What is Linux?

What is Linux?

Linux is an operating system that evolved from a **kernel** created by **Linus Torvalds** when he was a student at the University of Helsinki. Generally, it is obvious to most people what Linux is. However, both for political and practical reasons, it needs to be explained further. To say that Linux is an operating system means that it's meant to be used as an alternative to other operating systems, Windows, Mac OS, MS-DOS, Solaris and others. Linux is not a program like a word processor and is not a set of programs like an office suite. Linux is an interface between computer/server hardware, and the programs which run on it.

A brief history of Linux

When Linus Torvalds was studying at the University of Helsinki, he was using a version of the **UNIX** operating system called '**Minix**'. Linus and other users sent requests for modifications and improvements to Minix's creator, **Andrew Tanenbaum**, but he felt that they weren't necessary. That's when Linus decided to create his own operating system that would take into account users' comments and suggestions for improvements.

Free Software pre-Linux

This philosophy of asking for users' comments and suggestions and using them to improve computer programs was not new. **Richard Stallman**, who worked at the Massachusetts Institute of Technology, had been advocating just such an approach to computer programming and use since the early 1970's. He was a pioneer in the concept of **'free software'**, always pointing out that 'free' means 'freedom', not zero cost. Finding it difficult to continue working under conditions that he felt went against his concept of 'free software' he left MIT in 1984 and founded **GNU**. The goal of GNU was to produce software that was free to use, distribute and modify. Linus Torvalds' goal 6 years later was basically the same: to produce an operating system that took into account user feedback.

The kernel

We should point out here that the focal point of any operating system is its 'kernel'. Without going into great detail, the kernel is what tells the big chip that controls your computer to do what you want the program that you're using to do. To use a metaphor, if you go to your favorite Italian restaurant and order '**Pizza**', this dish is like your **operating system**. There are a lot of things that go into making that dish like pizza base, tomato, chilly, capsicum, onion, mushrooms and cheese etc. Well, the **kernel** is like the **pizza base**. Without pizza base, that dish doesn't exist. You might as well find some bread and make a sandwich. A piece of just pizza base is tasteless.

Without a kernel, an operating system doesn't exist. Without programs, a kernel is useless.

1991, a fateful year

In 1991, ideal conditions existed that would create Linux. In essence, Linus Torvalds had a kernel but no programs of his own, Richard Stallman and GNU had programs but no working kernel. Read the two men's own words about this:

"Linus:
Sadly, a kernel by itself gets you nowhere. To get a working system you need a shell, compilers, a library etc."

"RMS:
The GNU Hurd is not ready for production use. Fortunately, another kernel is available. [It is called] Linux. So combining the necessary programs provided by GNU in Cambridge, Massachusetts and a kernel, developed by Linus Torvalds in Helsinki, Finland, Linux was born. Due to the physical distances involved, the means used to get Linus' kernel together with the GNU programs was the Internet, then in its infancy. We can say then that Linux is an operating system that came to life on the Internet. The Internet would also be crucial in Linux's subsequent development as the means of coordinating the work of all the developers that have made Linux into what it is today."

Linux is introduced

Late in 1991, Linus Torvalds had his kernel and a few GNU programs wrapped around it so it would work well enough to show other people what he had done. And that's what he did. The first people to see Linux knew that Linus was on to something. At this point, though, he needed more people to help him. Here's what Linus had to say back in 1991.

"Linus:
Are you without a nice project and dying to cut your teeth on an OS you can try to modify for your needs?... This post might just be for you."

People all over the world decided to take him up on it. At first, only people with extensive computer programming knowledge would be able to do anything with that early public version of Linux. These people started to offer their help. The version numbers of Linux were getting higher and higher. People began writing programs specifically to be run under Linux. Developers began writing drivers for different video cards, sound cards and other gadgets inside and outside your computer could use Linux. Nevertheless, throughout most of first part of the 1990's Linux did not get out of the 'GURU' stage. GURU is a term that has evolved to mean anyone who has special expertise in a particular subject. That is, you had to have special expertise in how computers worked to be able to install Linux in those days.

Linux, at first, was not for everybody

Other popular software companies sold you a CD or a set of floppies and a brief instruction booklet and in probably less than a half an hour, you could install a fully working operating system on your PC. The only ability you needed was knowing how to read. Those companies had that intention when they actually sat down and developed their operating systems. Linus Torvalds didn't have that in mind when he developed Linux. It was just a hobby for him. Later on, companies like Red Hat made it their goal to bring Linux to the point where it could be installed just like any other operating system; by anyone who can follow a set of simple instructions, and they have succeeded. For some reason, though, Linux hasn't completely lost its 'Gurus only' image. This is largely because of the popular tech press' inability to explain in a meaningful way what Linux is. The truth is that few tech reporters have real life experience with Linux and it is reflected in their writing.

Where Linux is Today

Today, Linux is enjoying a favorable press for the most part. This comes from the fact that Linux has proven to be a tremendously stable and versatile operating system, particularly as a network server. When Linux is deployed as a web server or in corporate networks, its down-time is almost negligible. There have been cases when Linux servers have been running for more than a year without re-booting and then only taken down for a brief period for routine maintenance. Its cost effectiveness has sold it more than anything else. Linux can be installed on a home PC as well as a network server for a fraction of the cost of other companies' software packages. More reliability and less cost - it's ideal.

If you're reading this, you're obviously here to learn how to use Linux. Any learning experience means opening up to new ideas and new ways of doing things. As mentioned before, Linux is in the UNIX family of operating systems. UNIX is primarily designed to be used by professionals. You will have to learn some UNIX concepts in this lesson, but that doesn't mean that Linux is a professionals-only operating system. In fact, most major versions of Linux are designed to be as user-friendly and as easy to install as any other operating system on the market today.

Now that you know what Linux is and how good it is.

The Linux File System

Now's the time to learn a little bit about the Linux file system. We'll learn about where Linux puts it stuff, where to find stuff and a little bit about what that stuff is.

When you're running Linux and start to type things on that black screen, you are using a **shell**. Any operating system uses a shell to get commands from the keyboard to the computer. It's a lot easier than punching holes in cards like they used to do in the old days. There are actually programs for Linux where you talk through a microphone and Linux will carry out commands that you've programmed in advance. It's really cool. For now, though, we'll concentrate on the keyboard. The most popular shell used for Linux is the bash shell. bash means "Bourne Again Shell". It is a free version of the Bourne shell and uses a little play on words, as you can see.

Getting in and out of directories with 'cd'

We saw a few commands in the last lesson, but we didn't go into them much. We will handle a lot of commands in more detail in later lessons. This lesson will cover those commands which you will need to see what's under Linux's hood.

The first one we should look at is 'cd'.

cd

Will get you in and out of directories. CD = Change Directory. Pretty simple eh?

Try this one:

```
cd /
```

This will get you into the 'root' or main directory. It's the directory of directories, the king of kings, your show of shows. The root directory shouldn't be confused with root's directory. That is /root.

Basic Directory Structure of Linux

Now type this:

```
ls
```

You will probably see something like this:

```
cdrom
boot
bin
dev
etc
floppy
home
lib
lost+found
mnt
opt
proc
root
sbin
tmp
usr
var
```

They will be blue in color. Those are directories.

The /bin directory


```
cd bin
ls
```

This is the famous bin/ directory. You know, I have always felt this one was misnamed. For example, when people say, 'That's no good, throw it in the bin'. Actually, bin/ is one of the most important directories in Linux. You'll find all of the most used commands there. Right now you should be seeing a lot of red (or green, depending on your version of Linux). Those are programs.

The /etc directory

Now let's look at another directory. There's a long way and a short cut. First the long way.

```
cd .. [Enter]
cd etc [Enter]
```

Or you can just type:

```
cd /etc
```

Anyway, you are now in the etc/ directory. This houses most of the configuration files for Linux. lilo.conf, the file that tells you which OS to boot is in there.

you'll see: lilo.conf or grub in newer distributions

And you don't even have to type the whole thing. You could just type 'ls li' and push the tab key. Linux will type the rest for you. Isn't that cool!

In Linux, Everything is a File

Everyone knows what a files is... It's that "photo", "document", or "music" that you use. Programs are made of files, in fact, the whole Linux operating system is just a collection of files... But, now for the weird part. Not only is that digital photo that you uploaded to your computer a file, but your monitor is a file too! You see, in Linux, everything is a file! WOW!!! How can that be? Let's try to explain it.

The /dev directory


```
cd /dev [Enter]
ls [Enter]
```

You'll see a lot of yellow outlined in black. These are the devices that your system uses or can use. Everything is considered a file in Linux, so your hard disk is kept track of as a file that sits there. If you're using an IDE hard drive (as opposed to SCSI), your hard drive will be known as /dev/hda. Don't delete that, because your hard disk will spin around, come jumping out of your computer, land on the floor and spill out ooze all over the place. No, not really. You will probably not have to look in /dev very much, so don't worry about that.

The /boot directory


```
cd /boot [Enter]
```

Will get you into the /boot directory. You will not find any boots or shoes or footwear of any kind there. That's where the Linux kernel usually is. Power users may change the location of the kernel for reasons of their own (they may prefer /shoe), but it is normally placed there on most systems. You will eventually have to use this directory, because you may need to use two or more different types of kernels in the future. That will be taken up in a more advanced lesson.

root's directory /root

If you are not working as 'root' and you type cd /root, you will be taken to the directory /root. However, you won't be able to do anything while you're there. Root's home directory is a restricted area for everybody else. Linux response is sort of like, 'You don't have to know that'. Users' home directories are under certain restrictions for other users as well.

