

# Overview of Linux

This book belongs to

Name :

Batch :

**SQLSTAR**  
KNOWLEDGE TO KEEP YOU IN BUSINESS™

**Copyright © 2009**

First Edition

**SQL STAR INTERNATIONAL LIMITED**

**SQL STAR HOUSE, No. 8-2-293/174/A 25, Road No. 14,  
Banjara Hills, Hyderabad - 500 033.**

**Tel. No. 91- 40-23101600(30 lines)**

**Fax No. 23101663**

**Toll Free No: 1800 425 2944**

**Email: [knowledge.services@sqlstar.com](mailto:knowledge.services@sqlstar.com)**

No part of this publication may be reproduced (incl. photocopying) in any way, without prior agreement and written permission of SQL Star International Ltd., Hyderabad. SQL Star International Ltd., Hyderabad assumes no responsibility for its use, nor for any infringements of patents or other rights of third parties which could result.

## CONTENTS

Linux Overview .....	01-18
The bash Shell .....	19-23
Standard I/O and Pipes .....	24-26
User, Groups and Permissions .....	27-35
vi Editor Basics and Printing .....	36-48

**SQLSTAR**  
KNOWLEDGE TO KEEP YOU IN BUSINESS™

# Chapter 1

## Linux Overview

### Objectives:

After completing this chapter, you will be able to:

- ◆ Describe the history of Unix
- ◆ List the principles of Unix on which the OS runs
- ◆ Define General Public License (GNU)
- ◆ Explain the reasons of preferring Linux over other operating systems
- ◆ List the features of Red Hat enterprise Linux
- ◆ Recommend hardware specifications required to install Linux
- ◆ Use some basic commands of Linux
- ◆ Describe the Linux file system hierarchy

## 1. UNIX History

Linux is a freely distributed, multiuser, multitasking operating system that behaves like UNIX.

UNIX is one of the most popular operating systems in the world. It was developed, in **1969**, by Ken Thompson, Dennis Ritchie and others at **AT&T**. UNIX was one of the first *multiuser* and *multitasking* operating systems. (A multiuser operating system (OS) handles multiple users and multiple peripheral devices at a time. And, a multitasking OS enables a user to carry on multiple tasks at a time, like copying, editing, and printing.

Earlier, UNIX was released in various universities for academic research. Later, it was released for commercial purpose where each vendor came with an improved version of UNIX. Over a period, many versions of UNIX were introduced in the market, with each version more polished in terms of superiority, reliability, and speed.

However, the commercial UNIX has certain limitations. Firstly, it is a big OS, which means it requires huge amount of memory. Secondly, it is expensive, especially for the PC version. This is where Linux comes in - Linux is small, fast, and inexpensive.

## 2. UNIX Principles

### 2.1. The Kernel

In Unix (often spelt UNIX), the basic software controlling the hardware is known as the Kernel. The Kernel does all the difficult and nasty things like managing all the running processes (a name for a running program), the memory, the network connections, disks, tapes and virtually any bit of hardware on the computer.

When you write a program in a computer language like Fortran or C, a 'runtime library' deals with the kernel allowing you to use nicer commands to get memory and write files in your language, like WRITE in Fortran or printf in C.

### 2.2. Shells

A shell is a program, which allows you to run other programs. Although there are graphical shells for Unix, such as Gnome and KDE, they are not installed on the computing cluster. Most interaction is done through a traditional shell, which allows you to type commands into it to run other programs. A "**command line interface**" is harder to learn than a graphical shell, but much more efficient once you've got the hang of it.

The standard astronomy shell is known as **tcsh** (Extended C- Shell, as it bears some resemblance to the C language), but other popular ones include: the standard **sh** (Also known as the Bourne Shell) and **bash** (The Bourne Again Shell).

### 2.3. Files

In Unix, like in many environments, data are stored in things called files. A file contains data made up of a particular number of bytes. Each byte can hold a number between 0 and 255. Characters (like letters, numbers and symbols) are encoded as a particular number using ASCII encoding, although things are a bit more complex than this on modern systems. The letter 'A', for instance is stored as the number 65, 'B' is 66, and the lowercase 'a' is 97.

Many files in Unix are stored as text; often data in a program are translated into text for saving and loading. Other files, such as those your programs could write, or programs themselves, are encoded such that you can load them back into memory.

### 2.4. Directories

Files are stored on disk (which may be a CD-ROM or a disk in memory, like the /tmp file system). Files are organised in a "File System Hierarchy" or "Directory Tree" (it looks like a tree), which sounds a bit complex. A directory is a set of files with a name, which can also contain other directories. All the directories can be traced back to the root directory, '/'.

In Unix, the layout of directories is largely historical and conventional.

## 2.5. Permissions

Each file and each directory has an owner saying who has control of the file, and a group which can be several users, who also can have access to the file. The important files on the computer are owned by the user ``**root**'' who is your local friendly system administrator. Certain programs and files are restricted for use by root.

Each file and directory has permissions saying whether the owner, group or any other users are allowed to read from, write to, or run the file (r, w and x permissions). Root can always do things to a file. ``**ls -l**'' lets you examine the ownership and permissions of a file, and **chmod** lets you modify the permissions if you own it.

## 2.6. Processes

When you run a program, you load it into memory and it starts running as a "process". Processes compete for the CPU time on the computer, according to their priority or "iciness". Each process gets its own identification number (PID), which are recycled eventually. Processes are identified by this number in the "top" and "ps" process examination tools, and can be used to kill or stop the process with the "kill" command.

When you start a process, it starts in the "foreground", letting you type input into it and letting you see what its output is. Processes may also be started in the background, letting you carry on using the terminal (where you are typing into the shell).

## 2.7. The X Window System

X is a system for displaying graphical applications on Unix (and other) systems. An X server runs on your computer, and receives requests to open windows, draw windows, accept input, and so on from a client application (which is the program you are running).

X is clever, because it doesn't matter where the client application is running. You can run a program in Australia and have it appear on your screen, as if it were running on your local computer (with a bit of slowness).

## 3. GPL - GNU General Public License

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the **GNU General Public License (GPL)** is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This GPL applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs too.

When we speak of free software, we are referring to freedom, not price. Our GPLs are designed to make sure that:

- You have the freedom to distribute copies of free software (and charge for this service, if you wish);
- You receive source code;
- You can change the software or use pieces (modules) of it in new free programs.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you, if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And, you must show them these terms, so they know their rights.

**We protect your rights with two steps:**

- (1) Copyright the software;
- (2) Offer you this license which gives you legal permission to copy, distribute and/ or modify the software.

Also, for each author's protection and ours, we wish to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we would want its recipients to know that what they have is not the original; so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we make it clear that any patent must be licensed for everyone's free use or not licensed at all.

## 4. Why Linux?

### 4.1. Linux Origins

Linus Torvalds developed Linux OS during his graduation at the University of Helsinki in Finland. He wrote a small PC-based implementation of UNIX and named it Linux. Linus made the source code freely available on the Internet and encouraged other programmers to develop it further. Linux continues to be developed by a worldwide team.

Linux is written in C language and is portable. It provides a graphical interface that makes the Linux system user friendly.

## 4.2. Features of Linux

With several operating systems available in the market, why should one opt for the Linux OS? The reasons are obvious. The Linux OS contains several features that distinguish it from other operating systems.

**The features of Linux OS are as follows:**

- a) **Multitasking:** Like UNIX, Linux systems support multitasking where multiple tasks run in the background.
- b) **Hardware Support:** Linux supports nearly all hardware architectures and devices, with best support for legacy hardware.
- c) **Virtual Memory:** Linux uses a portion of the hard disk as virtual memory, which increases the efficiency of the system by keeping active processes in RAM and placing less frequently used or inactive portions of memory on the disk.
- d) **The X System:** The X system is a set of programs that provide graphical interface across different platforms.
- e) **Built-in Networking Support:** Linux has a built-in network support. It uses the standard TCP/IP protocol, Network File System (NFS), Network Information Service (NIS), and Session Message Block (SMB) to perform networking.
- f) **Shared Libraries:** Linux uses dynamically shared libraries extensively. Dynamically shared libraries use a common library section for many different applications, thereby reducing the size of each application.
- g) **Memory Efficiency:** To enable you to work with large memory requirements when only small amount of RAM is available, Linux supports *swap* space. The Linux kernel also supports *demand paging*, which means that only sections of a program that are necessary are read into RAM.

