LINUX

WITH

SCRIPTING

Disclaimer

The copyright of the content used in the courseware will remain with Principle Company.

A Brief History of Red Hat Linux

Arguably one of the most highly regarded and widely used enterprise Linux distributions available today is Red Hat Enterprise Linux (RHEL). It is considered to be amongst the most stable and reliable operating systems and is backed by the considerable resources and technical skills of Red Hat, Inc.

RHEL 6 Essentials is designed to provide detailed information on the use and administration of the Red Hat Enterprise Linux 6 distribution. For beginners, the book covers the basics of configuring the desktop environment, resolving screen resolution issues and configuring email and web servers. Installation topics such as network installation and dual booting with Microsoft Windows are covered together with all important security topics such as configuring a firewall and user and group administration.

For the experienced user, topics such as remote access, logical volume management (LVM), disk partitioning, swap management, KVM virtualization, Secure Shell (SSH) and file sharing using both Samba and NFS are covered in detail to provide a thorough overview of this enterprise class operating system.

Red Hat Enterprise Linux is the product of a U.S. company called Red Hat, Inc., based in Raleigh, North Carolina near the campus of NC State University. The company was founded in the mid-1990s when Marc Ewing and Bob Young merged two companies.

In 1993 Bob Young created a company called ACC Corporation that he says he ran from his wife's "sewing closet". The name ACC was intended to represent a catalog business but was also an abbreviation of a small business his wife ran called Antiques and Collectables of Connecticut. Amongst the items sold through the ACC catalog business were Linux CDs and related open source software.

Around the same time Marc Ewing had created his own Linux distribution which he named Red Hat Linux (after his propensity to wear a red hat).

In 1995, ACC acquired Red Hat and adopted the name Red Hat, Inc. Red Hat went public in 1999 making both owners fabulously wealthy.

Early releases of Red Hat Linux were shipped to customers on floppy disks and CDs (this, of course, predated the widespread availability of broadband internet connections). When users

encountered problems with the software they could contact Red Hat only by email. In fact, Bob Young often jokes that this was effective in limiting support requests since by the time a customer realized they needed help, their computer was usually inoperative and therefore unavailable to be used to send an email message seeking assistance from Red Hat's support team.

In later years Red Hat provided better levels of support tied to paid subscriptions (and you can now get phone support 24 hours a day).

Red Hat Enterprise Linux 6 is the latest commercial offering from Red Hat and is targeted at corporate, mission critical installations. It is an open source product in that you can download the source code free of charge and build the software yourself if you wish to do so (a task not to be undertaken lightly) but if you wish to download a pre-built binary version of the software you have to buy a maintenance subscription which also provides you with support and updates.
Red Hat also sponsors the Fedora Project. The goal of this project is to provide access to a free Linux operating system (in both source and binary distributions) in the form of Fedora Linux. Fedora also acts as a proving ground for new features that eventually are adopted into the Red Hat Enterprise Linux operating system family.

For users that cannot afford a Red Hat Enterprise Linux subscription, another option is provided in the form of the CentOS operating system. CentOS is a community driven project that takes the source code to Red Hat Enterprise Linux, removes the Red Hat branding and subscription requirements, compiles it and provides the distribution for download.

Installing Red Hat Enterprise Linux 6

Insert the RHEL 6 DVD into the appropriate drive and power on the system. If the system tries to boot from the hard disk drive you will need to enter the BIOS set up for your computer and change the boot order so that it boots from the DVD drive first. Once the system has booted you will be presented with the following screen:

Figure 1 RHEL-6 Booting from DVD

To install using the graphical installer, simply select the first installation option and press the <Enter> key. If you encounter problems with the display when the graphical installer begins, reboot and try again with the basic video driver option. Note also that if the host system has insufficient memory or a graphics card is not detected, the installer will run in text mode.

Options are also available to boot from the current operating system on the local drive (if one is installed), test the system memory, or rescue an installed RHEL 6 system. The last option alone is reason enough to keep the installation DVD in a safe place in case you need to perform a rescue at some future date.

The RHEL installer will then provide the option to test the installation media for errors. Use the arrow keys to navigate between the options and make a selection with the <Enter> key. After a

short delay the first screen of the graphical installer will appear. Navigate through the next few pages to configure your preferred language, keyboard type and storage devices (unless you plan to use a Storage Area Network device, the Basic option is recommended). If the installer detects that the target disk needs to be initialized a dialog will appear seeking confirmation.

Timezone and the Root Password

Subsequent screens will request information about Timezone and the root password of the system.

On the Timezone screen, make a selection corresponding to your geographical location. The option is also provided to use UTC which automatically adjusts the time to account for daylight savings time. If the computer on which RHEL is being installed also runs another operating system which already uses UTC (such as Windows), leave this option unselected.

On the next screen, enter a password for the root account on the system. The root, or super-user account is a special user that has administrative privileges on the system. Whilst you will generally use your own account to log into the system, you will need to gain root privileges in order to configure the system and to perform other administrative tasks.

The installer will subsequently move on to the disk partitioning screen.

Partitioning a Disk for RHEL 6

When you reach the disk partitioning phase of the installation, the installer will present a screen similar to the one illustrated in the following figure:

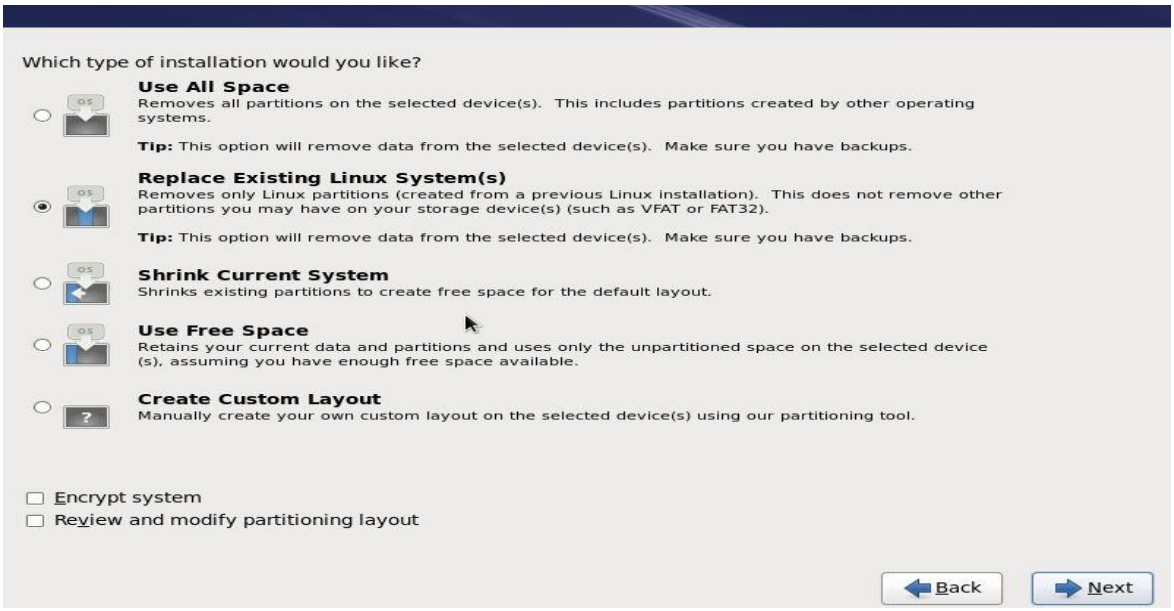


Figure 2 Partitioning while installing

A number of options are provided for allocating space for the installation of RHEL 6:

- **Use All Space** - The entire disk drive will be assigned to the RHEL 6 operating system installation. Any pre-existing partitions, together with any existing operating systems and associated data files contained therein will be deleted to make room for RHEL. This option should only be used if you are absolutely sure you no longer need anything that is currently stored on that disk, or have already backed up all user files.

- **Replace existing Linux System(s)** - If the drive was previously configured to support a Windows/Linux dual boot environment or was devoted entirely to another Linuxinstallation, this option may be selected to instruct the installer to delete the pre-existing Linux partition and replace it with RHEL 6. Once again, it is important to backup any user data that may still be needed.

- **Shrink Current system** - Allows an existing partition to be reduced in size to make room on the drive for the RHEL 6 installation. More details on this option are provided in a later chapter entitled Installing RHEL 6 with Windows in Dual Boot Environment.

- **Use Free Space** - If the current partitions on the drive do not take up the entire disk space available, any unallocated space may be assigned to the RHEL installation using this option.

- **Create Custom Layout** - When selected, this option displays the disk partitioning tool allowing each partition on the disk to be manually configured. Unless you have experience with low level disk partitioning this option is not recommended.

On this screen, make a selection based on your requirements. If, for example, the entire disk is to be used for RHEL 6, select the Use All Space option. In order to implement a dual boot configuration, refer to Installing RHEL 6 with Windows in Dual Boot Environment.

For the purposes of this chapter we are assuming the entire disk is available to accommodate the RHEL 6 installation so select the Use All Space option.

Beneath the partition menu is the option to encrypt the system. The choice of whether to use encryption will depend on the purpose for which the system is being used, its physical location and type of data it is going to store. Keep in mind that as with any form of encryption there are performance overheads associated with selecting this option.

Having made the partitioning selection click Next to begin the partitioning process.

Package Selection

Linux is a modular operating system in that it provides a basic operating system kernel and infrastructure upon which a range of different packages may be installed depending on your specific requirements for the system. If, for example, you plan on using the system as a web server you would need to install the Apache web server package.

At this point in the installation the installer needs us to decide which packages should be installed along with the base operating system and displays the screen shown in the following figure:

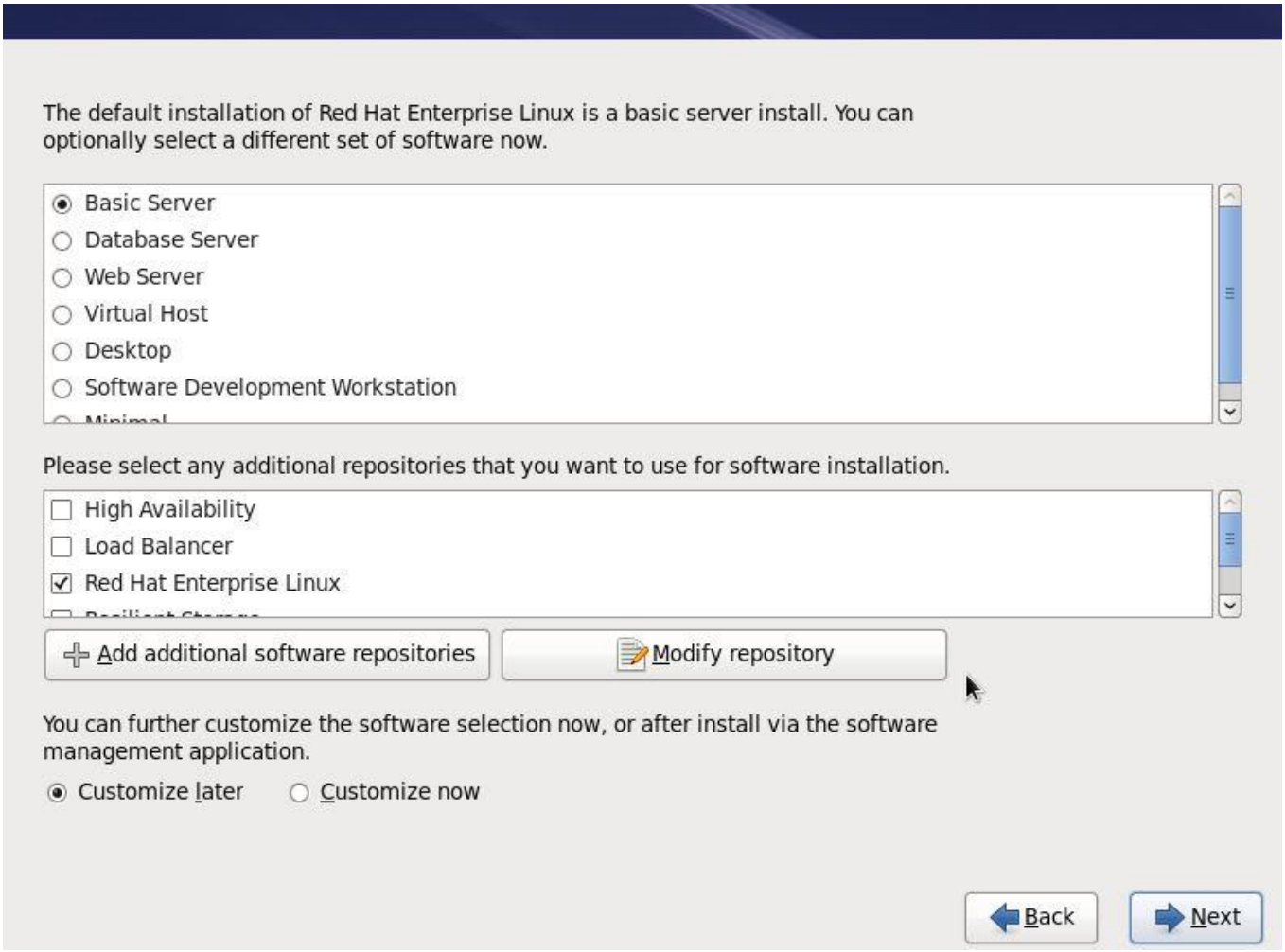


Figure 3 customize menu

This screen allows you to make general choices about the type of functions you need the system to perform. Ideally, you should select the option that most closely resembles the intended purpose of the system.

To view or modify the specific package selections, make sure that the Customize Now option is selected before proceeding. You will then be provided a complete overview of which packages are selected for installation and which are not together with the ability to make changes to these selections. Don't worry too much about getting this exactly right at this stage. Packages can be added and removed at any time after the installation is complete by selecting the desktop System -> Administration -> Add/Remove Software menu option.

The Physical Installation

Having made the appropriate package selections, clicking next will initiate the installation process. During the installation process, the installer will provide a running commentary of the selected packages as they are installed and a progress bar. If you are using the DVD the installation will complete without further interaction. Once the installation process is complete a screen will appear containing a button to reboot the system. Remove the installation media and click the button.

Final Configuration Steps

After the system has started for the first time, the RHEL 6 Setup Agent will appear with a welcome message. Click on the Forward button to display licensing terms and conditions and select the option to accept the terms of the agreement (assuming of course that you do agree to them).

On the Setup Software Updates screen, register the system with the Red Hat Network (RHN). This enables additional packages to be installed from the RHN repositories and also allows the system to receive software updates. In order to register the system, enter your RHN user login and password when prompted. If you would prefer to register the system at a later time, do so simply by running the rhn_register command from a terminal window or selecting the System -> Administration -> RHN Registration menu option.

Choose whether or not to enable Kdump support and then work through the remaining screens to create a user account for yourself and verify the date and time. If you would like the date and time of your RHEL system to be synchronized with an external Network Time Protocol server, select the Synchronize date and time over network option before proceeding.

Having worked through all the set up pages, click Finish to exit the setup agent and log in using your newly created account credentials.

Linux File System

The **ext4** or **fourth extended filesystem** is a journaling file system for Linux, developed as the successor to ext3.

It was born as a series of backward compatible extensions to ext3, many of them originally developed by Cluster File Systems for the Lustre file system between 2003 and 2006, meant to extend storage limits and add other performance improvements. However, other Linux kernel developers opposed accepting extensions to ext3 for stability reasons, and proposed to fork the source code of ext3, rename it as ext4, and do all the development there, without affecting the current ext3 users. This proposal was accepted, and on 28 June 2006, Theodore Ts'o, the ext3 maintainer, announced the new plan of development for ext4.

A preliminary development version of ext4 was included in version 2.6.19 of the Linux kernel. On 11 October 2008, the patches that mark ext4 as stable code were merged in the Linux 2.6.28 source code repositories, denoting the end of the development phase and recommending ext4 adoption. Kernel 2.6.28, containing the ext4 filesystem, was finally released on 25 December 2008. On 15 January 2010, Google announced that it would upgrade its storage infrastructure from ext2 to ext4. On December 14, 2010, they also announced they would use ext4, instead of YAFFS, on Android 2.3.

Features

Large file system

The ext4 filesystem can support volumes with sizes up to 1 exbibyte (EiB) and files with sizes up to 16 tebibytes (TiB). The current e2fsprogs can only handle a filesystem of 16 TiB, but support for larger filesystems is under development.

Extents

Extents replace the traditional block mapping scheme used by ext2/3 filesystems. An extent is a range of contiguous physical blocks, improving large file performance and reducing fragmentation. A single extent in ext4 can map up to 128 MiB of contiguous space with a

4 KiB block size. There can be 4 extents stored in the inode. When there are more than 4 extents to a file, the rest of the extents are indexed in an Htree.

Backward compatibility

The ext4 filesystem is backward compatible with ext3 and ext2, making it possible to mount ext3 and ext2 filesystems as ext4. This will slightly improve performance, because certain new features of ext4 can also be used with ext3 and ext2, such as the new block allocation algorithm.

The ext3 file system is partially forward compatible with ext4, that is, an ext4 filesystem can be mounted as an ext3 partition (using "ext3" as the filesystem type when mounting). However, if the ext4 partition uses extents (a major new feature of ext4), then the ability to mount the file system as ext3 is lost.

Persistent pre-allocation

The ext4 filesystem allows for pre-allocation of on-disk space for a file. The current method for this on most file systems is to write the file full of 0s to reserve the space when the file is created. This method is no longer required for ext4; instead, a new fallocate() system call was added to the Linux kernel for use by filesystems, including ext4 and XFS, that have this capability. The space allocated for files such as these would be guaranteed and would likely be contiguous. This has applications for media streaming and databases.

Delayed allocation

Ext4 uses a filesystem performance technique called allocate-on-flush, also known as *delayed allocation*. It consists of delaying block allocation until the data is going to be written to the disk, unlike some other file systems, which may allocate the necessary blocks before that step. This improves performance and reduces fragmentation by improving block allocation decisions based on the actual file size.

Break 32,000 subdirectory limit

In ext3 the number of subdirectories that a directory can contain is limited to 32,000. This limit has been raised to 64,000 in ext4, and with the "dir_nlink" feature it can go beyond this (although it will stop increasing the link count on the parent). To allow for continued performance given the possibility of much larger directories, Htree indexes (a specialized version of a B-tree) are turned on by default in ext4. This feature is implemented in Linux kernel 2.6.23. Htree is also available in ext3 when the dir_index feature is enabled.