The /sbin directory

/sbin is another one of those off-limits directories. You may look, but you can't touch. This directory is like /bin in that it has frequently used programs in it, but they're only meant to be used by root. 'Shutdown' is in there. Only root can shutdown the system. If a user other than root tried to shutdown the system, he or she would get a message saying that only root can do that. Then that person would be followed by the secret police for three months.

The /tmp directory

/tmp is a directory that is used to store temporary files, as the name may suggest. You will find later on that when you use a Windows-style system with Linux like KDE, this window manager will create files there for temporary use. When you double click on an icon of a photo, the photo comes up for you to see but a temporary file is created while you're looking at the photo. The temporary file is deleted when you close the KDE image program. It's mainly the programs that work under a windows manager that take advantage of this directory.

The /var directory

/var is a directory for certain files that may change their size (i.e. variable size) For example, there are a few excellent databases

commerce website, I would have a database to register purchases. That database would obviously grow in size. And if it didn't then I'd be in trouble. It is also the normal place where email servers store their incoming mail. Again, email varies in size as well.

The /lib directory

/lib is for library files. That's where the name /lib comes from. Programs may use libraries to carry out their functions. Different programs use the same libraries, so Linux will store them here so that every program knows where to find them. You will probably not have to worry about this directory much unless you start getting messages like 'can't find shared library...'. That will sometimes happen when you've downloaded a program and had to compile it yourself from source. Even then, getting what are known as "dependency" problems are quite rare. Most programs, even when compiled from source, usually have a pre-configuration program that makes sure that they can find what libraries they "depend" on to run. If they don't, they'll tell you that you can't install the program.

The /home directory (home sweet home!)

We talked about /home before. This is a directory for storing users' personal files. All of us have certain preferences for using programs. These preferences are usually included in configuration files which are also stored in users' home directories. Most of these files start with a '.' (period/dot).
If you go to your home directory, '

```
cd /home/[username]
```

```
ls -a
```

You will see these files.

What's left

Most installations of Linux will also provide these directories:
/mnt
/cdrom
/floppy

These shouldn't contain anything. Later on, we'll explain in more detail what these are for. Let's just say that in Linux, if you want to see what's on a floppy disk or a CD, you're not going to be able to just click on an 'a:' icon or a 'd:' icon. You're going to do cd /floppy or cd /cdrom

If you try that now you probably won't see anything. As I said, more about these directories later in the course.
Well, we've looked under Linux's hood, so to speak. In the next lesson, we'll take her for a little spin.

Shutting down Linux

At this point you should have installed Linux, and you've looked around at what you have. And then when you're finished you'll have to shut off your computer. Actually, there are computers that are never shut off. Imagine if your ISP shut off the computer every night! The Internet is a 24/7 business so that wouldn't be practical. There are also people who probably just shut off their monitor. As you probably get some sleep occasionally, so we should maybe let our machine have a rest too once and a while. For this, we'll use the shutdown command

As anyone who's used a computer knows, if you shut off you're computer before you've finished saving work, or if there's a power outage that shuts it off for you, data will be lost. At first, if you shut off Linux incorrectly or there was an inopportune thunder storm and you lost electrical power, you could do severe damage to your Linux file system. That will very rarely happen these days, but you should always use the shutdown command when you want to shut off your computer. Linux will tell you about it if you don't - it will run a check on your hard disk automatically when you use it again. If you have a big hard disk, you might as well go and make yourself a sandwich because it's going to take a while. Linux will also run a routine check every once and a while automatically. You also have our permission to fix yourself a sandwich in these cases too.

Shutdown a single computer

The most common way of shutting down a single user Linux system is for you as root to issue the command:

```
shutdown -h now
```

You use this when you plan on shutting your computer off at that moment, as opposed to some later time.


```
Linux is going for system halt NOW
```

It will start to shut off programs that you're computer is using and you'll see it all happening. That's because Linux is a transparent system. It lets you see everything it's doing. It won't give you a simple message telling you to wait and then another one telling you you can shut it off now. If something is causing a problem, it will tell you about it when it starts up and when it shuts down. That way, if you are having a problem, you may be able to track it down. If you don't know how to solve it, you can tell another person what you saw and he or she may be able to help you.

With the shutdown command, you must wait until you see the message:


```
System halted
Power down before you shut off the computer.
Re-booting the computer
```

Rebooting Your Computer

```
shutdown -r now
```

If you have installed a dual-boot system and you want to use the other operating system, (why would you want to do that?) you would use this command. You will get a similar message as with the -h (halt) option that will say something like:


```
System going for reboot NOW
```

The basic reason behind all of these messages is that Linux was conceived to be a networked operating system. You have people at workstations on the network busily doing their work. The last part of the shutdown command now is fine for a single-user home PC, but on a network system this would be changed to indicate a time. That way people would have a chance to finish what they were doing before the system went down for maintenance. Using 'now', in a network, would probably be hazardous to the health of the person who sent that command.

The next time you shutdown your system, you may want to try using some time options instead of just now. For example, you may want to try shutting down the computer at a given time.

```
shutdown -h 20:01
```

Which will shutdown the computer at 8:01 PM. You could also try:

```
shutdown -h +5
```

That shuts down the computer in 5 minutes time.
Now you know the correct way to shutdown your Linux system.

MAN Pages & Useful Information

How to get more information with Linux

Now we'll talk about some other commands that you will probably need in your day to day work with Linux. They make your work a little easier and give you added information about your system.
'**man**' - manual pages in Linux

The first command is '**man**'. This command will show the manual for a command or program. The manual is a file that shows you how to use the command and list the different options for the command in question. You would type:

```
man [the command]
```

For example, if you type

```
man mkdir
```

The manual file for '**mkdir**' will come up and give you a detailed explanation of this command.

Managing Documentation in Linux

The manual file for '**mkdir**' is actually one of the more straight-forward ones. There are a lot that I think we're written by Harry Bigbrains and they were meant to be seen only by Richard Biggerbrain who's sitting in the cubicle next to him.

For example, this appears in the '**man**' file for '**cp**'

By default, sparse SOURCE files are detected by a crude heuristic and the corresponding DEST file is made sparse as well. That is the behavior selected by --**sparse=auto**. Specify --sparse=always to create a sparse DEST file whenever the SOURCE file contains a long enough sequence of zero bytes. Use --sparse=never to inhibit creation of sparse files.

I don't know about you, but I'm going to call my lawyer. I've been assaulted by "a crude heuristic".

If you use the command cp --help, you'll get a nutshell version of the 'cp' command.

If you use your pipe cp --help | less, it'll be a little easier to manage.

The 'info' format

Typing info [command name] will get you more information on a command and is more current than most man files and perhaps a little more readable. In fact, some 'man' files will actually tell you to consult the 'info' file. The 'info' files are not always installed automatically. so you may want to consult your own version of Linux about these files.

Apropos

The word '**apropos**' means pertinent to something else. There is a command that will show you all of the man page that may shed some light on a certain command. For example, if I typed:


```
apropos xterm
```

resize (1x) - set TERMCAP and terminal settings to current xterm window size
xterm (1x) - terminal emulator for X
terms (5) - database of blessed terminals for xtermset.
xtermset (1) - change settings of an xterm

These are all man pages related to xterm. You would then just choose one of these and type man terms for example.

Some versions of Linux that are made for languages other than English will give you this documentation in its particular language. There are also websites that specialize in documentation in other languages. You can use your favorite Internet search engine to find Linux documentation in your own language.

Working with Disks, Drives & Files

Linux File Systems

This is a beginner's course and one of the concepts that newcomers to Linux find different is the idea of a file system in Linux. That is to say, the way data is stored and managed in Linux. Most users are familiar with Files. In the simplest terms possible, one can say that files are just a collection of data items which are stored on a disk. However, the way that those files are stored on a disk can vary depending on several different factors. Speed, security, data redundancy etc...

Linux Supports lots of different types of File System formats. Here are just a few...

Ext2:	This is like UNIX file system. It has the concepts of blocks, inodes and directories.
Ext3:	It is backwards compatible with the ext2 file system, except that it has added journal ing capabilities. Journalling allows fast file system recovery. Includes support for Access Control Lists (ACL).
Isofs:	Used by CDROM file system (iso9660).
Procfs:	The proc file system acts as an interface to internal data structures in the kernel. It can be used to obtain information about the system and to change certain kernel parameters at runtime using sysctl command.

Mounting file systems

In this part of the lesson about file systems we'll learn how to use the commands mount and umount

We've mentioned previously that there's a different idea in Linux as to what constitutes a floppy disk drive, a CD-ROM drive and another partition of your hard disk (the Windows partition, for example). Though some windows managers for Linux have provisions for clicking on an icon to access a floppy drive, for example, the method behind this is quite different from other OSes. In Linux the floppy drive or other device must be "mounted". That means basically, incorporating it temporarily into your Linux file system or, in other words, telling Linux that it is a file to be written to or copied from.

You can use the mount command to copy to and from other devices.

If you would like to get some files from a CD-ROM, the standard command to do this is:

```
mount -t iso9660 /dev/hdb /cdrom
```

The type, iso9660 is the standard file system for a CD. The device (/dev/hdb) is the non-SCSI type of CD-ROM and the mount point (/cdrom) should exist. If it doesn't, you should create it in the root directory with 'mkdir', just as you may have done with the /floppy directory.

Remember that the concept of CD-ROM is read only. You won't be able to write to this type of CD-ROM drive. A message will tell you that when you mount this type of device.

Mounting another partition of the hard disk.

