

INTRODUCTION TO THE LINUX ENVIRONMENT

MODULE OBJECTIVE

- Become familiar with the Linux operating system, the Linux shell, and sources of help

LESSON OBJECTIVES

- Learn what Linux is - Define kernel, shell, desktop, user, and process
- Understand basic shell features and be familiar with the shell environment.
- Remote Access
- Use online documentation to display manual information.

WHAT IS LINUX?

Linux is a variant of the UNIX operating system. UNIX has become a very popular O/S in the networking world, in part, due to its strong built-in communications handling ability. UNIX has been implemented on a wider range of machines than any other operating system. One thing that is important to remember is that there is no standard UNIX operating system. 'UNIX' has actually become a generic term for many different implementations of the original AT&T Bell Labs UNIX code, which dates back to the sixties. Most UNIX implementations are written specifically for a particular manufacturer's hardware, and the operating system is bundled with their computer systems. The name "UNIX" and the definition of UNIX are now owned by Novell, who has turned control over to the neutral organization X/Open. The version of UNIX being employed widely within the NWS for a number of national systems as well as local is Linux, mostly Red Hat Linux (but there are others). Linux, in various flavors, is an open-systems form derived from various UNIX implementations.

Linux is a multitasking operating system. A user can perform more than one task at a time. These tasks or processes can be run in the background so that the user can continue with other activities. For example, a user could be sorting a file or making a calculation in the background at the same time they are editing a file.

Linux is a multi-user operating system. Many users can be using the same machine. This is directly related to multitasking since each user has their own set of processes. The multi-user aspect allows groups of people to work easily together, sharing files and utilities.

There are several sources for the Linux operating system itself and a myriad of sources for Linux-based applications. Let us briefly consider major suppliers.

Debian GNU/Linux

The GNU Project was launched in 1984 to develop a complete UNIX-style operating system as free software: the GNU system. (*GNU is a recursive acronym for 'GNU's Not UNIX' and is pronounced 'guh-noo'.*) Variants of the GNU operating system, which use the kernel Linux, are now widely used. The Debian Project is an association of volunteers. A large part of the basic tools that fill out their operating system come from the GNU project. The Debian distribution contains a large number of software packages and each has a maintainer who is primarily responsible for keeping the package up-to-date, tracking bug reports, and communicating with the author(s) of the packaged software. It's most popular to install Debian from a CD that you buy for the price of the media, but you can also download on the Internet.

SUSE Linux

SUSE Linux is created by the openSUSE project, a Novell-sponsored community project dedicated to driving Linux adoption everywhere. Through openSUSE.org, you may download a free version (unsupported) from any one of over 100 mirror sites worldwide. The free offering allows you to use the latest open source desktop, server and application functionalities. There are supported versions and these fall under the categories of Personal and Enterprise. For example, new users to Linux might elect the current Personal version which includes complete end-user documentation, installable media for x86 and x86 64-bit systems, plus 90 days of end-user installation support. The Enterprise versions are those more suitable to larger networked business applications.

Fedora Project (Red Hat)

The goal of the Fedora Project is to work with the Linux community to build a complete general-purpose operating system exclusively from free software. It is also a proving ground for new technology that may eventually make its way into Red Hat products. It is not a supported product of Red Hat, Inc. The project will produce time-based releases of Fedora Core about 2-3 times a year with a public release schedule. Fedora downloads through Red Hat will initially focus on the x86 family of architectures. Fedora Core releases will be available as ISO images for both CDs and DVDs, and will also be available through other channels such as third-party online sales of physical media (or as inserts to textbooks). Fedora downloads are available at www.redhat.com/en_us/USA/fedora/

Red Hat Enterprise

Within NWS Red Hat Linux has generally been the Linux of choice and more than one release version is in use. **Red Hat** versions **7.x, 8 and 9** were designed as a vehicle for distributing the latest open source features to developers, enthusiasts and early adopters. Downloads were available free-of-charge and support was offered for a price. As Linux has matured the demand for a product that is designed for commercial IT deployments has grown. In response Red Hat has developed the **Red Hat Enterprise** Linux product family sold by annual subscription (see www.redhat.com/en_us/USA/rhel/). Enterprise includes a much longer release cycle, 18 months. Additionally products are all supplied with a year of support, and customers can continue to obtain support of any release for 7 years after release.

The Enterprise versions:

Server solutions

Red Hat Enterprise Linux AS

The top-of-the-line enterprise server, supporting high-end and mission-critical systems. Available with the highest levels of support.

Red Hat Enterprise Linux ES

The solution for small to mid-range servers used for the majority of today's business computing.

Client solutions

Red Hat Enterprise Linux WS

For technical workstation and single unit desktops/clients including software development, power desktop, targeted client applications, and High Performance Computing (HPC).

Red Hat Desktop

Ideal for volume client system deployments. Available in 10-unit and 50-unit packs bundled with Red Hat Network Proxy or Satellite Server.

Note: For Red Hat manuals check out www.redhat.com/docs/manuals/

ENVIRONMENT

OVERVIEW

The Linux operating system is really several components working in concert with each other. This section will define some terms such as kernel, shell, and desktop.

The **kernel** is the core of the UNIX/Linux operating system. It controls the computer's resources and allots time to different users and tasks. The kernel keeps track of the programs being run and is in charge of starting each user on the system. The kernel does NOT interact with the user to interpret commands. That is the function of the shell.

The **shell** is a program that the kernel runs for each user which sets up commands for execution. When you type in a command at your terminal, the shell interprets your command and calls the program that you want. The shell will support multiple users, multiple tasks, and multiple interfaces to itself. The shell uses standard syntax for its commands. There are several popular shells in use today. In an interactive mode, the shell is essentially running in an endless loop. When you enter a command and press <RETURN>, the shell assumes you are finished entering a command or commands. The shell then **parses** (*scans* and *interprets*) the tokens (input on the line separated by white space - <space>, <tab>, <return>) passed on the line. It then tries to interpret what you entered. Before it passes what you typed to the kernel (operating system), it must be interpreted. Commands are written to take advantage of shell features like filename expansion, metacharacters, redirection, pipelining, and variable replacement.

Some Commonly Used Shells

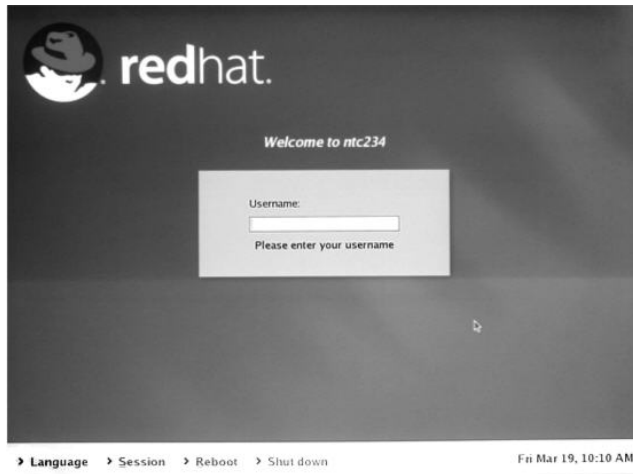
sh	BASH	BASH is a Unix shell written for the GNU project. It's name is an acronym for <i>Bourne-Again Shell</i> – a pun on the Bourne shell (see below). Bash is the default shell on most GNU/Linux systems (including Red-Hat) and can be run on most Unix-like operating systems.
sh	POSIX	POSIX-conforming command programming language and command interpreter residing in the file <i>/usr/bin/sh</i> . Can execute commands read from a terminal or a file. It conforms to current POSIX standards. It is similar to the Korn shell and it supports a history mechanism, job control, and various other features.

sh	Bourne	Bourne-shell command programming language and commands interpreter. It can execute commands read from a terminal or a file. This shell lacks features found in the POSIX and KORN shells, and often considered obsolete.
ksh	Korn	Korn shell command programming language and commands interpreter. It can execute commands read from a terminal or a file. This shell, like the POSIX shell supports a history mechanism, job control, and other features.
csh	C-shell	A command language interpreter that incorporates a command history buffer, C-language-like syntax, and job control facilities.
rsh	Restricted POSIX	A restricted version of the POSIX or Bourne shell command interpreter. It sets up a login name and environment whose capabilities are more controlled than normal user shells.

