to the directory /home/student/SoftwareLab/
- **mv SoftwareLab SoftwareLabMove** – Renames the directory SoftwareLab to SoftwareLabMove if SoftwareLabMove directory does not exists already. Else it will move the directory SoftwareLab inside the SoftwareLabMove directory

rm – removes files and directory

Usage: rm [OPTIONS] <List of Files/directories>

OPTIONS:

- **-i** – interactive removal, prompts the user before removing a file/directory
- **-f** – force removal, no prompt while removing a file
- **-r** – recursively removes the directories.

Examples:

- **rm Hello.c** – removes the file Hello.c without any prompt
- **rm -i Hello.c** – removes the file Hello.c after the prompt (If user wishes to remove)
- **rm -rf SoftwareLab** – removes the entire directory SoftwareLab without any prompt

rmdir – removes empty directory

Usage: rmdir [OPTIONS] <DIRECTORY_NAME>

OPTIONS: -p - removes the hierarchy of directories

Examples:

- **rmdir DIR1** – removes the empty directory DIR1. If the directory is not empty removal will not be successful

- **rmdir -p D1/D2/D3** – removes hierarchy of empty directories

Wild cards:

We can use verity of wild card characters where ever we need to give file names or directory names as arguments to a command.

Wild card Characters:

- * - matches zero or more characters
- ? - matches exactly one character
- [ ] - defines a class of characters (- for range, ! To exclude)
    - ❖ [abc] – any one character from a, b, c
    - ❖ [a-z] – any one character from small alphabets a to z
    - ❖ [a-zA-Z] – any one character from the alphabets a to z or A to Z
    - ❖ [!abc] – any one character other than a, b ,c

Examples:

- **rm -ir *** - removes all the files and directories from the current directory after prompting the user
- **ls -l 12MCMC[0-9][0-9]** – long lists all the files starting with 12MCMC and has digits as last two characters
- **cat 12MCMC*** - concatenates all the files whose names are starting with 12MCMC
- **cat Hello.?** - displays contents all the file whose name starts with Hello and has any single letter extension

chmod – change a file permissions

Usage: chmod [OPTIONS] <OCTAL_MODE> <FILE_NAME>

OPTIONS: -R – recursive change of permissions for a directory

Examples:

- **chmod 666 Hello.c** – gives read and write permissions to owner, group and others but no execution permission for all. (octal 666 in binary 110 110 110, rw- rw- rw-)
- **chmod -R 666 SoftwareLab** - gives read and write permissions to owner, group and others but no execution permission for all for all the files and directories inside the directory SoftwareLab

top – displays all the processes running on machine. It is very similar to task manager in windows environment

find – searches for files in a directory or hierarchy

Usage: find <DIRECTOR_PATH> [OPTIONS] <EXPRESSION>

OPTIONS:

- **-name** – matches expression in the file names

Example:

- **find . -name *.c** - finds all the .c files in the current directory(and sub directories)
- **find . -name Hell.c** - finds the file Hello.c in the current directory(and sub directories)
- **find /home/student/SoftwareLab -name Test.c** – finds the file Test.c in the directory /home/student/SoftwareLab and its subdirectories

grep – prints lines matching a pattern in a file (files)

Usage: grep [OPTIONS] PATTERN <LIST OF FILES>

OPTIONS:

- **-i** - ignore case while matching the text
- **-v** – invert the match (prints lines which does not match the pattern)
- **-c** – counts the number of lines matches the pattern
- **-l** – lists the file names which has the matched lines instead of printing the lines
- **-r** – recursively match in files and sub directories in a directory

Examples:

- **grep programming File.txt** - prints all the lines which contains the word programming in the file File.txt
- **grep -i File.txt** - prints all the lines which contains the word programming (ignoring the case) in the file File.txt
- **grep -v programming File.txt** - prints all the lines which does not contains the word programming in the file File.txt
- **grep -c programming File.txt** - prints number of lines which contains the word programming in the file File.txt
- **grep -ir 12MCMC01 /home/student/SoftwareLab** – displays all the lines in all the files of the directory /home/student/SoftwareLab which contains the word 12MCMC01 (ignoring the case)
- **grep main *.c** – display all the lines which contains the word main in the all the .c files

in current directory

Redirection Operators:

Input can be given to any command using the input redirection operator <

Output of any Linux command can be saved in a file using Linux output redirection operators > and >>

Examples: (Input Redirection)

- **cat < Hello.c** – it is equal to cat Hello.c

Examples: (Output Redirection)

- **ls -l > output** – output of the command ls -l will be saved in the file output in the current directory. If the file output is already exists contents will be overwritten.
- **ls -l >> output** - output of the command ls -l will be appended to the file output in the current directory. If the file output does not exist the command will produce an error.

Pipe operators (|):

Output of one command can be redirected as input to the another command using pipe (|) operator

Example:

- **cat Hello.c | grep main** – cat command will gives all the lines of the file Hello.c as input to the grep command and then grep command prints all the lines which has the word main