Many people may have preferred to install Linux along with another operating system. You may have Linux and Windows installed in the same computer. If you would like to access files on the Windows partition you would type the following command:

```
mount -t vfat /dev/hda1 /mnt
```

Windows is always in the primary partition, so that's why we've used the device /hda1 (hard disk partition 1). The choice for /mnt is the standard mount point in this case. You may use the /mnt directory to mount the other devices (floppies, CDs) as well. I use the different empty directories (/floppy /cdrom /mnt) to avoid confusion.

If you change to the mount directory (cd /mnt) and then type: ls and you'll see something interesting. The directories are blue but the files are green (or red - depending on your distribution of Linux). You won't have the various color combinations as you do in Linux. That's because Windows' file system doesn't distinguish file types. Everything looks like it's a program (binary) instead of a regular file.

When you copy files from the Windows partition to the Linux partition you should bear this in mind. For example, if you wanted to copy an mp3 file from the Windows partition to the Linux partition to test out your sound configuration, it would show up as a executable program and not just a standard file under Linux. This doesn't effect your playing it, but for a more accurate accounting of what you have on your system, you may want to change the permissions of the file so that it shows up as a regular file in your color scheme. We'll talk about file permissions and and making changes to them shortly.

unmount

Mounting file systems that aren't part of the standard Linux system is considered a temporary condition in Linux. Now that we know how to mount these outside file systems in Linux, the important thing now is to learn how to unmount it when we're finished using it.

In the early days of Linux, you could do serious damage to your system if you didn't unmount manually after you were finished. Nowadays if you have mounted a system and you shut down the computer without unmounting, the chances are pretty slim that you're going to trash a file system. Slim, however, isn't good enough for me. I'd rather not take a chance. I always take the time to unmount these external file systems when I'm finished with them.

The command for this is:

```
umount [/mount point]
```

In the examples I used in this lesson, the mount points were:

```
floppy:              /floppy     - therefore umount /floppy
CD:                              /cdrom              - umount /cdrom
Windows partition    /mnt                - umount /mnt
```

Remember also to NOT be accessing the disk. when you use umount or it will give you a drive busy message. If you get this, you may have left a terminal open where you were using to copy or write to the particular drive. Check your terminals if you get this error.

I just wanted to point out another thing too. You are 'unmounting' but the command is umount (that is, without theN of un) Being u(n)familiar with the mount concept when I started with Linux, I typed unmount instead of umount and much to my surprise, I got the message: command not found. U(n)deniably, I had done something wrong. It took me a while to figure out that the command didn't include an N. I haven't really done a survey on how many people have actually done this. For all I know, I may be the only person who has, but I just thought it best to warn you.

Well, now you can use the standard devices that most PC users need. Later in our advanced class we'll talk about installing and using other devices like scanners, CD writers, Zip drives and matter/anti-matter flow inducers. (well, maybe we'll save that last one for the super-advanced classes!)

SWAP Partition

Plunk that CD/DVD in your computers drive. Okay. Now is the moment of truth. We've got the CD/DVD in the drive, and we're ready to go.

Restart the computer.

One Big Partition?

This should now boot the Linux kernel located on your CD ROM. What you'll have to do first is partition the hard drive. There is an easy way to do this. You can dedicate the whole hard disk or non-Windows partition (depending on the type of install you're doing).

There is a better alternative?

Yes. The better alternative is to partition your hard disk even further and put "parts" of Linux on separate partitions.

For example, this is the scheme that works for me: Let's take a 1 Terabyte hard drive as an example.

First, you should see how much RAM you have. From this figure, you create what's known as a SWAP partition. This is simply a way that Linux uses to get an extra memory when it runs out of free memory. Physical RAM is very very fast. Many times faster than your hard drive I/O. Your computer will put things in and out of RAM as it needs too. However, if you run out of RAM, then your computer will start to use your hard drive to temporarily save information too. This is called "SWAPPING", and is generally a bad condition to be in. Systems that are "swapping", are not able to take advantage of the very very fast RAM, and instead rely on slow disk I/O. Thus, slowing down the entire system.

It is recommended that the amount of SWAP space that is allocated to a system is twice the amount of physical memory (RAM) that is installed in a computer. Of course, this depends upon the amount of hard disk space available for the system (although this "recommendation" dates way back and doesn't apply so well on modern systems).

So if you've got 8 gigabytes of RAM, the feel free to make an 16 gigabyte swap partition. In today's world, hard disk space is so plentiful, that you really can't go wrong with over doing it...

Then my partition scheme ends up looking like this:

```
Partition Location     Size
------------------------------
swap     /dev/hda2     20 gb
/        /dev/hda1     80 gb
/usr     /dev/hda3     300 gb
/home    /dev/hda4     600 gb
```

The nice folks over a Ubuntu have written up a wonderful SWAP-FAQ. Feel free to pop over to them and read all about how to change swap sizes, and what they recommend you use when setting up a swap partition.

Assign partitions to look like this. Don't worry about the /usr and /home parts. That will come after. You must indicate here that you want / to be the bootable partition. /usr will contain most of the programs that will run on your machine. /home will contain your personal files. This kind of a partition scheme may come in handy if you have problems with your hard disk. You may be able to save information if it's located in different partitions easier than if it were only one big partition.
Before we actually assign the other partitions their places and functions, we need to initialize and activate the swap partition. Do this now.

Now you should initialize the / partition - the one that will boot the Linux kernel.
Now, there is what I consider a little glitch in the Debian install. It doesn't really take into account that you want to initialize /usr and /home partitions. Don't go to the next step yet. You should go back and initialize these partitions now before proceeding.

File Permissions (chmod/chown)

Linux has inherited from UNIX the concept of ownerships and permissions for files. This is basically because it was conceived as a networked system where different people would be using a variety of programs, files, etc. Obviously, there's a need to keep things organized and secure. We don't want an ordinary user using a program that could potentially trash the whole system. There are security and privacy issues here as well. Let's face it, we don't want Bill to read Bob's love letters to the Janet who works in R & D. (because Janet is Bill's fiancé) In the end, it's important to know what belongs to me, to you and to everybody.

As we mentioned at the beginning of this course, the big advantage that Linux has is its multi-user concept- the fact that many different people can use the same computer or that one person can use the same computer to do different jobs. That's where the system of file permissions comes in to help out in what could be a very confusing situation. We're going to explain some basic concepts about who owns the file and who can do what with a file. We won't get into an enormous amount of detail here. We'll save that for the Linux system administration course. We will show you how to understand file permission symbols and how to modify certain files so that they're more secure.

File permission symbols

If you run the command

```
ls -l
```

in your home directory, you will get a list of files that may include something like this

```
-rw-r--r--   1  bob  users   1892   Jul 10  18:30 linux_course_notes.txt
```

This basically says, interpreting this from RIGHT to LEFT that the file, linux_course_notes.txt was created at 6:30 PM on July 10 and is 1892 bytes large. It belongs to the group users (i.e, the people who use this computer). It belongs to bob in particular and it is one (1) file. Then come the file permission symbols.

Let's look at what these symbols mean:

The dashes - separate the permissions into three types

The first part refers to the owner's (bob's) permissions.

The dash - before the rw means that this is a normal file that contains any type of data. A directory, for example, would have a d instead of a dash.

The rw that follows means that bob can read and write to (modify) his own file. That's pretty logical. If you own it, you can do what you want with it.

The second part of the these symbols after the second dash, are the permissions for the group. Linux can establish different types of groups for file access. In a one home computer environment anyone who uses the computer can read this file but cannot write to (modify) it. This is a completely normal situation. You, as a user, may want to take away the rights of others to read your file. We'll cover how to do that later.

After the two dashes (two here because there is no write permissions for the group) come the overall user permissions. Anyone who might have access to the computer from inside or outside (in the case of a network) can read this file. Once again, we can take away the possibility of people reading this file if we so choose.

Let's take a look at some other examples. An interesting place to look at different kinds of file permissions is the /bin directory. Here we have the commands that anybody can use on the Linux system. Let's look at the command for gzip, a file compression utility for Linux.

```
-rwxr-xr-x   1 root     root       53468 May  1  1999 gzip
```

As we see here, there are some differences.

The program name, date, bytes are all standard. Even though this is obviously different information, the idea is the same as before.
The changes are in the owner and group. Root owns the file and it is in the group "root". Root is actually the only member of that group.
The file is an executable (program) so that's why the letter x is among the symbols.

This file can be executed by everybody: the owner (root), the group (root) and all others that have access to the computer
As we mentioned, the file is a program, so there is no need for anybody other than root to "write" to the file, so there is no w permissions for it for anybody but root.

If we look at a file in /sbin which are files that only root can use or execute, the permissions would look like this:

```
-rwxr--r--  1 root    root       1065 Jan 14  1999 cron
```

'cron' is a program on Linux systems that allows programs to be run automatically at certain times and under certain conditions. As we can see here, only root, the owner of the file, is allowed to use this program. There are no xpermissions for the rest of the users.

We hope you enjoyed this little walk-through of file permissions in Linux. Now that we know what we're looking for, we can talk about changing certain permissions.

chmod

chmod is a Linux command that will let you "set permissions" (aka, assign who can read/write/execute) on a file.

```
chmod permissions file
```

```
chmod permission1_permission2_permission3 file
```

When using chmod, you need to be aware that there are three types of Linux users that you are setting permissions for. Therefore, when setting permissions, you are assigning them for "yourself", "your group" and "everyone else" in the world. These users are technically know as:

- Owner
- Group
- World

Therefore, when setting permissions on a file, you will want to assign all three levels of permissions, and not just one user.

Think of the chmod command actually having the following syntax...

```
chmod owner group world FileName
```

Now that you understand that you are setting permissions for THREE user levels, you just have to wrap your head around what permissions you are able to set!

