Linux Directory Structure

When you open any file manager in the "tree" mode, you should see a similar file structure.**/**: is the beginning of the file system, better known as "root." This is where everything begins. This is the cornerstone upon which the rest of the structure is built.

/etc: System-wide configuration files are stored here.

/usr: User accessible programs, program source codes, and documents are here.

/bin: (for binary): The programs themselves are stored here. It's a bin of binaries, or more exactly, the applications themselves. **/bin** also contains stuff like bash, cat, cp, kill, pwd, rm, and truly core things.

/sbin: This contains server and administration programs. Kernel and hardware-related programs such as shutdown, reboot, etc. may be found here.

/home: All users who have an account on the PC have a directory in home. Home is also the "catch-all," where you stick stuff until you find out where the heck it belongs. You may also create many convenient directories here to hold texts, images, working files, junk, and miscellaneous goodies.

/tmp: All temporary files that need to be created are stored here. It's the dusty attic, and it should be cleaned each springtime. It must be purged from time to time. Never keep anything important here, or it may just get dumped by mistake.

/var: Process information, such as system history and access logs, and error logs are here. This is the "bad conscience" of your system.

/root: This is "root's" home. It's the administrator's hiding place.

/dev: This is the location of the "devices" the system uses: CD-ROMs, cards, anything "mounted" will be found here. In Linux, a device is equipment, or even device-emulating code, providing methods for performing input or output (I/O). For example, a keyboard and a mouse are input devices. In Linux, SCSI devices such as ZIP drives are often "emulated" with code. This directory is important, and entries may be edited with extreme caution from the root account. A word of caution: the slightest editing error here can easily make a system inoperable. You will need exact documentation and instructions before delving into this file for edition.

/proc: A file system for running processes

Linux will have many processes running at once. Window managers, email clients, and Web browsers will be visible to the end user. Others, like servers and helper processes, run in the background and are not visible. Tasks that do not require the user's interaction are often invisible. Running "ps -ax" in a shell will print a list of all the currently running processes. You may also use the graphical representation of running processes by clicking the "System Monitor" icon in the system menu of Gnome, or the "Process Management" icon in KDE. This is also a good way to find out who's doing what to your station on your network, if your station's permissions have not been set properly or are set to allow remote user access and data sharing.

/mnt: Additional devices that need to be mounted are here. In Linux, there are no drive letters. Instead, each partition of your harddrive (or floppy/cdrom, etc) must be mounted somewhere to the root filesystem. **/mnt** is where you may find your cdroms and floppy filesystems.

/cdrom: Some distributions (such as Debian) give the CD-ROM device(s) this predetermined mount point.

/floppy: Same as above, but for the floppy drive. Other mount point files such as "zip" may also be added to the directory tree in certain Linux distributions.

MAN Pages and Shells

Fortunately, Linux comes with nearly all the documentation you will need "on the fly" while performing basic installations and operations. The key is to know which commands to use and how to look for them.

Ready references may be accessed from the command line. Each program, function, and utility in Linux has a related man or "manual" page. Follow "man" with a command name to get the syntax, description, and a list of options for that particular command. As an example, here is the "man" on "man" accessed by typing [man man] (without the brackets) in a terminal command line after the $ or the # prompt, followed by hitting "enter".

Logging in and out

Just a word of caution here. I assume you understand that you have both root (ie, Administrator) and user login options. Until you become very familiar with the file structure, after having installed a few applications and a few updates, I recommend highly that you do not change permissions while in root mode simply to permit user access to any particular files or directories. Arbitrary modifications of permissions can begin a chain of events that could very well make your system unusable. Of course, this is not "written in stone." All I am saying is to play it very carefully at the beginning. Use your regular user account for casual browsing of the web, and only log in as root to install/remove packages for now.

Navigation

Navigating around the files and directories of your hard drive could be a dreaded task for you, but it is necessary knowledge. If you were a user of command prompt interfaces such as MS-DOS, you'll have little trouble adjusting. You'll only need to learn a few new commands. If you're used to navigating using a graphical file manager, I don't know how it'll be like, but some concepts might require a little more clarification. Or maybe it'll be easier for you. Who knows? Everyone is different.

cd

As you might already have guessed, the **cd** command changes directories. It's a very common navigation command that you'll end up using, just like you might have done in MS-DOS.

You must put a space between **cd** and the ".." or else it won't work; Linux doesn't see the two dots as an extension to the cd command, but rather a different command altogether. It'll come to make sense if it doesn't already.

ls

The **ls** letters stand for **l**i**s**t. It basically works the same way as the **dir** command in DOS. Only being a Unix command, you can do more with it. :-)

Typing **ls** will give you a listing of all the files in the current directory. If you're new to Linux, chances are that the directories you are commonly in will be empty, and after the **ls** command is run, you aren't given any information and will just be returned to the command prompt (the shell).

