1. **The Shell**

What is the shell? The shell is basically a program that takes your commands from the keyboard and sends them to the operating system to perform. If you’ve ever used a GUI, you’ve probably seen programs such as “Terminal” or “Console” these are just programs that launch a shell for you.

almost all Linux distributions will default to the bash shell. There are other shells available such as ksh, zsh, tsch, but we won’t get into any of those.

Let’s jump right in! Depending on the distribution your shell prompt might change, but for the most part it should adhere to the following format:

username@hostname:current\_directory

pete@icebox:/home/pete $

Notice the $ at the end of the prompt? Different shells will have different prompts, in our case the $ is for a normal user using Bash, Bourne or Korn shell, you don't add the prompt symbol when you type the command, just know that it's there.

Let’s start with a simple command, echo. The echo command just prints out the text arguments to the display.

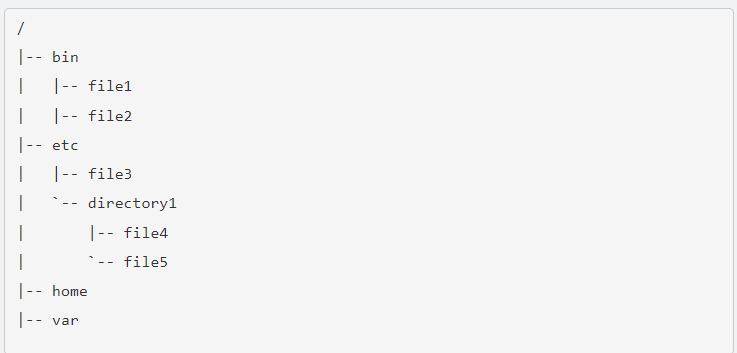
$ echo Hello World

**Exercises**

Try some other Linux commands and see what they output:

1. $ date
2. $ whoami
3. **PWD (Print working Directory)**

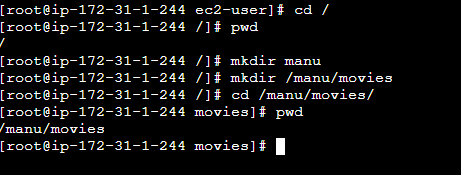
Everything in Linux is a file, as you journey deeper into Linux you’ll understand this, but for now just keep that in mind. Every file is organized in a hierarchical directory tree. The first directory in the filesystem is aptly named the root directory. The root directory has many folders and files which you can store more folders and files, etc. Here is an example of what the directory tree looks like:



The location of these files and directories are referred to as paths. If you had a folder named home with a folder inside of it named manu and another folder in that folder called Movies, that path would look like this: /home/manu/Movies, pretty simple huh?

Navigation of the filesystem, much like real life is helpful if you know where you are and where you are going. To see where you are, you can use the pwd command, this command means “print working directory” and it just shows you which directory you are in, note the path stems from the root directory.

$ pwd



1. **cd (Change Directory)**

Now that you know where you are, let’s see if we can move around the filesystem a bit. Remember we’ll need to navigate our way using paths. There are two different ways to specify a path, with absolute and relative paths.

* Absolute path: This is the path from the root directory. The root is the head honcho. The root directory is commonly shown as a slash. Every time your path starts with / it means you are starting from the root directory. For example, /home/manu/Desktop.
* 
* Relative path: This is the path from where you are currently in filesystem. If I was in location /home/manu/Documents and wanted to get to a directory inside Documents called taxes, I don’t have to specify the whole path from root like /home/manu/Documents/taxes, I can just go to taxes/ instead.



Now that you know how paths work, we just need something to help us change to the directory we want to. Luckily, we have cd or “change directory” to do that.

So now I've changed my directory location to /home/manu/movies.

Now from this directory I have a folder inside called Hawaii, I can navigate to that folder with:

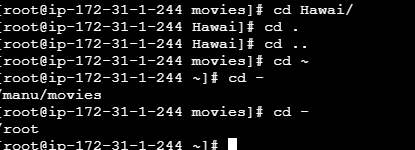
$ cd Hawai



Notice how I just used the name of the folder? It’s because I was already in /home/manu/movies.

It can get pretty tiring navigating with absolute and relative paths all the time, luckily there are some shortcuts to help you out.

* **.** (current directory). This is the directory you are currently in.
* **..** (parent directory). Takes you to the directory above your current.
* **~** (home directory). This directory defaults to your “home directory”. Such as /home/manu.
* **-** (previous directory). This will take you to the previous directory you were just at.