There are three types of permissions that Linux allows for each file.

- read
- write
- execute

Putting it all together:

So, in laymen terms, if you wanted a file to be readable by everyone, and writable by only you, you would write the chmod

command with the following structure.

COMMAND : OWNER : GROUP : WORLD : PATH

```
chmod read & write read read FileName
chmod 6 4 4 myDoc.txt
```

Wait! What are those numbers?!?

Computers like numbers, not words. Sorry. You will have to deal with it. Take a look at the following output of **`ls -l`**


```
-rw-r--r-- 1 gcawood iqnection 382 Dec 19 6:49 myDoc.txt
```

You will need to convert the word *read* or *write* or *execute* into the numeric equivalent (octal) based on the table below.

- 4 read (r)
- 2 write (w)
- 1 execute (x)

Practical Examples

chmod 400 mydoc.txt read by owner
chmod 040 mydoc.txt read by group
chmod 004 mydoc.txt read by anybody (other)
chmod 200 mydoc.txt write by owner
chmod 020 mydoc.txt write by group
chmod 002 mydoc.txt write by anybody
chmod 100 mydoc.txt execute by owner
chmod 010 mydoc.txt execute by group
chmod 001 mydoc.txt execute by anybody

Wait! I don't get it... there aren't enough permissions to do what I want!

Good call. You need to add up the numbers to get other types of permissions...

So, try wrapping your head around this!!
7 = 4+2+1 (read/write/execute)
6 = 4+2 (read/write)
5 = 4+1 (read/execute)
4 = 4 (read)
3 = 2+1 (write/execute)
2 = 2 (write)
1 = 1 (execute)

chmod 666 mydoc.txt read/write by anybody! (the devil loves this one!)
chmod 755 mydoc.txt rwx for owner, rx for group and rx for the world
chmod 777 mydoc.txt read, write, execute for all! (may not be the best plan in the world...)

Good luck! Hope this helps.

Never set things to 777 unless you have a really good reason to do so.

File Permissions - chown

chown

The command chown is the chmod's cousin. It is used for changing the ownership rights of a file (hence the name 'chown' - change owner). It does not change the read, write and execution permissions however.

This command, though available to every user, is probably going to be used when you're working as root. The command is uses like this:

```
chown owner.group filename
```

Let's say you want to copy something from your Windows partition (if you have one). You mount the partition (as root) and to save time, you copy the file to your user directory /home/bob/. If you type ls -l the_file you'll get something like this:

```
-rw-r--r--   1 root     root         2428 Nov 17 13:18 the_file
```

As we now know from the previous lesson, root is the owner of the file. Therefore, root is the only one who has write permissions for the file (permission to modify its content). If you plan on working with the file as "bob", there isn't a snowball's chance in hell to modify that file until, as root, you run chown on the file. So let's do it!

```
chown bob.bob the_file
```

This example presupposes that your Linux version creates groups for each user. There are others that will create a generic group called users for everybody who uses the computer. On a network, groups are created according to the needs of the organization. On your single home computer, just type ls -l and see what system corresponds to you.

As you can see, 'chown' is absolutely necessary if you're working as more than one user with the computer.

Commands for System Administration

The following commands are frequently used by systems administrators to keep an eye on what's going on with their systems

last

The command last will show you the people who have logged into the computer today and the terminals they are/were using. If you type:

```
last
```

You may get something like this:

```
fred     tty6         Thu Oct 5  16:55 - 20:05          (3:10)
bob      tty1         Thu Oct 5  still logged in        (3:10)
root     tty1         Thu Oct 5  16:23 - 16:43          (0:20)
reboot   system boot              Thu Oct 5 16:22
```

As you can see, you worked as 'fred' for 3 hours and 10 mins. You are still working as 'bob'. You worked as 'root' for 20 minutes (probably some administration tasks) and you booted your computer at 4:22 PM.

This is a good way to see who's been using the computer if it's networked. For example, if you saw a an entry for 'satan' and you hadn't given the Prince of Darkness permission to login, you could fire off a nasty e-mail to him about mis-use of your server. His address, by the way, is "thedevil@hell.com"

df

'df' is a command that you're going to use a lot if you're pressed for hard disk space. Once again, there are many programs that run graphically that will inform you of the space available on your Linux partition. But this is a very good, quick, non-graphic way to keep track of your hard disk space.

If you type

```
df
```

You may get something like this. (This is actually taken from a system I use for testing versions of Linux. My 'df' is going to be a bit confusing because I run a XenServer virtual machine attached to a SAN. Sorry!)

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup-lv_root
                      48964432   2838132  43639004   7% /
tmpfs                  1987488         0   1987488   0% /dev/shm
/dev/xvda1              495844     52160    418084  12% /boot
/dev/mapper/VolGroup-lv_home
                      47626600    370604  44836696   1% /home
```

If you start seeing a 'df' output like this, it's time to get down to your local computer shop and buy a new hard disk. Anyway, 'df' is a good way to keep track of this.

free

'free' is a command that you can use if you want to know how much RAM memory you have free on your system. By typing:

```
free
```

and you will get something like this

```
total       used       free     shared    buffers     cached
Mem:         14452      13904        548      28208        492       7312
-/+ buffers/cache:       6100       8352
Swap:        33260       1556      31704
```

This output isn't very friendly. Try converting things to Megabytes by typing

```
free -m
```

Sometimes, if a program is running particularly slowly, you may find out that your memory usage is high using this command. Linux's memory management is quite good but a certain program may be "hogging" memory. You could exit that program and then type free again to see if it was the culprit.

du

'du' is the way to see how big files are. You can use it on a directory or on a particular file. This is another command I use a lot. It's probably best to use the option du -b (-b for bytes) and it will give you the exact figure in bytes. By default, 'du' shows the closest kilobyte figure. Let's look at a couple of examples:

If I type:

```
du people_I_owe_money.note
```

I may get an output like this: 193 people_I_owe_money.net

But instead, if I type:

```
du -b people_I_owe_money.note
```

I'll get: 197120 people_I_owe_money.note

As you can see, it's a big file. I owe a lot of people money. On the other hand look at the output for 'people_who_owe_me_money.note':
1 people_who_owe_me_money.note

No, that's not the kilobyte figure. That's the byte figure!

You can also use this on a directory, and it will list the files and subdirectories and give you the byte or kilobyte count, whichever you prefer

If you turn out to be a human, then you may want to set the output to something even more friendly.

Try this:

```
du -h
```

top

To show you the use of the 'top' command. Here you will see what processes are running 'top' is a good command to use when you want to see what your system's doing. 'top' is designed to show you how your CPU is being used. It will give you a pretty complete list of everything that's going on in your computer. Here's a sample output of the 'top' command:

```
top
```

```
top - 14:11:38 up 12 days, 22:38,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  97 total,   1 running,  96 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3974980k total,  3515800k used,   459180k free,   244404k buffers
Swap:  6209528k total,       0k used,  6209528k free,  2684644k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      20   0 19204 1512 1220 S  0.0  0.0   0:01.84 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.55 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:02.41 watchdog/0
    7 root      20   0     0    0    0 S  0.0  0.0   1:11.89 events/0
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cgroup
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
   11 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pm
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.00 xenwatch
   14 root      20   0     0    0    0 S  0.0  0.0   0:01.47 xenbus
   15 root      20   0     0    0    0 S  0.0  0.0   0:05.50 sync_supers
   16 root      20   0     0    0    0 S  0.0  0.0   0:05.87 bdi-default
   17 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/0
   18 root      20   0     0    0    0 S  0.0  0.0   0:03.82 kblockd/0
   19 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ata/0
   20 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ata_aux
   21 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksuspend_usbd
   22 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khubd
   23 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kseriod
   24 root      20   0     0    0    0 S  0.0  0.0   0:00.00 md/0
   25 root      20   0     0    0    0 S  0.0  0.0   0:00.00 md_misc/0
   26 root      20   0     0    0    0 S  0.0  0.0   0:00.33 khungtaskd
   27 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kswapd0
   28 root      25   5     0    0    0 S  0.0  0.0   0:00.00 ksmd
   29 root      20   0     0    0    0 S  0.0  0.0   0:00.00 aio/0
   30 root      20   0     0    0    0 S  0.0  0.0   0:00.00 crypto/0
   35 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthrotld/0
   37 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khvcd
   38 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kpsmoused
   39 root      20   0     0    0    0 S  0.0  0.0   0:00.00 usbhid_resumer
   69 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kstriped
  229 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdmflush
  231 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdmflush
  250 root      20   0     0    0    0 S  0.0  0.0   0:13.44 jbd2/dm-0-8
```

```
 251 root     20   0     0    0    0 S  0.0  0.0   0:00.00 ext4-dio-unwrit
```

ps

'ps' will give you a list of the processes running on your system.

Just typing ps will give you the processes you're running as a user. It may look like this:

```
ps
```

```
PID TTY          TIME CMD
22987 pts/0    00:00:00 ps
29552 pts/0    00:00:00 bash
```

If you happen to forget what your name is, you can type ps u. This is the user mode and your user name will appear in the first column. Actually, there's more than that.

There will be other columns about memory usage, the time you started running the processes and others.

You can see other users processes if you type ps -au. If you're not using a networked computer, you will see yours and root's processes. If you're into detective work, you can type just ps -a and try to guess who's using the process.

The information that you'll probably be most interested in is the column that shows the "process ID" or "PID". We'll get into why these are so important in the next part of the lesson.

kill

"kill" is a very explicit word. It implies 'death'. In the last part of this lesson we talked about process IDs or "PIDs". Well, with the command 'kill' plus a PID, you can terminate a program. In other words, you kill the program. You will probably only use this with troublesome processes; programs that may not let you exit regularly. A good example of this is when you try a new program that hasn't got all the bugs worked out of it yet. If the 'exit' button doesn't work, you can 'kill' it.

To do this, first, you would type ps and you would get this output that we talked about before.

PID TTY STAT TIME COMMAND
293 2 S 0:00 -bash
422 2 S 0:00 sh /usr/X11R6/bin/startx
437 2 S 0:00 tee /home/bob/.X.err
438 2 S 0:00 xinit /home/bob/.xinitrc --
441 2 S 0:01 /usr/X11R6/bin/evilaliens

Let's say you're trying a new game that was just ported to Linux. The game is called "Evil and Nasty Alien Visitors". The name of the "binary" or program itself is called evilaliens. (the last one on my 'ps' example) Now you can't exit the program either. The aliens just took over Oklahoma and you swore you got them all. You're playing in your x-windows enviroment and you have an x-terminal open (probably the one you typed 'ps' into). You would just type:
kill and the PID number, in my example, 441.

That is:

```
kill 441
```

and the program disappears, along with all those nasty aliens.

Here's a little trick. If you don't want to do two steps, 'ps' and 'kill' and you know the name of the "binary" or program itself, as I mentioned before, you can just type:

```
killall evilaliens
```

and that should also do the trick.

Using 'kill' as root

Remember that when you work as root, you are the all-powerful master of the universe. (or at least the computer). When you use 'kill' as root, you are the 007 on Her Majesty's Linux Service. You have license to 'kill' the process you desire. If you're working as 'root' and you need to use the 'kill' command, it's a good idea to run ps -au and look closely at your PIDs. You don't want to end up killing some process that you need.

Another way to kill a process.

Try typing this:

```
find *.
```

(Kind of an absurd thing to do, but good as an example) It will just start finding everything. If you have done something like this by mistake, you can use the keys

```
CTR+ C
```

to stop the 'find' process. In these cases, there's no need to use the 'kill' command.

Useful Commands - The 'mkdir' command

'**mkdir**' is the command for making directories. '**mkdir**' may be familiar to MS-DOS users out there. As you have noticed, the people who wrote these programs tried to give them names that described what they do more or less, not as long as 'makemeadirectoryplease' and not too cryptic like 'xr77b'.

Using the 'mkdir' command

To create the directory 'my_friends' that we talked about in the last lesson, you would type:

```
mkdir my_friends
```

There are no whistles or buzzers. If you'd like some sort of acknowledgment, you could type

```
mkdir --verbose my_friends
```

and it will tell you that you created the directory.

If you type ls -l You'll see it there along with information about it.

Now you know how to use 'mkdir'. You can even use it to create a directory called 'my_enemies' if you're into that sort of thing.

The 'rmdir' command

'**rmdir**' is the opposite of '**mkdi**r'- it gets rid of directories. It should be pointed out that in order to use it, the directory has to be empty. If you copied or moved anything to 'my_friends' and you typed

```
rmdir my_friends/
```

Linux would politely tell you that you can't do that.

So, you have to use your 'rm' command on the files first to remove them or use 'mv' to get them into another directory. Then you're free to use 'rmdir'

So, next we'll deal with the 'rm' command.

Useful Commands - The 'rm' command

'**rm**' is for removing or deleting files. That means, sending them into non-existence, oblivion, bye-bye.

The correct use of 'rm'

So you have to be careful with '**rm**'. That's why we put an entry into our '**.bashrc**' file: **alias rm='rm -i'** so that it asks you if that's what you really want to do.

If you created a file called 'bad_jokes' and you wanted to get rid of the file, you would type rm bad_jokes, and because you made an alias, it will ask you.

```
rm: remove 'bad_jokes'?:
```

You would press the 'y' key unless of course you remembered that you have a real good one in there and then you would answer: with the 'n' key. Actually, any key other than 'y' is the same as responding with the 'n' key, so if you accidentally type 'w', don't worry.

'rm' - some words of caution

You can also do stuff like rm b* with the asterisk, but I would use my best judgment with that. You may get 'y-itis' and just keep pressing the 'y' key. I have done that before.

Occasionally, when I've used the text editor 'joe' a lot I end up with a lot of files that end in a tilde (~). You get files like 'note_to_myself1' and if you've modified it, another one 'note_to_myself1~' Then I decide that I don't want all of those ~ files littering up my directory and I innocently type: 'rm note_to_myself*' and then 'y' 'y' 'y' and then I realize too late and my brain types: 'Y did you do that!!' The problem is that 'note_to_myself2' contained my important plans for taking over the world and 'note_to_myself3' contained a note about how much money I owe at the dry cleaners. Maybe taking over the world can wait, but my dry cleaner's not going to give me my suits the next time unless I pay him.

There's another case of 'rm' that's potentially more dangerous than the '**rm ***' case. That's adding the '-f' option on the end. If you do this, it will override the '-i' option and won't ask you anything. It just goes ahead and deletes the files in question. For example, if you were in a directory and typed 'rm * -f' you would delete everything, no questions asked. I generally use the '-f' option very sparingly, like when Valentine's day falls on Friday during leap year and coincides with a full moon.

Also, please be very careful when you're using the '**rm**' command as 'root'. You could do some very serious system damage if you delete the wrong files. Wait until you have some experience before you start removing files 'by hand' as 'root'. Use the configuration tools provided in your version of Linux to un-install programs that you don't want

Useful Commands - The 'cp' command

To show you how to copy files with Linux we talked about 'cp' in the lesson on aliases. 'cp' is for copying files from one place to another, or for making a duplicate of one file under a different name.

Let's go back to Tony's 'stuff' file For example, if you saved Tony's e-mail attachment to your main /home directory, /home/[your name], you may want to create a directory to keep Tony's files You could make the directory for Tony tonyd (Tony's last name is Dweebweiler)

Type this:

```
mkdir tonyd
```

Then you can do:

```
cp stuff tonyd
```

Remember use your TAB key to save time.

Now you're going to have TWO files named 'stuff' because you copied that file to the directory 'tonyd/' - you didn't move it there. You'll have the original 'stuff' in your home directory and then the copy in /home/[your name]/tonyd/.

You'll be able to tell the difference between the two files because the copy of 'stuff' in the directory 'tonyd' will show a different time. Use the command **ls -l stuff** on both files to see this.

If you had used the command cp -p instead of just cp you would end up with two identical files in two different places. If you don't want that, there's a better way of doing it so that 'stuff' is only in the directory 'tonyd'. That's themv command. We'll talk about that shortly.

More uses of the 'cp' command

To show you how to copy directories and create duplicates of files. Now let's talk about two more basic uses of the cp command and some short cuts.

You can also copy entire directories to another place. As I mentioned in a previous lesson, you may want to work as two different users for two different jobs. You may be working as 'fred' and your directory 'tonyd' is in the directory /home/bob, where you work as 'bob'

As 'fred', you can use the command:

```
cp -r /home/bob/tonyd/ /home/fred/
```

If you're in your home directory you can use this command

```
cp -r /home/bob/tonyd/ ./
```

To copy the directory 'tonyd' to your home directory.

You may also use the command

```
cp -r /home/bob/tonyd/ ~
```

With the tilde wherever you happen to be and that will automatically copy the directory 'tonyd' to your other home directory.

The other use of '**cp**' we talked about was to get a copy of a file with a different name. For example, Tony's file 'stuff' is loaded with jokes. You may want to add some more jokes and then pass it along to another person You could do this:

```
cp stuff stuff2
```

or choose a name that's meaningful for you other than 'stuff2'

Now you have another file that you can add jokes to while you preserve the original file. You can open it in 'pico' and start writing: "Why did the chicken cross the road..."

Always remember to use that TAB key and the up and down arrows to save yourself some time..

Useful Commands - The 'mv' command

'**mv**' is a command that we're going to use to move files around or to rename them. '**mv**' sort of has a split-personality because it serves these two functions at the same time.

'mv' command for renaming files

Let's go back yet again to Tony's file, 'stuff' again. 'stuff' is not a good name for a file just as 'book' isn't a good name for a book. Just imagine: "The number one bestselling book this week is 'Book' by John Author.

You should probably re-name this file to something meaningful. I would suggest doing something like this:

```
mv stuff tonys_jokes
```

You may have noticed the underscore '_' in the title. It's there because Linux doesn't really like spaces in the file names. You can do it and Linux will accept it but it will put a \\ between the different words. Spaces are sort of 'faux pas' in Linux but not 'verboten'. It would be to your advantage to use '_' between words though.

Moving files with the 'mv' command

Now you can use the 'mv' command to move Tony's jokes into the directory you made to keep his files.

```
mv tonys_jokes tonyd/
```

If you do cd tonyd and then ls to* you will see his file there along with 'toms_jokes' and 'tomato_soup_recipe'. (if you have another friend named Tom and you like to cook)

You can also move entire directories with this command. You do not have to use the '**-r'** option as you did with '**cp**'. You would just substitute the file name for a directory name

```
mv tonyd/ my_friends/
```

would move the directory 'tonyd' to the directory 'my_friends'.

Even More Useful Commands

The command 'touch'

Now we're going to talk about a touchy subject. The command '**touch**' which is used to change the time and/or date of a file.

You can use '**touch**' if your boss yells at you about not having a report ready at lunchtime. You should quickly finish the report, then type:

```
touch -t 05070915 my_report.txt
```

and it makes it look like you did it at 9:15. The first four digits stand for May 7 (0507) and the last four (0915) the time, 9:15 in the morning. Make sure your digits match your story. You don't want to have it look like you did it in February. Of course, if you punched in at 9:40, then you're in trouble.

'**touch**' can be used also to create an empty file. You would just enter.