Journal checksumming

Ext4 uses checksums in the journal to improve reliability, since the journal is one of the most used files of the disk. This feature has a side benefit; it can safely avoid a disk I/O wait during the journaling process, improving performance slightly. The technique of journal checksumming was inspired by a research paper from the University of Wisconsin titled *IRON File Systems* (specifically, section 6, called "transaction checksums") with modifications within the implementation of compound transactions performed by the IRON file system (originally proposed by Sam Naghshineh in the RedHat summit).

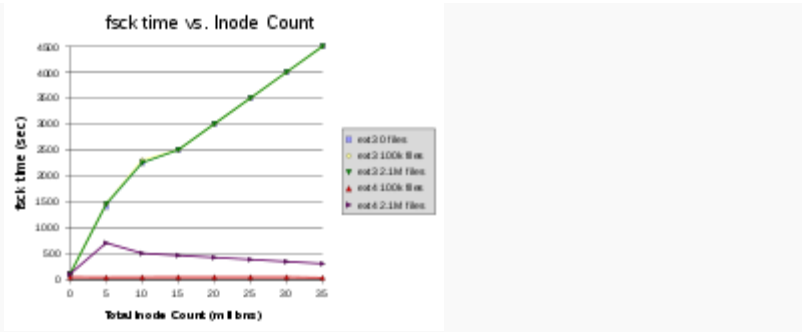


Figure 4 fsck time/Inode Count(ext3 vs. ext4)

Faster file system checking

In ext4, unallocated block groups and sections of the inode table are marked as such. This enables e2fsck to skip them entirely on a check and greatly reduces the time it takes to check a file system of the size ext4 is built to support. This feature is implemented in version 2.6.24 of the Linux kernel.

Multiblock allocator

When a file is being appended to, ext3 calls the block allocator once for each block individually; with multiple concurrent writers, files can easily becomefragmented on disk. With delayed allocation, however, ext4 buffers up a larger amount of data, and then allocates a group of blocks in a batch. This means that the allocator has more information about what's being written and can make better choices for allocating files contiguously on disk. The multiblock allocator is used when delayed allocation is enabled for a file system, or when files are opened in O_DIRECT mode. This feature does not affect the disk format.

Improved timestamps

As computers become faster in general and as Linux becomes used more for mission-critical applications, the granularity of second-based timestamps becomes insufficient. To solve this, ext4 provides timestamps measured in nanoseconds. In addition, 2 bits of the expanded timestamp field are added to the most significant bits of the seconds field of the timestamps to defer the year 2038 problem for an additional 204 years.

Ext4 also adds support for date-created timestamps. But, as Theodore Ts'o points out, while it is easy to add an extra creation-date field in the inode (thus technically enabling support for date-created timestamps in ext4), it is more difficult to modify or add the necessary system calls, like stat() (which would probably require a new version), and the various libraries that depend on them (like glibc). These changes would require coordination of many projects. So, even if ext4 developers implement initial support for creation-date timestamps, this feature will not be available to user programs for now.

Delayed allocation and potential data loss

Because delayed allocation changes the behavior that programmers have been relying on with ext3, the feature poses some additional risk of data loss in cases where the system crashes or loses power before all of the data has been written to disk. Due to this, ext4 in kernel versions 2.6.30 and later automatically detects these cases and reverts to the old behavior.

The typical scenario in which this might occur is a program replacing the contents of a file without forcing a write to the disk with fsync. There are two common ways of replacing the contents of a file on Unix systems:

```
fd=open("file", O_TRUNC); write(fd, data); close(fd);
```

In this case, an existing file is truncated at the time of open (due to O_TRUNC flag), then new data is written out. Since the write can take some time, there is an opportunity of losing contents even with ext3, but usually very small. However, because ext4 can delay allocating file data for a long time, this opportunity is much greater.
There are several problems with this approach:

If the write does not succeed (which may be due to error conditions in the writing program, or due to external conditions such as a full disk), then both the original version *and* the new version of the file will be lost, and the file may be corrupted because only a part of it has been written.

If other processes access the file while it is being written, they see a corrupted version.

If other processes have the file open and do not expect its contents to change, those processes may crash. One notable example is a shared library file which is mapped into running programs.
Because of these issues, often the following idiom is preferred over the above one:

```
fd=open("file.new"); write(fd, data); close(fd); rename("file.new", "file");
```

A new temporary file ("file.new") is created, which initially contains the new contents. Then the new file is renamed over the old one. Replacing files by the "rename" call is guaranteed to be atomic by POSIX standards – i.e. either the old file remains, or it's overwritten with the new one. Because the ext3 default "ordered" journalling mode guarantees file data is written out on disk before metadata, this technique guarantees that either the old or the new file contents will persist on disk. ext4's delayed allocation breaks this expectation, because the file write can be delayed for a long time, and the rename is usually carried out before new file *contents* reach the disk.

Using fsync more often to reduce the risk for ext4 could lead to performance penalties on ext3 filesystems mounted with the `data=ordered` flag (the default on most Linux distributions). Given that both file systems will be in use for some time, this complicates matters for end-user application

developers. In response, ext4 in Linux kernels 2.6.30 and newer detect the occurrence of these common cases and force the files to be allocated immediately. For a small cost in performance, this provides semantics similar to ext3 ordered mode and increases the chance that either version of the file will survive the crash. This new behavior is enabled by default, but can be disabled with the "noauto_da_alloc" mount option.

The new patches have become part of the mainline kernel 2.6.30, but various distributions chose to backport them to 2.6.28 or 2.6.29. For instance Ubuntu made them part of the 2.6.28 kernel in version 9.04 ("Jaunty Jackalope").

These patches don't completely prevent potential data loss or help at all with new files. No other filesystem is perfect in terms of data loss either, although the probability of data loss is lower on ext3. The only way to be safe is to write and use software that does fsync when it needs to. Performance problems can be minimized by limiting crucial disk writes that need fsync to occur less frequently.

Getting Started On Your Linux Desktop

If you have logged in from the graphical login screen, the graphical desktop will be started automatically for you. The graphical desktop presents a Graphical User Interface (GUI) for the user to interact with the system and run applications. If you have used the text-based screen login, you will have to start the graphical desktop manually by entering the command startx followed by the ENTER key.

The graphical desktop that we will be using throughout most of this guide is called the GNOME Desktop. There is another desktop environment in popular use on Linux systems - the KDE Desktop.

In order to start using your system you will usually have to perform what is known as a user login. This procedure is necessary to identify yourself to the system. The system allows multiple users to use it concurrently and so it has to be able to identify a user in order to grant them the necessary privileges and rights to use the system and applications. Each user upon successful login will be assigned to his home directory (folder).

Depending on how you have set up your system, you will either have a graphical login screen or a text-based login prompt for you to perform the login process.

To login, enter the username followed by the ENTER key and when the password prompt appears, enter the password followed by the ENTER key.

Some systems may have been set up so that upon power-on, a default user is logged in automatically. If this is so, then you will not be presented with a login screen

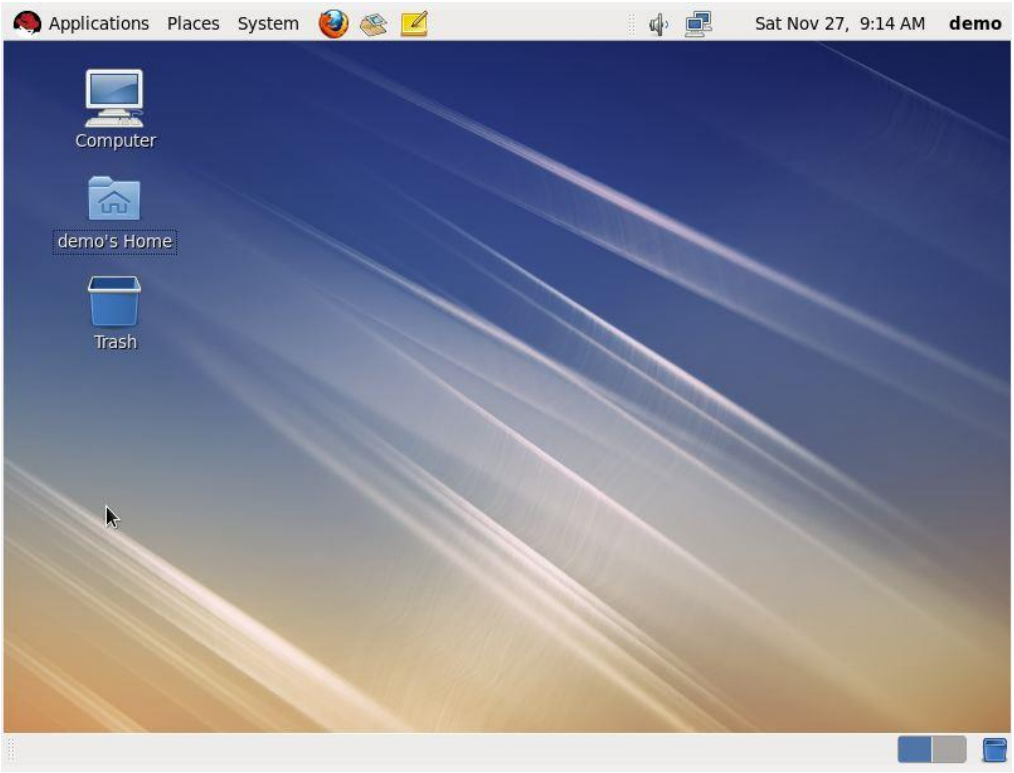


Figure 5 The Desktop

The Desktop Background

The desktop background constitutes the swirling graphics in the above figure. If you don't like the default desktop background you can change it to either a selection of pre-installed images, or to a digital picture or image of your choosing.

The desktop also provides a location for folders and applications for which quick access is required. A link to the user's home folder is added to the desktop by default, as are the Computer and Trash icons. The former provides access to storage devices and the network. The latter contains any files that have been placed in the trash can.

Clicking with the right mouse button on any area of the desktop background presents the following popup menu containing a number of useful options for changing or adding items to the RHEL GNOME desktop:

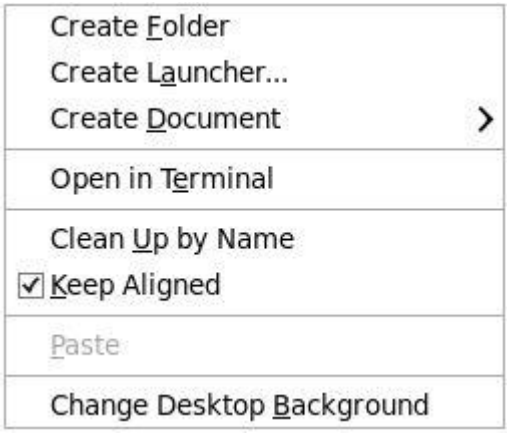


Figure 6 Right click dialog box

Create Folder - Creates a new folder on the desktop. A new folder icon appears on the desktop with a field provided for the user to enter a folder name. The folder is physically located in the Desktop folder of the user's home directory.

Create Launcher - Allows an icon to be placed on the desktop which, when double clicked, launches a specified application. Another way to add application launchers to the desktop is to find them in the menu systems at the top of the screen, right click on them and select Add this launcher to desktop.

Create Document - Creates a new document on the desktop (the resulting file is physically located in the Desktop folder of the current user's home directory).

Open in Terminal – Launches a terminal window containing a shell command prompt where commands may be entered and executed. The current working directory of the shell within the terminal is set to the Desktop subdirectory of the user's home folder.

Clean Up by Name - Sorts and organizes the desktop icons in alphabetical order.

Keep Aligned - A toggle setting which dictates whether icons should be neatly aligned on the desktop or allowed to be placed arbitrarily around the desktop.

Change Desktop Background - Allows a different background image to be specified.

Right clicking on an icon produces the following menu allowing tasks to be performed on the selected item:

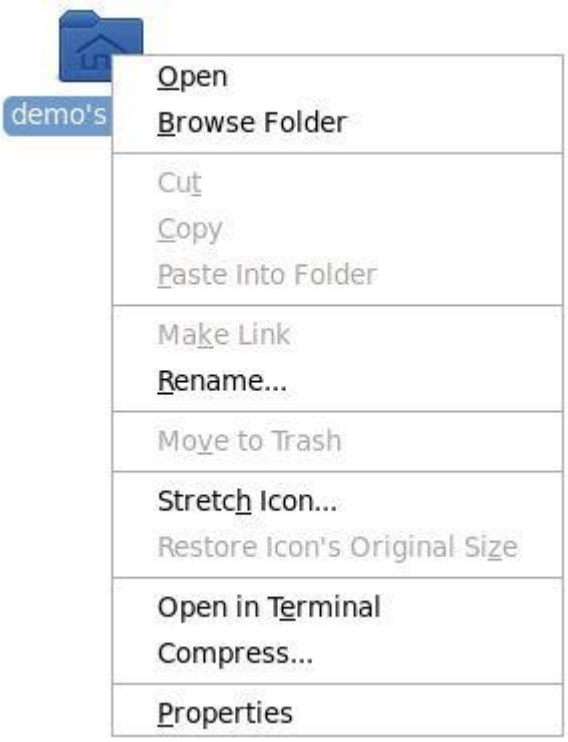


Figure 7 Right click dialog box on a folder

The Desktop Panels

The bars across the top and bottom of the desktop are called panels. Both the content and position of these panels can be configured

By default the top panel appears as follows:

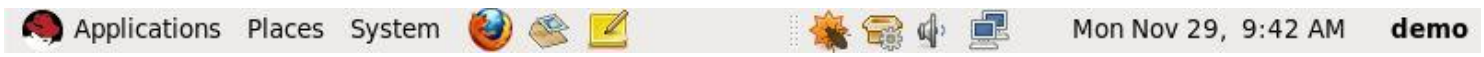


Figure 8 Desktop Panel

The Application menu provides access to the applications installed in the system. The Places menu provides a list of locations which, when selected, are opened in file browser windows. Available locations include devices (such as disk or CD/DVD Drives), the current user's home folder and

other systems on the network. The System menu provides options for configuring the system and desktop environment (including factors such as desktop theme and screen resolution).

The icons next to the system menu provide quick access to common applications and tools. The default icons will depend on the packages installed (such as Firefox, the Evolution email client and the OpenOffice office productivity tools) but may be configured to add additional launchers for tools or applications you find you use regularly.

The right hand side of the panel includes the current time, a volume control and a network status indicator. To establish a network connection, access a variety of configuration options by clicking with the left and right mouse buttons on the icon.

Also present on laptop based installations is an icon indicating current power status. A variety of other status icons will appear in this section from time to time. Typically hovering over or clicking on the status icon will display a brief description of the icon's purpose. For example, two additional icons are present in the above figure indicating that updates are available for the system (represented by the bug icon) and that the package manager has messages that are need to be read (the cardboard box icon).

As you perform administrative tasks on the system that require authentication using the root password, you will see a set of keys appear in the top panel. This indicates that elevated root (administrative) privileges have been assigned to the desktop session allowing sequential tasks to be performed without repeatedly entering the root password. To de-authorize these elevated privileges simply click on the icon.

The bottom desktop panel contains three items by default but may similarly be configured as By default the panel appears as follows:

Figure 9 Bottom Desktop Panel

The area containing squares controls the currently displayed virtual workspace. The RHEL GNOME desktop allows multiple workspaces to be active at any one time with different applications and windows visible on each workspace. To switch to a different desktop, simply click on one of the squares in the bottom toolbar. If your mouse has a scroll wheel, click on a workspace in the panel and then scroll back and forth through the workspaces. By default two workspaces are configured, though this number can be increased or decreased. To move an application window from one workspace to another, right click on the window toolbar and select the destination workspace from the Move to Another Workspace sub-menu.

When applications are running on the desktop, a button will appear for each application located on the currently selected workspace. If the application's windows are currently visible on the desktop, clicking the corresponding button in the panel will minimize the windows so that they are no longer visible. Clicking the button again will re-display the windows once again. The final item on the bottom panel is the trash can. Items may be dragged and dropped into the trash can. Right click on the icon to access options to open or empty the trash can.

Now that we have covered the basics of the desktop we can begin looking at customizing the desktop. The first step involves changing the RHEL GNOME desktop background.

The RHEL 6 GNOME Slideshow Background

The default background for Red Hat Enterprise Linux 6 is actually made of multiple images that are shown in sequence in the form of a slide show. The images are located in the/usr/share/background directory and the order in which the images appear and the duration for which each is displayed is configured in the /usr/share/background/default.xml file. Use this file as a template to create your own slide shows if desired.

Changing the RHEL 6 GNOME Desktop Background

There are three methods for changing the background, either by clicking with the right hand mouse button on the current desktop background and selecting Change Desktop Background, selecting the Administration -> Preferences -> Appearance menu option and selecting the Background tab,

or clicking with the right hand-mouse button on an image in the web browser and selecting Set As Desktop Background from the resulting popup menu. The latter method allows you find images from one of the many web sites that provide them and quickly make one of them your background.