## 4.3. Advantages of Linux

There are several advantages to using a Linux operating system. They are as follows:

- a) **Free Software Foundation:** Free Software Foundation (FSF) developed GNU (Gnu's Not Unix) to provide royalty free software to programmers and developers. Linux includes many GNU utilities like various languages (C, C++, FORTRAN, Pascal, Ada, BASIC and Smalltalk), compilers, debuggers, text processors and print utilities. Most of the Linux compilers, tools, debuggers, assemblers, linkers, loaders and editors are provided by the FSF.
- b) **Upgradation** - Linux OS can easily be upgraded.
- c) **Forward Compatibility** - Linux supports forward compatibility. New hardware is introduced every year.



## 5. Red Hat Enterprise Linux

### 5.1. History

**Red Hat Linux** is one of the most popular Linux distributions, assembled by Red Hat.

It is one of the "middle-aged" Linux distributions; 1.0 was released in November 3, 1994. It is not as old as Slackware, but certainly older than many other distributions. It was the first Linux distribution to use RPM as its packaging format, and over time, has served as the starting point for several other distributions, such as the desktop-oriented Mandriva Linux (originally Red Hat Linux with KDE), Yellow Dog Linux (which started from Red Hat Linux with PowerPC support) and ASPLinux (Red Hat Linux with better non-Latin character support).

However, Red Hat Linux lacks several features due to possible copyright and patent problems. For example, MP3 support is disabled in both Rhythmbox and XMMS; instead, Red Hat recommends using Ogg Vorbis, which has no patents.

### 5.2. Features of Red Hat Linux

More or less, all Linux distributions are similar as they use the Linux kernel, the GNU free software, command-line programs, manual pages and documentation. However, Linux distributions differ in a number of features they exhibit. Some of the reasons to use Red Hat Linux are as follows:

- a) **Red Hat Linux is the most popular Linux distribution** - For several years, Red Hat Linux has been on top of the list.
- b) **Red Hat Package Manager** - Red Hat Linux uses a sophisticated software management tool called the Red Hat Package Manager or the RPM.
- c) **Graphical Administration Tools** - Red Hat Linux can be entirely managed from a graphical user interface (GUI).
- d) **Security** - Red Hat leads the distributors' list for providing the most up-to-date security features.
- e) **Documentation** - Red Hat provides a user and installation guide, in several formats available in electronic form, on its CD-ROMs. Also, it makes on-line help available throughout the Linux installation process through various help buttons.
- f) **Installation** - Installation in Red Hat Linux is flexible. You can select a graphical, text-based or automatic installation.

## 6. Recommended Hardware Specifications for Linux

A Linux system offers a Unix workstation environment that works even on old PCs. The hardware requirements are not very demanding, unless you require application development and extensive use of GUI. In this session, you will learn about minimum hardware requirements for Linux installation.

Before attempting to install Red Hat Linux, **you should collect information about your system**. This information will help prevent any surprises during the installation. You should have a basic understanding of the hardware installed on your computer, including the hard drive(s), memory, CD-ROM, network card and mouse.

### 6.1. Minimum Hardware Requirement

The hardware requirement for a Linux system is similar to that of a PC. Hardware compatibility is particularly important if you have an older system.

Red Hat Linux 7.1 is compatible with most hardware in systems that were built within last few years. However, hardware specifications change almost daily; so it is hard to guarantee that your hardware would be 100% compatible.

The minimum realistic requirements for Linux systems are as follows:

- a) A motherboard with 80386SX processor or better,
- b) 2MB of RAM or more,
- c) A floppy disk drive,
- d) A hard drive with 40 MB or more,
- e) A video card,
- f) A monitor.

Most systems fulfill these requirements, but are suitable only for character-based installation. If you wish to use a GUI, such as X Window system, you need powerful motherboard and more memory.

#### 6.1.1. Motherboard

One of the first hardware requirements for Linux systems is a motherboard, which must be an Intel 80386 or better. Although Linux runs on 80386 machines, for application development, this would not be sufficient as the compilers and linkers utilize a lot of CPU space. The same applies to Linux systems with graphical support. The X Window uses excessive CPU space. If you try to compile an application on an 80386 machine, it may compile the application programs, but the performance degrades in terms of speed.

Therefore, for a realistic system that should support application development, as well as X, you should at least have 80486DX or Pentium 200 MHz.

### 6.1.2. RAM

RAM requirements for a Linux system vary, depending upon the size of the Linux system you want to run. A character-based Linux system performs well with 2MB. Although, a higher RAM for character-based systems would give a better performance. For application development and X Windows, minimum requirement is 8MB; however, at the cost of performance.

To achieve a really good performance with X Windows and compilers for application development (because compilers consume a lot of memory), you should definitely have more than 16 MB of RAM. Being a multiuser operating system, Linux has increased RAM requirement.

To extend the amount of physical RAM on the system, Linux creates a swap partition, called the *swap space*. The swap space is used as an extension of the actual memory, where data can be exchanged with the physical RAM.

### 6.1.3. Hard Disks

Although Linux can run completely from a floppy disk, generally it is not run that way. It is designed primarily for hard disk use, and supports all the common hard disk controller systems. These include:

- The Integrated Drive Electronics (IDE),
- Extended Integrated Drive Electronics (EIDE),
- Enhanced Small Device Interface (ESDI),
- Small Computer System Interface (SCSI).

Linux is independent of the manufacturer and the type of hard disk. If DOS or Windows can handle the drive, so can Linux. However, the SCSI drives require special handling.

The amount of hard disk space required by Linux depends on the parts of the operating system that are installed. To load an entire Linux system, including development tools and X windows, you require at least 1250MB of hard disk space - just for the files. You require additional space for personal files and temporary storage. For this, you should double the space requirement. In addition, you should also leave room for swap space. It is advisable to assign 16MB or 32MB for swap space, depending on the amount of physical RAM on your system.

You can use more than one hard drive on your Linux system. However, you should place the bootable root Linux partition on the first drive. The number of drives depends on the hard disk controller systems and the Basic Input Output Systems (BIOS). IDE systems can handle two drives, EIDE systems can handle four drives and SCSI systems can handle 15 drives. SCSI is the most versatile of all.

**Note:**

BIOS – On PC compatible systems, the BIOS is used to perform all necessary functions to properly initialize the system's hardware when the power is first applied. The BIOS also controls the boot process, provides low-level input/output routines and allows the user to modify details of the system's hardware configuration.

#### 6.1.4. Mouse

As Linux supports X Windows, a mouse is necessary. Linux handles almost every type of mouse and supports both *bus* and *serial* mice. It does not use the mouse for character-based sessions.

Linux also supports other pointing devices, including joysticks and pen systems that are used for cursor movements.

Commonly supported mice are as follows:

1. Microsoft serial mouse,
2. Mouse Systems serial mouse,
3. Logitech Mouseman serial mouse.
4. Logitech serial mouse.

#### 6.1.5. CD-ROMs

Linux supports ISO-9660 file systems. CD-ROMs that are not written using ISO-9660 file systems cannot be mounted properly on Linux systems.

#### 6.1.6. CD-ROM Drive

As most CD-ROMs use either IDE or an SCSI interface, you need an SCSI controller card or an interface on the motherboard.

Any SCSI CD-ROM drive with a block size of 512 or 2048 bytes should work under Linux. This includes the vast majority of CD-ROM drives on the market.

#### 6.1.7. Modems

Linux supports most serial asynchronous modems, as well as some synchronous modems. It also supports ISDN modems.

Linux can use any modem that is used by DOS or Windows.

#### 6.1.8. Printers

Linux supports all printers connected to the parallel or serial port, except for some printers that are supported by Windows 95 only.

### 6.1.9. Terminals

Linux supports character-based terminals that are connected through a serial port or a multi port.

