What is "the shell"?

Simply put, the shell is a program that takes your commands from the keyboard and gives them to the operating system to perform. In the old days, it was the only user interface available on a Unix computer. Nowadays, we have *graphical user interfaces (GUIs)* in addition to *command line interfaces (CLIs)* such as the shell.

On most Linux systems a program called **bash** (which stands for Bourne Again SHell, an enhanced version of the original Bourne shell program, **sh**, written by Steve Bourne) acts as the shell program. There are several additional shell programs available on a typical Linux system. These include: **ksh**, **tcsh** and **zsh**.

What's an xterm, gnome-terminal, konsole, etc.?

These are called "terminal emulators." They are programs that put a window up and let you interact with the shell. There are a bunch of different terminal emulators you can use. Most Linux distributions supply several, such as: **xterm**, **rxvt**, **konsole**, **kvt**, **gnome-terminal**, **nxterm**, and **eterm**.

Starting a Terminal

Your window manager probably has a way to launch programs from a menu. Look through the list of programs to see if anything looks like a terminal emulator program. In KDE, you can find "konsole" and "terminal" on the Utilities menu. In Gnome, you can find "color xterm," "regular xterm," and "gnome-terminal" on the Utilities menu. You can start up as many of these as you want and play with them. While there are a number of different terminal emulators, they all do the same thing. They give you access to a shell session. You will probably develop a preference for one, based on the different bells and whistles each one provides.

Testing the Keyboard

Ok, let's try some typing. Bring up a terminal window. You should see a shell prompt that contains your user name and the name of the machine followed by a dollar sign. Something like this:

```
[me@linuxbox me]$
```

Excellent! Now type some nonsense characters and press the enter key.

```
[me@linuxbox me]$ kdkjflajfks
```

If all went well, you should have gotten an error message complaining that it cannot understand you:

```
[me@linuxbox me]$ kdkjflajfks
bash: kdkjflajfks: command not found
```

Wonderful! Now press the up-arrow key. Watch how our previous command "kdkjflajfks" returns. Yes, we have *command history*. Press the down-arrow and we get the blank line again.

Recall the "kdkjflajfks" command using the up-arrow key if needed. Now, try the left and right-arrow keys. You can position the text cursor anywhere in the command line. This allows you to easily correct mistakes.

You're not logged in as root, are you?

Don't operate the computer as the superuser. You should only become the superuser when absolutely necessary. Doing otherwise is dangerous, stupid, and in poor taste. Create a user account for yourself now!

Using the Mouse

Even though the shell is a command line interface, you can still use the mouse for several things. That is, if you have a 3-button mouse; and you should have a 3-button mouse if you want to use Linux.

First, you can use the mouse to scroll backwards and forwards through the output of the terminal window. To demonstrate, hold down the enter key until it scrolls off the window. Now, with your mouse, you can use the scroll bar at the side of the terminal window to move the window contents up and down. If you are using **`xterm`**, you may find this difficult, since the middle button is required for this operation. If you have a 2-button mouse, it may have been configured to emulate a 3-button mouse. This means the middle button can be simulated by pressing down both the left and right buttons at the same time.

pressing down both the left and right buttons at the same time.

Next, you can copy text with the mouse. Drag your mouse over some text (for example, "kdkjflajfks" right here on the browser window) while holding down the left button. The text should highlight. Now move your mouse pointer to the terminal window and press the middle mouse button. The text you highlighted in the browser window should be copied into the command line. Did I mention that you will need a 3-button mouse?

A few words about focus...

When you installed your Linux system and its window manager (most likely Gnome or KDE), it was configured to behave in some ways like that legacy operating system.

In particular, it probably has its *focus policy* set to "click to focus." This means that in order for a window to gain focus (become active) you have to click in the window. This is contrary to traditional X windows behavior. If you take my advice and get a 3-button mouse, you will want to set the focus policy to "focus follows mouse". This will make using the text copying feature of X windows much easier to use. You may find it strange at first that windows don't raise to the front when they get focus (you have to click on the title bar to do that), but you will enjoy being able to work on more than one window at once without having the active window obscuring the the other. Try it and give it a fair trial; I think you will like it. You can find this setting in the configuration tools for your window manager.