Regardless of which method you use, the Desktop Background Preferences dialog will appear as illustrated in the following figure:

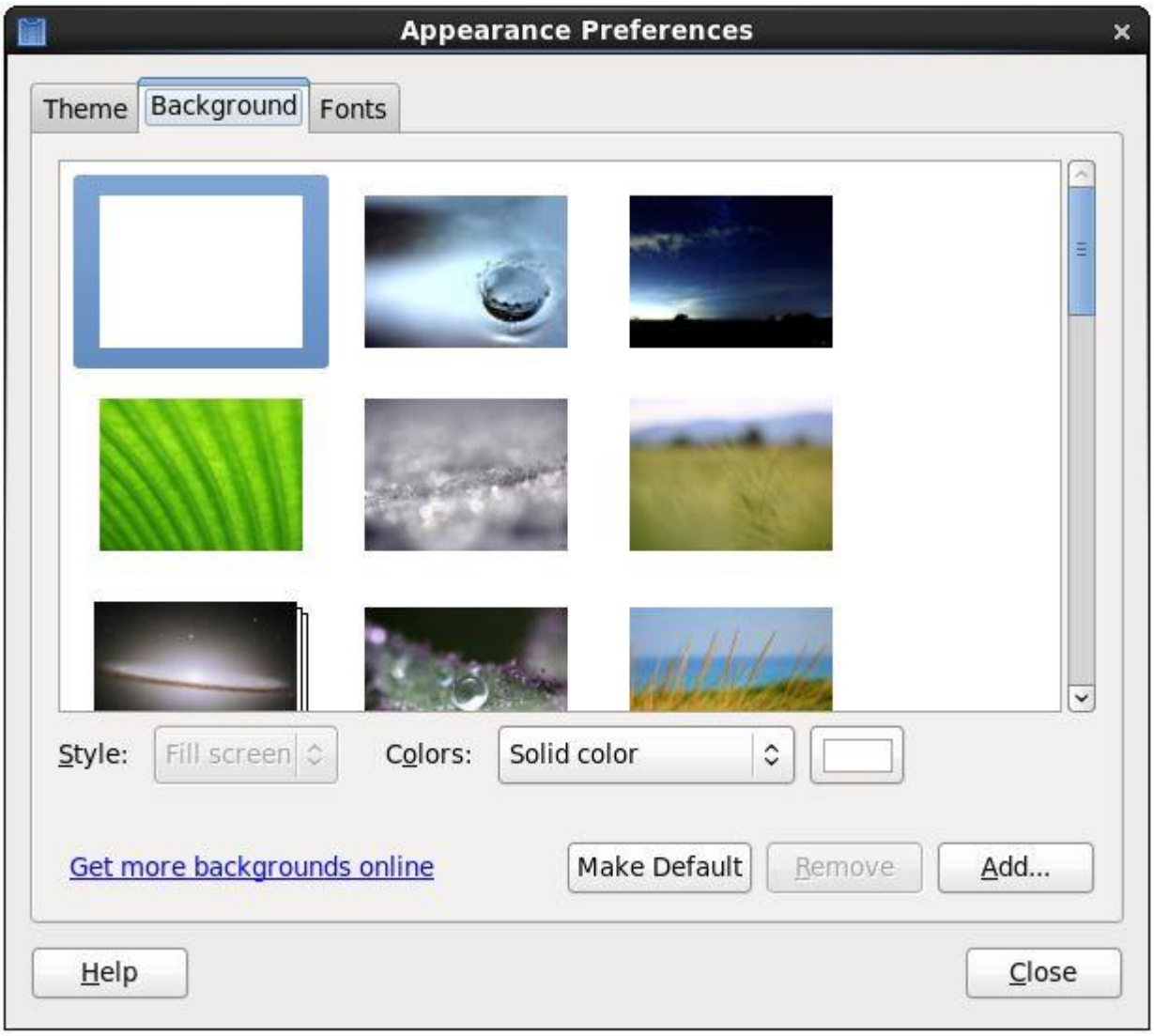


Figure 10 Background Selection

Selecting a Background from the Wallpaper List

The background preferences dialog provides a list of pre-installed wallpaper background images from which to choose. Those images that are shown stacked are slideshows.

As each example is selected the background will automatically change to reflect the new selection. Additional images may be added to the gallery by clicking on the Add... button and navigating to the image file. The Style menu provides a range of options for handling images that do not precisely match the screen dimensions. Choices include zooming, tiling, scaling and fitting.

Creating a Solid or Graded Background

If you prefer a background consisting of a solid color simply select the blank option option located at the top of the gallery list, make sure that Solid Color is selected next to Colors and click on the color box to select a color.

The background can also be created by specifying a color gradient transition from one color to another. The gradient may be horizontal or vertical. With the blank wallpaper option selected

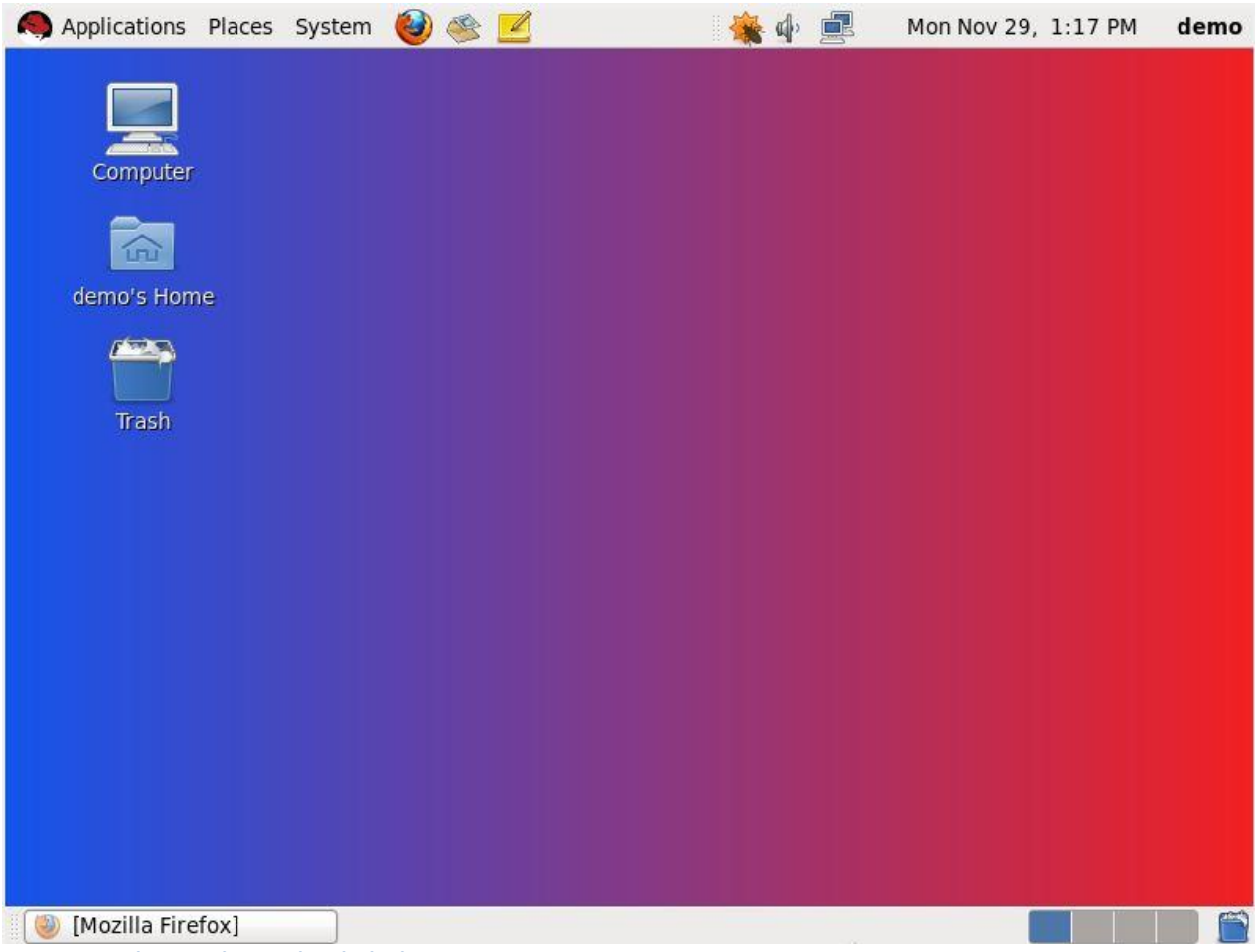


Figure 11 Desktop Background with shades

change the Desktop Color menu to Vertical or Horizontal Gradient. Using the two color boxes located to the right of the menu, specify the start and end colors. The desktop will then create a background image based on these settings. For example, the following figure shows a horizontal gradient from blue to red:

Specifying a Background Image

As mentioned previously, there are two ways to specify an image file for the RHEL desktop background. One way is to open an image (for example in a web browser) clicking with the right mouse button and selecting Set as Desktop Background from the resulting menu. This will cause the following dialog to appear asking how the image is to be positioned on the desktop:

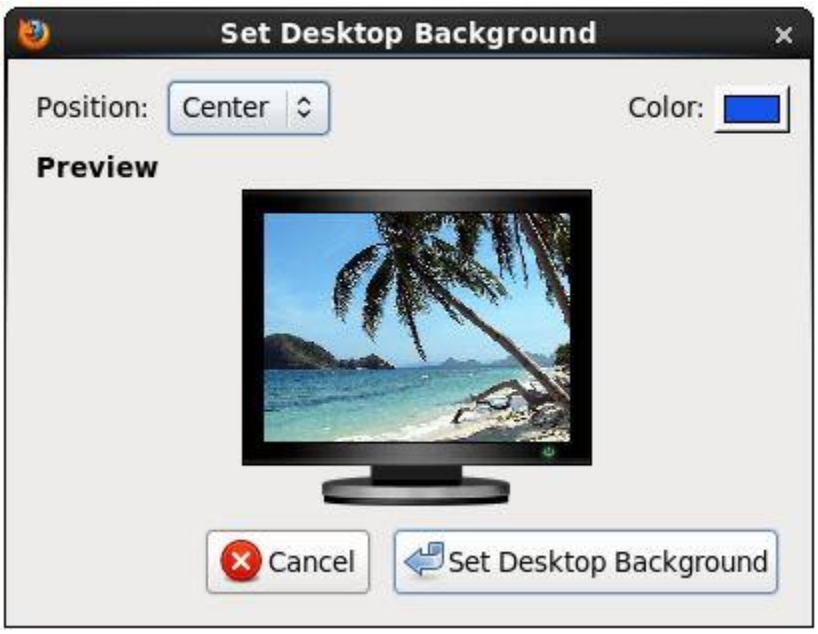


Figure 12 Set Desktop Background

From this dialog select whether the image should be tiled, stretched or centered. If space is going to be visible around the edges of the image be sure to select a suitable color for the border area using the color box. Click on Set Desktop Background to apply the settings

Configuring the RHEL 6 GNOME Desktop Panels

The Red Hat Enterprise Linux 6 GNOME Desktop Panels are one of the most useful aspects of the desktop in terms of providing information, ease of use and convenience to the user. As with just about every other aspect of the GNOME desktop, these panels are accompanied by extensive configuration options. In this chapter we will look at each of these configuration options in detail.

What are Desktop Panels?

On a newly installed RHEL 6 system there are two panels configured by default. These appear at the top and bottom of the desktop respectively. The top panel contains the main desktop menus and a number of status areas with information such as the system time and audio settings:

Figure 13 Desktop Panels

The bottom panel contains a button to hide and show all windows on the screen, the trash can, access to the various virtual workspaces and also has a button representing each application currently running in the current workspace.

Figure 14 Desktop Panels

These panels can be configured in a variety of ways including:

Adding additional panels to the desktop
Moving the panels to different locations on the desktop
Adding mini applications (such as stock ticker, system monitor or weather applet) to a panel
Size, background, visibility and transparency of each panel

In the remainder of this chapter we will look at each of these customization options in detail.

Changing the Position of a Panel

A panel can be placed in one of four different locations on the desktop: the top, bottom, far left or far right of the desktop. The location of the panel can be adjusted using the Properties dialog of the panel. This dialog is accessed by clicking with the right hand mouse button on any blank area of the panel and selecting Properties from the popup menu:

Figure 15 Panel Properties

From the Panel Properties dialog, change the Orientation setting to the desired value to position the panel as required.

Adding and Deleting Desktop Panels

New panels may be added to the desktop by right clicking on any existing panel and selecting New Panel from the resulting menu. The new panel will be placed in an orientation where no panel currently exists. If you already have panels on all four sides of your desktop, the new panel will

be placed next to a pre-existing panel. It is possible to fill your entire desktop with panels if you wish, though it is hard to imagine a situation where this would be desirable.

An existing panel may be removed from the desktop by right clicking on the panel and selecting Delete This Panel from the popup menu.

Changing the Appearance of a Desktop Panel

A number of configuration options are available for changing the appearance of each desktop panel. These changes are implemented using the Panel Properties dialog (accessed by right clicking on the panel to be changed and selecting Properties from the popup menu).

The Size property controls the height (when the panel is in a horizontal orientation) or width (when in vertical orientation). It is not possible to reduce the size to the extent where items will no longer be visible (the minimum value is typically around 23 pixels).

A different color or even an image may be specified for the panel background. These settings are available from the Background page of the panel properties dialog:

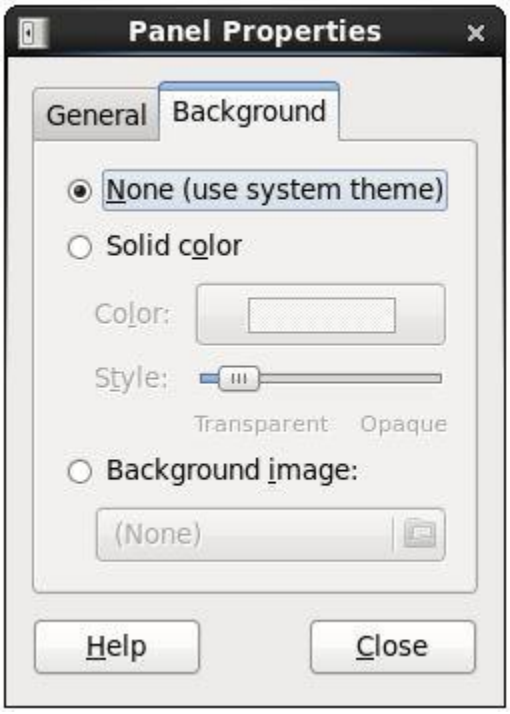


Figure 16 Panel Background properties

The level of transparency can be specified by selecting the Solid color option and adjusting the Style slider.

The panel may be further configured so that it remains hidden until the mouse pointer moves to the side of the desktop where the panel is located. To activate this feature, select the Autohide option on the General panel of the properties window.

Finally, if you do not wish to have the panel occupy the full width or height of the desktop (depending on the orientation of the selected panel), unset the Expand option. The panel will then only appear as wide or as tall as it needs to be to accommodate the items it is configured to display.

Adding Items to a RHEL 6 GNOME Desktop Panel

When RHEL 6 is first installed, a number of items are added by default to the two panels. Additional items, however, may easily be added to any panel. There are a couple of ways to add items to the panel.

If there is an application you frequently launch from the Applications menu there is a quick way to add a launch icon to the panel. Simply open the Applications menu and navigate to the menu option for the desired application. Rather than clicking on this menu item with the left mouse button as you normally would, click instead with the right mouse button and select the Add this launcher to panel option. An icon representing the application will subsequently appear on the panel which, when clicked, will launch the corresponding application.

Alternatively, launch the Add to Panel dialog by right clicking on a blank area of the desired panel and selecting Add to Panel… from the resulting menu. In the resulting dialog select Application Launcher... and click Forward. The dialog will list each menu item and all sub-menu items. If, for example, you wanted to be able to launch the Terminal window from the panel simply click on the arrow next to System Tools in the Add to Panel dialog to unfold the list of accessory applications. Scroll down the list of accessories, select Terminal and click on the Add button:

Figure 17 Adding Item to panels

In addition to applications from the menu, it is also possible to add a range of items specifically designed to appear on panels. To add such items to a panel click with the right mouse button over the panel to be configured and select the Add to Panel menu option from the resulting popup menu. The Add to Panel dialog will then appear as follows:

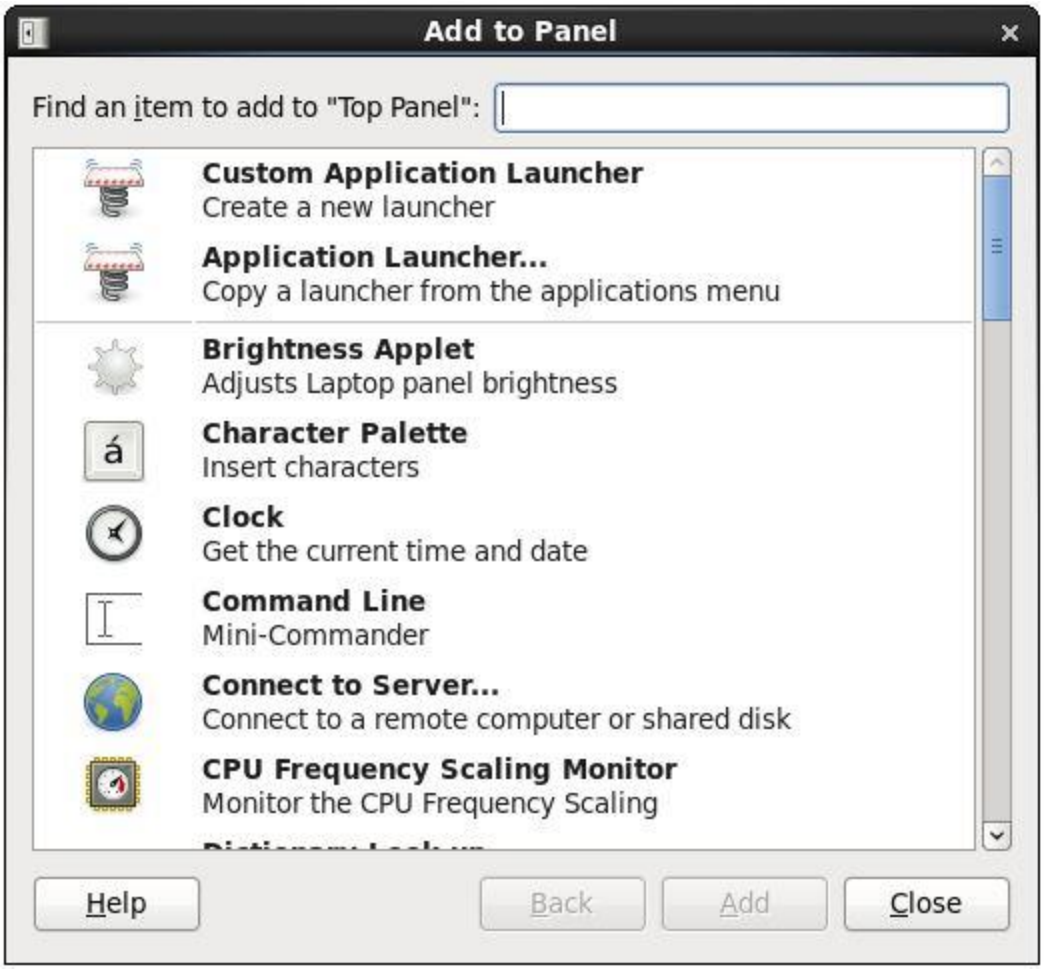


Figure 18 Adding Item to panels

To add an item to the panel, simply scroll through the list provided, select the required item and press the Add button. The following RHEL 6 desktop panel includes the System Monitor and Weather Report items:

Figure 19 Preview after adding items to panel

To edit the properties or preferences for a panel item, move the mouse pointer over the item in the panel and click the right mouse button. Select either Properties or Preferences from the menu and the appropriate dialog will appear. For example, the Preferences panel for the Weather Report item

allows various settings to be changed, such as the geographical location and units for displaying temperature (Celsius or Fahrenheit). The system monitor can similarly be configured to also display metrics such as system load, memory and network usage.