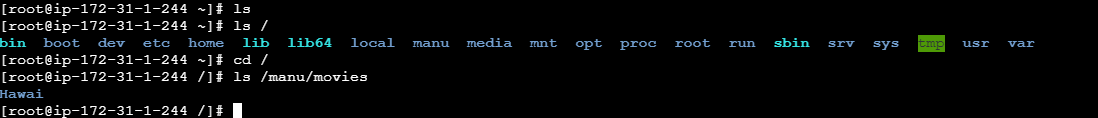


1. **ls (List Directories)**

Now that we know how to move around the system, how do we figure out what is available to us? Right now it’s like we are moving around in the dark. Well, we can use the wonderful ls command to list directory contents. The ls command will list directories and files in the current directory by default, however you can specify which path you want to list the directories of.

$ ls

$ ls /manu/movies



ls is a quite useful tool, it also shows you detailed information about the files and directories you are looking at.

Also note that not all files in a directory will be visible. Filenames that start with . are hidden, you can view them however with the ls command and pass the -a flag to it (a for all).

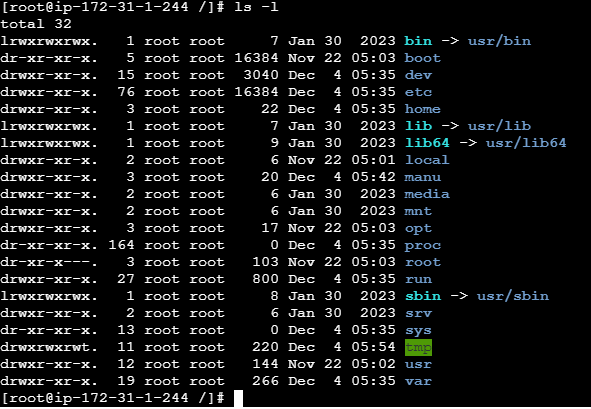
$ ls -a



ls is a quite useful tool, it also shows you detailed information about the files and directories you are looking at.

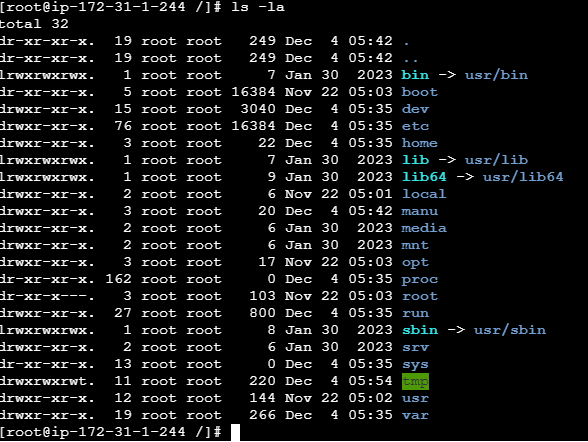
Also note that not all files in a directory will be visible. Filenames that start with . are hidden, you can view them however with the ls command and pass the -a flag to it (a for all).

$ ls -l



Commands have things called flags (or arguments or options, whatever you want to call it) to add more functionality. See how we added -a and -l, well you can add them both together with -la. The order of the flags determines which order it goes in, most of the time this doesn’t really matter so you can also do ls -al and it would still work.

$ls -la



**Exercises**

Run ls with different flags and see the output you receive.

* ls -R: recursively list directory contents
* ls -r: reverse order while sorting
* ls -t: sort by modification time, newest first

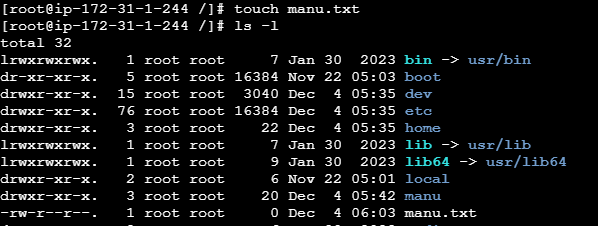
1. **touch**

Let’s learn how to make some files. A very simple way is to use the touch command. Touch allows you to the create new empty files.

$ touch mysuperduperfile

And boom, new file!

Touch is also used to change timestamps on existing files and directories. Give it a try, do an ls -l on a file and note the timestamp, then touch that file and it will update the timestamp.



There are many other ways to create files that involve other things like redirection and text editors, but we’ll get to that in the Text Manipulation course.