### 6.1.10. Network Cards

A Linux system supports networking. Its primary network protocol is TCP/IP - Transmission Control Protocol – Internet Protocol. Every system requires a Network Interface Card (NIC), also called network adapter, to support networking. One of them is *Ethernet*. TCP/IP usually uses Ethernet. Therefore, most of the networking systems in Linux are designed around Ethernet. The NICs from Novell, Western Digital, Hewlett-Packard and Intel work efficiently with Linux.

### 6.1.11. Video Cards

Linux can use almost any video card that works without special drivers under DOS and Windows. These include: CGA, EGA, VGA and Super VGA.

## 7. Some Simple Commands

Some of the commonly used commands are discussed below:

- **alias** - Create an alias.  
\$alias exmple='rm -r'  
\$exmple
- **cal** - Display a calendar.  
\$cal
- **cat** - Display the contents of a file.  
\$cat file1
- **chgrp** - Change group ownership.  
\$chgrp group1 file1
- **chmod** - Change access permissions.  
\$chmod 777 file1
- **chown** - Change file owner and group.  
\$chown user1:user1 file1
- **chroot** - Run a command with a different root directory.  
\$chroot /mnt/sysimage
- **crontab** - Schedule a command to run at a later time.  
\$crontab -e
- **pwd** - To identify the exact location of current directory.  
\$ pwd
- **ls** - To view the contents of the directory.

\$ ls

- **mkdir** - To create a new directory.

\$ mkdir dir1

- **cd** - To change the directory.

\$ cd dir1

- **cp** - To copy a file from one location to another.

\$ cp file1 dir1

- **mv** - To move a file from one location to another.

\$ mv file1 dir1

- **su** - To change the login directory.

\$ su user1

- **locate** - To find a file or a directory.

\$ locate file1

- **clear** - To clear the contents in the shell window.

\$ clear

## 8. Linux File System Hierarchy

Like UNIX, Linux chooses to have a single hierarchal directory structure. Everything starts from the root directory, represented by /, and then expands into sub-directories instead of having so-called 'drives'. In the Windows environment, one may put one's files almost anywhere: on C drive, D drive, E drive etc. Such a file system is called a hierarchical structure and is managed by the programs themselves (program directories), not by the operating system. On the other hand, Linux sorts directories descending from the root directory / according to their importance to the boot process.

To comply with the FSSTND, the following directories, or symbolic links to directories, are required in '/':

/	--	the root directory
/bin		Essential command binaries
/boot		Static files of the boot loader
/dev		Device files
/etc		Host-specific system configuration
/lib		Essential shared libraries and kernel modules
/mnt		Mount point for mounting a filesystem temporarily
/opt		Add-on application software packages
/sbin		Essential system binaries
/tmp		Temporary files
/usr		Secondary hierarchy
/var		Variable data



The first thing that most new users shifting from Windows will find confusing is navigating the Linux file system. The Linux file system does things a lot more differently than the Windows file system. This section explains those differences and takes you through the layout of the Linux file system.

For starters, there is only a single hierarchal directory structure. Everything starts from the root directory, represented by '/', and then expands into sub-directories. Where DOS/ Windows have various partitions and then directories under those partitions, Linux places all the partitions under the root directory by 'mounting' them under specific directories. **Closest to root under Windows would be c:.**

Under Windows, the various partitions are detected at boot and assigned a drive letter. Under Linux, unless you mount a partition or a device, the system does not know of the existence of that partition or device. This might not seem to be the easiest way to provide access to your partitions or devices, but it offers great flexibility.

This kind of layout, known as the unified file system, does offer several advantages over the approach that Windows uses. Let's take the example of the **/usr** directory. This directory of the root directory contains most of the system executables. With the Linux file system, you can choose to mount it off another partition or even off another machine over the network. The underlying system will not know the difference, because /usr appears to be a local directory that is part of the local directory structure.

How many times have you wished to move around executables and data under Windows, only to run into registry and system errors? Try moving c:windowssystem to another partition or drive.

Another point likely to confuse newbies is the use of the frontslash '/' instead of the backslash as in DOS/Windows. So, c:windowssystem would be /c/windows/system. Well, Linux is not going against convention here. Unix has been around a lot longer than Windows and was the standard a lot before Windows was. Rather, DOS took the different path, using '/' for command-line options and '' as the directory separator.

To liven up matters even more, Linux also chooses to be case sensitive. This means that the case, whether in capitals or not, of the characters becomes very important. So, "this" is not the same as "THIS" or "ThIs" for that matter. This single feature probably causes the most problems for newbies.

Like all modern operating systems, Linux also uses a hierarchical directory structure. That is, files are stored under directories.

The Linux file hierarchy is based on UNIX. Every part of the file system can be traced back to one central point, the 'root'.

The file hierarchy appears like an inverted tree shown in figure 2.1, where 'root' is at the top with several branches at the bottom. This reduces the search time and provides faster access to a file.

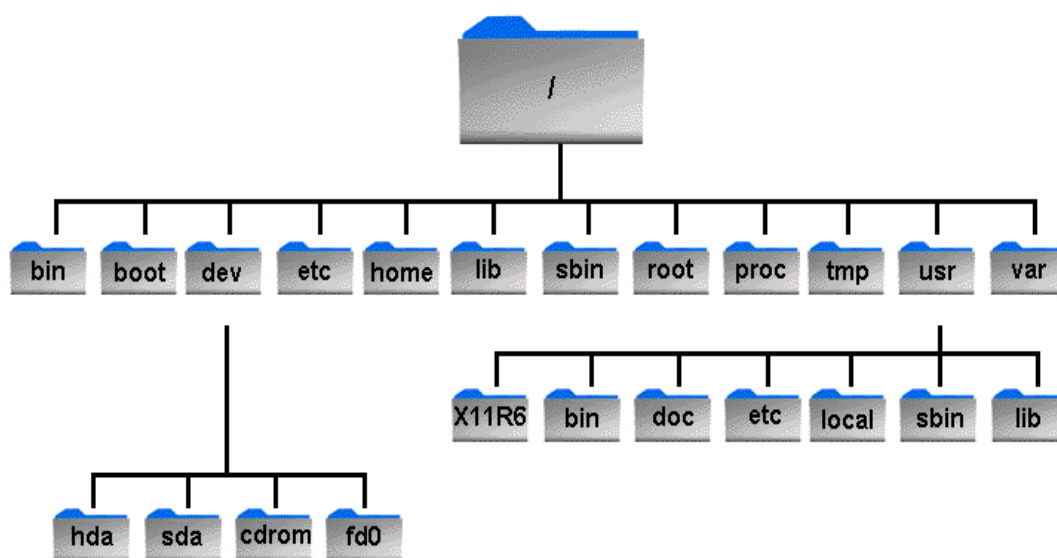


Figure 1.1 The Linux File Hierarchy

The Linux file system is spread over several physical devices. Different parts of the file hierarchy may exist on totally separate partitions on the hard disks, CD-ROMs, Network File System (NFS) shares and floppy disks. Some directories are put on separate partitions or systems for easier backups or security concerns. Other directories have to be on the root partition, because they are vital for the boot process.

Directory	Contains
<b>bin/</b>	Required Boot-time binaries
<b>boot/</b>	Boot configuration files for the OS loader and kernel image
<b>dev/</b>	Device files
<b>etc/</b>	System configuration files and scripts
<b>home/</b>	User/Sub branch directories
<b>lib/</b>	Main OS shared libraries and kernel modules
<b>lost+found/</b>	Storage directory for "recovered" files
<b>mnt/</b>	Temporary point to connect devices to
<b>proc/</b>	Pseudo directory structure containing information about the kernel, currently running processes and resource allocation

Directory	Contains
<b>root/</b>	Linux (non-standard) home directory for the root user. Alternate location being the / directory itself
<b>sbin/</b>	System administration binaries and tools
<b>tmp/</b>	Location of temporary files



<b>usr/</b>	Difficult to define - it contains almost everything else including local binaries, libraries, applications and packages (including X Windows)
<b>var/</b>	Variable data, usually machine specific. Includes spool directories for mail and news

Table 1.1 – The directories under the root.

