Command-Line

1. **ls -a** : To list all the files including the hidden files
2. **cat**: used to type out a file (or combine files)
3. **head**: used to show the first few lines of a file.
4. **tail**: used to show the last few lines of a file.
5. **man**: used to view documentation.
6. **|** : The pipe symbol (|) used to have one program take as input the output of another.
7. Most input lines entered at the shell prompt have three basic elements. They are:
    - Command
    - Options
    - Arguments
8. The command is the name of the program you are executing. It may be followed by one or more options (or switches) that modify what the command may do. Options usually start with one or two dashes, for example, -p or --print, in order to differentiate them from arguments, which represent what the command operates on.
9. **sudo**: sudo allows users to run programs using the security privileges of another user, generally root (superuser).
10. Some systems might be having sudo installed. In that case we need to install the sudo manually

    Steps to install sudo:

    1. You will need to make modifications as the administrative or superuser, root. While sudo will become the preferred method of doing this, we do not have it set up yet, so we will use su instead. At the command line prompt, type su and press Enter. You will then be prompted for the root password, so enter it and press Enter.
       $ su Password:
       #
    2. Now, you need to create a configuration file to enable your user account to use sudo. Typically, this file is created in the /etc/sudoers.d/ directory with the name of the file the same as your username. For example, for this demo, let's say your username is student. After doing step 1, you would then create the configuration file for student by doing this
       # echo "student ALL=(ALL) ALL" > /etc/sudoers.d/student
    3. Finally, some Linux distributions will complain if you do not also change permissions on the file by doing the below
       # chmod 440 /etc/sudoers.d/student

Turning Off the Graphical Desktop:

1. Linux distributions can start and stop the graphical desktop in various ways.
2. To stop the graphical display manager(gdm) run the below command
   sudo systemctl stop gdm(or lightdm)
   or
   sudo telinit 3

3. To restart (after logging into the console) run the below command
   sudo systemctl start gdm(or lightdm)
   Or
   sudo telinit 5

Rebooting and Shutting Down:

1. The preferred method to shut down or reboot the system is to use the shutdown command. This sends a warning message, and then prevents further users from logging in.
2. The init process will then control shutting down or rebooting the system. It is important to always **shut down** properly; failure to do so can result in damage to the system and/or loss of data.
3. The **halt** and **poweroff** commands issue shutdown -h to halt the system; reboot issues shutdown -r and causes the machine to reboot instead of just shutting down.
4. Both rebooting and shutting down from the command line requires superuser (root) access.
   sudo shutdown -h 10:00 "Shutting down for scheduled maintenance."

Locating Applications:

1. In general, executable programs and scripts should live in the /bin, /usr/bin, /sbin, /usr/sbin directories, or somewhere under /opt. They can also appear in /usr/local/bin and /usr/local/sbin, or in a directory in a user's account space, such as /home/student/bin.
2. To find out exactly where the diff program resides on the filesystem use the below command
   Ex: which diff
   o/p: /usr/bin/diff
3. If which does not find the program, **whereis** is a good alternative because it looks for packages in a broader range of system directories
   Ex: whereis diff
   o/p: diff: /usr/bin/diff /usr/share/man/man1/diff.1.gz /usr/share/man/man1p/diff.1p.gz

Accessing Directories:

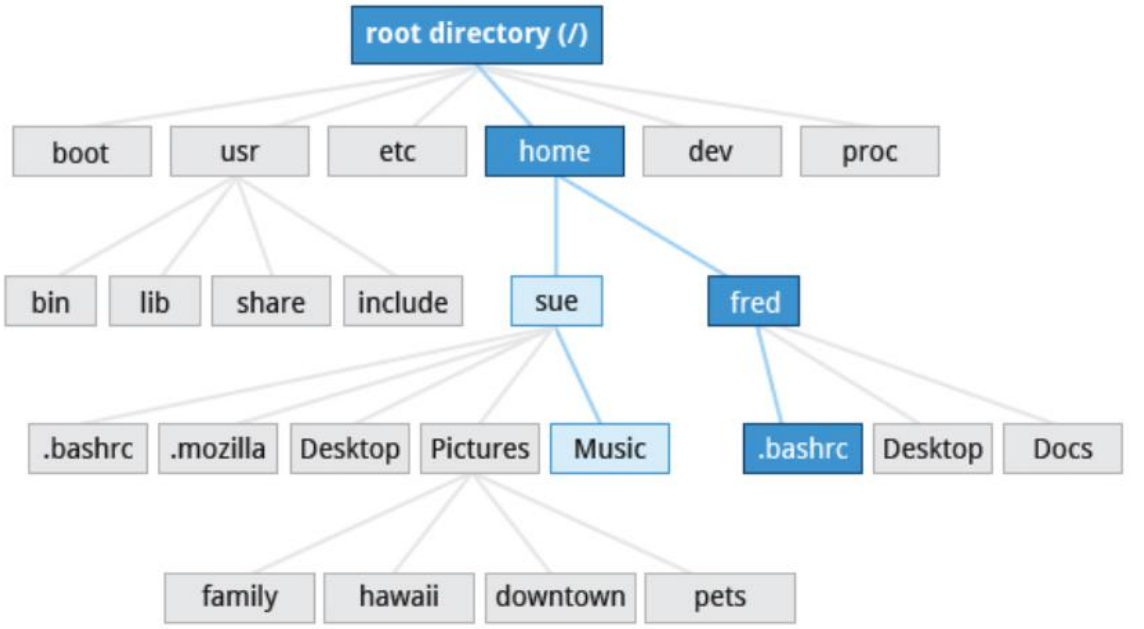
1. When you first log into a system or open a terminal, the default directory should be your home directory
2. We can print the exact path of this by typing **echo $HOME**.

3. Many Linux distributions actually open new graphical terminals in **$HOME/Desktop**.

Command	Result
`pwd`	Displays the present working directory
`cd ~` or `cd`	Change to your home directory (shortcut name is ~ (tilde))
`cd ..`	Change to parent directory (..)
`cd -`	Change to previous directory (- **(minus)**)

Understanding Absolute and Relative Paths:

1. **Absolute pathname:**
   An absolute pathname begins with the root directory and follows the tree, branch by branch, until it reaches the desired directory or file. Absolute paths always start with /.
2. **Relative pathname:**
   A relative pathname starts from the present working directory. Relative paths never start with /.
3. Multiple slashes (/) between directories and files are allowed, but all but one slash between elements in the pathname is ignored by the system.
4. ////usr//bin is valid, but seen as /usr/bin by the system.
5. For example, suppose you are currently working in your home directory and wish to move to the /usr/bin directory. The following two ways will bring you to the same directory from your home directory.
   - Absolute pathname method
     cd /usr/bin
   - Relative pathname method
     cd ../../usr/bin



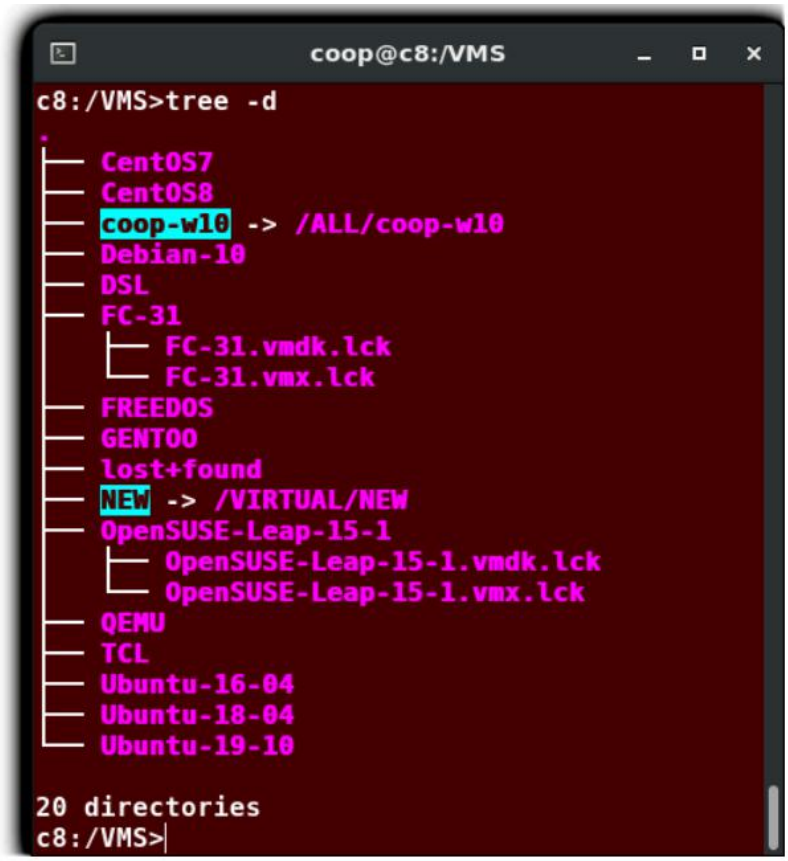
1.In the above example, we use the relative path method to list the files under Music from your current working directory (sue)
$ ls ../sue/Music