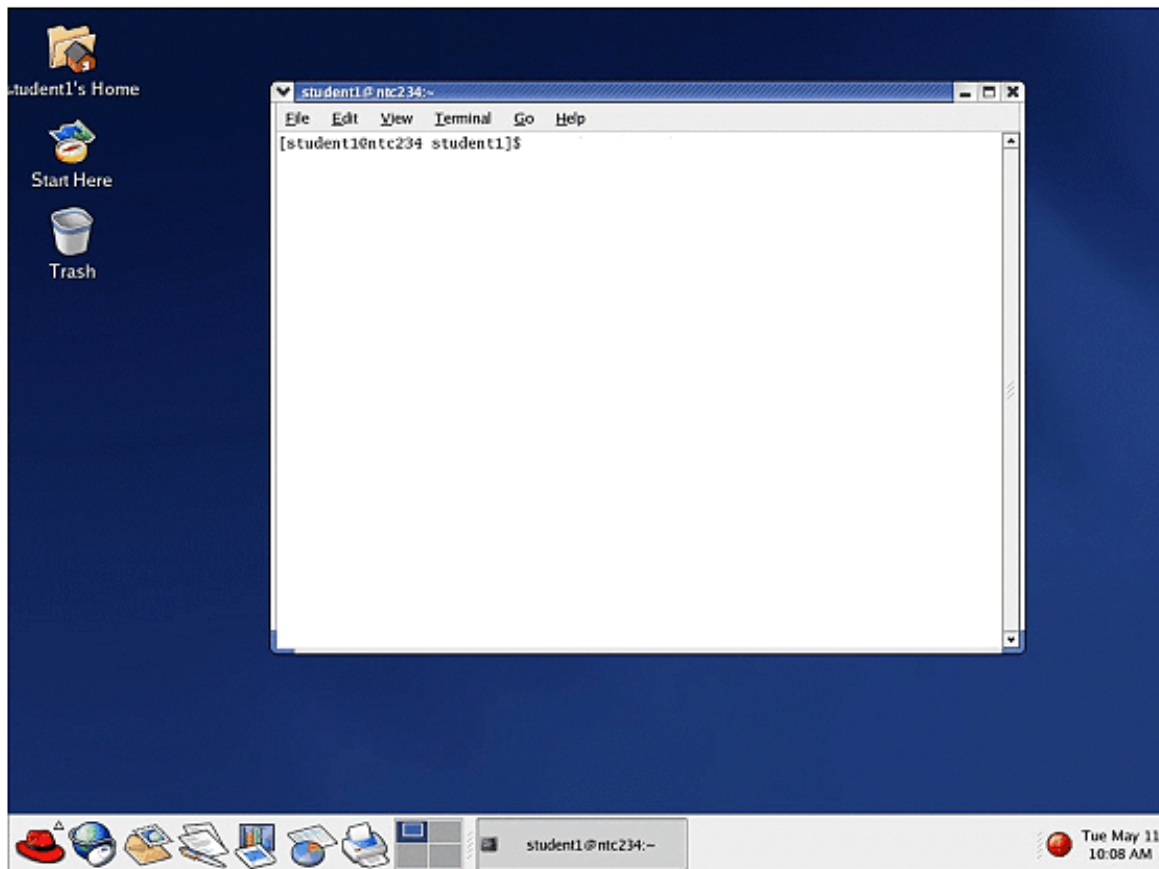
On a fully operational Linux system, before you deal directly with the shell you will interact with it through a graphical user interface (GUI) that runs as part of the **desktop** environment. The desktop environment creates a common graphical user environment and development platform. The desktop refers to the windows, pop-up menus, panels, icons and all the other graphical components on your screen. The purpose of the desktop is to make the operation of your computer easier; for example by making user applications, such as word processors, spreadsheets, and Web browsers, available through icons. Most Red Hat users are familiar with the highly developed, interactive **Gnome desktop environment**. Red Hat Linux also provides an alternative desktop environment called **KDE** that is used in AWIPS.

Red Hat has made the “look and feel” for both GNOME and KDE very similar. For example, the **Red Hat icon** represents the **main menu** for both desktops. The panel (at the bottom of the desktop screen) is similar as well. It starts the same five applications, a viewer for virtual desktops, an area to show running programs and a clock. System Tools, System Settings, and Server Settings accessed from the main menu are the same on both desktops. Both KDE and GNOME start with Home, Start Here, and Trash icons on the desktop, but KDE will have one additional. Although the desktops look alike they are very different. KDE has more tools to configure preferences and has more integrated applications. GNOME is more streamlined and offers a simple, efficient desktop for users.

The Gnome desktop before logging in.



The Gnome desktop after logging in.



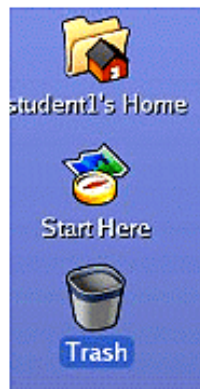
The desktop will make available working space and a number of icons and menus from which to start Linux activity. The Gnome environment under Red Hat Linux Enterprise 3 is shown above. You see the desktop window and working space, icons, and in this case where the user has opened up a terminal window to submit shell commands.

The exact way the desktop environment will be presented will vary from Linux implementation to Linux implementation. [Note in the exercises you will perform on the NWSTC student server as part of this course, you will not be using a desktop environment at all, rather you will be entering shell commands through a text-only remote shell access program). We'll present a few samples screen here from Red Hat Linux Enterprise 3, but be aware another system's desktop may look and function differently.

As we examine some sample screens, we'll start with the desktop icons, again these appear the same for Gnome and KDE:

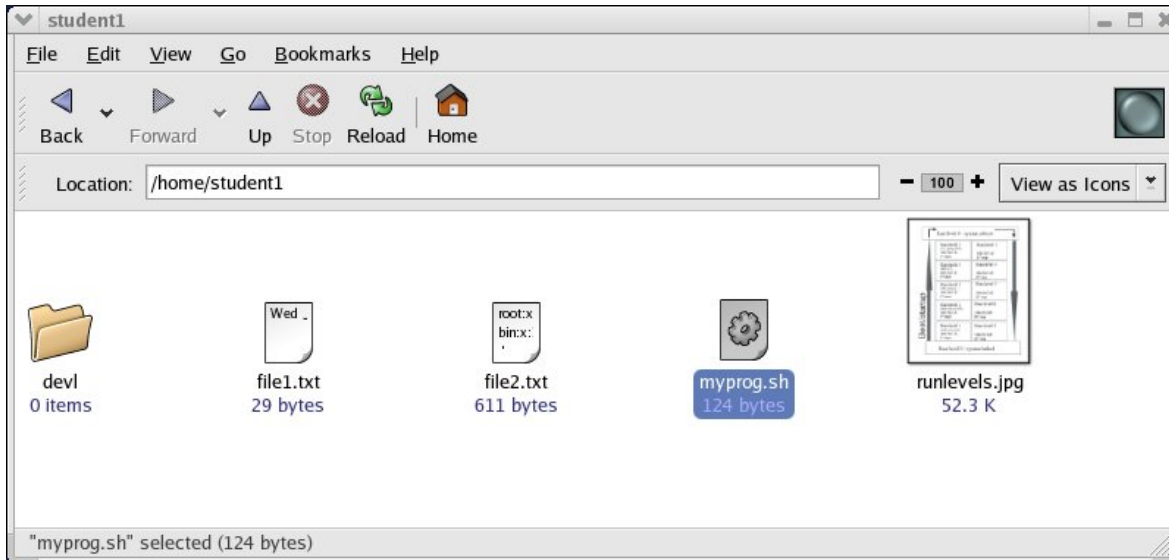
student1's Home

Double-click on this icon ----->

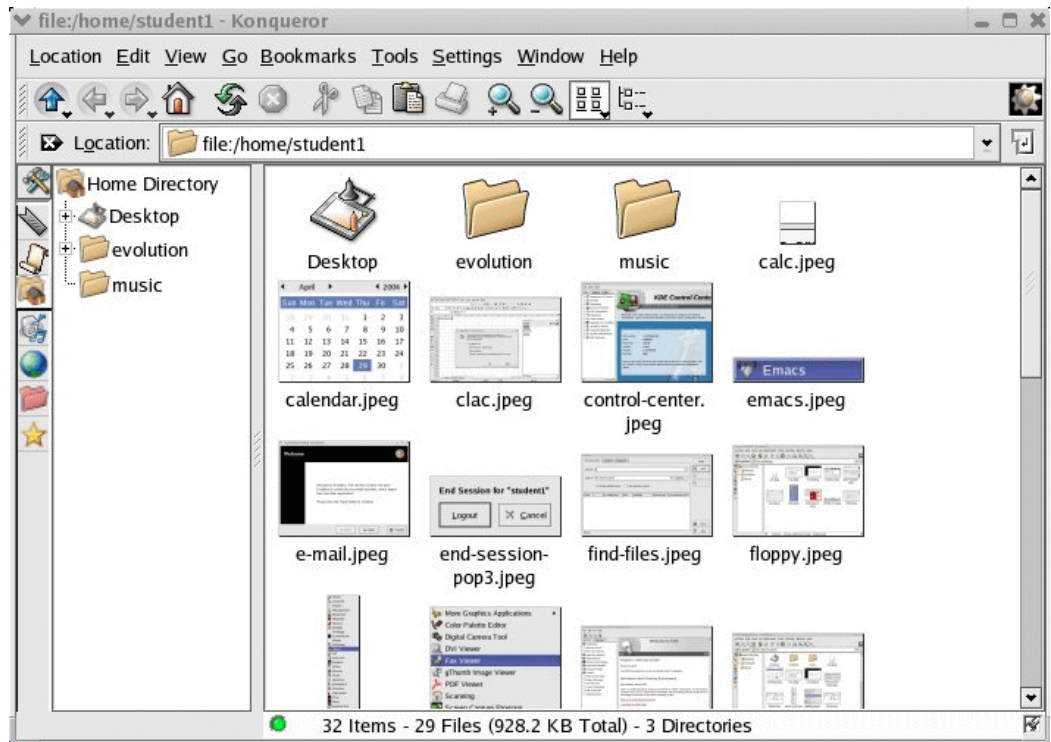


When either desktop starts, these three icons will appear in the upper-left portion of the screen. The top one represents the home directory for username's account, in this case student1. Double-click on this icon and a file manager program will display the contents of student1's folder....