## 9.1. The Root (/) Directory

All files are stored on the disk under one main directory called the **root**. The hierarchy of the file system begins at the *root directory*, written as "/" (slash) on Linux, and "\" (backslash) on DOS/ Windows. Beneath the **root** directory are many files and directories. Each directory may contain files and other directories.

As the figure 2.1 shows, there are several subdirectories like **bin**, **boot**, **dev**, **etc**, **home**, **lib**, **sbin**, **root**, **tmp**, **proc**, **usr** and **var** under the **root** (/) directory. These directories further contain other directories.

## 9.2. The /bin Directory

The **/bin** directory contains all the startup utilities and must be present for the operating system to boot. It contains:

1. Shells,
2. Basic file and text utilities like **ls**, **pwd**, **cut**, **head**, **tail** and **ed**,
3. Commands that may be used by the system administrator and the users.

There should be no subdirectories within **/bin**.

## 9.3. The /boot Directory

The **/boot** directory contains all the files required for the boot process except the configuration files. The configuration files are stored in the **/etc** directory. The kernel can be located either in **/'** or **/boot** directory.

## 9.4. The /dev Directory

The **/dev** directory contains files which represent the devices that are attached to the system. The devices could be the hard disk drives, CD-ROM drives, floppy drives, printers, or network card. These files are essential for the system to function properly.

## 9.5. The /etc Directory

The **/etc** directory is the nerve center of your system. It contains all system related configuration files. It is reserved for configuration files that are local to your machine. No binaries are to be put in **/etc**. The **X11** and **skel** directories are subdirectories of

**/etc**. The **X11** directory contains X Windows configuration files such as **XF86Config**. The **X11** configuration files are essential for the graphical support. Some files that are kept in the **/etc** directory are **passwd**, **inittab**, **mttools**, **profile**, **group**, **lilo.conf**, **syslog.conf**, **exports**, **hosts**, **protocols** and **services**.

## 9.6. The **/home** Directory

The **/home** directory contains the home directories of all users (except the root user).

Every user needs a directory to work. This is the home directory of the user and is created by the same name as the user. Home directories of all users (except 'root' user) are placed under the directory **/home**. For instance,

**/home/james**

where **james** is the home directory of the user 'james'.

If there are groups of users, a directory for each group is created and placed under the **/home** directory. Then, a directory for each user is created and placed under the respective group directory. For instance,

**/home/admin**

where **admin** is the group directory, which may contain directories of users belonging to the group 'admin'.

## 9.7. The **/root** Directory

The **/root** directory is the home directory of 'root' users. When logged, the 'root' user is taken to this directory.

A 'root' user has to perform several tasks related to system administration and configuration. These tasks can be performed being in the '/' directory. In order to save the day-to-day work of the 'root' users, the **/root** directory is created. This helps in reducing the clutter in the '/' directory.

Due to some reason, if the **/root** directory does not exist, the 'root' user will be working in the '/' directory, which is the traditional location for root users. There is a **root** directory under **/home** directory also. Therefore, a 'root' user's home directory may be '/', **/root** or **/home/root**.

## 9.8. The **/lib** Directory

The **/lib** directory contains only library files that are needed to execute the binaries in **/bin** and **/sbin**. These shared library files are particularly important for booting the system and executing commands within the 'root' filesystem.

## 9.9. The **/sbin** Directory

The **/sbin** directory is meant for **system binaries**. This directory contains files that

should only be used by the 'root' user. The **/sbin** directory contains essential system administration scripts and programs, including those related to:

1. User management,
2. Disk administration,
3. System event control (restart and shutdown programs),
4. System repair and recovery,
5. Certain networking programs.

## 9.10. The /proc Directory

The **/proc** directory contains files associated with the **executing** kernel. These files contain information about:

1. The state of the system's resource usage (memory, swap space and CPU);
2. Each process running on the system.

Note:



The **/proc** file system is the main source of information for a program called **top**. This is a very useful administration tool as it displays the information of the CPU and memory resources being used by each process on the system.

## 9.11. The /tmp Directory

As the name suggests, the **/tmp** directory is used by programs that require temporary files. The files and directories located in **/tmp** directory are deleted whenever the system is booted, and are not preserved when a program is closed and opened again.

## 9.12. The /usr Directory

The **/usr** directory contains a vast majority of files on every Linux system. The **/usr** directory, in turn, has a hierarchy. The documentation, libraries, and header files of all programs are stored in the **/usr** directory. It is the second major section of the file system that consists of shareable, read-only data. Therefore, the **/usr** directory is shared between various users.

The subdirectories under the **/usr** directory are as follows:

<b><u>usr</u></b>	<b><u>Secondary Hierarchy</u></b>
X11R6	X Window System.
X386	X Window System.
bin	Most user commands, compilers, editors, and games
games	Games and educational binaries
include	Header files included by C and C++ programs
lib	Libraries
local	Local hierarchy (empty after main installation)
sbin	Non-vital system binaries used by system administrator
share	Architecture-independent data
src	Source code

### 9.13. The /var Directory

The /var directory contains variable data files, which include spool directories and files for mail and news, data bank data and log files; administrative and logging data, and transient and temporary files.

Some of the subdirectories of /var are as follows:

1. /var/backups
2. /var/cron
3. /var/lib
4. /var/local
5. /var/messages
6. /var/preserve

### 9.14. The /mnt Directory

To use CD-ROMs or floppy disks, the file systems on these disks should be loaded on your system. This is known as *mounting* the file systems. When you are done, you must unmount the file systems.

Programs related to mounting and unmounting the file systems are placed in the **/mnt** directory.

## Summary:

You should now be able to:

- ◆ Describe the history of Unix
- ◆ List the principles of Unix on which the OS runs.
- ◆ Define General Public License (GNU)
- ◆ Explain the reasons of preferring Linux over other operating systems
- ◆ List the features of Red Hat enterprise Linux
- ◆ Recommend hardware specifications required to install Linux
- ◆ Use some basic commands of Linux
- ◆ Describe the Linux file system hierarchy



## Chapter 2

### The bash Shell

#### Objectives:

After completing this chapter, you will be able to:

- Describe the bash shell and its utilities
- Use command line shortcuts and expansion
- Describe history tricks
- Describe and use gnome terminal

#### 1. bash Introduction

##### 1.1. What is a bash?

**Bash** is the shell, or command language interpreter, for the GNU operating system. The name is an acronym for "**B**ourne-**A**gain **S**hell", a pun on Stephen Bourne, the author of the direct ancestor of the current Unix shell /bin/sh, which appeared in the Seventh Edition Bell Labs Research version of Unix.

Bash is largely compatible with sh and incorporates useful features from the Korn shell ksh and the C shell csh. It is intended to be a conformant implementation of the IEEE POSIX Shell and Tools specification (IEEE Working Group 1003.2). It offers functional improvements over sh for both interactive and programming use.

While the GNU operating system provides other shells, including a version of csh, Bash is the default shell. Like other GNU software, Bash is quite portable. It currently runs on nearly every version of Unix and a few other operating systems - independently-supported ports exist for MS-DOS, OS/2, Windows 95/98, and Windows NT.

## 2. Command Line Shortcuts and Expansion

This is necessary to know about command line shortcuts:

- ◆ Press [Ctrl]B to move backward one character.
- ◆ Press [Ctrl]F to move forward one character.
- ◆ Press [Esc]B to move one word backward.
- ◆ Press [Esc]F to move one word forward.
- ◆ Press [Ctrl]A to move to the first character in the line.
- ◆ Press [Ctrl]E to move to the end of the line.
- ◆ Press [Ctrl]U to delete the current line.
- ◆ Press [Ctrl]K to delete from the cursor's current position to the end of the line.

Figure 2.1

When typing commands, it is often necessary to issue the same command on more than one file at the same time. The use of Wild cards allows one pattern to expand to multiple filenames with a process called file globbing .