Navigation

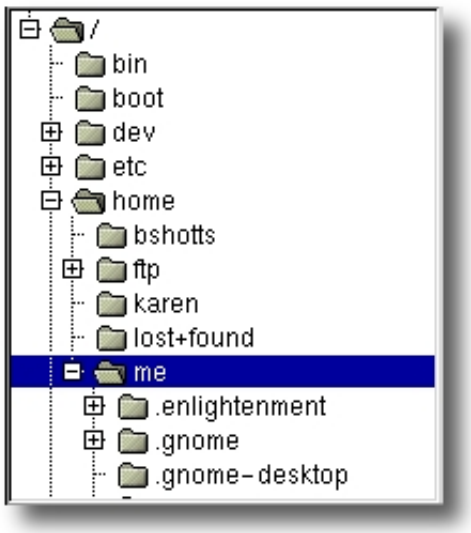
In this lesson, I will introduce your first three commands: **pwd** (print working directory), **cd** (change directory), and **ls** (list files and directories).

If you have not worked with a command line interface before, you will need to pay close attention to this lesson, since the concepts will take some getting used to.

File System Organization

Like that legacy operating system, the files on a Linux system are arranged in what is called a *hierarchical directory structure*. This means that they are organized in a tree-like pattern of directories (called folders in other systems), which may contain files and other directories. The first directory in the file system is called the *root directory*. The root directory contains files and subdirectories, which contain more files and subdirectories and so on and so on.

Most graphical environments today include a file manager program to view and manipulate the contents of the file system. Often you will see the file system represented like this:



One important difference between the legacy operating system and Unix/Linux is that Linux does not employ the concept of drive letters. While drive letters split the file system into a series of different trees (one for each drive), Linux always has a single tree. Different storage devices may contain different branches of the tree, but there is always a single tree.

pwd

Since a command line interface cannot provide graphic pictures of the file system structure, it must have a different way of representing it. Think of the file system tree as a maze, and you are standing in it. At any given moment, you stand in a single directory. Inside that directory, you can see its files and the pathway to its parent directory and the pathways to the subdirectories of the directory in which you are standing.

The directory you are standing in is called the *working directory*. To find the name of the working directory, use the **pwd** command.

```
[me@linuxbox me]$ pwd
/home/me
```

When you first log on to a Linux system, the working directory is set to your home directory. This is where you put your files. On most systems, your home directory will be called /home/your_user_name, but it can be anything according to the whims of the system administrator.

To list the files in the working directory, use the **ls** command.

```
[me@linuxbox me]$ ls

Desktop     Xrootenv.0    linuxcmd
GNUstep     bin           nedit.rpm
GUILG00.GZ  hitni123.jpg  nsmail
```

I will come back to **ls** in the next lesson. There are a lot of fun things you can do with it, but I have to talk about pathnames and directories a bit first.

cd

To change your working directory (where you are standing in the maze) you use the **cd** command. To do this, type **cd** followed by the *pathname* of the desired working directory. A pathname is the route you take along the branches of the tree to get to the directory you want. Pathnames can be specified in one of two different ways; *absolute pathnames* or *relative pathnames*. Let's deal with absolute pathnames first.

An absolute pathname begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed. For example, there is a directory on your system in which programs are installed for the X window system. The pathname of the directory is /usr/X11R6/bin. This means from the root directory (represented by the leading

directory is /usr/X11R6/bin. This means from the root directory (represented by the leading slash in the pathname) there is a directory called "usr" which contains a directory called "X11R6" which contains a directory called "bin".

Let's try this out:

```
[me@linuxbox me]$ cd /usr/X11R6/bin

[me@linuxbox bin]$ pwd
/usr/X11R6/bin

[me@linuxbox bin]$ ls

Animate               import              xfwp
AnotherLevel          lbxproxy            xg3
Audio                 listres             xgal
Auto                  lndir               xgammon
Banner                makedepend          xgc
Cascade               makeg               xgetfile
Clean                 mergelib            xgopher
Form                  mkdirhier           xhexagons
Ident                 mkfontdir           xhost
Pager                 mkxauth             xieperf
Pager_noxpm           mogrify             xinit
RunWM                 montage             xiterm
RunWM.AfterStep       mtv                 xjewel
RunWM.Fvwm95          mtvp                xkbbell
RunWM.MWM             nxterm              xkbcomp

and many more...
```

Now we can see that we have changed the current working directory to /usr/X11R6/bin and that it is full of files. Notice how your prompt has changed? As a convenience, it is usually set up to display the name of the working directory.