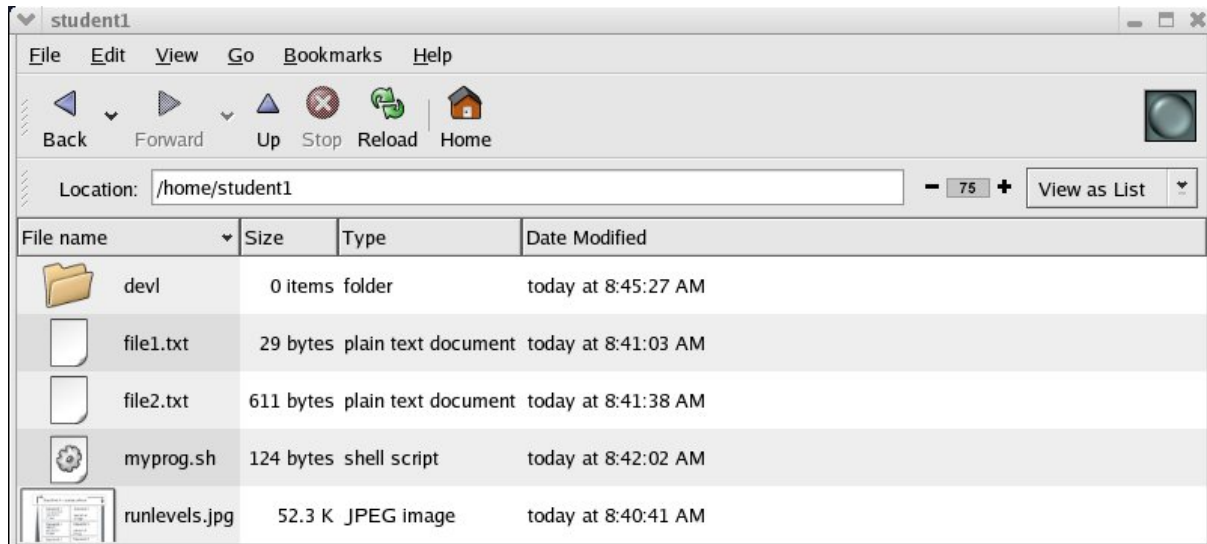
In Gnome the file manager is called Nautilus.



In KDE the file manager is called Konqueror.

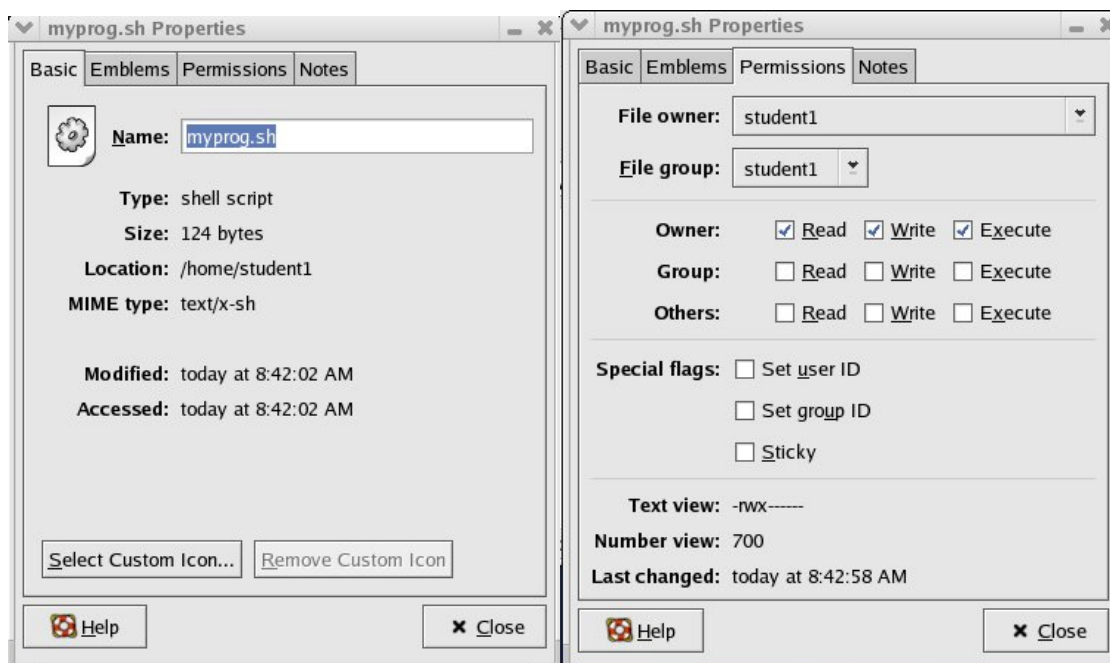


You can view file manager displays in list format by clicking on **View** on the top menu.



From either display you can double-click on files to open them

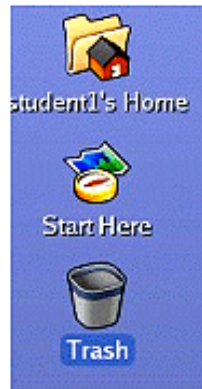
The File menu option will, among other things, let you open a selected file (for example edit a text file); or by selecting **Properties** off the **File** menu you can get file information such as dates and permissions....



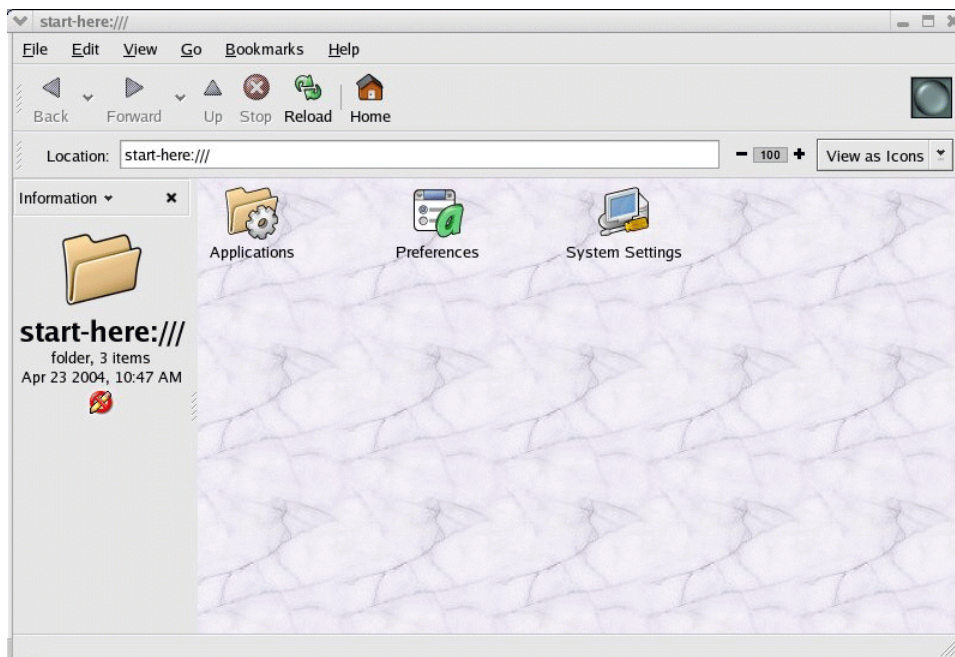
Though not shown, another highly useful Nautilus menu option is **Edit**, where among other things by choosing **Preferences** you can select to include hidden files in the files display. Hidden files are also called 'dot files' since the file names begin with a period. (We'll discuss file types, properties and operations in detail in another module).

Start Here

Double-click on this icon →



The “Start Here” icon in either desktop offers these three areas to work on: Applications, Preferences, and System Settings. The Gnome window is displayed as shown on the next page:



Panel at bottom of screen



This is the GNOME panel (KDE very similar). This is the portion of the screen that allows you to manage your desktop. Some of the functionalities available include:

1. Starting applications
2. Seeing which programs are active

There are many ways to change the panel. You can add applications or monitors, modify the placement of icons, or change the behavior of the panel itself.

Right-click in any open space on the panel and the Panel Menu will appear. From this Panel Menu you can perform the following:

1. Add/remove some application icons to the panel.
2. Change position, size and background of the panel.
3. Add panels to your desktop.