**Exercises**

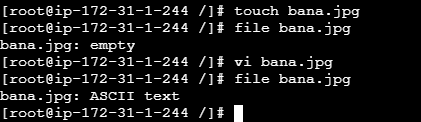
1. Create a new file
2. Note the timestamp
3. Touch the file and check the timestamp once again
4. **file**

we learned about touch, let’s go back to that for a bit. Did you notice that the filename didn’t conform to standard naming like you’ve probably seen with other operating systems like Windows? Normally you would expect a file called banana.jpeg and expect a JPEG picture file.

In Linux, filenames aren’t required to represent the contents of the file. You can create a file called funny.gif that isn’t actually a GIF.

To find out what kind of file a file is, you can use the file command. It will show you a description of the file’s contents.

$ file banana.jpg



1. **cat**

We’re almost done navigating files, but first let’s learn how to read a file. A simple command to use is the cat command, short for concatenate, it not only displays file contents but it can combine multiple files and show you the output of them.

$ cat bana.jpg

It’s not great for viewing large files and it’s only meant for short content. There are many other tools that we use to view larger text files that we’ll discuss in the next lesson.



1. less

If you are viewing text files larger than a simple output, less is more. (There is actually a command called more that does something similar, so this is ironic.) The text is displayed in a paged manner, so you can navigate through a text file page by page.

Go ahead and look at the contents of a file with less. Once you’re in the less command, you can actually use other keyboard commands to navigate in the file.

$less bana.jpg

Use the following command to navigate through less:

* q - Used to quit out of less and go back to your shell.
* Page up, Page down, Up and Down - Navigate using the arrow keys and page keys.
* g - Moves to beginning of the text file.
* G - Moves to the end of the text file.
* /search - You can search for specific text inside the text document. Prefacing the words you want to search with /
* h - If you need a little help about how to use less while you’re in less, use help.

### Exercises

Run less on a file, then page up and around the file. Try searching for a specific word. Quickly navigate to the beginning or the end of the file.

1. **History**

In your shell, there is a history of the commands that you previously entered, you can actually look through these commands. This is quite useful when you want to find and run a command you used previously without actually typing it again.

$ history

Want to run the same command you did before, just hit the up arrow.

Want to run the previous command without typing it again? Use !!. If you typed cat file1 and want to run it again, you can actually just go !! and it will run the last command you ran.

Another history shortcut is ctrl-R, this is the reverse search command, if you hit ctrl-R and you start typing parts of the command you want it will show you matches and you can just navigate through them by hitting the ctrl-R key again. Once you found the command you want to use again, just hit the Enter key.

Our terminal is getting a little cluttered no? Let’s do a little cleanup, use the clear command to clear up your display.

$ clear

There that looks better doesn’t it?

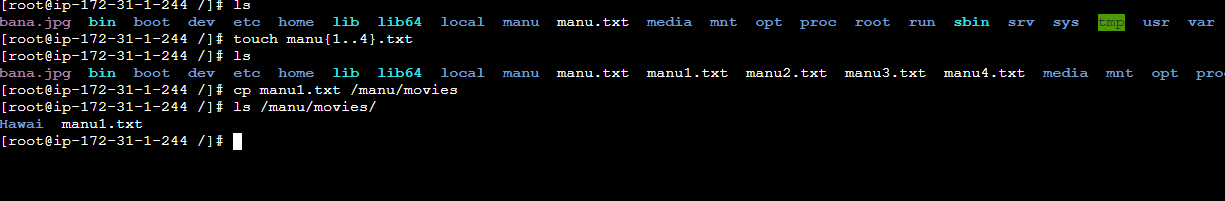
While we are talking about useful things, one of the most useful features in any command-line environment is tab completion. If you start typing the beginning of a command, file, directory, etc and hit the Tab key, it will autocomplete based on what it finds in the directory you are searching as long as you don’t have any other files that start with those letters. For example if you were trying to run the command chrome, you can type chr and press Tab and it will autocomplete chrome.

1. CP (Copy)

Let’s start making some copies of these files. Much like copy and pasting files in other operating systems, the shell gives us an even simpler way of doing that.

$cp <source file location> <destination to copy path>

$ cp cp manu1.txt /manu/movies



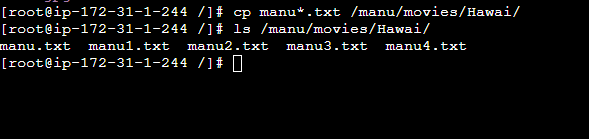
Manu1.txt is the file you want to copy and /manu/movies is where you are copying the file to.