Typing the following command **\$ rm \*mp3** is same as typing: **\$ rm gonk.mp3 zonk.mp3.**

The table below shows you the wild card characters:

<b>*</b>	<b><i>matches zero or more characters</i></b>
<b>?</b>	matches any single character
<b>[a-z]</b>	matches a range of characters
<b>[^a-z]</b>	matches all except the range

### 2.1. The '\*' Wildcard

You will use the asterisk (\*) most frequently when you are searching for files. The asterisk allows you search everything that matches the pattern you are looking for. You can specify the pattern, and the operating system performs the search and displays all the files that match the pattern. The '\*' wildcard helps to narrow down the



search as much as possible. Therefore, when you type the following command, it displays a list of all the files with the extension **'txt'**.

```
$ ls *.txt
```

Type the following command:

```
$ ls sn*
```

This command not only displays the file **'sneakers.txt'**, but also displays other files with names that begin with **'sn'**.

## 2.2. The '?' Wildcard

Another way to narrow down the search is to use the question mark symbol (**'?'**). Like the asterisk, using **'?'** helps locate a file that matches a pattern.

The **'?'** wildcard is useful for matching a single character. For instance, give the following command:

```
$ ls sneaker?.txt
```

It displays all files with the name **'sneaker'**, followed by a single character, and the extension **'txt'**. This would not only display the file **'sneakers.txt'**, but also display the file **'sneakerz.txt'** (if it exists).

## 2.3. The '[' Wildcard

You can further limit the search for files by using the **'[]'** (bracket) wildcard. You can specify a set of values or a range of values in the bracket. The search is based on the values specified in the bracket. For instance, to search a file whose name begins with **'M'** followed by either **'A'** or **'B'** or **'C'**, give the following command:

```
$ ls M[ABC]
```

You can also give a range of characters to be matched in the **'[]'** bracket. For instance, if you are looking for a file whose name starts with **'M'**, followed by any character between **'A'** to **'Z'**, then the command is:

```
$ ls M[A-Z]
```

### Note:

If a special character has to be used as its original meaning and not as a wildcard, precede the character with a backslash (**'\'**). For instance, the command **ls sneaker\\*.txt** displays a file with the name **'sneaker'**, with an asterisk at the end, and the extension **'txt'**, that is, the **sneaker\*.txt** file.



Type <TAB> to complete command lines:

1. For the command name, it will complete command;
2. For an argument, it will complete a file name.

Example:

```
$ xte<TAB>
```

```
$ xterm
```

```
$ ls myf<TAB>
```

```
$ ls myfile.txt
```

'bash' stores a history of commands you've entered, which can be used to repeat commands.

You can use '**history**' command to see list of "remembered" commands.

```
$ history
```

In addition to basic command recall with the arrow keys, the bash history mechanism supports a variety of advanced ways of retrieving command from the list.

**!!** - To repeat last command

**!c** - To repeat last command that started with c

**!n** - To repeat command by its number in history output

### 3. History Tricks

Use the **up** and **down** arrow keys to scroll through previous commands. Type <**CTRL-R**> to search for a command on command history. To recall last argument from previous command:

<**Esc**>. - The escape key followed by a period.

<**Alt**> . - Hold down the alt key while pressing the period.

### 4. Gnome-terminal

Gnome-terminal is a graphical terminal emulator, with support for maintaining multiple "tabbed" shells simultaneously.

The path to open gnome-terminal is:

#### Applications/System Tool/Terminal

- |                             |                  |
|-----------------------------|------------------|
| < <b>CTRL-SHIFT-t</b> >     | - Open a new tab |
| < <b>CTRL-PgUp/PgDN</b> >   | - Next/ Prev tab |
| < <b>CTRL-shift – c/v</b> > | - Copy/ Paste    |

## Summary:

You should now be able to:

- Describe the bash shell and its utilities
- Use command line shortcuts and expansion
- Describe history tricks
- Describe and use gnome terminal



## Chapter 3

### Standard I/O Pipes

#### Objectives:

After completing this chapter, you will be able to:

- ◆ Use redirection in the command line
- ◆ Use standard input
- ◆ Use standard output
- ◆ Use pipes

At first glance, many people think the Unix Shell is a more cryptic version of the MS-Windows/DOS command prompt. However, the Unix shell can do much more than launch programs and copy files. Sequences of commands can be strung together in "Shell Scripts" to automate tedious or repetitive tasks.

Unix systems provide a series of tools that are helpful when writing shell scripts. Typically, each tool performs a specific task, accepting input from the "**standard input**" source and sending its output to the "**standard output source**".

#### 1. Redirection

The standard input, output and error streams can be redirected to files. This is useful for saving the output of commands. Redirecting the output of a command can be done by using the '**greater than**' symbol ( > ) followed by the name of the file the output is to be saved in.

For example, to save a list of the files in the current directory, use the command:

```
ls >/tmp/filelist
```

Running this command will create a file named **filelist** in the **/tmp** directory containing the output of **ls**. If a file named **filelist** already exists, it will be overwritten by this command.

#### 2. Standard Input

Many commands can accept input from a facility called *standard input*. By default, standard input gets its contents from the keyboard, but can be redirected. To redirect

standard input from a file instead of the keyboard, the "<" character is used as follows:

```
[me@linuxbox me]$ sort < file_list.txt  
[me@linuxbox me]$ sort < file_list.txt > sorted_file_list.txt
```

### 3. Standard Output

Most command line programs that display their results do so by sending their results to a facility called *standard output*. By default, standard output directs its contents to the display. To redirect standard output to a file, the ">" character is used as follows:

```
[me@linuxbox me]$ ls > file_list.txt
```

In this example, the **ls** command is executed and the results are written in a file named `file_list.txt`. Since the output of **ls** was redirected to the file, no results appear on the display.

Each time the command above is repeated, `file_list.txt` is overwritten (from the beginning) with the output of the command **ls**. If you want the new results to be *appended* to the file instead, use ">>":

```
[me@linuxbox me]$ ls >> file_list.txt
```

When the results are appended, the new results are added to the end of the file, thus making the file longer each time the command is repeated. If the file does not exist when you attempt to append the redirected output, the file will be created.

As you can see, a command can have both its input and output redirected. Also, be aware that the order of the redirection does not matter. The only requirement is that the redirection operators (the "<" and ">") must appear after the other options and arguments in the command.

### 4. Pipes

By far, the most useful and powerful thing you can do with I/O redirection is to connect multiple commands together with what are called ***pipes***. With pipes, the standard output of one command is fed into the standard input of another. Here is an example:

```
me@linuxbox me]$ ls -l | less
```

In this example, the output of the **ls** command is fed into less. By using this "| less" trick, you can make any command have scrolling output.

You can use input redirection for looking the contents of a file by '<' operator.

```
$ cat < file1
```

You can also use output redirection to add information to the file using '>' operator.

```
$ cat file1 > file2
```

## Summary:

You should now be able to:

- ◆ Use redirection in the command line
- ◆ Use standard input
- ◆ Use standard output
- ◆ Use pipes

**SQLSTAR**  
KNOWLEDGE TO KEEP YOU IN BUSINESS™

## Chapter 4

# Users, Groups and Permissions

### Objectives:

After completing this chapter, you will be able to:

- ◆ Manage users and groups
- ◆ Change permissions on files and folders for users, groups and others

## 1. Managing Users and Groups

First, you will learn about the process of **logging in** and **logging out** of the Linux system. Although you may be the only user on your Linux system, you should know about user accounts and managing users. As you should have your own account (other than root) for your daily routines, you should know how to establish a user account. If your system allows access to other users, you should be able to create user accounts for every one who wants access.

### 1.1. Logging In

To begin your journey in the world of Linux, you must log in. By logging, you are basically introducing yourself to the system. To log in, you should enter a name, followed by a password.

The text mode login will appear as shown below:

- Red Hat Linux release 7.0
- Kernel 2.xx on an i686
- localhost login: **root**
- Password: ***yourrootpassword***

The graphical mode login screen will appear as follows:





Figure 4.1- The Graphical Login Screen

If you log in as root user, you must type the user login name and the password. The system authenticates the password and presents the desktop screen as shown in figure 2.2, from where you can start working with the Red Hat Linux system.