To delete an item from a panel right click with the mouse on the item and select Remove from Panel.

To move an item to a different location in a panel right click with the mouse on the item and select Move. Then use the mouse to drag and drop the item to a new location. Alternatively, click with either the middle mouse button, or the left and right mouse buttons simultaneously and drag and drop the panel item to the new location.

To add an application not currently available from the desktop menus simply select the Custom Application Launcher, click Add and specify the path to the application executable and provide a name. Click on the No icon button to select an icon to represent the item on the panel.

Adding Menus to a Panel

Any of the sub-menus in the Applications, Places or System menus may be added as icons to the main panel. To add, for example, an icon to access the Application -> Accessories menu to the panel, open the Applications menu and right click on any item in the Accessories menu item. From the popup menu, select the Entire menu... followed by Add this as menu to panel. An icon will then appear on the panel which, when clicked, drops down the Accessories menu:

Figure 20 Preview of item added to panel

Alternatively, the menu may be added to the panel as a Drawer by selecting Add this as drawer to panel. A drawer is similar to the menu with the exception that only the icons are displayed and an Up arrow

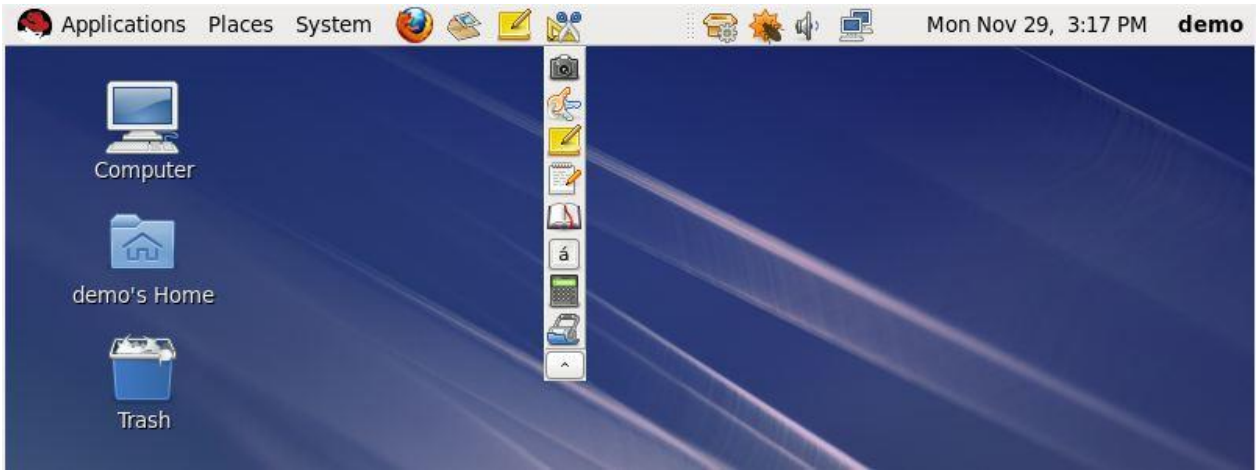


Figure 21

Changing the Number of RHEL Desktop Workspaces

The RHEL GNOME desktop supports multiple desktop workspaces (these are essentially individual screens which the user can switch between to avoid having to have all applications cluttered onto one screen). Switching between workspaces is achieved by clicking on the workspaces in the bottom desktop panel. The image below shows a section of the panel from a desktop with four workspaces:

Figure 22 Workspaces

By default, RHEL 6 configures two workspaces. To increase the number of available workspaces, right click with the mouse on the workspace control (as shown above) in the panel and selectPreferences. Simply increase the Number of workspaces value to the desired number. The

name of each workspace may be changed by double clicking on the default name in the list (Workspace 1, Workspace 2 etc) and typing in a new name.

Linux Directory Structure

/ -

This is referred to as the root directory (not to be confused with the /root directory, which is the root user's directory!). It's the baseline for the directory structure, which is important when navigating from one directory tree to another.

/bin -

Contains essential programs for the operating system in executable form.

/boot -

This directory, as the name suggests, contains the boot information for Linux, including the kernel.

/dev -

In Linux, devices are treated in the same way as files, so they can be read from and written to in the same way. So, when a device is attached, it will show up in this folder. Bear in mind that this includes every device in the system, including internal motherboard devices and more.

/etc -

The default configuration files for Linux applications are stored in this directory. These are text files can be easily edited, but bear in mind that they can sometimes be overridden by copies of the files elsewhere on the system.

/home -

This is where the computer's users files are kept, so in a way it is Linux's equivalent to Windows' "My Documents". Each user gets their own named directory, and security permissions can be set so that users can view and edit, view only, or if required not even see the contents of other users' home directories.

/lib -

This directory contains shared libraries for use by applications running on Linux, similar to Windows' DLL files.

/lost+found -

This is Linux's rescue directory, so that in the event of a crash or other serious event files are stashed here to enable recovery.

/mnt -

In Linux, every storage device is treated as just another directory. This includes floppy disks, hard drives, CD/DVD ROMs and USB card devices. Since it is very unlikely to ever concern you with a dedicated server it is not covered here, but just know that this is the directory in which storage devices are "mounted."

/proc -

This "virtual" directory contains a lot of fluid data about the status of the kernel and its running environment. Since Linux treats everything as files, you can view this data using text viewing software, and though even editing these files is sometimes possible, doing so is not advised unless you really know what you're doing.

/root -

Rather than being part of the /home directory, the superuser (or root user)'s directory is placed here. Remember that this is not the same thing as the root directory of the system (/).

/sbin -

This is where system administration software is stored. Unlike applications in the /bin folder, the root user is usually the only user who can run these.

/tmp -

Applications store their temporary files in this directory.

/usr -

This directory is where users' applications are stored, including the executables, and also sometimes the source code, along with any images and documentation.

/var -

Similar to /proc, this directory contains data concerning the status of running application, including many log files. This is worth knowing, because these can be viewed in the event of a system error to help in diagnosing the problem.

The Linux Console

In the early days of Linux, when you booted up your system you would see a login prompt on your monitor, and that's all. As mentioned earlier, this is called the Linux console. It was the only place you could enter commands for the system.

With modern Linux systems, when the Linux system starts it automatically creates several *virtual consoles*. A virtual console is a terminal session that runs in memory on the Linux system. Instead of having six dumb terminals connected to the PC, the Linux system starts six (or sometimes even more) virtual consoles that you can access from the single PC keyboard and monitor.

In most Linux distributions, you can access the virtual consoles using a simple keystroke combination. Usually you must hold down the Ctl +Alt key combination, then press a function key

(F2 through F6) for the virtual console you want to use. Function key F2 produces virtual console 1, key F3 produces virtual console 2, and so on.

The first five virtual consoles use a full-screen text terminal emulator to display a text login screen,

After logging in with your user ID and password, you are taken to the Linux bash shell CLI. In the Linux console you do not have the ability to run any graphical programs. You can only use text programs to display on the Linux text consoles.

After logging in to a virtual console, you can keep it active and switch to another virtual console without losing your active session. You can switch between all of the virtual consoles, with multiple active sessions running.

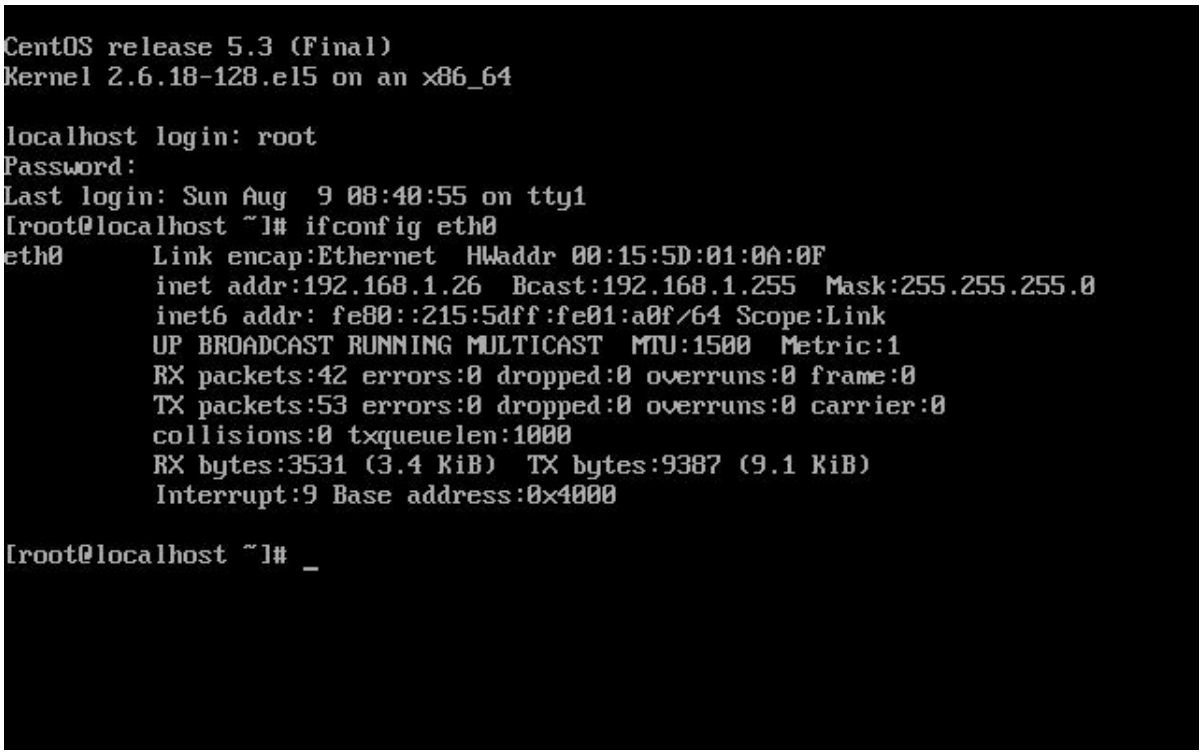


Figure 23 Linux Console

The first virtual console is normally reserved for X Windows graphical desktops.

Absolute filepaths

You can reference a directory name within the virtual directory using an *absolute filepath*. The absolute filepath defines exactly where the directory is in the virtual directory structure, starting at the root of the virtual directory. Sort of like a full name for a directory.

Thus, to reference the apache directory, that's contained within the lib directory, which in turn is contained within the usr directory, you would use the absolute filepath:

/usr/lib/apache

With the absolute filepath there's no doubt as to exactly where you want to go. To move to a specific location in the filesystem using the absolute filepath, you just specify the full pathname in the cd command:

root@1[~]#cd /etc
root@1[etc]#

The prompt shows that the new directory for the shell after the cd command is now /etc. You can move to any level within the entire Linux virtual directory structure using the absolute filepath:

root@1[~]# cd /usr/lib/apache
root@1[apache]#

However, if you're just working within your own home directory structure, often using absolute filepaths can get tedious. For demo, if you're already in the directory /home/root, it seems somewhat cumbersome to have to type the command:

cd /home/root/Documents

just to get to your Documents directory. Fortunately, there's a simpler solution.

Relative filepaths

Relative filepaths allow you to specify a destination filepath relative to your current location, without having to start at the root. A relative filepath doesn't start with a forward slash, indicating the root directory.

Instead, a relative filepath starts with either a directory name (if you're traversing to a directory under your current directory), or a special character indicating a relative location to your current directory location. The two special characters used for this are:

■ The dot (.) to represent the current directory
■ The double dot (..) to represent the parent directory

The double dot character is extremely handy when trying to traverse a directory hierarchy. For demo, if you are in the Documents directory under your home directory and need to go to your Desktop directory, also under your home directory, you can do this:

root@1[Documents]# cd ../Desktop
root@1[Desktop]#

The double dot character takes you back up one level to your home directory, then the /Desktop portion then takes you back down into the Desktop directory. You can use as many double dot characters as necessary to move around. For demo, if you are in your home directory (/home/root) and want to go to the /etc directory, you could type:

root@1[~]# cd ../../etc
root@1[etc]#

Of course, in a case like this, you actually have to do more typing to use the relative filepath rather than just typing the absolute filepath, /etc!

Beginning with command line utility

File and Directory Listing

The most basic feature of the shell is the ability to see what files are available on the system. The list command (ls) is the tool that helps do that. This section describes the ls command, and all of the options available to format the information it can provide.

Basic listing

The ls command at its most basic form displays the files and directories located in your current directory:

```
# ls
4root Desktop Download Music Pictures store store.zip test
backup Documents Drivers myfile Public store.sql Templates Videos
#
```

Notice that the ls command produces the listing in alphabetical order (in columns rather than rows). If you're using a terminal emulator that supports color, the ls command may also show different types of entries in different colors. The LS COLORS environment variable controls this feature. Different Linux distributions set this environment variable depending on the capabilities of the terminal emulator.

If you don't have a color terminal emulator, you can use the -F parameter with the ls command to easily distinguish files from directories. Using the -F parameter produces the following output:

```
# ls -F
4root/ Documents/ Music/ Public/ store.zip Videos/
backup.zip Download/ myfile* store/ Templates/
Desktop/ Drivers/ Pictures/ store.sql test
#
```

The -F parameter now flags the directories with a forward slash, to help identify them in the listing. Similarly, it flags executable files (like the myfile file above) with an asterisk, to help you find the files that can be run on the system easier.

The basic ls command can be somewhat misleading. It shows the files and directories contained in the current directory, but not necessarily all of them. Linux often uses *hidden files* to store configuration information. In Linux, hidden files are files with filenames that start with a period. These files don't appear in the default ls listing (thus they are called hidden). To display hidden files along with normal files and directories, use the -a parameter.

Wow, that's quite a difference. In a home directory for a user who has logged in to the system from a graphical desktop, you'll see lots of hidden configuration files. This particular demo is from a user logged in to a GNOME desktop session. Also notice that there are three files that begin with .bash. These files are hidden files that are used by the bash shell environment.

The -R parameter is another command ls parameter to use. It shows files that are contained within directories in the current directory. If you have lots of directories, this can be quite a long listing. Here's a simple demo of what the -R parameter produces:

```
# ls -F -R
.:
file1 test1/ test2/
./test1:
myfile1* myfile2*
./test2:
#
```

Notice that first, the -R parameter shows the contents of the current directory, which is a file (file1) and two directories (test1 and test2). Following that, it traverses each of the two directories, showing if any files are contained within each directory. The test1 directory shows two files (myfile1 and myfile2), while the test2 directory doesn't contain any files. If there had been further subdirectories within the test1 or test2 directories, the -R parameter would have continued to

traverse those as well. As you can see, for large directory structures this can become quite a large output listing.

Modifying the information presented

As you can see in the basic listings, the ls command doesn't produce a whole lot of information about each file. For listing additional information, another popular parameter is -l. The -l parameter produces a long listing format, providing more information about each file in the directory:

```
# ls -l
total 2064
drwxrwxr-x  2 root root 4096        2007-08-24 22:04 4root
-rw-r--r--  1 root root 1766205     2007-08-24 15:34 backup.zip
drwxr-xr-x  3 root root 4096        2007-08-31 22:24 Desktop
drwxr-xr-x  2 root root 4096        2001-11-01 04:06 Documents
drwxr-xr-x  2 root root 4096        2001-11-01 04:06 Download
drwxrwxr-x  2 root root 4096        2007-07-26 18:25 Drivers
drwxr-xr-x  2 root root 4096        2001-11-01 04:06 Music
-rwxr--r--  1 root root 30          2007-08-23 21:42 myfile
drwxr-xr-x  2 root root 4096        2001-11-01 04:06 Pictures
drwxr-xr-x  2 root root 4096        2001-11-01 04:06 Public
drwxrwxr-x  5 root root 4096        2007-08-24 22:04 store
-rw-rw-r--  1 root root 98772       2007-08-24 15:30 store.sql
-rw-r--r--  1 root root 107507      2007-08-13 15:45 store.zip
drwxr-xr-x  2 root root 4096        2001-11-01 04:06 Templates
drwxr-xr-x  2 root root 4096        2001-11-01 04:06 Videos
[root@testbox ~]#
```

The long listing format lists each file and directory contained in the directory on a single line. Besides the filename, it shows additional useful information. The first line in the output shows the total number of blocks contained within the directory. Following that, each line contains the following information about each file (or directory):

■ The file type (such as directory (d), file (-), character device (c), or block device (b)

■ The permissions for the file

■ The number of hard links to the file

■ The username of the owner of the file

■ The group name of the group the file belongs to

■ The size of the file in bytes

■ The time the file was modified last

■ The file or directory name

The -l parameter is a powerful tool to have. Armed with this information you can see just about any information you need to for any file or directory on the system.

There are lots of parameters for the ls command that can come in handy as you do file management. If you use the man command for ls, you'll see several pages of available parameters for you to use to modify the output of the ls command.

Single Letter	Full Word	Description
-o		In long listing don't display owner names.
-r	--reverse	Reverse the sorting order when displaying files and directories.
-R	--recursive	List subdirectory contents recursively.
-s	--size	Print the block size of each file.
-S	--sort=size	Sort the output by file size.
-t	--sort=time	Sort the output by file modification time.
-u		Display file last access time instead of last modification time.
-U	--sort=none	Don't sort the output listing.
-v	--sort=version	Sort the output by file version.
-x		List entries by line instead of columns.
-X	--sort=extension	Sort the output by file extension.