There are "hidden" files in Linux, too. Their file names start with a dot, and doing a normal **ls** won't show them in a directory. Many configuration files start with a dot on their file names because they would only get in the way of users who would like to see more commonly used items. To view hidden files, use the **-a** flag with the **ls** command, i.e. **ls -a**.

To view more information about the files in a directory, use the **-l** flag with **ls**. It will show the file permissions as well as the file size, which are probably what are the most useful things to know about files.

You might occasionally want to have a listing of all the subdirectories, also. A simple **-R** flag will do, so you could look upon **ls -R** as a *rough* equivalent of the **dir /s** command in MS-DOS.

You can put flags together, so to view all the files in a directory, show their permissions/size, and view all the files that way through the subdirectories, you could type **ls -laR**.

pwd

This command simply shows what directory you're in at the moment. It stands for "Print Working Directory". It's useful for scripting in case you might ever want to refer to your current directory.

File Management

A lot of people, surprisingly for me, prefer to use graphical file managers. Fortunately for me, I wasn't spoiled like that and used commands in DOS. That made it a bit easier for me to make the transition to Linux. Most of the file management Linux gurus do is through the command line, so if you learn to use the commands, you can brag that you're a guru. Well, almost.

cp

Copying works very much the same. The **cp** command can be used just like the MS-DOS **copy** command, only remember that directories are separated with slashes (/) instead of backslashes (\). So a basic command line is just **cp filename1 filename2**.

There are other extensions to the **cp** command. You can use the **-f** command to force it. You can use the **-p** command to preserve the permissions (and also who owns the file, but I'm not sure).

You can move an entire directory to its new destination. Let's say you want to copy a directory (and all of its contents) from where you are to be /home/jack/newdirectory/. You would type **cp -rpf olddirectory /home/jack/newdirectory**. To issue this command you would have to be in the directory where the subdirectory "olddirectory" is actually located.

ln

A feature of **lin**king files is available in Linux. It works by "redirecting" a file to the actual file. It's referred to as a **symbolic link**. Don't confuse this term with the linking of programs, which is when binary programs are connected with libraries that they need to load in order to run.

The most simple way that I've ever used **ln** to create symbolic links is **ln -s existing_file link**. Evidently there's a hard link and a symbolic link; I've been using a symbolic link all along. You can also use the **-f** flag to force the command line to overwrite anything that might have the symbolic link's file name already.

To remove a symbolic link, simply type **rm symbolic_link**. It won't remove the file that it's linked to.

mv

The **mv** command can be used both to move files and to rename them. The syntax is **mv fileone filetwo**, where "fileone" is the original file name and "filetwo" will be the new file name.

You can't move a directory that is located in one partition to another, unfortunately. You can copy it, though, using **cp -rpf**, and then remove it with **rm -rf** later on. If you have only a single partition that makes up your filesystem then you have very little to worry about in this area.

rm

The **rm** command is used for **r**e**m**oving files. You use it just like the **del** or **delete** command in MS-DOS. Let's say you want to remove a file called foobar in your current directory. To do that, simply type **rm foobar**. Note that there is no "Recycle Bin" like in Windows 95. So when you delete a file, it's gone for good.

To delete something in some other directory, use the full path as the file name. For example, if you want to delete a file called "windows" that's in the directory /usr/local/src/, you would type **rm /usr/local/src/windows**.

To remove an entire directory and its contents, type **rm -rf /directory** where "/directory" is the path to the directory that you want to delete. If you're wondering, the "rf" stands for "recursive" and "force". Be very careful with this command, as it can wreak havoc easily if misused.

Shutting Down and Rebooting

To shut down your system, type **shutdown -h now**, which tells the **shutdown** program to begin system halt immediately. You can also tell it to halt the system at a later time, I think, but you'll have to consult the **shutdown** manual page for that (**man shutdown**).

To do a reboot, you can either type **reboot** or **shutdown -r**. You can also use the famous Ctrl-Alt-Delete combination to reboot, which you might already be familiar with.

Shutting down and restarting properly (as described above) will prevent your filesystem from being damaged. Filesystem damage is the most obvious of the consequences, but there are probably other things out there that I don't know about. The point is, shut down your system properly.

There are (rare!) cases in which the machine might lock up entirely, and prevent you from being able to access a command prompt. Only then will your last resort be to do a forced reboot (just pressing the restart button on the case).

Sources

http://linuxguide.automatedshops.com/LinuxGuide/linux-commands.html
http://www.extremetech.com/article2/0,3973,1154286,00.asp