2.In the above example, we use the Absolute pathname method to edit the .bashrc file:
$ gedit /home/fred/.bashrc

Understanding Absolute and Relative Paths

Exploring the Filesystem:

1. Traversing up and down the filesystem tree can get tedious. The **tree** command is a good way to get a bird's-eye view of the filesystem tree. Use **tree -d** to view just the directories and to suppress listing file names.



Command	Usage
`cd /`	Changes your current directory to the root (/) directory (or path you supply)
`ls`	List the contents of the present working directory
`ls -a`	List all files, including hidden files and directories (those whose name start with .)
`tree`	Displays a tree view of the filesystem

Hard Links:

1. A hard link acts as a copy (mirrored) of the selected file.
2. It accesses the data available in the original file.
3. If the earlier selected file is deleted, the hard link to the file will still contain the data of that file.
4. The **ln** utility is used to create hard links
5. Suppose that **file1** already exists. A hard link, called **file2**, is created with the below command
   Ex: **ln file1 file2**
6. Note that two files now appear to exist. However, a closer inspection of the file listing shows that this is not quite true.
   Ex: **ls -li file1 file2**

7. The **-i** option to **ls** prints out in the first column the inode number, which is a unique quantity for each file object. This field is the same for both of these files; what is really going on here is that it is only one file, but it has more than one name associated with it, as is indicated by the **2** that appears in the ls output. Thus, there was already another object linked to **file1** before the command was executed.

```
harika@harika-VirtualBox:~$ ls -li file?
655783 -rw-rw-r-- 2 harika harika 0 Feb  3 18:12 file1
655783 -rw-rw-r-- 2 harika harika 0 Feb  3 18:12 file2
```

8. Hard links are very useful and they save space, but you have to be careful with their use, sometimes in subtle ways.
9. For one thing, if you remove either file1 or file2 in the example, the inode object (and the remaining file name) will remain, which might be undesirable, as it may lead to subtle errors later if you recreate a file of that name.

soft links or symbolic links or symlinks:

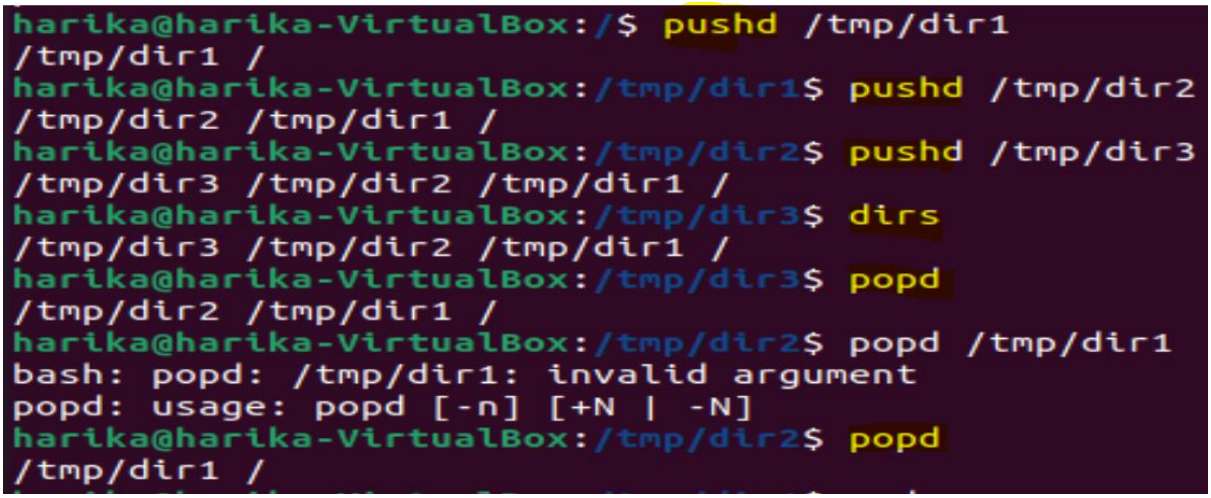
1. A soft link (also known as Symbolic link) acts as a pointer or a reference to the file name.
2. It does not access the data available in the original file.
3. If the earlier file is deleted, the soft link will be pointing to a file that does not exist anymore.
4. Soft link is accessed via -s option
   Ex: **ln -s file1 file3**
5. Notice file3 no longer appears to be a regular file, and it clearly points to file1 and has a different inode number.
   Ex: **ls -li file1 file3**

```
harika@harika-VirtualBox:~$ ls -li file1 file3
655783 -rw-rw-r-- 2 harika harika 0 Feb  3 18:12 file1
657314 lrwxrwxrwx 1 harika harika 5 Feb  3 18:22 file3 -> file1
```