Figure 24 ls parameters

Some Popular ls Command Parameters

Single Letter	Full Word	Description
-a	--all	Don't ignore entries starting with a period.
-A	--almost-all	Don't list the . and .. files.
	--author	Print the author of each file.
-b	--escape	Print octal values for nonprintable characters.
	--block-size=size	Calculate the block sizes using size-byte blocks.
-B	--ignore-backups	Don't list entries with the tilde (~) symbol (used to denote backup copies).
-c		Sort by time of last modification.
-C		List entries by columns.
	--color=when	When to use colors (always, never, or auto).
-d	--directory	List directory entries instead of contents, and don't dereference symbolic links.
-F	--classify	Append file-type indicator to entries.
	--file-type	Only append file-type indicators to some filetypes (not executable files).
	--format=word	Format output as either across, commas, horizontal, long, single-column, verbose, or vertical.
-g		List full file information except for the file's owner.
	--group-directories-first	List all directories before files.
-G	--no-group	In long listing don't display group names.
-h	--human-readable	Print sizes using K for kilobytes, M for megabytes, and G for gigabytes.
	--si	Same as -h, but use powers of 1000 instead of 1024.
-i	--inode	Display the index number (inode) of each file.
-l		Display the long listing format.
-L	--dereference	Show information for the original file for a linked file.
-n	--numeric-uid-gid	Show numeric `userid` and `groupid` instead of names.

Figure 25 ls Parameters

File Handling

The bash shell provides lots of commands for manipulating files on the Linux filesystem. This section walks you through the basic commands you will need to work with files from the CLI for all your file-handling needs.

Creating files

Every once in a while you will run into a situation where you need to create an empty file. Sometimes applications expect a log file to be present before they can write to it. In these situations, you can use the touch command to easily create an empty file:

```
# touch test1
# ls -il test1
1954793 -rw-r--r-- 1 root root 0 Sep 1 09:35 test1
#
```

The touch command creates the new file you specify, and assigns your username as the file owner. Since I used the -il parameters for the ls command, the first entry in the listing shows the inode number assigned to the file. Every file on the Linux system has a unique inode number. Notice that the file size is zero, since the touch command just created an empty file. The touch command can also be used to change the access and modification times on an existing file without changing the file contents:

```
# touch test1
# ls -l test1
-rw-r--r-- 1 root root 0 Sep 1 09:37 test1
#
```

The modification time of test1 is now updated from the original time. If you want to change only the access time, use the -a parameter. To change only the modification time, use the –m parameter.

By default touch uses the current time. You can specify the time by using the –t parameter with a specific timestamp:

```
# touch -t 200812251200 test1
# ls -l test1
-rw-r--r-- 1 root root 0 Dec 25 2008 test1
#
```

Now the modification time for the file is set to a date significantly in the future from the current time.

Copying files

Copying files and directories from one location in the filesystem to another is a common practice for system administrators. The cp command provides this feature. In it's most basic form, the cp command uses two parameters: the source object and the destination object:

cp source destination

When both the *source* and *destination* parameters are filenames, the cp command copies the source file to a new file with the filename specified as the destination. The new file acts like a brand new file, with an updated file creation and last modified times:

```
# cp test1 test2
# ls -il
total 0
1954793 -rw-r--r-- 1 root root 0 Dec 25 2008 test1
1954794 -rw-r--r-- 1 root root 0 Sep 1 09:39 test2
#
```

The new file test2 shows a different inode number, indicating that it's a completely new file. You'll also notice that the modification time for the test2 file shows the time that it was created.

If the destination file already exists, the cp command will prompt you to answer whether or not you want to overwrite it:

```
# cp test1 test2
cp: overwrite `test2'? y
#
```

If you don't answer y, the file copy will not proceed. You can also copy a file to an existing directory:

```
# cp test1 dir1
# ls -il dir1
total 0
1954887 -rw-r--r-- 1 root root 0 Sep 6 09:42 test1
#
```

The new file is now under the dir1 directory, using the same filename as the original. These demos all used relative pathnames, but you can just as easily use the absolute pathname for both the source and destination objects.
To copy a file to the current directory you're in, you can use the dot symbol:

```
# cp /home/root/dir1/test1 .
cp: overwrite `./test1'?
```

As with most commands, the cp command has a few command line parameters to help you out. Use the -p parameter to preserve the file access or modification times of the original file for the copied file.

```
# cp -p test1 test3
# ls -il
```

```
total 4
1954886 drwxr-xr-x 2 root root 4096 Sep 1 09:42 dir1/
1954793 -rw-r--r-- 1 root root 0 Dec 25 2008 test1
1954794 -rw-r--r-- 1 root root 0 Sep 1 09:39 test2
1954888 -rw-r--r-- 1 root root 0 Dec 25 2008 test3
#
```

Now, even though the test3 file is a completely new file, it has the same timestamps as the original test1 file.

The -R parameter is extremely powerful. It allows you to recursively copy the contents of an entire directory in one command:

```
# cp -R dir1 dir2
# ls -l
total 8
drwxr-xr-x      2 root root 4096        Sep 6   09:42 dir1/
drwxr-xr-x      2 root root 4096        Sep 6   09:45 dir2/
-rw-r--r--      1 root root 0           Dec 25 09:30 test1
-rw-r--r--      1 root root 0           Sep 6   09:39 test2
-rw-r--r--      1 root root 0           Dec 25 11:30 test3
#
```

Now dir2 is a complete copy of dir1. You can also use wildcard characters in your cp commands:

```
# cp -f test* dir2
# ls -al dir2
total 12
drwxr-xr-x 2 root root 4096 Sep 6 10:55 ./
drwxr-xr-x 4 root root 4096 Sep 6 10:46 ../
-rw-r--r-- 1 root root 0 Dec 25 2008 test1
-rw-r--r-- 1 root root 0 Sep 6 10:55 test2
-rw-r--r-- 1 root root 0 Dec 25 2008 test3
```

#

The cp Command Parameters

Parameter	Description
-a	Archive files by preserving their attributes.
-b	Create a backup of each existing destination file instead of overwriting it.
-d	Preserve.
-f	Force the overwriting of existing destination files without prompting.
-i	Prompt before overwriting destination files.
-l	Create a file link instead of copying the files.
-p	Preserve file attributes if possible.
-r	Copy files recursively.
-R	Copy directories recursively.
-s	Create a symbolic link instead of copying the file.
-S	Override the backup feature.
-u	Copy the source file only if it has a newer date and time than the destination (update).
-v	Verbose mode, explaining what's happening.
-x	Restrict the copy to the current filestytem.

Figure 26 cp command parameters

Linking files

You may have noticed a couple of the parameters for the cp command referred to linking files. This is a pretty cool option available in the Linux filesystems. If you need to maintain two (or more) copies of the same file on the system, instead of having separate physical copies, you can

use one physical copy and multiple virtual copies, called *links*. A link is a placeholder in a directory that points to the real location of the file. There are two different types of file links in Linux:

■ A symbolic, or soft, link
■ A hard link

The hard link creates a separate file that contains information about the original file and where to locate it. When you reference the hard link file, it's just as if you're referencing the original file:

```
# cp -l test1 test4
# ls -il
total 16
1954886 drwxr-xr-x 2 root root 4096 Sep 1 09:42 dir1/
1954889 drwxr-xr-x 2 root root 4096 Sep 1 09:45 dir2/
1954793 -rw-r--r-- 2 root root 0 Sep 1 09:51 test1
1954794 -rw-r--r-- 1 root root 0 Sep 1 09:39 test2
1954888 -rw-r--r-- 1 root root 0 Dec 25 2008 test3
1954793 -rw-r--r-- 2 root root 0 Sep 1 09:51 test4
#
```

The -l parameter created a hard link for the test1 file called test4. When I performed the file listing, you can see that the inode number of both the test1 and test4 files are the same, indicating that, in reality, they are both the same file. Also notice that the link count (the third item in the listing) now shows that both files have two links.

(NOTE:- You can only create a hard link between files on the same physical medium. You can't create a hard link between files under separate mount points. In that case, you'll have to use a soft link.)

On the other hand, the -s parameter creates a symbolic, or soft, link:

```
# cp -s test1 test5
```

```
# ls -il test*
total 16
1954793 -rw-r--r-- 2 root root 6 Sep 1 09:51 test1
1954794 -rw-r--r-- 1 root root 0 Sep 1 09:39 test2
1954888 -rw-r--r-- 1 root root 0 Dec 25 2008 test3
1954793 -rw-r--r-- 2 root root 6 Sep 1 09:51 test4
1954891 lrwxrwxrwx 1 root root 5 Sep 1 09:56 test5 -> test1
```

There are a couple of things to notice in the file listing, First, you'll notice that the new test5 file has a different inode number than the test1 file, indicating that the Linux system treats it as a separate file. Second, the file size is different. A linked file needs to store only information about the source file, not the actual data in the file. The filename area of the listing shows the relationship between the two files.

(TIP: Instead of using the cp command, if you want to link files you can also use the ln command. By default the ln command creates hard links. If you want to create a soft link, you'll still need to use the -s parameter.)

Be careful when copying linked files. If you use the cp command to copy a file that's linked to another source file, all you're doing is making another copy of the source file. This can quickly get confusing. Instead of copying the linked file, you can create another link to the original file. You can have many links to the same file with no problems. However, you also don't want to create soft links to other soft-linked files. This creates a chain of links that can not only be confusing but also be easily broken, causing all sorts of problems.

Renaming files

In the Linux world, renaming files is called *moving*. The mv command is available to move both files and directories to another location:

```
# mv test2 test6
```

```
# ls -il test*
1954793 -rw-r--r-- 2 root root 6 Sep 1 09:51 test1
1954888 -rw-r--r-- 1 root root 0 Dec 25 2008 test3
1954793 -rw-r--r-- 2 root root 6 Sep 1 09:51 test4
1954891 lrwxrwxrwx 1 root root 5 Sep 1 09:56 test5 -> test1
1954794 -rw-r--r-- 1 root root 0 Sep 1 09:39 test6
#
```

Notice that moving the file changed the filename but kept the same inode number and the timestamp value. Moving a file with soft links is a problem:

```
# mv test1 test8
# ls -il test*
total 16
1954888 -rw-r--r-- 1 root root 0 Dec 25 2008 test3
1954793 -rw-r--r-- 2 root root 6 Sep 1 09:51 test4
1954891 lrwxrwxrwx 1 root root 5 Sep 1 09:56 test5 -> test1
1954794 -rw-r--r-- 1 root root 0 Sep 1 09:39 test6
1954793 -rw-r--r-- 2 root root 6 Sep 1 09:51 test8
[root@test2 clsc]# mv test8 test1
```

The test4 file that uses a hard link still uses the same inode number, which is perfectly fine. However, the test5 file now points to an invalid file, and it is no longer a valid link. You can also use the mv command to move directories:

```
# mv dir2 dir4
```

The entire contents of the directory are unchanged. The only thing that changes is the name of the directory.

Deleting files

Most likely at some point in your Linux career you'll want to be able to delete existing files. Whether it's to clean up a filesystem or to remove a software package, there's always opportunities to delete files.

In the Linux world, deleting is called *removing*. The command to remove files in the bash shell is rm. The basic form of the rm command is pretty simple:

```
# rm -i test2
rm: remove `test2'? y
# ls -l
total 16
drwxr-xr-x 2 root root 4096 Sep 1 09:42 dir1/
drwxr-xr-x 2 root root 4096 Sep 1 09:45 dir2/
-rw-r--r-- 2 root root 6 Sep 1 09:51 test1
-rw-r--r-- 1 root root 0 Dec 25 2008 test3
-rw-r--r-- 2 root root 6 Sep 1 09:51 test4
lrwxrwxrwx 1 root root 5 Sep 1 09:56 test5 -> test1
#
```

Notice that the command prompts you to make sure that you're serious about removing the file. There's no trashcan in the bash shell. Once you remove a file it's gone forever. Now, here's an interesting tidbit about deleting a file that has links to it:

```
# rm test1
# ls -l
total 12
drwxr-xr-x 2 root root 4096 Sep 1 09:42 dir1/
drwxr-xr-x 2 root root 4096 Sep 1 09:45 dir2/
-rw-r--r-- 1 root root 0 Dec 25 2008 test3
-rw-r--r-- 1 root root 6 Sep 1 09:51 test4
lrwxrwxrwx 1 root root 5 Sep 1 09:56 test5 -> test1
```

```
# cat test4
hello
# cat test5
cat: test5: No such file or directory
#
```

I removed the test1 file, which had both a hard link with the test4 file and a soft link with the test5 file. Noticed what happened. Both of the linked files still appear, even though the test1 file is now gone (although on my color terminal the test5 filename now appears in red). When I look at the contents of the test4 file that was a hard link, it still shows the contents of the file. When I look at the contents of the test5 file that was a soft link, bash indicates that it doesn't exist any more.

Remember that the hard link file uses the same inode number as the original file. The hard link file maintains that inode number until you remove the last linked file, preserving the data! All the soft link file knows is that the underlying file is now gone, so it has nothing to point to. This is an important feature to remember when working with linked files. One other feature of the rm command, if you're removing lots of files and don't want to be bothered with the prompt, is to use the -f parameter to force the removal. Just be careful!

Directory Handling

In Linux there are a few commands that work for both files and directories (such as the cp command), and some that only work for directories. To create a new directory, you'll need to use a specific command, which I'll discuss in this section. Removing directories can get interesting, so we'll look at that as well in this section.

Creating directories

There's not much to creating a new directory in Linux, just use the mkdir command:

```
# mkdir dir3
# ls -il
total 16
```

```
1954886 drwxr-xr-x 2 root root 4096 Sep 1 09:42 dir1/
1954889 drwxr-xr-x 2 root root 4096 Sep 1 10:55 dir2/
1954893 drwxr-xr-x 2 root root 4096 Sep 1 11:01 dir3/
1954888 -rw-r--r-- 1 root root 0 Dec 25 2008 test3
1954793 -rw-r--r-- 1 root root 6 Sep 1 09:51 test4
#
```

The system creates a new directory and assigns it a new inode number.

Deleting directories

Removing directories can be tricky, but there's a reason for that. There are lots of opportunity for bad things to happen when you start deleting directories. The bash shell tries to protect us from accidental catastrophes as much as possible. The basic command for removing a directory is rmdir:

```
# rmdir dir3
# rmdir dir1
rmdir: dir1: Directory not empty
#
```

By default, the rmdir command only works for removing empty directories. Since there is a file in the dir1 directory, the rmdir command refuses to remove it. You can remove nonempty directories using the --ignore-fail-on-non-empty parameter.

Our friend the rm command can also help us out some when handling directories. If you try using it with not parameters, as with files, you'll be somewhat disappointed:

```
# rm dir1
rm: dir1: is a directory
#
```

However, if you really want to remove a directory, you can use the -r parameter to recursively remove the files in the directory, then the directory itself:

```
# rm -r dir2
rm: descend into directory `dir2'? y
rm: remove `dir2/test1'? y
rm: remove `dir2/test3'? y
rm: remove `dir2/test4'? y
rm: remove directory `dir2'? y
#
```

While this works, it's somewhat awkward. Notice that you still must verify every file that gets removed. For a directory with lots of files and subdirectories, this can become tedious. The ultimate solution for throwing caution to the wind and removing an entire directory, contents and all, is the rm command with both the -r and -f parameters:

```
# rm -rf dir2
#
```

That's it. No warnings, no fanfare, just another shell prompt. This, of course, is an extremely dangerous tool to have, especially if you're logged in as the root user account. Use it sparingly, and only after triple checking to make sure that you're doing exactly what you want to do.

Viewing File Contents

So far we've covered everything there is to know about files, except for how to peek inside of them. There are several commands available for taking a look inside files without having to pull out an editor . This section demonstrates a few of the commands you have available to help you examine files.

Viewing file statistics

You've already seen that the ls command can be used to provide lots of useful information about files. However, there's still more information that you can't see in the ls command (or at least not all at once).

The stat command provides a complete rundown of the status of a file on the filesystem:

```
# stat test10
File: "test10"
Size: 6 Blocks: 8 Regular File
Device: 306h/774d Inode: 1954891 Links: 2
Access: (0644/-rw-r--r--) Uid: ( 501/ root) Gid: ( 501/ root)
Access: Sat Sep 1 12:10:25 2007
Modify: Sat Sep 1 12:11:17 2007
Change: Sat Sep 1 12:16:42 2007
#
```

The results from the stat command show just about everything you'd want to know about the file being examined, even down the major and minor device numbers of the device where the file is being stored.

Viewing the file type

Despite all of the information the stat command produces, there's still one piece of information missing — the file type. Before you go charging off trying to list out a 1000-byte file, it's usually a good idea to get a handle on what type of file it is. If you try listing a binary file, you'll get lots of gibberish on your monitor, and possibly even lock up your terminal emulator.

The file command is a handy little utility to have around. It has the ability to peek inside of a file and determine just what kind of file it is:

```
# file test1
test1: ASCII text
# file myscript
myscript: Bourne shell script text executable

# file myfile
```

```
myfile: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), not stripped
#
```

The file command classifies files into three categories:

- Text files: Files that contain printable characters
- Executable files: Files that you can run on the system
- Data files: Files that contain nonprintable binary characters, but that you can't run on the system

The first demo shows a text file. The file command determined not only that the file contains text but also the character code format of the text. The second demo shows a text script file. While the file is text, since it's a script file you can execute (run) it on the system. The final demo is a binary executable program. The file command determines the platform that the program was compiled for and what types of libraries it requires. This is an especially handy feature if you have a binary executable program from an unknown source.

Viewing the whole file

If you have a large text file on your hands, you may want to be able to see what's inside of it. There are three different commands in Linux that can help you out here.

The cat command

The cat command is a handy tool for displaying all of the data inside a text file:

```
# cat test1
hello
This is a test file.
That we'll use to test the cat command.
#
```

Nothing too exciting, just the contents of the text file. There are a few parameters you can use with the cat command, though, that can help you out.

The -n parameter numbers all of the lines for us:

```
# cat -n test1
1 hello
2 This is a test file.
3 That we'll use to test the cat command.
#
```

That feature will come in handy when you're examining scripts. If you just want to number the lines that have text in them, the -b parameter is for you:

```
# cat -b test1
1 hello
2 This is a test file.
3 That we'll use to test the cat command.
```

Fancy! If you need to compress multiple blank lines into a single blank line, use the -s parameter:

```
# cat -s test1
hello
This is a test file.
That we'll use to test the cat command.
#
```

Finally, if you don't want tab characters to appear, use the -T parameter:

```
# cat -T test1
hello
This is a test file.
That we'll use to^Itest the cat command.
#
```

The -T parameter replaces any tabs in the text with the ^I character combination.