```
touch [a file name]
```

There may be times in the future when you need an empty file that will be filled up later automatically by the workings of some program. We'll deal with the uses of 'touch' in our later courses.

Finding things with the command 'find'

There's so much on a computer's hard drive, nobody could ever know from memory where everything is. Perhaps the smart lad who won the spelling bee by spelling 'prestidigitator' might be able to, but most of us are going to have to find things now and then.
If you use a windows manager like KDE, you can use the find tool. It's very useful because it has a lot of options and you can use them to modify your searches.

How to use the 'find' command

But if you're getting used to using command line stuff, just type in:

```
find -name *hawaii*
```

and find out where you put you pictures of you Hawaiian vacation. If you're in your /home directory, it will go through every directory and find every file that has the name 'hawaii' in it. The two asterisks make sure it does that. If they started with 'hawaii' you wouldn't need the first asterisk but you can leave it there if you want.

You may have created some file recently. For example, you may want to find some file that you were working on, let's say from now up to 10 minutes ago, you could type.

```
find -mmin +0 -mmin -10
```

This will list the files that you created or modified within the last ten minutes. If you choose to use a higher number for -mmin -? you should probably use a pipe, for example:

```
find -mmin +0 -mmin -120 | less
```

will find things that you created or modified up to 2 hours ago and the '| less' part will make it easier to read.

The 'grep' command

In the last section we talked about the 'find' command which finds files. Now we'll talk about the 'grep' command which finds words in files. Your windows manager may have this incorporated into its find tool but then again, the beauty of Linux is having alternatives.

What does 'grep' mean?

'grep' is a Vulcan word that means "find". Actually it isn't, but it sort of looks like it, doesn't it?

Kirk: "Find the solar system L10J, Mr. Spock."

Spock: "Grepping now, Captain." *

* Star Trek stuff copyright Paramount Pictures

Let's have a little practice session with 'grep'. The best way is learning by doing, so let's do it.

A 'grep' mini-tutorial
With 'pico' or any Linux text editor, create a file called 'mary1.txt'

```
pico mary1.txt
```

Then type: "Mary had a little lamb"

Press CTRL-X in 'pico' (if you're using that) and it will prompt you to save.

Then create: mary2.txt and enter the text Mary had a little cow.

Save that and create the file: mary3.txt and type: Mary had a little too much to drink. Now we know what Mary was doing when she wasn't watching her lambs! Now save that file.

OK, now we're ready to try out '**grep**', so phasers on stun and let's go.

Type the following command:

```
grep Mary mary*.txt
```

Let's explain this a little. 'grep' looks for the word "Mary" in any text file that is called "mary(something).txt". You've created three files that start with 'mary', so the asterisk makes sure that 'grep' will look for the word 'Mary' in all three.

You should get this output:

```
mary1.txt Mary had a little lamb
mary2.txt Mary had a little cow
mary3.txt Mary had a little too much to drink
```

The word 'Mary' is in all of those files, so you'll get this output.

If you type grep little mary*.txt you'll get the same output because the word "little" is also in each of those files. But if you type the word "cow", you'll get this output:

mary2.txt: Mary had a little cow

because the word "cow" is only in mary2.txt.

Typing grep drink mary*.txt will get us more or less the same, only that mary3.txt will show up instead of mary2.txt.
Well, there's 'grep' in a nutshell. It's been a pleasure 'grepping' with you!

Power user commands

Here is a brief overview of some other commands that you may find interesting at some point as you use Linux. They will help you to get all of the power out of Linux.

'**who**'

'**who**' is a command to find out who's working on your system. As you now know, Linux is a multi-user system. Even if you're using one computer at your home, you may be working as more than one person. For example, if you logged in as 'root' but are working as 'bob'. You may see something like this:

```
root        tty1        May 20 09:48
bob         tty2        May 20 10:05
```

This is just Linux's way of saying that 'root' started working on terminal 1 on May 20 at 9:48 in the morning and bob started working on terminal 2 at 10:05. This is mainly used in networked situations so the system administrator knows who's working. It can be used by your boss to find out that you've come in late too. You may use it to find out if you've opened more that one terminal so that you remember to log out.

'tee', '>', '2>'

In the lesson on the pipe command, I mentioned plumbing with Linux. I think I'm going to resist the temptation to make some sort of golf reference here in the lesson on the command 'tee'.

'tee' is used to write out what appears on your screen into a file. You will be using this with the after a pipe '|' .

You might do this:

```
ls -l | tee directory_listing
```

to get a file with the listing of a directory. If you've placed files in a directory to be backed up, you could use this command to create a listing of that directory. You could print out the file on a label and stick it to the disk, tape, zip cartridge or whatever you used to make the backups.

If you're using the 'tee' command for the backups I described before, you may want to put a date on the file. You can use this command:

```
date | tee -a directory_listing
```

The command 'date' will enter the date and time in the file at the end. Remember to use the -a option if you're going to write to that file a second time. If you don't you will erase everything on the file in favor of whatever the second command was.

The '>' command

The "greater than" symbol '>' will do the same as 'tee'. You don't need the pipe command (|) with this one.

```
ls -l > directory_listing
```

will give you the same result. If you want to add the date at the end, use the command:

```
date >> directory_listing
```

with two "greater than" symbols (>>)

The two symbols will add to the file without erasing its contents (appending).

The '2>' command

This command, the number two (2) with the "greater than" symbol >, is used for creating a file for an error message that you may get.

You will probably not be using it a lot because we all know how perfect Linux is and how few errors there are when you're using it. But every once and a while you may want to download some software from the Internet You install it and - whoops! - there's some error. You may not have something installed that the program needs to run. You could just do something like this:

```
[program X that doesn't work] 2> program_X_error
```

You create a file with the error message. You could show it to someone who might know what's missing or you could send it to the author of the program. He or she would also like to know about it and will probably help you fix it.

whoami

whoami is a nice little program that tells you who you are, just in case you didn't know already. You amnesia victims are in luck! Actually it tells you who you are in terms of how Linux understands who you are, that is to say, your user name. So if your user name is bob and you type whoami you'll get: bob This comes in handy if you switch terminals a lot and work as a different user. You may, in terms of computer use anyway, forget who you are!

whatis

To show you how to use the '**whatis**' command '**whatis**' is a command so you can find out what a program does. If you explore your Linux system, you will find a lot of programs and you may not know what they do. You would simply type:

```
whatis grep
```

for example, and you would get this:

grep (1) - print lines matching a pattern

Linux is good, but it's not all-knowing, so if you type:

```
whatis orange juice
```

You will get this message:

orange: nothing appropriate.

juice: nothing appropriate.

basically telling you that Linux has no idea what orange juice is

whereis

whereis is a nice command for finding other commands or programs. If you decide to download any program from the internet, that program may need other programs in order to work. If you want to know whether or not you have it, you can type:

```
whereis [program name]
```

and find out.

If you wanted to find out if you have the 'pico' editor and where it is, you would type:

```
whereis pico
```

and you may get this:

pico: /usr/bin/pico /usr/man/man1/pico.1.gz

It shows you where the command is as well as the location of its manual file.

whereis isn't designed to find people, so if you type

whereis Harry

Linux is just going to say Harry:

which

To show you another tool for locating programs 'which' is similar to 'whereis'. It will give you the location of a program. At times, a program may not find another program it needs to make it run. It will need to know its location or "path". For example, a program may need Java to run it but thinks its in another place. You would simply type:

```
which java
```

and Linux will inform you of its location

/usr/lib/java/bin/java

This is a handy command because some locations of programs vary from one version of Linux to the next. A software developer may have designed his/her program to access Java, for example, from a different location. As Open Source software will always let you modify configuration files to get your program working according to your needs, you can get the program to work for your system.

echo

To show you some uses of the 'echo' command 'echo' is a little command that repeats anything you type. For example if you type

```
echo hello
```

Linux will display the word 'hello' .

There is really no need to do that under normal conditions. You may find 'echo' useful in the future if you start writing "shell scripts" which are like little programs that you could use to do a few commands at one time. You could use 'echo' in those scripts to tell you what the script is doing at any given time, or to prompt you to do something, like enter text. Shell scripts will be taken

up in a later course.

There is a practical use for 'echo' in everyday life. I sometimes use it to write short notes. If we use 'echo' along with 'pipe' (|) and 'tee', you've got a poor-man's post-it-note.

For example:

```
echo remember to tell Bill Gates he owes me 5 bucks | tee -a bill_gates.note
```

Will make you a nice reminder note about dear 'ole Bill. Just remember to read your note. You could type:

```
echo remember to open Gates note | tee -a remember_gates.note
```

to make yourself a reminder for the other reminder note. Use less bill_gates.note or less remember_gates.note to read your notes

wc

People following this course from Europe may recognize this as the symbol for 'bathroom'. Unfortunately, if you type this in your terminal it will not show you the way to the 'facilities'.

Actually, this command will give you the number of lines, words and letters (characters) in a file and in that order.

Let's go back to the file about the people I owe money. If I type:

```
wc people_I_owe_money.note
```

I will get this output:

439 6510 197120 wc people_I_owe.note

As you can see, there are 439 lines, so that means if each line represents one person, then I owe 439 people money. There are 6510 words and a total of 197120 characters.

I might add that this is a good tool for people who write letters professionally and get paid by the word.

dir

The people who traveled down the MS-DOS road will know this one. Actually, dir=ls -l. It will give you the same result. If you do any downloading or uploading of files via FTP by way of a non-GUI FTP program in your terminal, you may find this command useful. I once ran into a case where the remote computer didn't recognize thels -l command. I just typed in dir and that did the trick. Then I fired off a nasty e-mail asking why in the world they weren't using Linux!

pwd

The command pwd will show complete information on the directory you're working in. For example, if you type