Where an absolute pathname starts from the root directory and leads to its destination, a relative pathname starts from the working directory. To do this, it uses a couple of special symbols to represent relative positions in the file system tree. These special symbols are "." (dot) and ".." (dot dot).

The "." symbol refers to the working directory and the ".." symbol refers to the working directory's parent directory. Here is how it works. Let's change the working directory to /usr/X11R6/bin again:

```
[me@linuxbox me]$ cd /usr/X11R6/bin
[me@linuxbox bin]$ pwd
```

```
[me@linuxbox bin]$ pwd
/usr/X11R6/bin
```

O.K., now let's say that we wanted to change the working directory to the parent of /usr/X11R6/bin which is /usr/X11R6. We could do that two different ways. First, with an absolute pathname:

```
[me@linuxbox bin]$ cd /usr/X11R6
[me@linuxbox X11R6]$ pwd
/usr/X11R6
```

Or, with a relative pathname:

```
[me@linuxbox bin]$ cd ..
[me@linuxbox X11R6]$ pwd
/usr/X11R6
```

Two different methods with identical results. Which one should you use? The one that requires less typing!

Likewise, we can change the working directory from /usr/X11R6 to /usr/X11R6/bin in two different ways. First using an absolute pathname:

```
[me@linuxbox X11R6]$ cd /usr/X11R6/bin
[me@linuxbox bin]$ pwd
/usr/X11R6/bin
```

Or, with a relative pathname:

```
[me@linuxbox X11R6]$ cd ./bin
[me@linuxbox bin]$ pwd
/usr/X11R6/bin
```

Now, there is something important that I must point out here. In almost all cases, you can omit the "./". It is implied. Typing:

the "./". It is implied. Typing:

```
[me@linuxbox X11R6]$ cd bin
```

would do the same thing. In general, if you do not specify a pathname to something, the working directory will be assumed. There is one important exception to this, but we won't get to that for a while.

A couple of shortcuts

If you type **cd** followed by nothing, **cd** will change the working directory to your home directory.

A related shortcut is to type **cd ~*user_name***. In this case, **cd** will change the working directory to the home directory of the specified user.

Important facts about file names

1. File names that begin with a period character are hidden. This only means that **ls** will not list them unless you say **ls -a**. When your account was created, several hidden files were placed in your home directory to configure things for your account. Later on we will take a closer look at some of these files to see how you can customize your environment. In addition, some applications will place their configuration and settings files in your home directory as hidden files.

2. File names in Linux, like Unix, are case sensitive. The file names "File1" and "file1" refer to different files.

3. Linux has no concept of a "file extension" like legacy operating systems. You may name files any way you like. The contents/purpose of a file is determined by other means.

4. While Linux supports long file names which may contain embedded spaces and punctuation characters, limit the punctuation characters to period, dash, and underscore. *Most importantly, do not embed spaces in file names.* If you want to represent spaces between words in a file name, use underscore characters. You will thank yourself later.

Linux® is a registered trademark of Linus Torvalds.

Looking Around

Now that you know how to move from working directory to working directory, we're going to take a tour of your Linux system and, along the way, learn some things about what makes it tick. But before we begin, I have to teach you some tools that will come in handy during our adventure. These are:

- **`ls`** (list files and directories)
- **`less`** (view text files)
- **`file`** (classify a file's contents)

ls

The **`ls`** command is used to list the contents of a directory. It is probably the most commonly used Linux command. It can be used in a number of different ways. Here are some examples:

Examples of the ls command

Command	***Result***
`ls`	List the files in the working directory
`ls /bin`	List the files in the /bin directory (or any other directory you care to specify)
`ls -l`	List the files in the working directory in long format
`ls -l /etc /bin`	List the files in the /bin directory and the /etc directory in long format
`ls -la ..`	List all files (even ones with names beginning with a period character, which are normally hidden) in the parent of the working directory in long format

	directory in long format

These examples also point out an important concept about commands. Most commands operate like this:

```
command -options arguments
```

where *command* is the name of the command, *-options* is one or more adjustments to the command's behavior, and *arguments* is one or more "things" upon which the command operates.