For large files, the cat command can be somewhat annoying. The text in the file will just quickly scroll off of the monitor without stopping. Fortunately, there's a simple way to solve this problem.

The more command

The main drawback of the cat command is that you can't control what's happening once you start it. To solve that problem, developers created the more command. The more command displays a text file, but stops after it displays each page of data.

The more Command Options

Option	Description
H	Display a help menu.
spacebar	Display the next screen of text from the file.
z	Display the next screen of text from the file.
ENTER	Display one more line of text from the file.
d	Display a half-screen (11 lines) of text from the file.
q	Exit the program.
s	Skip forward one line of text.
f	Skip forward one screen of text.
b	Skip backward one screen of text.
/expression	Search for the text *expression* in the file.
n	Search for the next occurrence of the last specified expression.
'	Go to the first occurrence of the specified expression.
!cmd	Execute a shell command.
v	Start up the vi editor at the current line.
CTRL-L	Redraw the screen at the current location in the file.
=	Display the current line number in the file.
.	Repeat the previous command.

Figure 27 more commands parameter

The less command

Although from its name it sounds like it shouldn't be as advanced as the more command, the less command is actually a play on words and is an advanced version of the more command (the less command uses the phrase ''less is more''). It provides several very handy features for scrolling both forward and backward through a text file, as well as some pretty advanced searching capabilities.

The less command also has the feature of being able to display the contents of a file before it finishes reading the entire file. This is a serious drawback for both the cat and more commands when using extremely large files. The less command operates much the same as the more command, displaying one screen of text from a file at a time.

Notice that the less command provides additional information in its prompt, showing the total number of lines in the file, and the range of lines currently displayed. The less command supports the same command set as the more command, plus lots more options. To see all of the options available, look at the man pages for the less command. One set of features is that the less command recognizes the up and down arrow keys, as well as the page up and page down keys (assuming that you're using a properly defined terminal). This gives you full control when viewing a file.

Viewing parts of a file

Often the data you want to view is located either right at the top or buried at the bottom of a text file. If the information is at the top of a large file, you still need to wait for the cat or more commands to load the entire file before you can view it. If the information is located at the bottom

of a file (such as a log file), you need to wade through thousands of lines of text just to get to the last few entries. Fortunately, Linux has specialized commands to solve both of these problems.

The tail command

The tail command displays the last group of lines in a file. By default, it'll show the last 10 lines in the file, but you can change that with command line parameters. The -f parameter is a pretty cool feature of the tail command. It allows you to peek inside a file as it's being used by other processes. The tail command stays active and continues to display new lines as they appear in the text file. This is a great way to monitor the system log file in real-time mode.

The head command

While not as exotic as the tail command, the head command does what you'd expect, it displays the first group of lines at the start of a file. By default, it'll display the first 10 lines of text. Similar to the tail command, it supports the -c, and -n parameters so that you can alter what's displayed.

The tail Command Line Parameters

Parameter	Description
`-c bytes`	Display the last bytes number of bytes in the file.
`-n lines`	Display the last lines number of lines in the file.
`-f`	Keeps the tail program active and continues to display new lines as they're added to the file.
`--pid=PID`	Along with `-f`, follows a file until the process with ID PID terminates.
`-s sec`	Along with `-f`, sleeps for sec seconds between iterations.
`-v`	Always displays output headers giving the filename.
`-q`	Never displays output headers giving the filename.

Figure 28 tail command parameter

VI EDITOR

The VI editor is a screen-based editor used by many Unix users. The VI editor has powerful features to aid programmers. 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 demo, to edit a file called *XYZ*, you would type vi XYZ 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. 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]

If you started VI without a filename, the bottom line of the screen will just be blank when VI starts. If the screen does not show you these expected results, your terminal type may be set wrong. Just type :q and return to get out of VI.

Getting Out of VI

Now that you know how to get into VI, it would be a good idea to know how to get out of it. The VI editor has two modes and in order to get out of VI, you have to be in *command* mode. Hit the key labelled "**Escape**" or "**Esc**" to get into *command* mode.

The command to quit out of VI is :q. Once in *command* mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of VI without saving is :q!. This lets you exit VI without saving any of the changes.

you would want to save the changes you have made. The command to save the contents of the editor is :w. You can combine the above command with the quit command, or :wq. You can specify a different file name to save to by specifying the name after the :w. For demo, if you wanted to save the file you were working as another filename called *filename2*, you would type: w filename2 and return.

Another way to save your changes and exit out of VI is the ZZ command. When in *command* mode, type ZZ and it will do the equivalent of :wq. If any changes were made to the file, it will be saved. This is the easiest way to leave the editor, with only two keystrokes.

The Two Modes of VI

The first thing most users learn about the VI editor is that it has two modes: *command* and *insert*. The *command* mode allows the entry of commands to manipulate text. These commands are usually one or two characters long, and can be entered with few keystrokes. The *insert* mode puts anything typed on the keyboard into the current file.

VI starts out in *command* mode. There are several commands that put the VI editor into *insert* mode. The most commonly used commands to get into insert mode are a and i.

Some Simple VI Commands

Here is a simple set of commands to get a beginning VI user started. There are many other convenient commands, which will be discussed in later sections.

a

enter *insert* mode, the characters typed in will be inserted after the current cursor position. If you specify a count, all the text that had been inserted will be repeated that many times.

h

move the cursor to the left one character position.

i

enter *insert* mode, the characters typed in will be inserted before the current cursor position. If you specify a count, all the text that had been inserted will be repeated that many times.

j

move the cursor down one line.

k

move the cursor up one line.

l

move the cursor to the right one character position.

r

replace one character under the cursor. Specify *count* to replace a number of characters

u

undo the last change to the file. Typing u again will re-do the change.

x

delete character under the cursor. *Count* specifies how many characters to delete. The characters will be deleted after the cursor.

Cutting and Yanking

The command commonly used command for cutting is d. This command deletes text from the file. The command is preceded by an optional *count* and followed by a movement specification. If you double the command by typing dd, it deletes the current line. Here are some combinations of these:

d^
deletes from current cursor position to the beginning of the line.

d#
deletes from current cursor position to the end of the line.

dw
deletes from current cursor position to the end of the word.

3dd
deletes three lines from current cursor position downwards.

There is also the y command which operates similarly to the d command which take text from the file without deleting the text.

Pasting

The commands to paste are p and P. The only differ in the position relative to the cursor where they paste. p pastes the specified or general buffer after the cursor position, while P pastes the specified or general buffer before the cursor position. Specifying *count* before the paste command pastes text the specified number of times.

Word and Character Searching

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 a single character.

*

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 demo: /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 demo: /\<the\> should find only word the, but not words like these: there and other.

>

See the '<' character description above.

Abbreviations and Mapping Keys to Other Keys

One EX editor command that is useful in the VI editor is the **abbreviate** command. This lets you set up abbreviations for specific strings. The command looks like this: :ab string thing to substitute for. For demo, if you had to type the name, "**Humuhumunukunukuapua`a**" but you didn't want to type the whole name, you could use an abbreviation for it. For this demo, the command is entered like this:

:ab 9u Humuhumunukunukuapua`a

Now, whenever you type 9u as a separate word, VI will type in the entire word(s) specified. If you typed in 9university, it will not substitute the word. To remove a previously defined abbreviation, the command is unabbreviate. To remove the previous demo, the command would be ":una 9u" To get your listing of abbreviations, simply just type :ab without any definitions.

Another EX editor command that is useful for customization is the mapping command. There are two kinds of mapping commands. One for command mode, and the other for insert mode. These two commands are :map and :map! respectively. The mapping works similarly to the abbreviation, and you give it a key sequence and give it another key sequence to substitute it with. (The substituted key sequences are usually VI commands.)

Inserting New Text

A
Append at the end of the current line.

I
Insert from the beginning of a line.

O
(letter oh) Enter *insert* mode in a new line above the current cursor position.
a

Enter *insert* 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.

i

Enter *insert* 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.

o

Enter *insert* mode in a new line below the current cursor position.

Linux Files and File Permission

There are three types of access:

1. read
2. write
3. execute

Each file belongs to a specific user and group. Access to the files is controlled by user, group, and what is called other. The term, other, is used to refer to someone who is not the user (owner) of the file, nor is the person a member of the group the file belongs to. When talking about setting permissions for "other" users to use, it is commonly referred to as setting the world execute, read, or write bit since anyone in the world will be able to perform the operation if the permission is set in the other category.

File names and permission characters

File names can be up to 256 characters long with "-", "_", and "." characters along with letters andnumbers.

When a long file listing is done, there are 10 characters that are shown on the left that indicate type and permissions of the file. File permissions are shown according to the following syntax demo:drwxrwxrwx

There are a total of 10 characters in this demo, as in all Linux files. The first character indicates the type of file, and the next three indicate read, write, and execute permission for each of the three user types, user, group and other. Since there are three types of permission for three users, there are a total of nine permission bits. The table below shows the syntax:

1	2	3	4	5	6	7	8	9	10
File	User Permissions			Group Permissions			Other Permissions		
Type	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
D	R	w	x	R	w	x	r	w	x

- Character 1 is the type of file: - is ordinary, d is directory, l is link.
- Characters 2-4 show owner permissions. Character 2 indicates read permission, character 3 indicates write permission, and character 4 indicates execute permission.
- Characters 5-7 show group permissions. Character 5=read, 6=write, 7=execute
- Characters 8-10 show permissions for all other users. Character 8=read, 9=write, 10=execute

There are 5 possible characters in the permission fields. They are:

- r = read - This is only found in the read field.

- w = write - This is only found in the write field.
- x = execute - This is only found in the execute field.
- s = setuid - This is only found in the execute field.
- If there is a "-" in a particular location, there is no permission. This may be found in any field whether read, write, or execute field.

Demos

Type "ls -l" and a listing like the following is displayed:

total 10

drwxrwxrwx 4 FIRST FIRST 122 Dec 15 11:02 Projects

-rw-rw-rw- 1 FIRST FIRST 1873 Aug 25 03:34 Test

-rw-rw-rw- 1 FIRST FIRST 1234 Sep 12 11:13 datafile

Which means the following:

Type and Permission	# of field Links	Files's Owner	File's Group	Size in Bytes	Date of last modification	Filename
drwxrwxrwx	4	FIRST	FIRST	127	Dec 19 12:02	Project2

The fields are as follows:

1. Type field: The first character in the field indicates a file type of one of the following:
    - d = directory
    - l = symbolic link
    - s = socket

- p = named pipe
- - = regular file
- c= character (unbuffered) device file special
- b=block (buffered) device file special

2. Permissions are explained above.
3. Links: The number of directory entries that refer to the file. In our demo, there are four.
4. The file's owner in our demo is FIRST.
5. The group the file belongs to. In our demo, the group is FIRST.
6. The size of the file in bytes
7. The last modification date. If the file is recent, the date and time is shown. If the file is not in the current year, the year is shown rather than time.
8. The name of the file.

Set User Identification Attribute

The file permissions bits include an execute permission bit for file owner, group and other. When the execute bit for the owner is set to "s" the set user ID bit is set. This causes any persons or processes that run the file to have access to system resources as though they are the owner of the file. When the execute bit for the group is set to "s", the set group ID bit is set and the user running the program is given access based on access permission for the group the file belongs to. The following command:

```
chmod +s myfile
```

sets the user ID bit on the file "myfile". The command:

```
chmod g+s myfile
```

sets the group ID bit on the file "myfile".

The listing below shows a listing of two files that have the group or user ID bit set.

```
-rws--x--x   1 root    root    14024 Sep  9 1999 chfn
-rwxr-sr-x   1 root    mail    12072 Aug 16 1999 lockfile
```

The files chfn and lockfile are located in the directory "/usr/bin". The "s" takes the place of the normal location of the execute bit in the file listings above. This special permission mode has no meaning unless the file has execute permission set for either the group or other as well. This means that in the case of the lockfile, if the other users (world execute) bit is not set with permission to execute, then the user ID bit set would be meaningless since only that same group could run the program anyhow. In both files, everyone can execute the binary. The first program, when run is executed as though the program is the root user. The second program is run as though the group "mail" is the user's group.

For system security reasons it is not a good idea to set many program's set user or group ID bits any more than necessary, since this can allow an unauthorized user privileges in sensitive system areas. If the program has a flaw that allows the user to break out of the intended use of the program, then the system can be compromised.

Directory Permissions

There are two special bits in the permissions field of directories. They are:

- s - Set group ID
- t - Save text attribute (sticky bit) - The user may delete or modify only those files in the directory that they own or have write permission for.

Save text attribute

The /tmp directory is typically world-writable and looks like this in a listing:

```
drwxrwxrwt   13 root     root         4096 Apr 15 08:05 tmp
```

Everyone can read, write, and access the directory. The "t" indicates that only the user (and root, of course) that created a file in this directory can delete that file.

To set the sticky bit in a directory, do the following:

```
chmod +t data
```

This option should be used carefully. A possible alternative to this is

1. Create a directory in the user's home directory to which he or she can write temporary files.
2. Set the TMPDIR environment variable using each user's login script.
3. Programs using the tempnam(3) function will look for the TMPDIR variable and use it, instead of writing to the /tmp directory.

Directory Set Group ID

If the setgid bit on a directory entry is set, files in that directory will have the group ownership as the directory, instead of than the group of the user that created the file.

This attribute is helpful when several users need access to certain files. If the users work in a directory with the setgid attribute set then any files created in the directory by any of the users will have the permission of the group. For demo, the administrator can create a group called ABC and add the users Ram and Jain to the group ABC. The directory ABCdir can be created with the set GID bit set and Ram and Jain although in different primary groups can work in the directory and have full access to all files in that directory, but still not be able to access files in each other's primary group.

The following command will set the GID bit on a directory:

```
chmod g+s ABCdir
```

The directory listing of the directory "ABCdir":

```
drwxrwsr-x 2 kathy ABC 1674 Sep 17 1999 ABCdir
```

The "s" in place of the execute bit in the group permissions causes all files written to the directory "ABCdir" to belong to the group "ABC" .

Demos

Below are demos of making changes to permissions:

chmod u+x myfile	Gives the user execute permission on myfile.
chmod +x myfile	Gives everyone execute permission on myfile.
chmod ugo+x myfile	Same as the above command, but specifically specifies user, group and other.
chmod 400 myfile	Gives the user read permission, and removes all other permission. These permissions are specified in octal, the first char is for the user, second for the group and the third is for other. The high bit (4) is for read access, the middle bit (2) os for write access, and the low bit (1) is for execute access.
chmod 764 myfile	Gives user full access, group read and write access, and other read access.
chmod 751 myfile	Gives user full access, group read and execute permission, and other, execute permission.
chmod +s myfile	Set the setuid bit.
chmod go=rx myfile	Remove read and execute permissions for the group and other.

Below are demos of making changes to owner and group:

chown Garry test1	Changes the owner of the file test1 to the user Garry.
chgrp Garry test1	Changes the file test1 to belong to the group "Garry".

Note: Linux files were displayed with a default tab value of 8 in older Linux versions. That means that file names longer than 8 may not be displayed fully if you are using an old Linux distribution. There is an option associated with the ls command that solves this problem. It is "-T". Ex: "ls al -T 30" to make the tab length 30.

Umask Settings

The umask command is used to set and determine the default file creation permissions on the system. It is the octal complement of the desired file mode for the specific file type. Default permissions are:

- 777 - Executable files
- 666 - Text files

These defaults are set allowing all users to execute an executable file and not to execute a text file. The defaults allow all users can read and write the file.

The permission for the creation of new executable files is calculated by subtracting the umask value from the default permission value for the file type being created. An demo for a text file is shown below with a umask value of 022:

```
 666 Default Permission for text file
-022 Minus the umask value
-----
 644 Allowed Permissions
```

Therefore the umask value is an expression of the permissions the user, group and world will not have as a default with regard to reading, writing, or executing the file. The umask value here means the group the file belongs to and users other than the owner will not be able to write to the file. In this case, when a new text file is created it will have a file permission value of 644, which means the owner can read and write the file, but members of the group the file belongs to, and all others can only read the file. A long directory listing of a file with these permissions set is shown below.

```
-rw-r--r--   1 root     FIRST         14233 Apr  24 10:32 textfile.txt
```

A demo command to set the umask is:

umask 022

The most common umask setting is 022. The /etc/profile script is where the umask command is usually set for all users.

Red Hat Linux has a user and group ID creation scheme where there is a group for each user and only that user belongs to that group. If you use this scheme consistently you only need to use 002 for your umask value with normal users.

Hard Link and Soft Link

Hard Link is a mirror copy of the original file. Hard links share the same inode. Any changes made to the original or Hard linked file will reflect the other. Even if you delete any one of the files, nothing will happen to the other. Hard links can't cross file systems.

Soft Link is a symbolic link to the original file. Soft Links will have a different Inode value. A soft link points to the original file. If you delete the original file, the soft link fails. If you delete the soft link, nothing will happen. Hard links can cross file systems.

Lets learn the difference with an demo

Demo: Create a file " original-file.txt "

:~/test# echo "Learning about Hard and Soft Link" > original-file.txt
:~/test# cat original-file.txt
Learning about Hard and Soft Link

Now lets create a **Hard Link**

Demo: HardLink-file.txt

:~/test# ln original-file.txt HardLink-file.txt

:~/test# ls –il

total 8

840388 -rw-r--r-- 2 FIRST FIRST 33 2009-05-18 09:16 HardLink-file.txt

840388 -rw-r--r-- 2 FIRST FIRST 33 2009-05-18 09:16 original-file.txt

Now lets create a **Soft Link**

Demo: SoftLink-file.txt

:~/test# ln -s original-file.txt SoftLink-file.txt

:~/test# ls –il

total 8

840388 -rw-r--r-- 2 FIRST FIRST 33 2009-05-18 09:16 HardLink-file.txt

840388 -rw-r--r-- 2 FIRST FIRST 33 2009-05-18 09:16 original-file.txt

840186 lrwxrwxrwx 1 FIRST FIRST 17 2009-05-18 09:23 SoftLink-file.txt -> original-file.txt

From the above ls -il result, you find the same inode for " HardLink-file.txt " and " original-file.txt ".

Inode value is different for the soft link " SoftLink-file.txt ".

Now lets try editing the original file:

Demo:

:~/test# cat >> original-file.txt

Editing this file to learn more!!