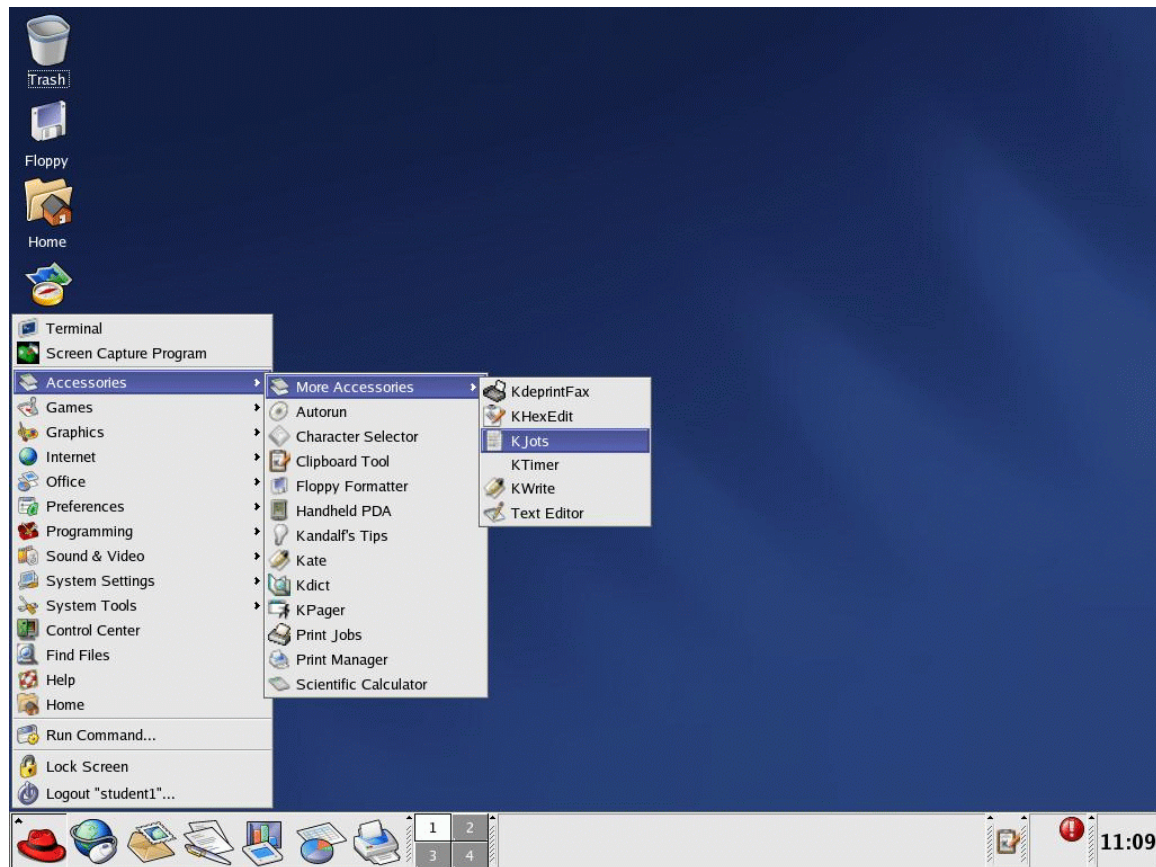
Red Hat Main Menu



This is the Red Hat desktop menu launcher button, more commonly known as the main menu icon. Clicking on this icon will bring up the main desktop menu. This menu has a set of applications that can be launched or started from the desktop.

Red Hat Main Menu (KDE)

A right-pointing arrow on the right of a selection line indicates that there is another pop-up with more choices. Click on that selection and the next pop-up screen will appear.

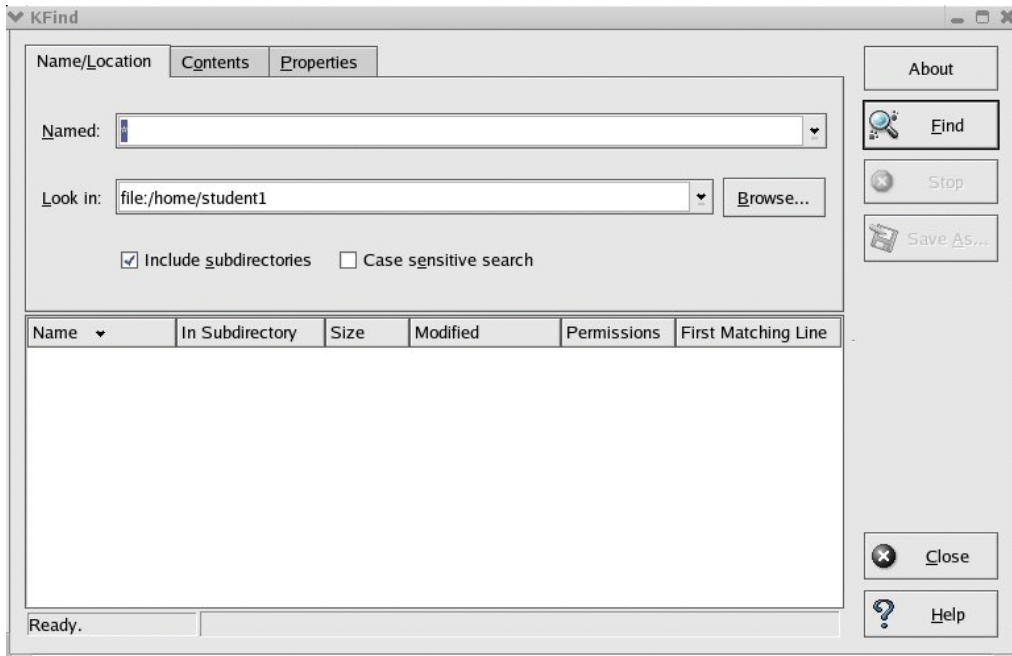


For example, click on accessories and the pop-up will appear as shown here. Click on “**Accessories**”, then in the new pop up window click on “**More Accessories**”. The next window has no right arrows so there are no more sub-menus. We will of course explore many of these menu options throughout the course.

For now, note that you can open a terminal window (text window) by selecting **Terminal** from this list by going to System Tools. In Gnome this can also be done by right-clicking on the desktop and selecting New Terminal from a pop-up menu.

And another very useful everyday option...

Find Files (under Gnome "Search for files" provides like functionality though with fewer options):



Back to the panel....

Web Browser



Click on this panel icon and the Mozilla web browser will automatically start up. The Mozilla software package includes a web browser and other client software for reading mail, participating in newsgroups, and creating web pages.

E-mail



Evolution is a program that allows you to get your e-mail, manage your messages, and send messages. Evolution is the preferred application for sending and receiving e-mail messages in Red Hat Linux. Of course, your WFO or RFC will be using some other product for e-mail.

Writer



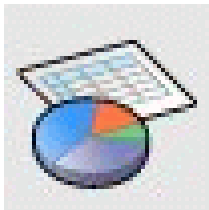
Click on this panel icon and OpenOffice Writer will automatically start. Writer is a word processing application that can work with documents using several different formats. It has a full set of features including, templates, fonts, file navigation, images and effects, besides doing a table of contents.

Impress



Click on this panel icon and OpenOffice Impress will automatically start. OpenOffice Impress is a presentation application that has several types of slide effects. You can create and save presentations in Microsoft Powerpoint, StarDraw, and StarImpress formats.

Calc



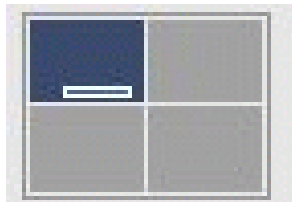
Click on the above panel icon and OpenOffice Calc will automatically start. This is a spreadsheet application that will incorporate data from Microsoft Excel, StarOffice, Dbase, and several other formats. You can create charts, set up database ranges, and use data arranged from different points of view.

Print Manager



Click on thi panel icon and the **print manager** will automatically start. This application can be used to add or remove printers and printer queues, There are no graphical tools to start printing. You can start a printing from a command in a terminal window.

Workspace Switcher



This is a feature that provides four virtual workspaces (or desktops). The idea of virtual desktops (not virtual consoles as examined earlier in this chapter) is to give you more space to run applications than will fit on one physical screen. Changing from one desktop to another is very simple. Just **click** on the desktop you want (in KDE they are numbered 1, 2, 3, or 4). The current desktop will disappear and you will move to the full screen for the one you selected. Of course you can still have multiple terminal windows within any virtual desktop. Here are some things you can do with a workspace switcher:

1. Choose the current workspace. Simply click in the virtual workspace and it will become the current workspace.
2. Click any window, represented by the small rectangle in the workspace, and drag-and-drop it to another workspace.
3. Right-click the switcher, select preferences and create up to 32 workspaces.

Window List



The window list area on the panel shows the tasks that are currently running on the desktop. So in this example, the trash application is currently running and there is a terminal window open for student1 on the current host, ntc234.

Clicking a task can either make the task minimized or maximized.