In the case of **`ls`**, we see that **`ls`** is the name of the command, and that it can have one or more options, such as **`-a`** and **`-l`**, and it can operate on one or more files or directories.

A Closer Look At Long Format

If you use the **`-l`** option with **`ls`**, you will get a file listing that contains a wealth of information about the files being listed. Here's an example:

```
-rw-------   1 bshotts  bshotts       576 Apr 17  1998 weather.txt
drwxr-xr-x   6 bshotts  bshotts      1024 Oct  9  1999 web_page
-rw-rw-r--   1 bshotts  bshotts    276480 Feb 11 20:41 web_site.tar
-rw-------   1 bshotts  bshotts      5743 Dec 16  1998 xmas_file.txt

----------     -------  -------  -------- ------------ -------------
    |              |        |         |          |              |
    |              |        |         |          |          File Name
    |              |        |         |          |
    |              |        |         |          +---  Modification Time
    |              |        |         |
    |              |        |         +-------------   Size (in bytes)
    |              |        |
    |              |        +------------------------        Group
    |              |
    |              +--------------------------------        Owner
    |
    +-------------------------------------------------   File Permissions
```

File Name
: The name of the file or directory.

Modification Time
: The last time the file was modified. If the last modification occurred more than six months in the past, the date and year are displayed. Otherwise, the time of day is shown.

Size
: The size of the file in bytes.

Group
: The name of the group that has file permissions in addition to the file's owner

The name of the group that has file permissions in addition to the file's owner.

Owner

The name of the user who owns the file.

File Permissions

A representation of the file's access permissions. The first character is the type of file. A "-" indicates a regular (ordinary) file. A "d" indicates a directory. The second set of three characters represent the read, write, and execution rights of the file's owner. The next three represent the rights of the file's group, and the final three represent the rights granted to everybody else.

less

`less` is a program that lets you view text files. This is very handy since many of the files used to control and configure Linux are human readable (as opposed to the legacy operating systems).

What is "text"?

There are many ways to represent information on a computer. All methods involve defining a relationship between the information and some numbers that will be used to represent it. Computers, after all, only understand numbers and all data is converted to numeric representation.

Some of these representation systems are very complex (such as compressed image files), while others are rather simple. One of the earliest and simplest is called *ASCII text*. ASCII (pronounced "As-Key") is short for American Standard Code for Information Interchange. This is a simple encoding scheme that was first used on Teletype machines to map keyboard characters to numbers.

Text is a simple one-to-one mapping of characters to numbers. It is very compact. Fifty characters of text translates to fifty bytes of data. Throughout a Linux system, many files are stored in text format and there are many Linux tools that work with text files. Even the legacy operating systems recognize the importance of this format. The well-known NOTEPAD.EXE program is an editor for plain ASCII text files.

The **`less`** program is invoked by simply typing:

```
less text_file
```

This will display the file.

Controlling less

Once started, **`less`** will display the text file one page at a time. You may use the Page Up and Page Down keys to move through the text file. To exit **`less`**, type "q". Here are some

Page Down keys to move through the text file. To exit **less**, type "q". Here are some commands that **less** will accept:

Keyboard commands for the less program

Command	*Action*
Page Up or b	Scroll back one page
Page Down or space	Scroll forward one page
G	Go to the end of the text file
1G	Go to the beginning of the text file
/characters	Search forward in the text file for an occurence of the specified *characters*
n	Repeat the previous search
q	Quit

file

As you wander around your Linux system, it is helpful to determine what a file contains before you try to view it. This is where the **file** command comes in. **file** will examine a file and tell you what kind of file it is.