:~/test# cat original-file.txt

Learning about Hard and Soft Link
Editing this file to learn more!!

:~/test# cat HardLink-file.txt
Learning about Hard and Soft Link
Editing this file to learn more!!

:~/test# cat SoftLink-file.txt
Learning about Hard and Soft Link
Editing this file to learn more!!

Now lets try changing the permissions:

Demo:

:~/test# chmod 700 original-file.txt
:~/test# ls -il
total 8
840388 -rwx------ 2 FIRST FIRST 67 2009-05-18 09:34 HardLink-file.txt
840388 -rwx------ 2 FIRST FIRST 67 2009-05-18 09:34 original-file.txt
840186 lrwxrwxrwx 1 FIRST FIRST 17 2009-05-18 09:23 SoftLink-file.txt -> original-file.txt

From the above demo its clear that changing the permission of " original-file.txt " will update the permission set of " HardLink-file.txt ".
The soft link remains unchanged.

Now lets try deleting the original file.

Demo:

:~/test# rm original-file.txt
:~/test# ls -il
total 4

840388 -rwx------ 1 FIRST FIRST 67 2009-05-18 09:34 HardLink-file.txt
840186 lrwxrwxrwx 1 FIRST FIRST 17 2009-05-18 09:23 SoftLink-file.txt -> original-file.txt

:~/test# cat HardLink-file.txt
Learning about Hard and Soft Link
Editing this file to learn more!!

:~/test# cat SoftLink-file.txt
cat: SoftLink-file.txt: No such file or directory

So removing the original file will affect the Soft link. The Soft link fails. Hard link is unaffected.

Users and Groups

Who Is The Super User?

The super user with unrestricted access to all system resources and files is the user named "**root**". You will need to log in as user **root** to add new users to your Linux box.

How To Add Users

When invoked without the **-D** option, the **useradd** command creates a new user account using the values specified on the command line and the default values from the system. The new user account will be entered into the system files as needed, the home directory will be created, and initial files copied, depending on the command line options. The version provided with Red Hat Linux will create a group for each user added to the system, unless the **-n** option is given. The options which apply to the **useradd** command are:

useradd [-c comment] [-d home_dir] [-e expire_date] [-f inactive_time] [-g initial_group] [-G group[,...]] [-m [-k skeleton_dir]] [-p passwd] [-s shell] [-u uid [-o]] login

useradd -D [-g default_group] [-b default_home] [-f default_inactive] [-e default_expire_date] [-s default_shell]

-c comment	The new user's password file comment field.
-d home_dir	The new user will be created using home_dir as the value for the user's login directory. The default is to append the login name to default_home and use that as the login directory name.
-e expire_date	The date on which the user account will be disabled. The date is specified in the format YYYY-MM-DD.
-f inactive_time	The number of days after a password expires until the account is permanently disabled. A value of 0 disables the account as soon as the password has expired, and a value of -1 disables the feature. The default value is -1.
-g initial_group	The group name or number of the user's initial login group. The group name must exist. A group number must refer to an already existing group. The default group number is 1.
-G group,[,...]	A list of supplementary groups which the user is also a member of. Each group is separated from the next by a comma, with no intervening whitespace. The groups are subject to the same restrictions as the group given with the -g option. The default is for the user to belong only to the initial group.
-m	The user's home directory will be created if it does not exist. The files contained in skeleton_dir will be copied to the home directory if the -k option is used, otherwise the files contained in /etc/skel will be used instead. Any directories contained in skeleton_dir or /etc/skel will be created in the user's home directory as well. The -k option is only valid in conjunction with the -m option. The default is to not create the directory and to not copy any files.
-p passwd	The encrypted password, as returned by crypt. The default is to disable the account.

-s shell	The name of the user's login shell. The default is to leave this field blank, which causes the system to select the default login shell.
-u uid	The numerical value of the user's ID. This value must be unique, unless the -o option is used. The value must be non-negative. The default is to use the smallest ID value greater than 99 and greater than every other user. Values between 0 and 99 are typically reserved for system accounts.
-b default_home	The initial path prefix for a new user's home directory. The user's name will be affixed to the end of default_home to create the new directory name if the -d option is not used when creating a new account.
-e default_expire_date	The date on which the user account is disabled.
-f default_inactive	The number of days after a password has expired before the account will be disabled.
-g default_group	The group name or ID for a new user's initial group. The named group must exist, and a numerical group ID must have an existing entry.
-s default_shell	The name of the new user's login shell. The named program will be used for all future new user accounts.

Changing/Setting password for USER

passwd - entering just passwd would allow you to change the password. After entering passwd you will receive the following three prompts:

Current Password:

New Password:

Confirm New Password:

Each of these prompts must be entered and entered correctly for the password to be successfully changed.

passwd newperson

If you have just setup an account using the useradd command you would then type a command similar to the above command to set the password for the "newperson" account. Only root users have the ability to set passwords for other accounts.

Adding user to a group

You can use the useradd or usermod commands to add a user to a group. The useradd command creates a new user or update default new user information. The usermod command modifies a user account i.e. it is useful to add user to existing group. There are two types of group. First is primary user group and other is secondary group. All user account related information is stored in /etc/passwd, /etc/shadow and /etc/group files to store user information.

useradd Demo - Add A New User To Secondary Group

You need to the useradd command to add new users to existing group (or create a new group and then add user). If group does not exist, create it. The syntax is as follows:

```
useradd -G {group-name} username
```

In this demo, create a new user called FIRST and add it to group called DEMO. First login as a root user (make sure group developers exists), enter:

If you do not see any group then you need to add group DEMO using groupadd command:

```
# groupadd DEMO
```

Next, add a user called FIRST to group DEMO:

```
# useradd -G DEMO FIRST
```

Setup password for user FIRST:

```
# passwd FIRST
```

Ensure that user added properly to group DEMO:

```
# id FIRST
```

Output:
uid=1122(FIRST) gid=1125(FIRST) groups=1125(FIRST),1124(DEMO)

Please note that capital G (-G) option add user to a list of supplementary groups. Each group is separated from the next by a comma, with no intervening whitespace. For demo, add user A to groups admins, ftp, www, and DEMO, enter:

```
# useradd -G admins,ftp,www,DEMO  A
```

useradd demo - Add a new user to primary group

To add a user AA to group DEMO use following command:

```
# useradd -g DEMO AA
```

```
# id AA
```

Sample outputs:
uid=1123(AA) gid=1124(DEMO) groups=1124(DEMO)

Please note that small -g option add user to initial login group (primary group). The group name must exist. A group number must refer to an already existing group.

usermod demo - Add a existing user to existing group

Add existing user tony to ftp supplementary/secondary group with usermod command using -a option ~ i.e. add the user to the supplemental group(s). Use only with -G option :

```
# usermod -a -G ftp AA
```

Change existing user AA primary group to www:

```
# usermod -g www AA
```

/etc/passwd file

Passwd file is a text file, that contains a list of the system's accounts, giving for each account some useful information like user ID, group ID, home directory, shell, etc. Often, it also contains the encrypted passwords for each account. It should have general read permission (many utilities, like ls use it to map user IDs to user names), but write access only for the superuser.

Password file format

account:password:UID:GID:GECOS:directory:shell

The /etc/passwd file consists of user records, one to a line. Each record contains multiple fields, separated by colons (:). The fields are:

- username
- encrypted password (or x if shadow passwords are in use)
- UID
- default GID
- real name (also known as the GECOS field)
- home directory
- default shell

/etc/shadow file

shadow file contains the encrypted password information for user's accounts and optional the password aging information.

Shadow file format

smithj:Ep6mckrOLChF.:10063:0:99999:7:::

if shadow passwords are being used, the /etc/shadow file contains users' encrypted passwords and other information about the passwords. Its fields are colon-separated as for /etc/passwd, and are as follows:

- username
- encrypted password
- Days since Jan 1, 1970 that password was last changed
- Days before password may be changed
- Days after which password must be changed
- Days before password is to expire that user is warned
- Days after password expires that account is disabled
- Days since Jan 1, 1970 that account is disabled
- A reserved field

The password expiry related fields are modified by the change program.

/etc/group file

There is one entry per line, and each line has the format:

Group file format

group_name:passwd:GID:user_list

The /etc/group file consists of group records, one to a line. Each record contains multiple fields, separated by colons (:). The fields are:

- group name
- encrypted group password (or x if shadow passwords are in use)
- GID
- group members' usernames, comma-separated

USING THE BASH SHELL

SHELL VARIABLES

demo, to print out the The shell has built in variables that control the behavior of the shell. To see a list of the names and values of all currently defined shell variables, use the **set** command without any paramaters.

To reference a shell variable in a command, simply place the # in front of the variable name. This will cause the shell to replace the variable reference with the current value of that shell variable. This can save a lot of typing. For current value for the PATH variable:

echo #PATH
/bin:/usr/bin:/home/fred/bin
The two variables you will be most concerned with are PATH and PS1.

PATH shell variable

The value of the PATH shell variable is a colon-separated list of directories in which the shell should automatically look for executable files when it is trying to run a command. For demo, when you run the command

netscape
the shell will look through all of the directories in the PATH looking for an executable file named "netscape" that it can run.

If you want the shell to always look in the cwd for executable files, you need to put the cwd on the PATH. If you don't have the cwd in your PATH variable, then in order to execute a file in the cwd you have to type ./filename. If the cwd is in the PATH variable you can just type the filename.

filename

command filename not found

./filename

//program runs

To set the value of a shell variable, simply type the name of the variable, an equal sign, followed by its new value. So, to add the cwd to the PATH variable, use the following command:

PATH=#PATH:.

This simply takes the current value of PATH, appends **:.** to it, and reassigns the new value to PATH. After executing this command, programs in the cwd can be executed by simply typing their names:

filename

// program runs

PS1 shell variable

PS1 is the variable controlling the prompt. You can change the prompt to be anything you want. For demo, the following command would change the prompt to be the string "DAN IS THE MAN #".

PS1="DAN IS THE MAN \#"

The prompt format string stored in PS1 may contain special variables that are automatically replaced with useful information. Some of these are:

\w	current working directory
\W	last element of current working directory
\u	user name
\h	host name
\d	current date
\t	current time

The most common format for PS1 is

PS1="[\u@\h \w]\#"

FILE NAME EXPANSION

Many Linux commands accept multiple file or directory names as arguments.

There are many file name expansion operators, some of which are listed below:

Operator	Meaning	Demo
*	match any string of zero or more characters	# cp *.txt backup
?	match any single character	# cp ?.txt backup
[abc…]	match any of the enclosed characters	# cp [abc].txt backup
[!abc…]	match anything but the enclosed characters	# cp [!abc].txt backup
[a-z]	match any character in the range	# cp [a-c].txt backup
[!a-z]	match any character not in the range	# cp [!a-c].txt backup
{str1,str2,…}	match any of the enclosed strings	# cp {dog,cat,duck}.txt backup
~	substitute user's home directory	# cp ~/foodle/*.txt backup
~name	substitute some other user's home directory	# cp ~ram/foodle/*.txt backup

QUOTING

File name expansion is very convenient, but what if we need to pass arguments that contain file expansion operators to a program? For demo, what if you wanted to pass an asterisk as an argument to a command?

```
# print the message "* is an asterisk"
# echo * is an asterisk
```

Normally, the shell will perform file name expansion on all arguments that look like patterns, even if we don't want it to. In the demo above, before calling echo the shell would replace the asterisk with the names of all files/directories in the current directory, which is not the intended effect.

The solution to this problem is quoting. Quoting is a mechanism for turning off file name expansion, causing the shell to pass arguments through unmodified, even if they contain file name expansion operators. File name expansion can be turned off for a single character by preceding it with \, like this:

print the message "* is an asterisk"
echo * is an asterisk
Any character that is preceded by \ is taken literally and passed through untouched (i.e., unexpanded).

You can also quote entire strings by enclosing them in single-quotes, like this:

print the message "* is an asterisk"
echo '* is an asterisk'
All characters between the single-quotes are taken literally.

You can use double-quotes instead of single-quotes, but the meaning is slightly different. The meaning of single-quotes is: "Shell, don't change the string between single- quotes, not matter what!". The meaning of double-quotes is: "Shell, don't change the string between double-quotes, unless it contains a shell variable reference. In that case, it's OK to replace the variable with its value.". The following demo demonstrates the difference between using single and double quotes.

echo ’Is your name #USERNAME ?’
Is your name #USERNAME ?

echo “Is your name #USERNAME ?”
Is your name user ?
In addition to turning off file name expansion, quoting is also useful for handling file/directory names that contain spaces. For demo, suppose that you have a directory named "my foodle files". Using the cd command to change to this directory would look like this:

cd my foodle files

The problem with this is that "my foodle files" looks like three arguments instead of one, and the cd command will not work. The solution to this problem is to quote the directory name:

cd 'my foodle files'

FILTER COMMANDS/TEXT PROCESSING TOOLS:

FINDING AND PROCESSING FILES:

wc

The wc command can be used to count the number of lines, words, and bytes in text files.

wc homework.txt
4 7 280 homework.txt

The output above means that the file homework.txt contains 4 lines, 7 words, and 280 bytes.

By default, wc prints out all three pieces of information The –l, -w, and –c options can be used to tell wc to display only one or two of these pieces of information

wc –l homework.txt
4 homework.txt

wc can also compute totals for multiple files

wc –l *.txt

grep

The grep command can be used to display all lines in a text file that match a particular pattern (i.e, regular expression). The user provides one or more file names, along with the pattern they are searching for. Grep will look through all of the specified files and print out all lines that match the specified pattern. Grep is useful for searching source code or other text files for particular strings. For demo, the following command searches all of the C++ source files in a directory for the string "main":

where is the main function?
grep main *.cpp

By default, grep performs case-sensitive searches (i.e., case matters). The –i option can be used to perform a case-insensitive search:

where is the main function?

grep –i mAiN *.cpp

So far we have used only literal strings as our search patterns. The search pattern can also be a regular expression. For demo, the following command searches a file for lines containing the text BYU, with any number of spaces between the letters:

grep ’B *Y *U’ file.txt

Grep supports a variety of pattern operators, some of which are listed below:

Operator	**Meaning**
.	match any character
*	match zero or more of the preceding (Note that the meaning of * here is different than with file name expansion)
^	match beginning of line
#	match end of line
\	escape the following character
[abc]	match one in the list
[a-z]	match one in the range
[^abc]	match one not in the list
[^a-z]	match one not in the range
\{n,m\}	match between n and m of the preceding
\{n\}	match exactly n of the preceding

\{n,\}	match n or more of the preceding

Some of the grep pattern operators are also shell file expansion operators. For this reason, when using grep it is frequently necessary to quote the pattern argument so that the shell does not perform file name expansion on it.

find

The find command is used to recursively search a directory structure for files/directories that match specified criteria. Possible search criteria include name, modification date, file type, and many other possibilities. Once a matching file/directory has been found, it can be operated upon by copying it, deleting it, running wc or grep on it, etc. Typical uses of find include the following:

- Recursively finding files with particular names
- Doing recursive search & replace on files
- Recursively deleting selected files

Find is a complex command with many options. Here we cover only a few of the most useful options. The find command-line has the following form:
find pathnames conditions operations

- pathnames is a list of directories to search
- conditions are the search criteria
- operations are the operations to be performed on each matching file/directory

Search Criteria

The following demos show how to provide search criteria for several common types of searches.

- **Search by Name**

  # find pathnames -name pattern operations

  # print the paths of all files/dirs named core in the CWD
  # find . -name core -print

print the paths of all .cpp files in foodle directory
find ~/foodle -name '*.cpp' -print

- **Search by Type**

  # find pathnames -type c operations

  # print the paths of all regular files in the CWD
  # find . -type f -print

  # print the paths of all directories in foodle directory
  # find ~/foodle -type d -print

- **Search by Modification Time**

  # find pathnames -mtime (+n|-n|n) operations

  # print the paths of all files/dirs in the CWD that
  # were last modified more than 7 days ago
  # find . -mtime +7 -print

  # print the paths of all files/dirs in the foodle
  # directory that were last modified less than 7 days ago
  # find ~/foodle -mtime -7 -print

Operations on Matching Files

The next demos show how to perform common operations on matching files/directories.

- **Print the full path of each matching file/directory**

  # find pathnames conditions -print

  # print the paths of all .cpp files in foodle directory
  # find ~/foodle -name '*.cpp' -print

- **Execute a command on each matching file/directory**

  # find pathnames conditions -exec command {} \;

  # search all .cpp files in the CWD for "main"
  # find . -name '*.cpp' -print -exec grep main {} \;

  # delete all .o files in the foodle directory
  # find ~/foodle -name '*.o' -print -exec rm {} \;

  In this case, {} is replaced by the name of the matching file/directory, and \; is used to mark the end of the command.

CUT

The **cut** command takes a vertical slice of a file, printing only the specified columns or fields. Like the **sort** command, the**cut** command defines a field as a word set off by blanks, unless you specify your own delimiter. It's easiest to think of a column as just the nth character on each line. In other words, "column 5" consists of the fifth character of each line.

Consider a slight variation on the **company.data** file we've been playing with in this section:

406378:Sales:Itorre:Jan
031762:Marketing:Nasium:Jim
636496:Research:Ancholie:Mel
396082:Sales:Jucacion:Ed

If you want to print just columns 1 to 6 of each line (the employee serial numbers), use the **-c1-6** flag, as in this command:

cut -c1-6 company.data

406378
031762

636496
396082

If you want to print just columns 4 and 8 of each line (the first letter of the department and the fourth digit of the serial number), use the **-c4,8** flag, as in this command:

cut -c4,8 company.data

3S
7M
4R
0S

And since this file obviously has fields delimited by colons, we can pick out just the last names by specifying the **-d:** and **-f3** flags, like this:

cut -d: -f3 company.data

Itorre
Nasium
Ancholie
Jucacion

It's often the case that you want to use a space as the delimiter. To do so, you must put the delimiter in single quotes, like this: **-d' '**

Also, when you want to cut from a starting point to the end of the line, just leave off the final field number, as shown in the demo below.

Let's say this is your test.txt file:

abc def ghi jkl

mno pqr stu vwx

yz1 234 567 890

To cut only columns 2-END, do this: **cut -d' ' -f2- test.txt**

And the results are:

def ghi jkl

pqr stu vwx

234 567 890

Here is a summary of the most common flags for the **cut** command:

-c [n | n,m | n-m] Specify a single column, multiple columns (separated by a comma), or range of columns (separated by a dash).