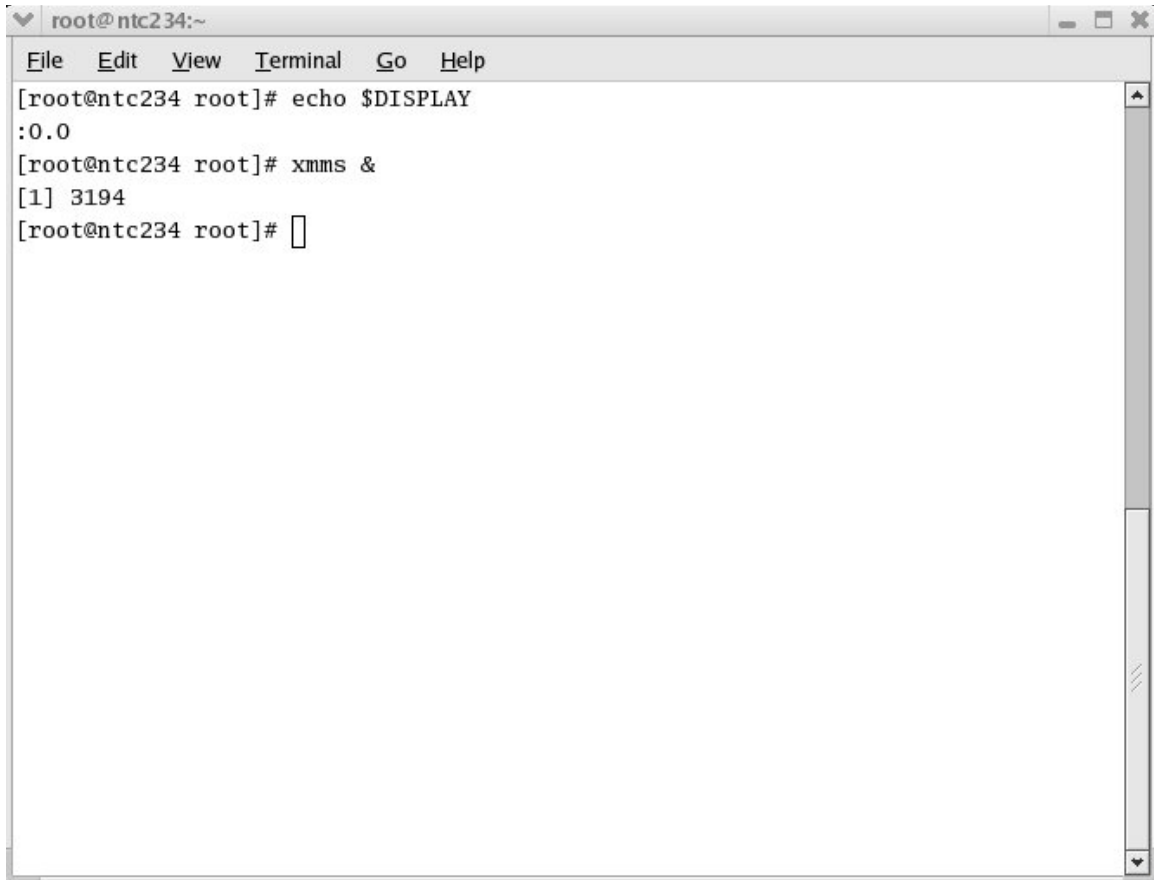
Red Hat Network Alert Notification



The round icon with the exclamation point is ready to help you check for critical updates to Red Hat Linux, for a price! After agreeing to the Terms of Service, the service will check for updates that might be available. If none are available, the red icon turns blue.

The Terminal Window

Access to the shell from either the Gnome or KDE desktop's is through a terminal window. A terminal window can be made available by right-clicking on open desktop space and selecting "Open Terminal" from the presented menu. The details of what is being done in this window is irrelevant for this course; the important thing to note is that by opening a terminal window you will be presented with a shell prompt from which to enter shell commands.



```
root@ntc234:~  
File Edit View Terminal Go Help  
[root@ntc234 root]# echo $DISPLAY  
:0.0  
[root@ntc234 root]# xmms &  
[1] 3194  
[root@ntc234 root]#
```

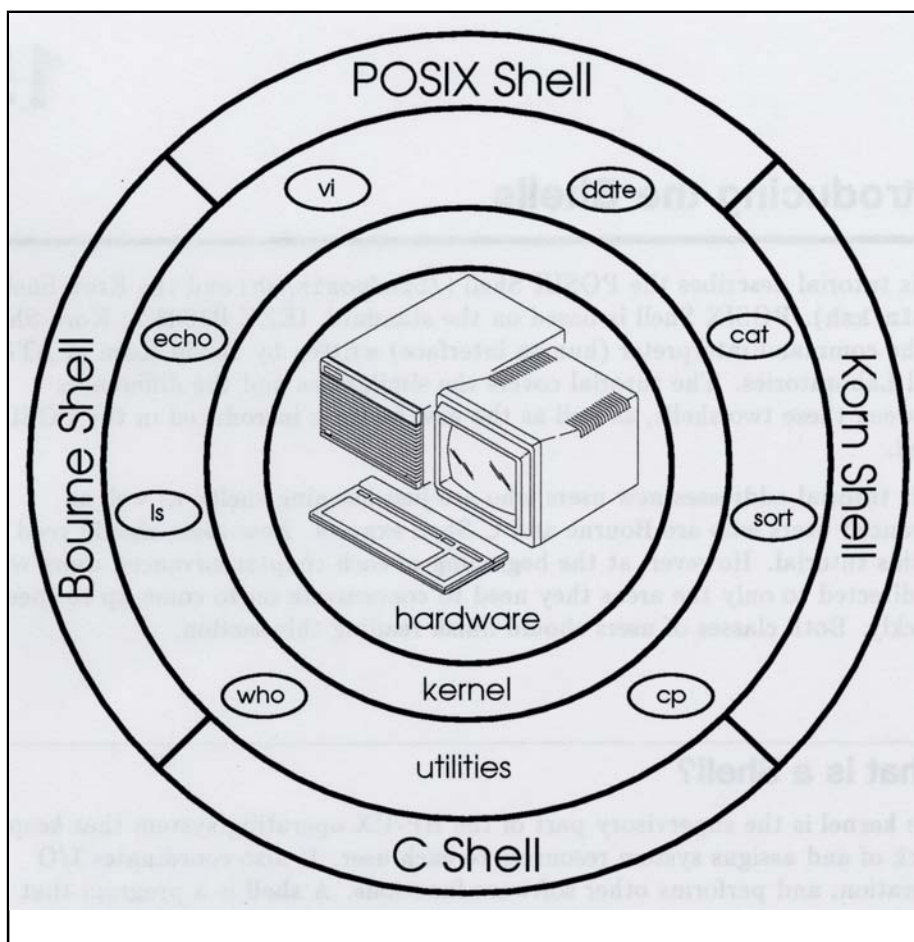


(To get more information on Linux desktop environments it is suggested at this time to do an internet search on the terms "gnome", and "kde" ; or get on a local system and explore the desktop screens, menus, and icons by yourself.)

USERS

As suspected, before being allowed access to either the desktop or a text shell session, you need to prove you are an authorized **user**, by supplying a username and password at a prompt. The **user** is, loosely translated, the human being pressing the keys. Humans wanting access to LINUX must have a username and password. This mechanism allows humans access to the system. The username is assigned by the system administrator. There are two types of user accounts, regular user and superuser. The superuser account (username "root") has special access and privileges for configuring and administering the system. Be forewarned that these privileges can, if misused, damage the system, possibly irreparably. After entering the user name you will be prompted for the password. [Note: On most Linux systems the user name you logged in as appears as part of the prompt as a reminder to you; otherwise on any LINUX/UNIX system you can see who you are with the '*whoami*' command or see all users logged onto the system with the '*who*' command.]

The following chart diagrams the relationship between a user login shell, UNIX commands, the operating system kernel, and hardware interrupts.



BASIC SHELL FEATURES

WHAT SHELL IS USED AND HOW IS IT IS EXECUTED

When we say a user is a BASH shell user, we mean the command displaying their systems level prompt, and accepting their commands is **sh** executing in an **interactive mode**.

As we've said, in **interactive mode**, the shell is essential running in an **ENDLESS LOOP**. When you enter a command and press <RETURN>, the shell assumes you are finished entering a command(s). The shell then **PARSES** (*scans and interprets*) the **TOKENS** (input on the line separated by **WHITE SPACE** - <SPACE>, <TAB>, <RETURN>) passed on the line. It then tries to interpret what you entered. Before it passes what you typed to the operating system, it must be **INTERPRETED**. Command line syntax such as wildcards, metacharacters or filename expansion, and such, must be resolved before the command itself is actually executed. The shell, not the command, does these types of functions. Commands are written to take advantage of shell features like **filename expansion, metacharacters, redirection, pipelining, and variable replacement** (further discussion of these things will occur later in the course).

WHY LEARN A COMMAND LINE INTERPRETER?

One issue of importance is the fact that graphical user interfaces will not always be available to the user, and thus dealing directly with the shell may be a requirement. This is especially true when doing Systems Administration. Users who interact with the Linux operating system via a command line interpreter must also be able to understand and manipulate their environment before learning higher level skills such as reading, writing, and modifying shell programs, or **shell scripts**. Luckily, shell environment and programming concepts are transferable - if you learn one shell others will come easier.

BUT I'M NOT A PROGRAMMER!

As we've said, when you log onto the system you are already running a LINUX command called **sh** in an **interactive mode**. Like any procedural programming language, you are entering one command at a time in a serial fashion. The only real difference between this and running a string of commands serially in a **batch mode** is that it is easier in the batch mode. Making use of the batch mode and scripts the user can use **programming constructs** like **decision making** commands (**IF** the answer to my question is true, **THEN** do this, **ELSE** do that), and **looping mechanisms** (**WHILE** something is true **DO** this) to repeat commands. (more on that in a later module)

You should know that you can change your login shell (the shell given to you at initial login) without the intervention of the **Systems Administrator (SA)** using the ***chsh*** command. The SA can restrict which programs or “shells” you chose. This command is not universally available in the UNIX world; on some systems, you may have to ask the SA to change it for you.