You can copy multiple files and directories as well as use wildcards. A wildcard is a character that can be substituted for a pattern based selection, giving you more flexibility with searches. You can use wildcards in every command for more flexibility.

* \* the wildcard of wildcards, it's used to represent all single characters or any string.
* ? used to represent one character
* [] used to represent any character within the brackets

$ cp manu\*.txt /manu/movies/Hawai/

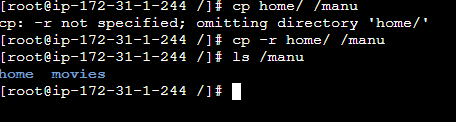
This will copy all files with the .jpg extension in your current directory to the Pictures directory



A useful command is to use the -r flag, this will recursively copy the files and directories within a directory.

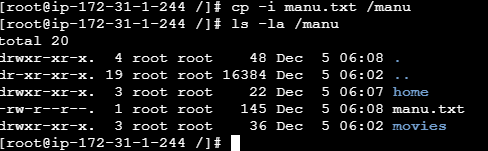
Try to do a cp on a directory that contains a couple of files to your Documents directory. Didn’t work did it? Well that’s because you’ll need to copy over the files and directories inside as well with -r command.

$ cp -r home/ /manu



One thing to note, if you copy a file over to a directory that has the same filename, the file will be overwritten with whatever you are copying over. This is no bueno if you have a file that you don’t want to get accidentally overwritten. You can use the -i flag (interactive) to prompt you before overwriting a file.

$ cp -i mycoolfile /home/pete/Pictures



**11. mv (Move)**

Used for moving files and also renaming them. Quite similar to the cp command in terms of flags and functionality.

You can rename files like this:

$ mv oldfile newfile

Or you can actually move a file to a different directory:

$ mv file2 /home/pete/Documents

And move more than one file:

$ mv file\_1 file\_2 /somedirectory

You can rename directories as well:

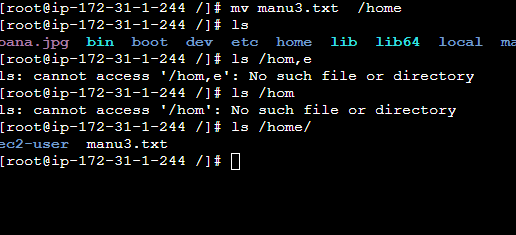
$ mv directory1 directory2

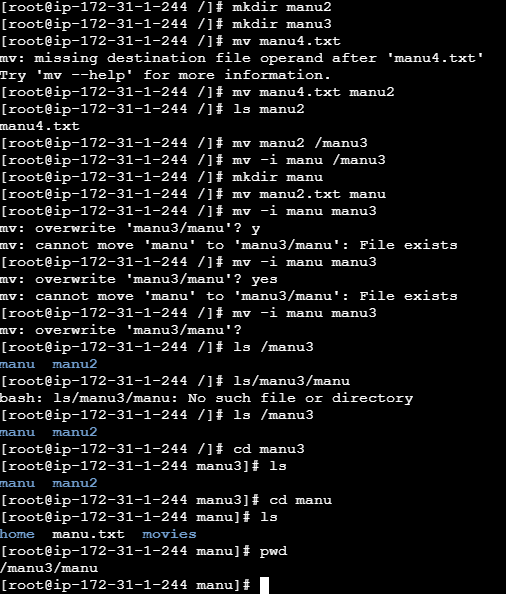
Like cp, if you mv a file or directory it will overwrite anything in the same directory. So you can use the -i flag to prompt you before overwriting anything.

mv -i directory1 directory2

Let’s say you did want to mv a file to overwrite the previous one. You can also make a backup of that file and it will just rename the old version with a ~.

$ mv -b directory1 directory2

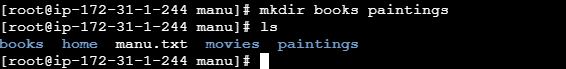




**12. mkdir (Make Directory)**

We’re gonna need some directories to store all these files we’ve been working on. The mkdir command (Make Directory) is useful for that, it will create a directory if it doesn’t already exist. You can even make multiple directories at the same time.

$ mkdir books paintings



You can also create subdirectories at the same time with the -p (parent flag).

$ mkdir -p books/hemmingway/favorites

