What Is "The Shell"?

Simply put, the shell is a program that takes 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-like system such as Linux. 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 Unix shell program, **`sh`**, written by Steve Bourne) acts as the shell program. Besides **`bash`**, there are other shell programs that can be installed in a Linux system. These include: **`ksh`**, **`tcsh`** and **`zsh`**.

What's A "Terminal?"

It's a program called a *terminal emulator*. This is a program that opens a window and lets you interact with the shell. There are a bunch of different terminal emulators you can use. Most Linux distributions supply several, such as: **`gnome-terminal`**, **`konsole`**, **`xterm`**, **`rxvt`**, **`kvt`**, **`nxterm`**, and **`eterm`**.

Starting A Terminal

Your window manager probably has a way to launch a terminal from the menu. Look through the list of programs to see if anything looks like a terminal emulator. If you are a KDE user, the terminal program is called "konsole," in Gnome it's called "gnome-terminal." 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?

If the last character of your shell prompt is # rather than $, you are operating as the *superuser*. This means that you have administrative privileges. This can be potentially dangerous, since you are able to delete or overwrite any file on the system. Unless you absolutely need administrative privileges, do not operate as the superuser.

Using The Mouse

Even though the shell is a command line interface, the mouse is still handy.

Besides using the mouse to scroll the contents of the terminal window, 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. Release the left button and move your mouse pointer to the terminal window and press the middle mouse button (alternately, you can press both the left and right buttons at the same time if you are working on a touch pad). The text you highlighted in the browser window should be copied into the command line.

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 Window behavior. You should consider setting the focus policy to "focus follows mouse". 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 window 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

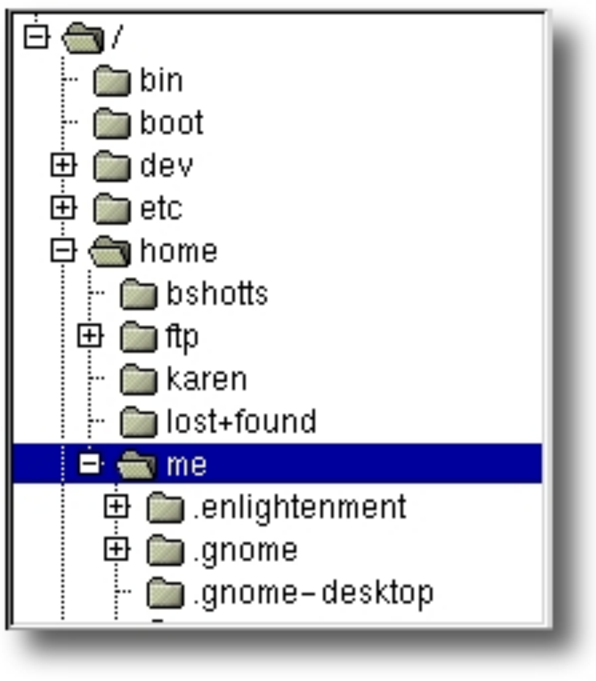
In this lesson, I will introduce your first three commands: **`pwd`** (print working directory), **`cd`** (change directory), and **`ls`** (list files and directories).

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-like operating systems such as 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 are located 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 look 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 most programs are installed. The pathname of the directory is /usr/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 "bin".

Let's try this out:

```
[me@linuxbox me]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
[me@linuxbox bin]$ ls
[                     lwp-request
2to3                  lwp-rget
2to3-2.6              lxterm
a2p                   lz
aalib-config          lzcat
aconnect              lzma
acpi_fakekey          lzmadec
acpi_listen           lzmainfo
add-apt-repository    m17n-db
addpart               magnifier

and many more...
```

Now we can see that we have changed the current working directory to /usr/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 notations to represent relative positions in the file system tree. These special notations are "." (dot) and ".." (dot dot).

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

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

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

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

Or, with a relative pathname:

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

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

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

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

Or, with a relative pathname:

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

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

```
[me@linuxbox usr]$ 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 Few 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.

Typing **`cd -`** changes the working directory to the previous one.

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. However, while Linux itself does not care about file extensions, many application programs do.

4. Though 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.