```
pwd
```

you may get something like this:

/home/bob/homework

which shows you that you're in the directory 'homework' in your user directory as 'bob', so you know exactly where you are.

date

Did you forget your wedding aniversary? Your boyfriend or girlfriend's birthday? Tax day? (everyone wants to forget that one!) No need for that to happen anymore with Linux. Just type:

```
date
```

You'll get this: (or something like it, actually. If you get the same thing as I do, then I'd consider buying lottery tickets)

Thu Sep 7 20:34:13 CEST 2000

You probably get everything here. If you're living in central Europe, you will recogize the 'CEST' part. That stands for 'Central European Standard Time'. Linux recognizes world time zones and you set this up when you installed Linux. If you live on Mars, you're out of luck, unfortunately.

There are other uses of the 'date' command. To see just the date, type:

```
date +%D
```

To see just the time, type:

```
date +%T
```

To see on what day Christmas falls this year (really, I'm not kidding!), type:

```
date --date 'Dec 25'
```

and you'll get the day that Christmas falls on this year. Substitute that for any date that you'd like to see.

There are many other options. Consult your manual file ('man date') or ('info date')

cal

Typing cal will give you the calendar of the present month on your screen, in the nice standard calendar format. There are a lot of useful options.
If you type:

```
cal 2000
```

You'll get the entire calendar for the year 2000. Substitute any year you like.

If you type:

```
cal 12 2025
```

You'll see the calender for December of 2025. Substitute any year or month you like.

If you add the option cal -m, the week will start on Monday, as it is preferred in many countries.

Just for fun, I typed

```
cal 10 1492
```

and I found out that Columbus discovered America on a Friday. That was good luck for him because that way he got to relax for the weekend.

exit

As you can guess, you can get out of a terminal with the exit command. If you're working in text mode, typing exitwill prompt you to login again. If you want to work as another user, use logout instead.

If you're in x-windows, exit will close the X-Terminal you're working with exit with the option "stage right" will get you an error message.

File Backup (tar/gzip)

Let's face it, computers aren't perfect. Linux is an "almost perfect" operating system, but things do happen and data is sometimes lost. The best way to avoid problems is to backup your files. Linux provides two key programs to do this: 'tar' and 'gzip'

First we'll start with 'tar'. This program assembles various files into one package, commonly called a "tarball". Let's say you have some files - notes that you've taken during this course.

You have:

```
notes_1.txt
notes_2.txt
notes_3.txt
notes_4.txt
notes_5.txt
```

and you've placed them in a directory called /linux_course. You want to back them up and keep them on a floppy, let's say. You would type the following command to package them in a tarball.

```
tar -cvf linux_notes.tar notes*.txt
```

First, you have tar, the name of the program. Then you have the options, c (--create) v (--verbose-show what files they are) (f--file -make a file - should always be the last option) Then you have the name of the file you want to create (linux_notes.tar) and the files you want to backup (notes*.txt).

This presupposes that you may have other files in the directory that you don't want to include. If you want to include ALL files in a directory, just substitute notes*.txt for *.*.

If you've got good data storage capabilities (Jaz or Zip drives, a CD writer or a tape backup drive), you might want to back up whole directories along with their corresponding subdirectories. Then you would enter in the directory, let's say /home/bob/ and issue the command:

```
tar -cvf bob_backup.tar *
```

With one asterisk, you will include directories and files without extensions (my_file as opposed to my_file.txt). Be prepared to get a fairly voluminous tarball.

This is the first step in the backup process. Now let's look at the second step; the compression of these files.

Using 'gzip'

As we mentioned, 'tar' just assembles the files together into only one file. There is no reduction in the size of these files (the tarball might even be bigger!) Now we would have to do one more thing in order to reduce this file into a more manageable size: use 'gzip'.

gzip is the preferred compression tool for Linux. To reduce the size of your tar file, you would issue the following command:

```
gzip your_tar_file.tar
```

and the tar file would be compressed. You can also compress a regular file using the same command, but gzip is used primarily with tarballs.
The result would be a file like this: your_tar_file.tar.gz

The two file extensions show us that the file is a tarball and it is compressed with the 'gzip' format. You can now proceed to

store this as you see fit.
Putting it all together

tar

tar has an option built into it to use 'gzip' to zip the file at the same time you make the tarball. If you add z to the options, and change the name of the file to create to a .gz extension, you have the whole shebang in one step. Our previous example would be modified to this:

```
tar -czvf bob_backup.tar.gz *
```

Remember f should always be the last option.

UnTar

Using 'tar' and 'gzip' sort of supposes that you're going to want to "untar" and "unzip" these files at one point or another.

The easiest way for doing this is to use 'tar' for the whole process. You would locate the zipped tarball in question and then ask yourself a question:
Did I make any changes to the files inside the tarball after I made it? If you did, then you've got an old tarball. If you untarred it in the same directory, you'd overwrite the existing ones. If you would like a copy of the old file, untar it in a different directory. If you don't want the old files, then you should make a new tarball. It's pretty standard backup practice.

When you've decided what you want to do, to proceed with the "untarring", issue this command:

tar -zxvpf my_tar_file.tar.gz

I've used my preferred options. I'll explain them:

```
-z - unzip the file first
-x - extract the files from the tarball
-v - "verbose" (i.e tar tells you what files it's extracting)
-p - preserves dates, permissions of the original files
-f - use the file in question (if you don't specify this, tar just sort of sits around doing nothing)
```

The files are extracted and your original tarball is preserved (my_tar_file.tar.gz).

You can also untar the file and then use gzip separately. Just leave the z option out of the previous example and type:

gzip -d my_tar_file.tar.gz or

gunzip my_tar_file.tar.gz

(gunzip runs gzip -d "automagically"!)

These commands are good if you've just zipped a regular file (not a tarball).

Other compression tools

zip

Most Linux distributions come with other tools to compress files. One of these is zip, famous in the MS-DOS/Windows world. If you're planning on compressing files to give to someone who (still) uses the Windows operating system, this might be your best bet. You can also use unzip if someone gives you a file compressed with 'zip'. Consult the man file (man zip) for specific instructions on using this tool.

bzip2

There is also another tool that is rapidly gaining acceptance in the Linux world: bzip2. As a matter of fact, the Linux kernel source package, usually comes "bzipped". When you compile a kernel (create a custom kernel for yourself from source) there is an option to create a bzipped kernel. This is supposed to become the official way of doing it in the near future, so it may be a good idea to get to know 'bzip2'

For all practical purposes you would use this tool in the same way as you would 'gzip'. The compression factor is supposed to be a little better. There are some differences in options for more advanced users. Consult man bzip2for more information.

Cool Shortcuts

Who doesn't like shortcuts? Well, the answer is my wife. But only because my shortcuts usually double the length of our trips... Anyway, if you are going to spend any time in the Linux command line interface, I highly recommend learning a few easy shortcuts that will make your life MUCH easier!

Auto Complete Anything You Type!

One of the best shortcuts is the auto-complete feature in Linux. Just start typing the name of a file, directory or program and then hit .


```
ls m
```

Then push the key. Linux is going to beep a couple of times, but you keep pushing. You will now see every file in the directory that begins with the letter 'm'

Now add an 'o' on to `ls m` so you get `ls mo` - now push the tab key.


```
ls mo
```

You should see 'motd'. This is a file that contains your startup message. SuSE has a famous one that says 'Have a lot of fun!'. I like that one so I haven't changed it, but you can change it so that Linux says anything you want when you log in.

Linux Saves Everything That You Type In Your History File!

Do you want to make sure that lilo.conf is still in there? You don't have to type 'lilo.conf' or even part of it and press the tab key anymore. You just have to press the up arrow. Your last commands will appear when you do that.

Your commands are saved in a history file located in your home directory. The more times you press the up button, the farther back in time you go. Pressing the down button gets you back to your most recent commands. Just stop on the command you want and press 'enter'. You can even type the command history and all of the last 400 or so commands you've typed will be presented. As you get more proficient in Linux, you'll find that thisreally comes in handy. You can often find out the answer to the question: How did I do that? by consulting your shell history.

The /usr directory

Let's talk about using some options with commands.