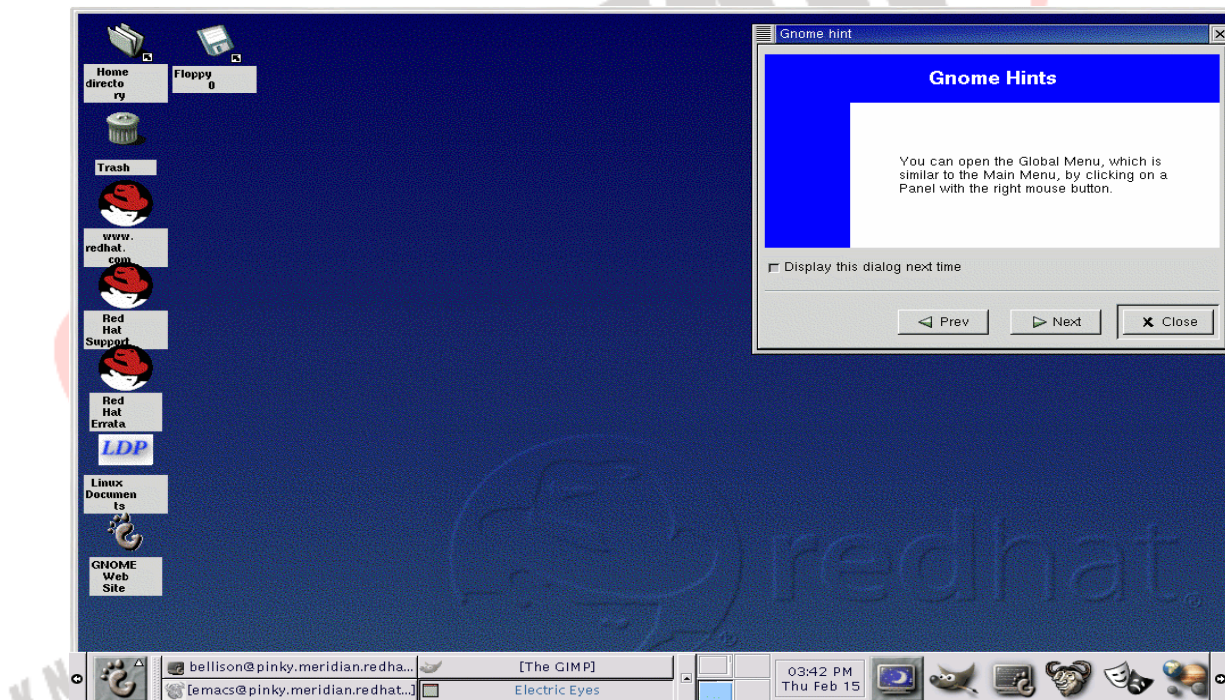


Figure 4.2 – The GNOME Desktop


## 1.2. Establishing User Accounts

Red Hat Linux system uses accounts to manage privileges and maintain security. To use the Linux system, you should have an account in the system. This account is known as a **user account**. All access to the Linux system is through the user account. The system administrator has the privilege to set up an account for each user, except the 'root' account. The 'root' account is created at the time of installation. Not all accounts are created equal; some accounts have fewer rights to access files or services than others.

## 1.3. Working as 'root'

The 'root' account is created at the time of installing Linux. The default name of the 'root' user is taken as **root**.

The 'root' is a special user account that is available on every Linux system. This special user has full access to the system.

<p>Note:</p> 	<p>As the Red Hat Linux system creates the 'root' account during installation, some new users are tempted to use only this account for all their activities. This is a bad idea. Since the 'root' account is allowed to do anything on the system, you can easily damage your system by mistakenly deleting or modifying sensitive system files.</p>
---	--

## 1.4. User Accounts

Every person using your system should have a unique account. By keeping separate accounts for each user, you can have a better idea of who is accessing your system. Every user account has a unique name and a password. The user information in the **/etc/passwd** is maintained in the following format:

**username:password:user ID:group ID:comment:home directory:login command**

Every entry in the **/etc/passwd** file comprises seven fields separated by colons. The fields contain the following values in a sequence:

- The username,
- The password,
- The user identification (UID),



- The group identification (GID),
- A comment (usually the user's real name and other details),
- The home directory (the directory where the user is placed when he /she logs in,
- The login command – The command executed when the user logs in.

### 1.4.1. useradd - Create new user account

**\$useradd -d <home Directory> -s <defaultshell> -u <user id> -g <group id> -c <comment> <username>**

<home Directory> - users home directory

Example:

/home/user1

<Default shell> - default shell on which user will log in

Example:

/bin/bash

<user id> - user's unique id number

Example

1001

<group id> - user's group id

Example

1001

<comment> - some comment for the user

Example

"local user"

**Example:**

useradd -d /home/user1 -s /bin/bash -u 1001 -g 1001 -c "local user" user1

### 1.4.2. usermod - Modify user account

**\$usermod -d <home Directory> -s <default shell> -u  
<user id> -g <group id> -c <comment> <username>**

<home Directory> - change users home directory

Example:

/home/user2

<Default shell> - change default shell on which user will log in

Example:

/bin/csh

<user id> - change user's unique id number

Example:

1002

<group id> - change user's group id

Example:

1002

<comment> - change comments for the user

Example

"change local user"

**Example:**

usermod -d /home/user2 -s /bin/csh -u 1002 -g 1002 -c "change local user" user2

## 1.5. Some Useful Commands

**cat** - Sends file contents to standard output. This is a way to list the contents of short files to the screen. It works well with piping.

Example:      cat .bashrc

Description: Sends the contents of the ".bashrc" file to the screen.

**cd** - Changes the current working directory to /home. The '/' indicates relative to root. No matter what directory you are in when you execute this command, the directory will be changed to "/home".

Example:      cd httpd

Description: Changes the current working directory to httpd, relative to the current location which is "/home". The full path of the new working directory is "/home/httpd".

**cd ..** - Moves to the parent directory of the current directory. This command will make the current working directory as `"/home"`.

**Cd ~** - Moves to the user's home directory which is `"/home/username"`. The `'~'` indicates the users home directory.

**Cp** - Copies files from one directory to another.

Example: `cp myfile yourfileCopy`

Description: The files "myfile" to the file "yourfile" in the current working directory. This command will create the file "yourfile" if it doesn't exist. It will normally overwrite it without warning if exists.

Example: `cp -i myfile yourfile`

Description: With the `"-i"` option, if the file "yourfile" exists, you will be prompted before it is overwritten.

Example: `cp -i /data/myfile`

Description: Copies the file `"/data/myfile"` to the current working directory and names it "myfile". It also prompts before overwriting the file.

Example: `cp -dpr srcdir destdirCopy`

Description: Copies all files from the directory "srcdir" to the directory "destdir" preserving links (`-p` option), file attributes (`-p` option), and copy recursively (`-r` option). With these options, a directory and all its contents can be copied to another directory.

**dd** - Disk duplicate. This command converts and copies a file.

Example: `dd if=/dev/hdb1 of=/backup/`

Description: "if" means input file, "of" means output file.

**df** - Shows the amount of disk space used on each mounted file system.

**less** - Allows the user to move page up and down through the file.

Example: `less textfile`

Description: Displays the contents of textfile.

**ln** - Creates a symbolic link to a file.

Example: `ln -s test symlink`

Description: Creates a symbolic link named symlink that points to the file test. Typing "ls -li test symlink" will show that the two files are different with different inodes. Typing "ls -l test symlink" will show that symlink points to the file test.

**locate** - A fast database driven file locator. This command builds the slocate database, taking several minutes to complete. It must be used before searching for files; however cron runs this command periodically on most systems.

Example: `slocate -u whereis`

Description: Lists all files whose names contain the string "whereis".

**logout** - Logs the current user off the system.

**ls** - Lists files in the current working directory except those starting with '.' and only shows the file name.

Example: `ls -al`

Description: Lists all files in the current working directory in long listing format, showing permissions, ownership, size, and time and date stamp.

**more** - Allows file contents or piped output to be sent to the screen, one page at a time.

Example: `more /etc/profile`

Description: Lists the contents of the "/etc/profile" file to the screen, one page at a time.

Example: `-al | more`

Description: Performs a directory listing of all files and pipes the output of the listing through more. If the directory listing is longer than a page, it will be listed one page at a time.