To use the **file** program, just type:

```
file name_of_file
```

The **file** program can recognize most types of files, such as:

Various kinds of files

File Type	*Description*	*Viewable as text?*
ASCII text	The name says it all	yes
Bourne-Again shell script text	A **`bash`** script	yes
ELF 32-bit LSB core file	A core dump file (a program will create this when it crashes)	no
ELF 32-bit LSB executable	An executable binary program	no
ELF 32-bit LSB shared object	A shared library	no
GNU tar archive	A tape archive file. A common way of storing groups of files.	no, use **`tar tvf`** to view listing.
gzip compressed data	An archive compressed with **`gzip`**	no
HTML document text	A web page	yes
JPEG image data	A compressed JPEG image	no
PostScript document text	A PostScript file	yes

document text		
RPM	A Red Hat Package Manager archive	no, use **`rpm -q`** to examine contents.
Zip archive data	An archive compressed with **`zip`**	no

While it may seem that most files cannot be viewed as text, you will be surprised how many can. This is especially true of the important configuration files. You will also notice during our adventure that many features of the operating system are controlled by shell scripts. In Linux, there are no secrets!

A Guided Tour

It's time to take our tour. The table below lists some interesting places to explore. This is by no means a complete list, but it should prove to be an interesting adventure. For each of the directories listed below, do the following:

- **`cd`** into each directory.
- Use **`ls`** to list the contents of the directory.
- If you see an interesting file, use the **`file`** command to determine its contents.
- For text files, use **`less`** to view them.

Interesting directories and their contents

Directory	***Description***
`/`	The root directory where the file system begins. In most cases the root directory only contains subdirectories.
`/boot`	This is where the Linux kernel and boot loader files are kept. The kernel is a file called **`vmlinuz`**.
`/etc`	The **`/etc`** directory contains the configuration files for the system. All of the files in **`/etc`** should be text files. Points of interest: **`/etc/passwd`** The **`passwd`** file contains the essential information for each user. It is here that users are defined. **`/etc/fstab`** The **`fstab`** file contains a table of devices that get mounted when your system boots. This file defines your disk drives. **`/etc/hosts`** This file lists the network host names and IP addresses that are intrinsically known to the system. **`/etc/init.d`** This directory contains the scripts that start various system services typically at boot time.
`/bin, /usr/bin`	These two directories contain most of the programs for the system. The **`/bin`** directory has the essential programs that the system requires to operate, while **`/usr/bin`** contains applications for the system's users.

	system's users.
/sbin, /usr/sbin	The **sbin** directories contain programs for system administration, mostly for use by the superuser.
/usr	The **/usr** directory contains a variety of things that support user applications. Some highlights: **/usr/share/X11** Support files for the X Windows system **/usr/share/dict** Dictionaries for the spelling checker. Bet you didn't know that Linux had a spelling checker. See **look** and **ispell**. **/usr/share/doc** Various documentation files in a variety of formats. **/usr/share/man** The man pages are kept here. **/usr/src** Source code files. If you installed the kernel source code package, you will find the entire Linux kernel source code here.
/usr/local	**/usr/local** and its subdirectories are used for the installation of software and other files for use on the local machine. What this really means is that software that is not part of the official distribution (which usually goes in **/usr/bin**) goes here. When you find interesting programs to install on your system, they should be installed in one of the **/usr/local** directories. Most often, the directory of choice is **/usr/local/bin**.
/var	The **/var** directory contains files that change as the system is running. This includes: **/var/log** Directory that contains log files. These are updated as the system runs. You should view the files in this directory from time to time, to monitor the health of your system. **/var/spool** This directory is used to hold files that are queued for some process, such as mail messages and print jobs. When a user's mail first arrives on the local system (assuming you have local mail), the messages are first stored in **/var/spool/mail**
/lib	The shared libraries (similar to DLLs in that other operating system) are kept here.