```
cd /usr  # this will take you to the /usr directory
```

TRY THIS CODE

```
ls -l  # This will give you a more detailed view about the contents of your current directory, which happens to be the /usr  # directory.
```

You will see more information, like dates, some numbers, letter combinations. It will say 'root' a lot. We'll get into more detail about what all of that means later in the course. You'll see mainly sub-directories here. The usr/ directory contains files and programs meant to be used by all of the users on the system.

Using 'Pipes' in Linux

In this lesson, we're going to do a little plumbing. Plumbing with a computer? Well, Linux is so flexible that it even allows you to do plumbing with it. Well.... actually, it's just a little witticism of mine because the command we're going to learn in this lesson is called 'pipe', and plumbers work with pipes. To use the pipe command, you don't type: pipe. You press the '|' key. The location will vary on keyboards from country to country. This symbol is like two vertical slashes, one on top of the other.

This is the first time that we're going to see a command that's meant to be used with other commands. That means that the pipe will separate two commands so that they will be done one after the other. Let's try some plumbing.

For example, if you looked at the contents of your /proc directory with:

```
ls -l /proc
```

it would be too big to fit in one screen. So if we typed

```
ls -l /proc | more
```

you could scroll down with the ENTER key and see it all.

Actually, **ls -l /proc | less** is a better solution because you can scroll up and down with the arrow keys. Remember that in a previous lesson we said: "Less is more than more".

You're probably going to end up using this a lot. You're personal directory in /home will fill up and pretty soon 'ls -l' will overflow in your terminal or x-terminal window.

There's a little short cut if you've forgotten to use the pipe. You can also scroll up and down in a terminal with the SHIFT-PAGE UP / SHIFT-PAGE DOWN keys.

Editing Files

Text Editors

If I were to choose one of the main reason why people use PCs, I would definitely say for writing. With a computer and a word processing program, cross outs, white out and crumpled up paper has disappeared forever. Linux is just as well suited for word processing as any other operating system. There are several excellent word processing programs for Linux like AbiWord, KWord, part of the KOffice suite and the OpenOffice.org suite's word processor. We'll talk about these kinds of programs in a later lesson. First, we should talk about the terminal mode text editors that are available for Linux.

Why use a text editor?

A text editor is just like a word processor without a lot of features. All operating systems come with a basic text editor. Linux comes with several. The main use of a text editor is for writing something in plain text with no formatting so that another program can read it. Based on the information it gets from that file, the program will run one way or another.

The text editor "vi"

The most popular text editor for Linux is called 'vi'. This is a program that comes from UNIX. There is a more recent version called 'vim' which means 'vi improved'. The problem with 'vi' or 'vim' is that a lot of people don't like it. You have to remember a lot of key combinations to do stuff that other text editors will do for you more easily.
We should go through some basic 'vi' commands, because I have found that 'vi' is good if I want to get into a text file quickly and change something or I want to write a short note to myself. I generally do not use "vi" for anything that requires more than about 30 seconds of work, but there are people who swear by 'vi' and do all kinds of things with it like designing entire websites.

Working with 'vi'

Let's make a text file. Type:

```
vi try_vi
```

You'll see a line of tildes down the left side and the name 'tryvi' at the bottom and [new file].

To write something, you have to press **ESC** and the 'i' key (i for insert). Even if you don't press 'ESC-i' it usually gets the idea that you want to type something and lets you do it after a few keystrokes. You should get used to the '**ESC-i**' keys so you don't end up writing 'ar John' instead of 'Dear John'.

Press **ESC + 'i'** then type: **hello vi**

If you wrote jello vi or jello bi or something I don't want to know about, you can always erase your mistakes with the backspace key.

To save this file, you would press **ESC** then the colon key '**:**' then '**w**' (write)

To save the file and quit vi, you would press ESC, ESC the colon key ':' then wq (write, quit)

To quit without saving, press ESC, ':' then 'q'. Vi may protest if you've written something and you don't want to save it. If you press ESC ':' 'q!' with an exclamation point, vi will accept it and not save your changes.

That's vi in a nutshell, or more like a sesame seed. There are a lot of commands in vi - and you may explore those on your own at a later date, on your own terms and in the privacy of your own home.

Using 'joe'

'joe'- sounds like a comic strip. Actually, they are two other text editors that I like and I think are a little easier to manage. They're like 'vi' in that you use them to create and edit non-formatted text, but they're a little more user-friendly. Using 'joe'
'joe' was created by Joseph Allen, so that's why it's called Joe. I suppose if his name had been Hrothgar Allen, it would have been

called 'hroth'.

To use 'joe', you could type:

```
joe try_joe
```

You won't see the tildes like vi. It looks a little friendlier. The majority of joe's commands are based on the **CTRL-K** keys and a third key. The most important of these is **CTRL-K-H** which gets you 'help'. Help shows you the key combinations to use with 'joe'.

The most important thing about 'joe' is the logical concept that you can just start writing if you want. Try writing anything you want.

To save it, press **CTRL-K-D**. To save and quit, **CTRL-K-X**.

To quit without saving, **CTRL-C**, (without the K).

If you want to see the other features of 'joe', press **CTRL-K-H**, as I mentioned before.

My favorite little added feature of 'joe' is that if you edit a file again, it will save the previous file with a tilde on the end, like '**tryjoe~'** That little tilde file has saved my life a couple of times. (well, maybe not my life) But it has saved me a lot of work. I've made some changes to a file and then found out that wasn't a good idea. I could always fall back on the tilde file, which is a copy of your previous edit.

'joe' is a very good option for writing those short text files that you'll need.

Using 'pico'

'pico' is another friendly text editor. If you type:

```
pico try_pico
```

You'll see the commands you need in 'pico' specified at the bottom. You can just start writing anything you want.

To save the file, press **CTRL-o**. To save and quit or to just quit, press **CTRL-x**

Pico will always ask you if you want to do what you're doing. That's good. Questions like that will keep you from sending a file into non-existence without wanting to. All the other commands you'll need are at the bottom of the page.

Well, this is our little overview of the main text editors available for Linux. In our next lesson, we're going to need to use one in order to make our work in Linux a little bit safer and easier.

Virtual Terminals

One of the coolest things that Linux has to offer is the concept of virtual terminals. Back in the days of MS-DOS, one program could only be run by one user at a time. Linux in non-graphics mode may resemble MS-DOS somewhat, but that's where the similarities end. Linux is a true multi-tasking, multi-user system. Unlike MS-DOS, you can work as more than one user with more than one program at a time

The ALT-F keys

Let's say, if you were working as a user, 'bob' for example, and you found that you needed to do something as 'root'. You wouldn't have to shutdown the program you were working with. You could just press ALT-F2 and Linux will prompt you to login as a different user, in this case, 'root'. You'd just type the root password and then you can do stuff as 'root'. Pretty cool, wouldn't you say?

The combination of ALT, plus the F keys will allow you to login as a different user, or as the same user, but to run a different program. All you then need to do is type: 'exit' when your finished, and then press ALT-F1 again to get back to your original terminal .

A preview of virtual terminals in X-window

It's true that the 1990's brought us the era of the graphic user interface, popularized by Macintosh and Microsoft Windows. This gave us the opportunity to have various programs running at the same time. The X-window system of Linux will let you do this as well, but then we can add the concept of multi-user to it.

If you've been experimenting with your windows manager already, you might want to try one more thing. The combination CRL-ALT-F6 will get you out of your windows manager momentarily so you can login as a different user. Pressing ALT-F7 will get you back to your windows manager again. We'll mention this again in the lesson on X-window.

When to work as root & When to Work as a System User

When you should work as *root*

You have now installed Linux and the first thing you did was login as 'root'. Then you provided a password so that you and ONLY you could login to the system as 'root'. When you decide to work as root, you had better go into a phone booth first and change into a blue suit with a big 'S' on the front because 'root' is known as the 'superuser' (you can skip the red tights if you want).

That's really not meant to be just a witty reference to the Man of Steel. Actually, it is much more glamorous to be 'Superman' but root is actually more like the 'janitor' of the Linux system. Root has the keys to everything. He can shut off the lights, shut off the heat, lock you out of the building; he has to clean up the system and in the end make sure everything runs. And the most important thing about being a janitor - he sees everything.

'root' is not for routine work

As I mentioned, Linux makes your computer a true multi-user system, which means that besides root, you can and should work as another person. I say 'should' because doing routine work as 'root' could be hazardous to your health. When I first started using Linux myself, information was not all that readily available and I still had that 'one computer- one user' concept in my brain. It was after I had trashed all of the files and programs that make Linux run that I realized that working regularly as root wasn't a good idea.

Working as another user

Well, then how do you do your day to day work with Linux? That's easy. You do it by working as a user other than root. You may pick the name you like. Try your name. If your name is "Bob" then you could create a user account for 'bob'. By the way, if Prince Charles is reading this, Charles Philip Arthur George is a bit too long and has spaces, which Linux doesn't like - he should try 'charlie'. One thing I find EXTREMELY helpful is that with Linux, I can work as different people. It should not be inferred here that I have split-personality disorder. I am just a guy with a couple of different jobs. This way I can organize my work a lot better and backups are easier this way too.

Adding a new user

Well, Bob, now it's time to create your account. If your name is Hrothgar, use 'Hrothgar' or 'Hrothie' instead of 'bob' for the remainder of the lesson.

Now 'root' has to do this stuff. Yes, I know I just warned you about working as root, but this is where you have to exert your authority. Most major distributions have tools to do this. SuSE, for example, has a nice tool called YAST which lets you add users painlessly. You just fill in the correct information. Consult your Linux version for information on their tools. There is also the get-your-hands-dirty way of doing this. Actually you won't get your hands dirty unless you're eating barbecued ribs at the same time.

Using 'useradd' and 'passwd'

To add a new user, you can also use the command 'useradd'. Kind of a logical name, isn't it?

Try this:

```
useradd bob
```

You probably won't see any fireworks go off. You might not see anything. That doesn't matter. Linux has been told that there's a new user and his name's 'bob'.

Now you should give yourself a password.

Do this:

```
passwd bob
```

Linux will ask you for your password. Follow the same advice I gave previously about passwords. Also, don't use your 'root' password. Like 'one man, one vote' it's 'one user - one password'. You will be asked to repeat it.

What's in your user directory

When you create a new user, there is a directory created for that user in /home. To see what's in this new directory, you have to do the following.
Go to the /home directory. - typing:

```
cd /home
```

you can make sure you're in the /home directory by typing:

```
pwd
```

You'll see this: /home.

Now you need to type:

```
cd bob
```

You can type:

```
ls -a
```

to see what's in the directory. We'll go into more detail with the ls command later. With the -a option, you'll see some files that begin with a '.' (period/dot). Those would normally be hidden from you if you didn't use the -a.

Now you can go to work as 'bob', 'pcharles' or whoever you happen to be.