SHELL COMMANDS

Commands are entered one at a time at a shell prompt – like:

```
$ ls fn*  
$ ls -l > listing  
$ ls -l | grep listing
```

These may look like a lot of gobbly-gook now, but you’ll learn not only how to read these but also how to enter them, along with many many more. The above command examples are not all that meaningful except to illustrate that any given command can be used in a number of ways. The commands above actually have multiple operations (tokens) being called for on each line. The question is what happens first ?

There is a specified order in which tokens are interpreted:

1. **I/O redirection**
2. **Control operator**
3. **Newline**
4. **Reserved word**
5. **Identifier**
6. **Word**
7. **Here document**

We will not go into the definitions of each of these at this point, and The ***man*** page and other reference materials listed at the end of the module and subsequent modules have more detailed explanations of these. As you become a more experienced user you will become familiar with the nuances of this concept.

One item worth noting, as often a new user will stumble upon this by accident is the concept of a **secondary prompt** (presented as the character `>`). If a token is incomplete, a secondary prompt (PS2) will appear. The example below is incomplete because one of the parsing rules being applied is the expectation of a closing quote. In English, we know that this single quote is an apostrophe used for denoting a contraction, but according to the rules of most shells this is forbidden. Your choice is to interrupt the command or provide the expected token **keyword**.

```
$ echo Don't
> # Secondary prompt (PS2)
```

We only note this of significance at this point to illustrate you may potentially enter a command without all the correct components and be left scratching your head at the `'>` prompt. To get out of such a pickle you simply need to complete the command at the `>` prompt by supplying whatever was missing on the original command line. If you are unsure then entering a CTRL C, will stop the process and return you to a `$` prompt. In the case above entering a closing single quote will complete the command and display *Dont*

MORE THAN ONE KIND OF COMMAND

There is more than one class of commands that the shell knows how to interpret and pass on to the operating system for execution. We will deal with each in detail later. The **precedence** for the POSIX shell is:

- | | |
|-----------------|-----------------------------|
| 1 reserved word | 3 built-in |
| 2 alias | 4 function |
| | 5 pathname resolution/other |

This is important to know for even a beginner. For instance, you can get yourself in trouble by **ALIASING** a “*real*” command program that resides on the disk, as the real command will never be executed and might lead to unexpected or erroneous results.

RESERVED WORDS

Reserved words (Keywords) hold special meaning to the shell and are used in various programming constructs common to most programming languages.

ALIASES

Bash, C, Korn, and POSIX allows the System, Superuser, or the user to define commands by new names.

To display any aliases set in your environment enter:

```
$ alias
```

If you create an alias on the command line, it is only good for this login session. When you log back in, it will be gone. If you want it set each time you log in, you can put it in your personal startup file - **\$HOME/.bashrc**, or the SA can put them in a system-wide file.

There are two other kinds of aliases - **preset** and **tracked**. Preset are system-dependent and users cannot change them. They are created when the software is installed. Tracked aliases essentially are resolved pathnames for commonly used commands and are also created by the shell.

It is a bad practice to use personal or system-wide aliases inside shell scripts. You might not even be able to pass your programs to other users on the same machine if they are defined outside the shell script. It also makes it very difficult to read and understand the program since there are no **man** pages defining the aliases.

The first word of each command is replaced by the text of an alias, if an alias for this word has been defined. An alias name consists of any number of characters excluding metacharacters, quoting characters, file expansion characters, parameter and command substitution characters, and =. The replacement string can contain any valid shell script, including the metacharacters mentioned above. Aliases can be used to redefine commands, but cannot be used to redefine keywords. Aliases are frequently used as shorthand for full path names.

We'll learn how to define your own aliases in a later module, for now you can see the aliases already defined for you with the *alias* command:

```
$ alias  
history='fc -l'  
stop='kill -STOP'  
suspend='kill -STOP $$'  
ll='ls -l'  
$
```

BUILT-IN COMMANDS

Built-in commands are called directly from inside shell code. Commands like *pwd* usually are part of the operating system available to other shells and procedural programming languages. They reside on the disk as separate executable binaries from the shell. They are usually commands that are used frequently. Instead of making a call to execute another program outside the shell, it is more efficient to call them from inside, or even emulate them internally. Sometimes their behavior is slightly different than the operating system commands of the same name. One way of being sure to get the operating system version of *pwd* is to enter the absolute (full) pathname to the command. Other built-ins like *read* only have meaning only when used inside a shell process and are not available to other shells.

FUNCTIONS

Bash, Bourne, KORN shell, and POSIX shell allow you to create your own commands. They can be stored in the same shell script file or called by **EXEC**'ing a file containing functions inside the same process space (`.function_file`). This is similar to how procedural languages share reusable code in **library archives**. The **function** must be defined before used. Use entails only invoking the name as you would any other command. Some SAs provide libraries of locally-written functions for system-wide use. The advantage of functions over aliases is that they can contain very complicated code, pass run-time arguments more easily, and may be used over again in one or more programs. They are stored on disk, instead of only in memory so they are not as volatile. Rather than re-enter lines of code everywhere, you just call what appears to be a single word command with perhaps some command line arguments.

PATHNAME RESOLUTION/OTHER

When you type in a command the shell searches all of the above classes of commands first. If not found inside the shell, it searches **LEFT to RIGHT** through the directories in your **PATH** variable for an **executable program file** by the name given. If not found, you will get an error message like "**sh: lx not found**". The "**lx**" command may still be on the system in a directory not included in your **PATH**, or may not be executable.

SECURITY NOTE: If you always want your **current** directory searched, place the current directory (`.`) at the end of the **PATH**. Never do this with the **superuser** account. The danger with this practice is that someone may plant a **Trojan horse** in the current directory with the same name as a common command. Placing it at the tail end of the **PATH** ensures that the real command will be found first.

Understanding Processes

Another term we need to define is **process**. A process is the execution of a command by Linux. Whenever you open a Linux shell and/or run a command, a script, or a program, a process is started. Processes can also be executed by the operating system itself. The process structure is hierarchical. It contains parents, children, and even a root. A parent can fork (or spawn) a child process. That child can, in turn, fork other processes.

The kernel manages the processes, switching back and forth among them according to their needs and priorities. As we said, a process can create other processes; the creator is the **parent process** and the processes created are **child processes**. Certain system processes called **daemons** reside in the system more or less permanently and perform ongoing tasks such as handling mail, scheduling tasks that should be performed at regular intervals, and transferring files from the print queue to printers.

By creating multiple processes, you can run several programs at once. For example, suppose you want to execute a program that takes a long time to complete. You can run the program at a low priority in the background and do something else at the terminal while it is running. Some shells provide a facility called job control that lets you switch back and forth among processes.

A process is the execution of a command by the Linux system. The shell that starts when you log in is a command, or a process, like any other. When you run a Linux utility or command, a process is initiated.

JOB CONTROL

Job control is a facility, that enables you to control processes from your terminal, using “control signals”:

Commands that allow the user to control the state of a process.

^C (Control-C) Interrupts or aborts the command

^D (Control-D) Quit shell or program

Commands that control output to the terminal.

^S (Control-S) Suspend output to the monitor

^Q (Control-Q) Resume output to the monitor

These job controls have many uses, but are most commonly used to interrupt a run-away script/program or run-away output. As a new Linux user you may enter an erroneous command that requires such interruption.

PROCESS ID

The first thing the operating system does to begin execution is to create a single process, called “init”. The init process holds the same tree position as the root directory in the file structure. The init process is the ancestor to all processes that each user works with. It forks a process for each terminal or desktop session. Whenever a user logs in, Linux copies the shell program from system disk into memory. When it is in memory, the shell begins executing, and it becomes a process that lasts until you log out.

When you give the shell a command, it (referred to as the parent) usually “forks” (or spawns) a child process to execute the command. While the child process is executing the command, the parent process sleeps. While a process is sleeping, it does not use any computer time; it remains inactive, waiting to wake up. When the child process finishes executing the command, it dies. The parent process (which is running the shell) wakes up and prompts you for another command.

Linux assigns every process a unique number, known as a process identifier (PID). The PID can also be shown with the process status (**ps**) command. We said that init is the first/initial process started by the system and thus is the parent to all other processes. And so you may be able to guess it's Process ID – 1 .

As long as a process is in existence, it keeps the same PID number. During one session, the same process is always executing your login shell. When you fork a new process (execute a command, etc) the new (child) process has a different PID number from its parent process (PPID). When you return to the login shell, you will find it is still being executed by the same process and has the same PID number as when you logged in.

Displaying the Processes

The `ps` command will display system processes. You can use the `ps` command to obtain the process id number (**PID**). Because processes rapidly progress in their execution, this report is only a snapshot real-time view of what was happening when you entered the `ps` command.

The command `ps` will show your processes, `ps -f` will provide a full/verbose display, and `ps -ef` will show all processes active on the system.

```
[student1@ntc237 student1]# ps -f
  UID     PID   PPID  C  STIME  TTY  TIME      CMD
student1 28325   663   0 17:30:20 pts/1 0:00      bash
student1 28326 28325   0 17:30:21 pts/1 0:00      ps -f
[student1@ntc237 student1]#
```

- UID** refers to the user identifier
- PID** refers to the process identifier
- PPID** the process identifier of the parent process (the process that spawned this process)
- C** shows processor utilization for scheduling (*i.e.*, CPU time)
- STIME** the starting time of the process
- TTY** the controlling terminal for the process
- TIME** the cumulative execution time for the process
- CMD** the command name

(It should also be noted that In a Red Hat Linux Gnome session you can get a GUI display of the same information via **gnome-system-monitor** (issued from the command line) or **Redhat | System Tools | System Monitor** – then selecting the **Process Listing** tab).