6. Symbolic links take no extra space on the filesystem (unless their names are very long).
7. They are extremely convenient, as they can easily be modified to point to different places.
8. An easy way to create a shortcut from your home directory to long pathnames is to create a symbolic link.
9. Unlike hard links, soft links can point to objects even on different filesystems, partitions, and/or disks and other media, which may or may not be currently available or even exist.

Navigating the Directory History:

1. **pushd**: use pushd to change the directory instead of cd. This pushes your starting directory onto a list.
2. **popd**: Using popd will then send you back to those directories, walking in reverse order (the most recent directory will be the first one retrieved with popd).
3. **dirs**: The list of directories is displayed with the dirs command.



Viewing Files:

Command	Usage
`cat`	Used for viewing files that are not very long; it does not provide any scroll-back.
`tac`	Used to look at a file backwards, starting with the last line.
`less`	Used to view larger files because it is a paging program. It pauses at each screen full of text, provides scroll-back capabilities, and lets you search and navigate within the file. ***NOTE**: Use / to search for a pattern in the forward direction and ? for a pattern in the backward direction. An older program named more is still used, but has fewer capabilities: "less is more".*
`tail`	Used to print the last 10 lines of a file by default. You can change the number of lines by doing `-n 15` or just `-15` if you wanted to look at the last 15 lines instead of the default.
`head`	The opposite of `tail`; by default, it prints the first 10 lines of a file.

touch:

1. touch is often used to set or update the access, change, and modify times of files
2. By default, it resets a file's timestamp to match the current time.
3. However, you can also create an empty file using touch
   touch <filename>
4. This is normally done to create an empty file as a placeholder for a later purpose
5. touch provides several useful options. For example, the -t option allows you to set the date and timestamp of the file to a specific value, as in the below. This sets the myfile file's timestamp to 4 p.m., December 9th (12 09 1600).
   touch -t 12091600 myfile

mkdir:

1. mkdir is used to create a director
2. The below example creates a sample directory named sampdir under the current directory.
   mkdir sampdir
3. The below example creates a sample directory called sampdir under /usr.
   mkdir /usr/sampdir

rmdir:

1. Removing a directory is done with **rmdir**.
2. The directory must be empty or the command will fail.
3. To remove a directory and all of its contents you have to do **rm -rf**.

Moving, Renaming or Removing a File:

1. **mv** Simply rename a file.
2. **mv** is used to Move a file to another location, while possibly changing its name at the same time.
3. If you are not certain about removing files that match a pattern you supply, it is always good to run rm interactively (**rm –i**) to prompt before every removal.

Command	Usage
`mv`	Rename a file
`rm`	Remove a file
`rm -f`	Forcefully remove a file
`rm -i`	Interactively remove a file

Renaming or Removing a Directory:

1. **rmdir** works only on empty directories, otherwise you get an error.
2. While typing **rm –rf** is a fast and easy way to remove a whole filesystem tree recursively, it is extremely dangerous and should be used with the utmost care, especially when used by root

Command	Usage
`mv`	Rename a directory
`rmdir`	Remove an empty directory
`rm -rf`	Forcefully remove a directory recursively