**mv** - Moves or renames files.

Example: `mv -i myfile yourfile`

Description: Moves the file from "myfile" to "yourfile". This effectively changes the name of "myfile" to "yourfile".

Example: `mv -i /data/myfile`

Description: Moved the file "myfile" from the directory "/data" to the current working directory.

**pwd** - Shows the name of the current working directory.

Example: `more /etc/profile`

Description: Lists the contents of the "/etc/profile" file to the screen, one page at a time.

**shutdown** - Shuts the system down.

Example: `shutdown -h now`

Description: Shuts the system down to halt immediately.

Example: `shutdown -r now`

Description: Shuts the system down immediately, and the system reboots.

**whereis** - Shows where the binary, source and manual page files are for a command.

## 2. Changing Permissions

It is possible to change file permissions, if you wish to grant or remove permissions from some users. You can change file permissions if you are:

- A 'root' user,
- The file owner.

You can change the file permissions using the command **chmod**. Permissions can be changed in two ways:

- ♦ Using letters with the **chmod** command. The letters represent -
  - Permissions,
  - Different users.
- ♦ Using *numbers* that represent permissions, along with the **chmod** command.

Following is a list of options that can be used with the **chmod** command:

### 1. Identities:


- a) u: the user who owns the file (the owner),
- b) g: the group to which the user belongs
- c) o: others (neither the owner nor the owner's group)
- d) a: everyone or all (u, g, and o)

### 2. Permissions:

- a) r: read access
- b) w: write access
- c) x: execute access

**3. Actions:**

- a) + : grants the permission
- b) - : removes the permission
- c) = : makes it the only permission

<b>Note:</b> 	Another permission symbol is <b>t</b> , for the sticky bit. If a sticky bit is assigned to a file, a user who wants to remove or rename that file must own the file, own the directory, have write permission, or be root
---	---

The existing file permissions of the file 'sneakers.txt' are as follows:

```
-rw-rw-r-- 1 newuser newuser 150 Mar 19 08:08 sneakers.txt
```

**Summary:**

You should now be able to:

- ◆ Manage users and groups
- ◆ Change permissions on files and folders for users, groups and others

## Chapter 5

### vi Editor

#### Objectives:

After completing this chapter, you will be able to:

- ◆ Describe vi Editor
- ◆ List different modes of vi Editor
- ◆ Manipulate (insert, delete and change) text within vi
- ◆ Navigate within the vi Editor
- ◆ Search text from the document
- ◆ Save files and quit from vi
- ◆ Summarize vi commands
- ◆ Print documents

### 1. Overview of Vi Editor

The VI editor is a screen-based editor used by many Unix users.

#### 1.1. Before You Begin

The VI editor uses the full screen, so it needs to know what kind of terminal you have. When you log in, wiliki should ask you what terminal you have. The prompt looks like this:

```
TERM = (vt100)
```

If you know your terminal is a vt100 (or an emulator that can do vt100), just hit return for the terminal type when you log in. If you have an hp terminal, type "hp" for the terminal type and hit return. If you are not sure what kind of terminal you have, ask a lab monitor, or have someone help you set the correct terminal type.

#### 1.2 Starting the VI Editor

The VI editor lets a user create new files or edit existing files. The command to start the VI editor is vi, followed by the filename. For example to edit a file called *temporary*, you

would type vi temporary and then return. You can start VI without a filename, but when you want to save your work, you will have to tell VI which filename to save it into later.

When you start VI for the first time, you will see a screen filled with tildes (A tilde looks like this: ~) on the left side of the screen. Any blank lines beyond the end of the file are shown this way. At the bottom of your screen, the filename should be shown, if you specified an existing file, and the size of the file will be shown as well, like this:

```
"filename" 21 lines, 385 characters
```

If the file you specified does not exist, then it will tell you that it is a new file, like this:

```
"newfile" [New file]
```

## 2. Three Modes of vi

The vi editor has three modes - command mode, insert mode and command line mode.

1. **Command mode:** Letters or sequence of letters interactively command vi. Commands are case sensitive. The ESC key can end a command.
2. **Insert mode:** Text is inserted. The ESC key ends insert mode and returns you to command mode. One can enter insert mode with the "i" (insert), "a" (insert after), "A" (insert at end of line), "o" (open new line after current line) or "O" (Open line above current line) commands.
3. **Command line mode:** One enters this mode by typing ":" which puts the command line entry at the foot of the screen.

## 3. Inserting Text

The vi program is now in command mode. Insert text into the file by pressing i, which places the editor into insert mode, and begin typing.

```
Now is the time for all good men to come to the aid of the party.
```

```
~
~
~
~
~
```

Type as many lines as you want (pressing Enter after each). You may correct mistakes with the Backspace key. To end insert mode and return to command mode, press Esc.

In command mode, you can use the arrow keys to move around in the file. (If you have



only one line of text, trying to use the up- or down-arrow keys will probably cause vi to beep at you.)

There are several ways to insert text other than the **i** command. The **a** command inserts text beginning after the current cursor position, instead of at the current cursor position. For example, use the left arrow key to move the cursor between the words ``good" and ``men."

**Now is the time for all good\_men to come to the aid of the party.**

~  
~  
~  
~  
~

Press **a** to start insert mode, type ``wo", and then press Esc to return to command mode.

**Now is the time for all good women to come to the aid of the party.**

~  
~  
~  
~  
~

To begin inserting text at the next line, use the **o** command. Press **o** and enter another line or two:

**Now is the time for all good humans to come to the aid of the party.**

**Afterwards, we'll go out for pizza and beer.**

~  
~  
~  
~

## 4. Deleting Text

From command mode, the **x** command deletes the character under the cursor. If you press x five times, you'll end up with:

**Now is the time for all good humans to come to the aid of the party.**

**Afterwards, we'll go out for pizza and\_**

~  
~  
~  
~

Now press **a** and insert some text, followed by esc:

**Now is the time for all good humans to come to the aid of the party.**

**Afterwards, we'll go out for pizza and Diet Coke\_**

~  
~  
~  
~

You can delete entire lines using the command dd (that is, press d twice in a row). If the cursor is on the second line and you type dd, you'll see:

**Now is the time for all good humans to come to the aid of the party.**

~  
~  
~  
~  
~

To delete the word that the cursor is on, use the **dw** command. Place the cursor on the word "good", and type **dw**.

**Now is the time for all humans to come to the aid of the party.**

~

```
~
~
~
```

## 5. Changing Text

*You can replace sections of text using the **R** command. Place the cursor on the first letter in ``party'', press **R**, and type the word ``hungry''.*

**Now** is the time for all good humans to come to the aid of the hungry.

```
~
~
~
~
~
~
```

Using **R** to edit text is like the **i** and **a** commands, but **R** overwrites, rather than inserts, text.

The **r** command replaces the single character under the cursor. For example, move the cursor to the beginning of the word ``Now'', and press **r** followed by **C**, you'll see:

Cow is the time for all good humans to come to the aid of the hungry.

```
~
~
~
~
~
~
```

The ``~" command changes the case of the letter under the cursor from upper- to lower-case, and back. For example, if you place the cursor on the ``o" in ``Cow" above and repeatedly press ~, you'll end up with:

**COW** IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.

```
~
```

~  
~  
~  
~

## 6. Cursor Movement

You already know how to use the arrow keys to move around the document. In addition, you can use the **h**, **j**, **k**, and **l** commands to move the cursor left, down, up, and right, respectively. This comes in handy when (for some reason) your arrow keys aren't working correctly.

The **w** command moves the cursor to the beginning of the next word; the **b** command moves it to the beginning of the previous word.

The **O** command (that's the zero key) moves the cursor to the beginning of the current line, and the **\$** command moves it to the end of the line.

When editing large files, you'll want to move forwards or backwards through the file a screenful at a time. Pressing **Ctrl-F** moves the cursor one screenful forward, and **Ctrl-B** moves it a screenful back.

To move the cursor to the end of the file, press **G**. You can also move to an arbitrary line; for example, typing the command **10G** would move the cursor to line 10 in the file. To move to the beginning of the file, use **1G**.