Looking at the display again you can trace the lineage of a forked process – notice that the PPID of the second process (`ps -f`) in the box below is the same as the PID of the first one (`bash`). This indicates that the “`ps -f`” was forked/spawned by the first – student1 started a `ps` command from a bash shell.

```
[student1@ntc237 student1]# ps -f
  UID     PID   PPID  C  STIME  TTY  TIME      CMD
student1 28325   663   0 17:30:20 pts/1 0:00      bash
student1 28326 28325   0 17:30:21 pts/1 0:00      ps -f
[student1@ntc237 student1]#
```

There are several flags used with the `ps` command, giving different results:

- f Lists UID, PID, PPID, C, STIME, TTY, TIME, etc... (`man ps`)
- e Shows status of all processes
- l Long listing
- a Lists all processes associated with a terminal port
- g Lists processes for a specified group
- u Lists processes for a specified user

Some times one of your processes may get stuck and you'll need to kill it. Often the job control mechanisms noted at the beginning of this section may do the trick. However, if the stuck process is not in control of your terminal, you can use the `ps` command to get its number and the `kill` command to remove it. (If you cannot regain control of your session you may have to contact the System Administrator and have them kill the process.)

```
$ kill {PID} (eg. $ kill 28325)
```

Signals and codes for the kill command:

The *kill* command actually works by sending a signal to the system to stop operations with the indicated process. There are various ways (signals) that can be sent to the system. These are called by their number (0 thru 30+). The CTRL commands actually send these signals to the system. We will discuss the signals 3, 9, and 15.

- signal 3 Terminal quit (with a memory dump)

- signal 9 Process kill. Care should be taken when using this signal. Issuing kill -9 will kill the process WITHOUT the housekeeping of signal 15.

- signal 15 This is what would be known as an orderly shutdown or Software termination. When this is issued the system will do some "house-keeping", closing files and ending processes.

`$ kill -{signal} {PID}` (eg. `$ kill -9 28325`)

If you must use the **kill** command, the default is 15. Signal 9 should be a last resort, files can be left open or damaged.

To find out more about signals and codes, you can do a **man page** of "signal" or the command **kill -l** will produce a list of signals.

MANIPULATING YOUR HISTORY FILE

Use mechanisms to repeat and edit previous commands stored

During the course of your Linux work you will often have a need to repeat a command you entered earlier. You may want to repeat it verbatim or repeat it with some modification. A log of everything you type on the command line before pressing the <RETURN> key, including your typos, is kept in a history log file. By default this is file **\$HOME/.bash_history**, in which up to a 1000 commands will be stored.

Later we'll introduce the *'cat'* command to look at the contents of file and thus you can view your commands with *'cat .bash_history'*). However it may be more convenient to read the file with the **history** command. When **history** is run, it assigns a unique sequential number to each command for optional use in retrieval.

Using the history command

```
$ history
276  more /etc/passwd
277  echo $HISTSIZE
278  echo $PATH
279  pwd
280  cd /home
281  history 1
```

The most convenient method of recalling, and optionally editing, previous commands directly on the command line – is simply by pressing the up-arrow key. This will present to your previously entered commands from the most recent backwards with each press of the key. Once the desired command is displayed it can simply be reentered as is, or modified and then resubmitted from the displayed command line.

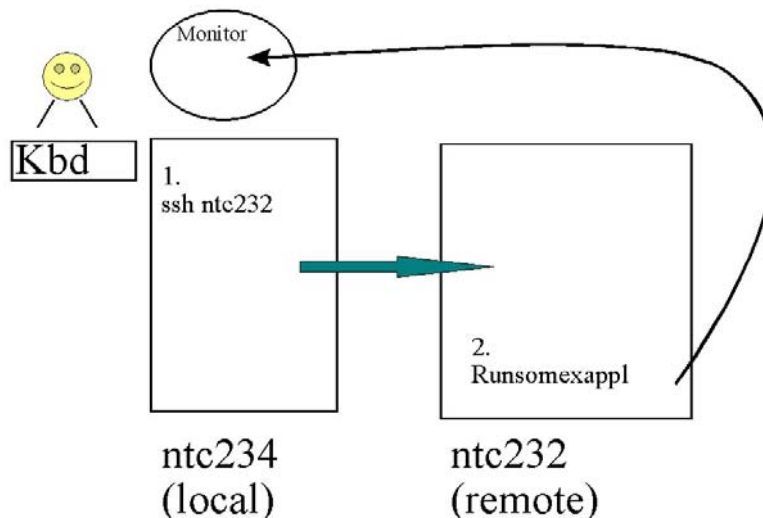
LOGGING OUT OF AN ASCII SESSION

In a desktop session you will typically logout by clicking an **EXIT** or **logout/logoff** icon. In an ASCII terminal session (eg. PuTTY or ssh), you simply type in the command **exit** back in your original login **sh** process or close the connection altogether. This, or **<CTRL> d** on some systems, is an **End of File** signal to the interactive shell process started when you logged in. It tells the shell that there are no further instructions. It ends itself and as you have no other process running your terminal session is disconnected.

REMOTE ACCESS

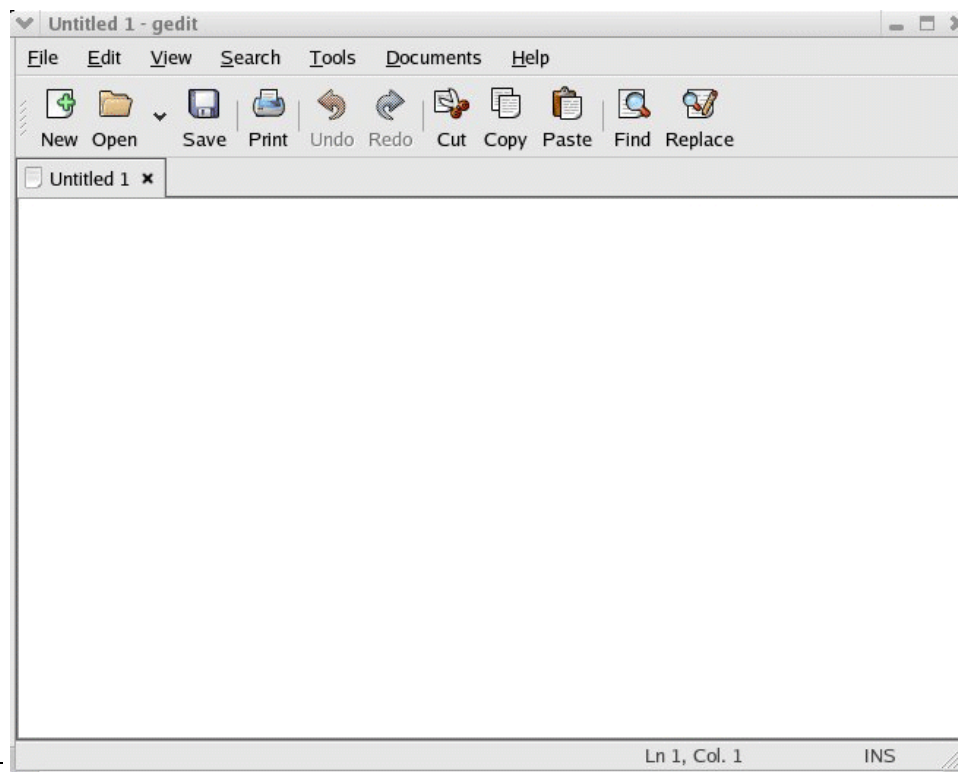
In a networked environment you can run an application or even a shell session on a remote computer by starting it there, but having it displayed on your local display. You can have access to files, printers, backup devices, and all other resources on the remote computer as though you were on that computer.

SSH can provide a secure mechanism for logging into a remote system, it also provides a way to run graphical applications securely. After you login using ssh, you can use that secure connection to forward the graphical application back to your display. You will use a form of ssh (PuTTY) to do the practical portions of this course.



In this example the application gedit is running on the remote computer ntc232 but is being displayed on the local computer ntc234.

```
[root@ntc234 root] # ssh -l student1 ntc232
student1 @ntc232's password:
[student1 @ntc232 student1]$ gedit
```



The remote session however does not have to be a GUI presentation. It may be text only, as will be the case with the student server on which you will do your practical exercises.

Your network identity can be displayed with the *hostname* command:

```
# hostname
```

Note: Depending on the Linux implementation and the configuration of the system, *hostname* will report the 'official' host/network/system-name, the system's ip-address or both.

Other settings of interest may be your gateway address and your DHCP (Dynamic Host Configuration protocol) settings (neither of which are presented here). Your network administrator is the source of information for your specific settings.



(The details of how networking works is beyond the scope of this course. To get more information it is suggested to do an internet search on the terms "TCP/IP", and/or complete the NWSTC Distance Learning courses on that topic (see <http://www.nwstc.noaa.gov/d.train/selfstudy.html>).

WHERE TO GET HELP: ON-LINE DOCUMENTATION OVERVIEW

What is a **man** page? The man pages are the official documentation for the UNIX operating system. All commands, arguments to commands, file formats, subroutines, library functions, utilities, tools, etc. are documented in the man pages.