	system) are kept here.
/home	**/home** is where users keep their personal work. In general, this is the only place users are allowed to write files. This keeps things nice and clean :-)
/root	This is the superuser's home directory.
/tmp	**/tmp** is a directory in which programs can write their temporary files.
/dev	The **/dev** directory is a special directory, since it does not really contain files in the usual sense. Rather, it contains devices that are available to the system. In Linux (like Unix), devices are treated like files. You can read and write devices as though they were files. For example **/dev/fd0** is the first floppy disk drive, **/dev/sda** (**/dev/hda** on older systems) is the first IDE hard drive. All the devices that the kernel understands are represented here.
/proc	The **/proc** directory is also special. This directory does not contain files. In fact, this directory does not really exist at all. It is entirely virtual. The **/proc** directory contains little peep holes into the kernel itself. There are a group of numbered entries in this directory that correspond to all the processes running on the system. In addition, there are a number of named entries that permit access to the current configuration of the system. Many of these entries can be viewed. Try viewing **/proc/cpuinfo**. This entry will tell you what the kernel thinks of your CPU.
/media,/mnt	Finally, we come to **/media**, a normal directory which is used in a special way. The **/media** directory is used for *mount points*. As we learned in the second lesson, the different physical storage devices (like hard disk drives) are attached to the file system tree in various places. This process of attaching a device to the tree is called *mounting*. For a device to be available, it must first be mounted. When your system boots, it reads a list of mounting instructions in the file **/etc/fstab**, which describes which device is mounted at which mount point in the directory tree. This takes care of the hard drives, but you may also have devices that are considered temporary, such as CD-ROMs and floppy disks. Since these are removable, they do not stay mounted all the time. The **/media** directory is used by the automatic device mounting mechanisms found in modern desktop oriented Linux distributions. On systems that require manual mounting of removable devices, the **/mnt** directory provides a convenient place for mounting these temporary devices. You will often see the directories **/mnt/floppy** and **/mnt/cdrom**. To see what devices and

	/mnt/floppy and **/mnt/cdrom**. To see what devices and mount points are used, type **mount**.

A weird kind of file...

During your tour, you probably noticed a strange kind of directory entry, particularly in the **/boot** and **/lib** directories. When listed with **ls -l**, you would have seen something like this:

```
lrwxrwxrwx     25 Jul  3 16:42 System.map -> /boot/System.map-2.0.36-3
-rw-r--r-- 105911 Oct 13  1998 System.map-2.0.36-0.7
-rw-r--r-- 105935 Dec 29  1998 System.map-2.0.36-3
-rw-r--r-- 181986 Dec 11  1999 initrd-2.0.36-0.7.img
-rw-r--r-- 182001 Dec 11  1999 initrd-2.0.36.img
lrwxrwxrwx     26 Jul  3 16:42 module-info -> /boot/module-info-2.0.36-3
-rw-r--r--  11773 Oct 13  1998 module-info-2.0.36-0.7
-rw-r--r--  11773 Dec 29  1998 module-info-2.0.36-3
lrwxrwxrwx     16 Dec 11  1999 vmlinuz -> vmlinuz-2.0.36-3
-rw-r--r-- 454325 Oct 13  1998 vmlinuz-2.0.36-0.7
-rw-r--r-- 454434 Dec 29  1998 vmlinuz-2.0.36-3
```

Notice the files, `System.map, module-info` and `vmlinuz`. See the strange notation after the file names?

These three files are called *symbolic links*. Symbolic links are a special type of file that point to another file. With symbolic links, it is possible for a single file to have multiple names. Here's how it works: Whenever the system is given a file name that is a symbolic link, it transparently maps it to the file it is pointing to.

Just what is this good for? This is a very handy feature. Let's consider the directory listing above (which is the `/boot` directory of an old Red Hat 5.2 system). This system has had multiple versions of the Linux kernel installed. We can see this from the files `vmlinuz-2.0.36-0.7` and `vmlinuz-2.0.36-3`. These file names suggest that both version 2.0.36-0.7 and 2.0.36-3 are installed. Because the file names contain the version it is easy to see the differences in the directory listing. However, this would be confusing to programs that rely on a fixed name for the kernel file. These programs might expect the kernel to simply be called `"vmlinuz"`. Here is where the beauty of the symbolic link comes in. By creating a symbolic link called `vmlinuz` that points to `vmlinuz-2.0.36-3`, we have solved the problem.

To create symbolic links, use the **ln** command.