You can couple moving commands with other commands, such as those for deleting text. For example, the **ds** command deletes everything from the cursor to the end of the line; **dG** deletes everything from the cursor to the end of the file, and so on.

## 7. Searching for Text

The Vi editor has two kinds of searches: string and character. For a string search, the **/** and **?** commands are used. When you start these commands, the command just typed will be shown on the bottom line, where you type the particular string to look for. These two commands differ only in the direction where the search takes place.

The **/** command searches forwards (downwards) in the file, while the **?** command searches backwards (upwards) in the file. The **n** and **N** commands repeat the previous search command in the same or opposite direction, respectively. Some characters have special meanings to Vi, so they must be preceded by a backslash (\) to be included as part of the search expression.

Special characters:

- ^ - Beginning of the line. (At the beginning of a search expression.)
- .

- \* - Matches zero or more of the previous character.
- \$ - End of the line (At the end of the search expression.)
- [ - Starts a set of matching, or non-matching expressions...  
     For example: /f[iae]t matches either of these: fit fat fet In this form,  
     it matches anything except these: /a[^bcd] will not match any of  
     these, but anything with an a and another letter: ab ac ad
- < - Put in an expression escaped with the backslash to find the ending or beginning  
     of a word. For example: /\<the\> should find only word the, but not words like  
     these: there and other.
- > - See the '<' character description above.

The character search searches within one line to find a character entered after the command. The **f** and **F** commands search for a character on the current line only. **f** searches forwards and **F** searches backwards and the cursor moves to the position of the found character.

The **t** and **T** commands search for a character on the current line only, but for **t**, the cursor moves to the position before the character, and **T** searches the line backwards to the position after the character.

These two sets of commands can be repeated using the **;** or **,** command, where **;** repeats the last character search command in the same direction, while **,** repeats the command in the reverse direction.

## 8. Saving Files and Quitting vi

To quit vi without making changes to the file, use the command **:q!**. When you press the **``:**, the cursor moves to the last line on the screen and you'll be in last line mode.

**COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.**

~  
~  
~  
~  
~  
~

In last line mode, certain extended commands are available. One of them is **q!**, which quits vi without saving. The command **:wq** saves the file and then exits vi. The command **ZZ** (from command mode, without the **``:**) is equivalent to **:wq**. If the file has not been changed since the last save, it merely exits, preserving the modification time of the last change. Remember that you must press Enter after a command entered in last line mode.

**To save the file without quitting vi, use :w.**

## 9. Summary of vi Commands

This list is a summary of vi commands, categorized by function.

### 9.1. Cutting and Pasting/Deleting Text

Command	Description
"	Specify a buffer to be used any of the commands using buffers. Follow the " with a letter or a number, which corresponds to a buffer.
D	Delete to the end of the line from the current cursor position.
P	Paste the specified buffer before the current cursor position or line. If no buffer is specified (with the " command), then 'P' uses the general buffer.
X	Delete the character before the cursor.
Y	Yank the current line into the specified buffer. If no buffer is specified, then the general buffer is used.
d	Delete until <i>where</i> . "dd" deletes the current line. A count deletes that many lines. Whatever is deleted is placed into the buffer specified with the " command. If no buffer is specified, then the general buffer is used.
p	Paste the specified buffer after the current cursor position or line. If no buffer is specified (with the " command.) then 'p' uses the general buffer.
x	Delete character under the cursor. A count tells how many characters to delete. The characters will be deleted after the cursor.
y	Yank until, putting the result into a buffer. "yy" yanks the current line. A count yanks that many lines. The buffer can be specified with the " command. If no buffer is specified, then the general buffer is used.

### 9.2. Inserting New Text

Command	Description
---------	-------------

Command	Description
<b>A</b>	Append at the end of the current line.
<b>I</b>	Insert from the beginning of a line.
<b>O</b>	(letter oh) Enter <i>insert</i> mode in a new line above the current cursor position.
<b>a</b>	Enter <i>insert</i> mode, the characters typed in will be inserted after the current cursor position. A count inserts all the text that had been inserted that many times.
<b>i</b>	Enter <i>insert</i> mode, the characters typed in will be inserted before the current cursor position. A count inserts all the text that had been inserted that many times.
<b>o</b>	Enter <i>insert</i> mode in a new line below the current cursor position.

### 9.3. Moving the Cursor within a File

Command	Description
<b>^B</b>	Scroll backwards one page. A count scrolls that many pages.
<b>^D</b>	Scroll forwards half a window. A count scrolls that many lines.
<b>^F</b>	Scroll forwards one page. A count scrolls that many pages.
<b>^H</b>	Move the cursor one space to the left. A count moves that many spaces.

Command	Description
---------	-------------



Command	Description
<b>^J</b>	Move the cursor down one line in the same column. A count moves that many lines down.
<b>^M</b>	Move to the first character on the next line.
<b>^N</b>	Move the cursor down one line in the same column. A count moves that many lines down.
<b>^P</b>	Move the cursor up one line in the same column. A count moves that many lines up.
<b>^U</b>	Scroll backwards half a window. A count scrolls that many lines.
<b>\$</b>	Move the cursor to the end of the current line. A count moves to the end of the following lines.
<b>%</b>	Move the cursor to the matching parenthesis or brace.
<b>^</b>	Move the cursor to the first non-whitespace character.
<b>(</b>	Move the cursor to the beginning of a sentence.
<b>)</b>	Move the cursor to the beginning of the next sentence.
<b>{</b>	Move the cursor to the preceding paragraph.
<b>}</b>	Move the cursor to the next paragraph.
<b> </b>	Move the cursor to the column specified by the count.
<b>+</b>	Move the cursor to the first non-whitespace character in the next line.
<b>-</b>	Move the cursor to the first non-whitespace character in the previous line.
<b>-</b>	Move the cursor to the first non-whitespace character in the current line.

Command	Description
<b>O</b>	(Zero) Move the cursor to the first column of the current line.
<b>B</b>	Move the cursor back one word, skipping over punctuation.
<b>E</b>	Move forward to the end of a word, skipping over punctuation.
<b>G</b>	Go to the line number specified as the count. If no count is given, then go to the end of the file.
<b>H</b>	Move the cursor to the first non-whitespace character on the top of the screen.
<b>L</b>	Move the cursor to the first non-whitespace character on the bottom of the screen.
<b>M</b>	Move the cursor to the first non-whitespace character on the middle of the screen.
<b>W</b>	Move forward to the beginning of a word, skipping over punctuation.
<b>b</b>	Move the cursor back one word. If the cursor is in the middle of a word, move the cursor to the first character of that word.
<b>e</b>	Move the cursor forward one word. If the cursor is in the middle of a word, move the cursor to the last character of that word.
<b>h</b>	Move the cursor to the left one character position.

Command	Description
<b>j</b>	Move the cursor down one line.
<b>k</b>	Move the cursor up one line.
<b>l</b>	Move the cursor to the right one character position.
<b>w</b>	Move the cursor forward one word. If the cursor is in the middle of a word, move the cursor to the first character of the next word.

## 10. Printing in Linux

The Linux printing system - the **lp** system - is a port of the source code written by the Regents of the University of California for the Berkeley Software Distribution version of the UNIX operating system.

*By far, the most simplistic way to print in the Linux operating system is to send the file to be printed directly to the printing device. One way to do this is to use the **cat** command. As the root user, one could do something like:*

```
# cat thesis.txt > /dev/lp
```

In this case, **/dev/lp** is a symbolic link to the actual printing device - be it a dot-matrix, laser printer, typesetter, or plotter. For the purpose of security, only the root user and users in the same group as the print daemon are able to write directly to the printer. This is why commands such as **lpr**, **lprm**, and **lpq** have to be used to access the printer.

## Summary:

You will now be able to:

- ◆ Describe vi Editor
- ◆ List different modes of vi Editor
- ◆ Manipulate (insert, delete and change) text within vi
- ◆ Navigate within the vi Editor
- ◆ Search text from the document
- ◆ Save files and quit from vi
- ◆ Summarize vi commands
- ◆ Print documents