Man pages can be searched via a title or keyword. To read a man page, type **man *command***. The man command will display on-line documentation about the command and its usage. Each man page contains structured documentation about the command it describes. [Note: Exit the *man* command with a “q”]

Accessing a Man Page

Man pages can be searched via a title “man man” or keyword “man -k manual”. The man page for man will describe how to use the man command.

```
$ man man
```

```
man(1)
```

```
man(1)
```

NAME

man – format and display the on-line manual pages

SYNOPSIS

```
man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]  
[-M pathlist] [-P pager] [-S section list] [section] name ...
```

DESCRIPTION

man formats and displays the on-line manual pages. If you specify section, **man** only looks in that section of the manual. name is normally the name of the manual page, which is typically the name of a command, function, or file. However, if name contains a slash (/) then **man** interprets it as a file specification, so that you can do **man ./foo.5** or even **man /cd/foo/bar.1.gz**

See below for a description of where man looks for the manual page Files.

OPTIONS

-C, config_file

Specify the configuration file to use; the default is **/etc/man.config**. (See **man.config(5)**.)

- More - (11%)

The message - **More - (11%)** Means you have viewed 11% of the file, and 89% remains. Some systems will just display - More -.

...

SEE ALSO

apropos(1), whatis(1), less(1) gruff(1), man.conf(5)

BUGS

The `-t` option only works if a troff-like program is installed

...

\$

Sections include: NAME (used in `man -k keyword` search) and DESCRIPTION – a summarization of the purpose of the command. The SYNOPSIS section tells how the command is to be used syntactically from the command line. OPTIONS explains the use of each option/parameter/argument. SEE ALSO is important as it will tell you of related files and commands. Depending on the man page you may also see EXAMPLES, WARNINGS, BUGS, and other pertinent information.

A few conventions are used:

Computer font strings are literals, and are to be typed exactly as they appear in the manual (except for parameters in the SYNOPSIS section of entries in Sections 2 and 3)



Try it now or try it later – you can try the `man` command by following the instructions for lab 1 (at the end of the module) on the NWSTC UU133 student server.

Another example:

\$ man date

DATE(1) User Commands DATE(1)

NAME

Date – print or set the system date and time

SYNOPSIS

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]

DESCRIPTION

Display the current time in the given FORMAT, or set the system date.

-d, --date=STRING

Display time described by STRING, not 'now'

-f, --file=DATEFILE

Like - **date** once for each line of DATEFILE

...

SEE ALSO

The full documentation for **date** is maintained as a Textinfo manual. If the **Info** and **date** programs are properly installed at your site, the command

Info coreutils date

Should give you access to the complete manual

...
\$

Note the SEE ALSO section in this example . This offers you other man pages to try for your information needs. In this case more information can be obtained through the use of the 'info' command

Another thing to note is the use of (#) by a command - eg. `date(1)` . The # indicates the “section” of the man pages which contains the information. For example information about commands is in section 1, while information about files is in section 5.

Red Hat Man page documentation sections include:

- 0 Everything
- 1 Commands
- 2 System Calls
- 3 Library Functions
- 4 Special Files
- 5 File Formats and Conversions
- 6 Games for Linux
- 7 Macro Packages and Conventions
- 8 System Management Commands
- 9 Kernel Routines

Using the proper section is important. For example there is a command named *'passwd'* and a file named *'passwd'* . The default for the man command is to look in section 1. Thus if you enter :

```
$ man passwd
```

You will get the man page information for the `passwd` command but no information about the `passwd` file. To get information about the file you would have to specify:

```
$ man 5 passwd
```

Keyword Search

If you don't know what command or file to locate the information you need, you can do a keyword search off all the man pages.

In the following example, the command returns all the man page names that contain the string "lp" (but note that this would also include words like "help and HelpTag").

\$ man -k lp

```
bos apropos (apropos) - Shows each help entry containing a specified string
bos help (help)      - Shows syntax of specified bos commands or lists functional
descriptions of all bos commands
cm apropos (apropos) - Shows each help entry containing a specified string
cm help (help)       - Shows syntax of specified cm commands or lists functional
descriptions of all cm commands
dfstrace apropos (apropos) - Shows each help entry containing a specified string
dfstrace help (help) - Shows syntax of specified dfstrace commands or lists
functional descriptions of all dfstrace commands
dtmailpr(1)          - electronic mail message print filter
fts apropos (apropos) - Shows each help entry containing a specified string
fts help (help)      - Shows syntax of specified fts commands or lists functional
descriptions of all fts commands
help (1m)            - Displays help information about dtscp commands.
help1m               help Displays list commands the options specified command
accept, reject(1M)  - allow/prevent LP printer queuing requests
dthelp_ctag1(1)     - first pass for formal SGML parse of HelpTag source
...
```



Your Turn – practical exercises

It is time to log onto the NWSTC student server (204.227.127.133) and enter your first Linux commands (using an ssh client such as PuTTY). If you need the instructions they can be found at the following link:

<http://webdev.nwstc.noaa.gov/IT/linuxessentials/linuxinstr.html>

It is rare that there is one **correct** solution to performing a task in a Linux/UNIX environment, only some that are faster, shorter, less process-intensive, safer, or more portable. Also, some of these exercises were written to fail, so that you would know what happens when bad things happen to nice people. Most of the errors shown are everyday problems for beginners. In any case you are encouraged to **EXPERIMENT** in this course and try various commands, so that you **SUCCEED** in the field. In the process you will learn problem-determination procedures. As an end-user, you do not have the access permissions to cause fatal system problems, so use this opportunity to experiment. Most importantly – DO NOT enter the commands robotically without trying to understand them in the process. Your success at further Linux training and actual work in the field is wholly dependant upon grasping the subject matter in this course.

EXERCISE 1 - SHELL ENVIRONMENT

Purpose: This exercise is designed to familiarize you with basic shell concepts

1. After logging in try your first commands (we'll only list the command you should enter (in bold) after the \$ sign, neither the full prompt or the result of the command will be in this text; press 'Enter' to execute each command):

```
$ pwd
```

```
$ date
```

```
$ whoami
```

```
$ who
```

```
$ id
```

```
$ hostname
```

2. Recall that we stated earlier that on occasion you may see a secondary prompt if the shell determines that your command entry is incomplete. Let's see an example:

```
$ echo Don't  
>
```

If you enter **<CTRL>c**, a software interrupt is sent to the **sh** process to abort the operation. If you understood that the shell was looking for a closing single quote you could have completed the command with a second single quote (try it if you like).

3. Let's look at command history

b. **\$ *history***

The above command will show up to your previous 1000 commands.

c. Now, the simplest way to repeat a command is with the up arrow key – try it now

d. You can modify previously entered commands and reissue them. Simply recall a previous command by arrowing through your history; Then make the desired change; then press enter to re-execute the modified command. Try it – page back to a 'whoami' command (or submit one if there is not one in your history; then when the prior-entered 'whoami' command displayed – backspace over the 'ami', and execute just the 'who'.

EXERCISE 2

- 1) Use find to demonstrate the use of ^S, ^Q, and ^D.
 - A. Start a long find command so you can suspend terminal output with ^S. Once output starts, type ^S to suspend terminal output. Note as you are Using a regular user account (not root/superuser) you will see many errors Ignore them, as the purpose is simply to interrupt the output.

```
$ find / -exec ls -l {} \;  
^S
```

- B. Restart terminal output by typing ^Q. Once output starts, interrupt the command with ^C, then exit the session with ^D.\

```
^Q  
^C  
$ ^D
```

2. Use 'ps' to list your processes. Use the simple format below, and then try some of the other switches. If you need help : 'man ps'

```
$ ps
```

EXERCISE 3 – man

Purpose: This exercise is teach you how to use the ‘man’ pages as a help resource

Call up “man” pages for the file and command “passwd”.

```
$ man passwd
```

```
$ man 5 passwd
```

Use the ‘*man*’ command to call up the manual pages of any Linux commands – enter man followed by any Linux command you have used thus far (pwd, who, date, ...)

```
$ man _____
```

Pay close attention to all parts of the returned information as this will come to your rescue time and time again.

After looking at a couple of different man pages for different commands, try a key work search via ‘man -k’ followed by a space and then whatever keyword you’d like to find (users, time, host, ...)

```
$ man -k _____
```

The man command can be used to get information on how to run a command, or can give you information about the layout of configuration files. Try this ...

```
$ man hosts
```

Now see if you can use the man pages to determine the meaning of all the columns of output from the *who* command.

```
$ man who
```

```
$ info who
```

And now that you know how to get information on Linux commands we’ll suggest you use the ‘man’ command to preview some of the commands that will be presented in the next module: pwd, df, cd, ls, file, cat, more, tail, grep and find. Don’t try to understand their syntax so much as simply identifying how they can be used.

The man pages are not the only source of help information. Many (but not all) Linux and UNIX systems have additional information available via a command call 'info' (try 'man info'); also there are in many cases help pages contained within a GUI environment such as Gnome. But by far one of the most helpful things actually is the internet.



Try an internet search on 'Red Hat Enterprise bash' and see the results

End LAB

This is the end of this module. At this time you should proceed to module 2.