-f [n | n,m | n-m] Specify a single field, multiple fields (separated by a comma), or range of fields (separated by a dash).

-dc Specify the field delimiter.

-s Suppress (don't print) lines not containing the delimiter.

Paste

paste command is used to paste the content from one file to another file. It is also used to set column format for each line.

The Syntax is
paste [options]

OPTIONS:

-s	Paste one file at a time instead of in parallel.
-d	Reuse characters from LIST instead of TABs .

SED (Stream Editor)

You can use the **sed** command to change all occurrences of one string to another within a file, just like the search-and-replace feature of your word processor. The **sed** command can also delete a range of lines from a file. Since **sed** is a stream editor, it takes the file given as input, and sends the output to the screen, unless you redirect output to a file. In other words, **sed** does not change the input file.

The general forms of the **sed** command are as follows:

Substitution sed 's/<oldstring>/<newstri ng>/g' <file>

Deletion sed '<start>,<end>d' < file>

Let's start with a substitution demo. If you want to change all occurrences of raga to hama in the **poem.txt** file in the **grep**demo, enter this:

sed 's/raga/hama/g' poem.txt

Mary had a little hama

Mary fried a lot of spam

Jack ate a Spam sandwich

Jill had a hama spamwich

In the quoted string, the "s" means substitute, and the "g" means make a global change. You can also leave off the "g" (to change only the first occurrence on each line) or specify a number instead (to change the first n occurrences on each line).

Now let's try an demo involving deletion of lines. The values for **start** and **end** can be either a line number or a pattern to match. All lines from the start line to the end line are removed from the output. This demo will delete starting at line 2, up to and including line 3:

sed '2,3d' poem.txt

Mary had a little raga

Jill had a raga spamwich

This demo will delete starting at line 1, up to and including the next line containing Jack:

sed '1,/Jack/d' poem.txt

Jill had a raga spamwich

The most common use of **sed** is to change one string of text to another string of text. But I should mention that the strings that **sed** uses for search and delete are actually regular expressions. This means you can use pattern matching, just as with **grep**. Although you'll probably never need to do anything like this, here's an demo anyway. To change any occurrences of raga at the end of a line to hama, and save the results in a new file, enter this:

sed 's/raga#/hama/g' poem.txt > new.file

Since we directed output to a file, **sed** didn't print anything on the screen. If you look at the contents of **new.file** it will show these lines:

Mary had a little hama

Mary fried a lot of spam

Jack ate a Spam sandwich

Jill had a raga spamwich

Sort

The **sort** command sorts a file according to fields--the individual pieces of data on each line. By default, **sort** assumes that the fields are just words separated by blanks, but you can specify an alternative field delimiter if you want (such as commas or colons). Output from **sort** is printed to the screen, unless you redirect it to a file.

If you had a file like the one shown here containing information on people who contributed to your presidential reelection campaign, for demo, you might want to sort it by last name, donation amount, or location. (Using a text editor, enter those three lines into a file and save it with **donor.data** as the file name.)

Bay Ching 500000 China

Jack Arta 250000 Indonesia

Cruella Lumper 725000 Malaysia

Let's take this sample donors file and sort it according to the donation amount. The following shows the command to sort the file on the second field (last name) and the output from the command:

sort +1 -2 donors.data

Jack Arta 250000 Indonesia

Bay Ching 500000 China

Cruella Lumper 725000 Malaysia

The syntax of the **sort** command is pretty strange, but if you study the following demos, you should be able to adapt one of them for your own use. The general form of the sort command is

sort <flags> <sort fields> <file name>

The most common flags are as follows:

-f Make all lines uppercase before sorting (so "Bill" and "bill" are treated the same).

-r Sort in reverse order (so "Z" starts the list instead of "A").

-n Sort a column in numerical order

-tx Use x as the field delimiter (replace x with a comma or other character).

-u Suppress all but one line in each set of lines with equal sort fields (so if you sort on a field containing last names, only one "Smith" will appear even if there are several).

Specify the sort keys like this:

+m Start at the first character of the m+1th field.

-n End at the last character of the nth field (if -N omitted, assume the end of the line).

Looks weird, huh? Let's look at a few more demos with the sample **company.data** file shown here, and you'll get the hang of it. (Each line of the file contains four fields: first name, last name, serial number, and department name.)

Jan Itorre 406378 Sales

Jim Nasium 031762 Marketing

Mel Ancholie 636496 Research

Ed Jucacion 396082 Sales

To sort the file on the third field (serial number) in reverse order and save the results in **sorted.data**, use this command:

sort -r +2 -3 company.data > sorted.data

Mel Ancholie 636496 Research

Jan Itorre 406378 Sales

Ed Jucacion 396082 Sales

Jim Nasium 031762 Marketing

Now let's look at a situation where the fields are separated by colons instead of spaces. In this case, we will use the **-t:** flag to tell the **sort** command how to find the fields on each line. Let's start with this file:

Itorre, Jan:406378:Sales

Nasium, Jim:031762:Marketing

Ancholie, Mel:636496:Research

Jucacion, Ed:396082:Sales

To sort the file on the second field (serial number), use this command:

sort -t: +1 -2 company.data

Nasium, Jim:031762:Marketing

Jucacion, Ed:396082:Sales

Itorre, Jan:406378:Sales

Ancholie, Mel:636496:Research

To sort the file on the third field (department name) and suppress the duplicates, use this command:

sort -t: -u +2 company.data

Nasium, Jim:031762:Marketing

Ancholie, Mel:636496:Research

Itorre, Jan:406378:Sales

Note that the line for Ed Jucacion did not print, because he's in Sales, and we asked the command (with the **-u** flag) to suppress lines that were the same in the **sort** field.

Investigating and Managing Processes (crontab zip tar)

tar

Sometimes it is necessary to store a set of files and directories in a single file called an "archive". This is useful for backing up files and for transmitting them over a network (email, web, ftp, etc.). In Linux, the tar command is used to create archive files, and to extract files/directories from archive files. Archive files created by tar are called "tar files", and frequently have a name with the .tar extension. Tar can be used to perform one of three primary tasks:

- Create a new archive file
- Extract files from an existing archive file
- List the contents of an existing archive file

The tar command-line has the following form:

tar options tar-file other-files

options is a string of characters that tells tar which options to use. Possible options are the following:

- You will always specify either **c**, **x**, or **t** (c = create, x = extract, t = list).
- The **v** option requests verbose output, and causes tar to print file names as they are added or extracted
- The **z** option tells tar to use gzip compression
- The **f** option is used to specify the name of the tar file

tar-file is the name of the tar file

other-files is a list of the files and directories that you want to include in the archive Here are some demos that show how to use tar to perform some common tasks.

create a tar file containing the source files for myfile

```
# ls
myfile.cpp myfile.h myfile.html
# tar cf myfile.tar *
# ls
myfile.cpp myfile.h myfile.html myfile.tar
```

list the contents of myfile.tar

```
# tar tf myfile.tar
myfile.cpp
myfile.h
myfile.html
```

extract the myfile source files

```
# rm *.cpp *.h *.html
# ls
myfile.tar
# tar xf myfile.tar
# ls
myfile.cpp myfile.h myfile.html myfile.tar
```

create a compressed a tar file containing the entire myfile directory

```
# cd ..
# ls
myfile
# tar czf myfile.tgz myfile
# ls
myfile myfile.tgz


### list the contents of myfile.tgz

# tar tzf myfile.tgz
myfile/myfile.cpp
myfile/myfile.h
myfile/myfile.html


### extract the myfile directory

# rm -R myfile
# ls
myfile.tgz
# tar xzf myfile.tgz
# ls
myfile myfile.tgz
```

COMPRESSION AND DECOMPRESSION

gzip

gzip does file compression/decompression, but not archiving. However, it works well when used together with an archiving tool - in particular, gzip and tar play nicely together.

gzip, like zip, uses a variation on the DEFLATE algorithm. However, when used on a tar archive, it'll compress better than zip, because it treats the whole archive together, rather than compressing file-by-file, and thus can look for (and eliminate!) duplication between files as well as within them.

Gzip commands:

- gzip file.tar
- compresses file.tar to create file.tar.gz.
- tar czf file.tar.gz mydir/
- archives and compresses mydir/ to create file.tar.gz
- gunzip file.tar.gz or gzip -d file.tar.gz
- uncompress file.tar.gz.
- tar zxf file.tar.gz
- uncompress file.tar.gz and also unpack the archive.

Note that you can also use gzcat (use exactly as gunzip) to decompress directly to standard output.

bzip2

bzip2, like gzip, does compression but not archiving. Again, it works well with tar if you want both archiving and compression. It compresses better than gzip, but the cost of that is that it takes longer to run.

It uses a series of stacked transforms to compress data, the most important of which is the Burrows-Wheeler transform. This reorders blocks to maximise the number of repeated symbols. Runs of repeated symbols can then be replaced by the symbol and a code which represents the length of the run, reducing the size taken up by each run. The Huffman coding used in DEFLATE (where common symbols are swapped in for shorter versions of them) is also used after that, to reduce the

space still further, along with a couple of other more minor changes. Given all of that, the time it takes is hardly surprising!

Bzip2 commands:

- bzip2 file.tar
- compresses file.tar to create file.tar.bz2.
- bunzip2 file.tbz
- uncompresses file.tbz to create file.tar.
- tar jxf file.tbz
- uncompresses file.tbz and unpacks the archive.

Note that you can also use bzcat to decompress directly to standard output.

If you're interested in compression algorithms, you could also check out 7-zip (the Linux command-line version is p7zip), which implements the LZM algorithm. In most cases it compresses better than bzip2, and it also decompresses faster (although it compresses slower). However, it's much less popular (I've never seen it in the wild).

CRONTAB

The crontab (cron derives from chronos, Greek for time; tab stands for table) command, found in Unix and Unix-like operating systems, is used to schedule commands to be executed periodically. To see what crontabs are currently running on your system, you can open a terminal and run:

crontab -e

This wil open a the default editor (could be vi or pico, if you want you can change the default editor) to let us manipulate the crontab. If you save and exit the editor, all your cronjobs are saved into crontab. Cronjobs are written in the following format:

```
* * * * * /bin/execute/this/script.sh
```

Scheduling explained

As you can see there are 5 stars. The stars represent different date parts in the following order:

1. minute (from 0 to 59)
2. hour (from 0 to 23)
3. day of month (from 1 to 31)
4. month (from 1 to 12)
5. day of week (from 0 to 6) (0=Sunday)

Execute every minute

If you leave the star, or asterisk, it means **every**. Maybe that's a bit unclear. Let's use the the previous demo again:

```
* * * * * /bin/execute/this/script.sh
```

They are all still asterisks! So this means execute /bin/execute/this/script.sh:

1. **every** minute
2. of **every** hour
3. of **every** day of the month
4. of **every** month
5. and **every** day in the week.

In short: This script is being executed every minute. Without exception.

Execute every Friday 1AM

So if we want to schedule the script to run at 1AM every Friday, we would need the following cronjob:

```
0 1 * * 5 /bin/execute/this/script.sh
```

Get it? The script is now being executed when the system clock hits:

1. minute: 0
2. of hour: 1
3. of day of month: * (every day of month)
4. of month: * (every month)
5. and weekday: 5 (=Friday)

Execute on workdays 1AM

So if we want to schedule the script to Monday till Friday at 1 AM, we would need the following cronjob:

```
0 1 * * 1-5 /bin/execute/this/script.sh
```

Get it? The script is now being executed when the system clock hits:

1. minute: 0
2. of hour: 1
3. of day of month: * (every day of month)
4. of month: * (every month)
5. and weekday: 1-5 (=Monday til Friday)

Figure 29 crontab

STANDARD INPUT/OUTPUT

C++ programs can read input from the keyboard. This is called “standard input”. C++ programs can also write output to the screen (i.e., the shell). This is called “standard output”.

Many Linux programs accept file names as command-line arguments (e.g., ls, cp, mv, rm, wc, etc.). If you don't provide any file names on the command-line, these programs will read their input from standard input (i.e., the keyboard). This allows the user to interactively type the input to the program rather than having it read from a file. For demo, the following command will count the number of lines typed in by the user (CTRL-D is used to terminate interactive input).

count the number of lines typed, hit CTRL-D to end

wc -l

Because no file name was provided on the command-line, wc will take its input from the keyboard. Once CTRL-D is hit, wc knows that the user is done typing.

STANDARD I/O REDIRECTION

The shell lets you redirect a program’s standard input so that it comes from a file instead of from the keyboard. The command below instructs the shell to run the program "myfile" with its standard input coming from a file named "file.input". This means that whenever "myfile" reads input from the keyboard, it will actually be reading from "file.input".

myfile < file.input

The shell also lets you redirect a program’s standard output so that it goes to a file instead of the screen. The command below instructs the shell to run the program "myfile" with its standard output going to a file named "file.output". This means that whenever "myfile" writes output to the screen, the output will actually go to "file.output" instead of the screen.

overwrite the output file

myfile > file.output

append to the output file

myfile >> file.output

You can also redirect both standard input and standard output at the same time, as shown below.

myfile < file.input > file.output

PIPELINES

The shell allows you to run multiple commands simultaneously, with the output of one command becoming the input to the next command. This allows commands to be assembled into a "pipeline" where each program reads input, transforms the data, and then sends the transformed data to the next command in the pipeline for further processing.

For demo, suppose that you have a text file and you want to count the number of lines in the file that contain the text "BYU". The grep command can be used to find the lines that contain "BYU", but it doesn't know how to count them. Similarly, the wc command can be used to count lines, but it doesn't know how to find only the lines that contain "BYU". By running grep and wc together in a pipeline, we can achieve the desired effect.

grep ’BYU’ file.txt | wc –l

First, run grep on the file, searching for the pattern "BYU". Take the output of grep and connect it to the input of wc. This will cause grep to find only those lines containing "BYU" and send them to wc. When wc receives the "BYU" lines from grep, it will count them and print out the answer.

As shown above, multiple commands may be assembled into a pipeline by placing | between the commands. This tells the shell to run the programs so that the standard output of one program becomes the standard input of the next program, thus connecting the programs together. The first command in the pipeline will read its input from the keyboard or a file. The last command in the pipeline will write its output to the screen or a file. All commands in the middle of the pipeline will read their input from the previous command, and write their output to the next command.

Of course, command pipelines may contain more than two commands. For demo, instead of counting "BYU" lines, we might want to sort them and send them to a printer.

grep 'BYU' file.txt | sort | print

Pipelines work because programs like grep, wc, sort, and print read their input from standard input if no file name is specified on the command line.

Basic Networking

Under Red Hat Enterprise Linux, all network communications occur between configured software interfaces and physical networking devices connected to the system.

The configuration files for network interfaces are located in the /etc/sysconfig/network-scripts/ directory. The scripts used to activate and deactivate these network interfaces are also located here. Although the number and type of interface files can differ from system to system, there are three categories of files that exist in this directory:

1. Interface configuration files
2. Interface control scripts
3. Network function files

The files in each of these categories work together to enable various network devices.

This part explores the relationship between these files and how they are used.

Network Configuration Files

Before delving into the interface configuration files, let us first itemize the primary configuration files used in network configuration. Understanding the role these files play in setting up the network stack can be helpful when customizing a Red Hat Enterprise Linux system.

The primary network configuration files are as follows:

/etc/hosts

> The main purpose of this file is to resolve hostnames that cannot be resolved any other way. It can also be used to resolve hostnames on small networks with no DNS server. Regardless of the type of network the computer is on, this file should contain a line

specifying the IP address of the loopback device (127.0.0.1) as localhost.localdomain. For more information, refer to the hosts man page.

/etc/resolv.conf

This file specifies the IP addresses of DNS servers and the search domain. Unless configured to do otherwise, the network initialization scripts populate this file. For more information about this file, refer to the resolv.conf man page.

/etc/sysconfig/network

This file specifies routing and host information for all network interfaces.

/etc/sysconfig/network-scripts/ifcfg-<interface-name>

For each network interface, there is a corresponding interface configuration script. Each of these files provide information specific to a particular network interface.

Ethernet Interfaces

One of the most common interface files is ifcfg-eth0, which controls the first Ethernet network interface card or NIC in the system. In a system with multiple NICs, there are multiple ifcfg-eth<X> files (where <X> is a unique number corresponding to a specific interface). Because each device has its own configuration file, an administrator can control how each interface functions individually.

The following is a sample ifcfg-eth0 file for a system using a fixed IP address:

```
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
NETWORK=10.0.1.0
NETMASK=255.255.255.0
```

IPADDR=10.0.1.27
USERCTL=no

The values required in an interface configuration file can change based on other values. For demo, the ifcfg-eth0 file for an interface using DHCP looks different because IP information is provided by the DHCP server:

DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes

The Network Administration Tool (system-config-network) is an easy way to make changes to the various network interface configuration files. However, it is also possible to manually edit the configuration files for a given network interface.

Below is a listing of the configurable parameters in an Ethernet interface configuration file:

BOOTPROTO=<protocol>

where <protocol> is one of the following:

- none — No boot-time protocol should be used.
- bootp — The BOOTP protocol should be used.
- dhcp — The DHCP protocol should be used.

BROADCAST=<address>

where <address> is the broadcast address. This directive is deprecated, as the value is calculated automatically with ifcalc.

DEVICE=<name>

where <name> is the name of the physical device (except for dynamically-allocated PPP devices where it is the logical name).

DHCP_HOSTNAME

Use this option only if the DHCP server requires the client to specify a hostname before receiving an IP address.

DNS{1,2}=<address>

where <address> is a name server address to be placed in /etc/resolv.conf if the PEERDNS directive is set to yes.

ETHTOOL_OPTS=<options>

where <options> are any device-specific options supported by ethtool. For demo, if you wanted to force 100Mb, full duplex:

ETHTOOL_OPTS="autoneg off speed 100 duplex full"

GATEWAY=<address>

where <address> is the IP address of the network router or gateway device (if any).

HWADDR=<MAC-address>

where <MAC-address> is the hardware address of the Ethernet device in the form AA:BB:CC:DD:EE:FF. This directive is useful for machines with multiple NICs to ensure that the interfaces are assigned the correct device names regardless of the configured load order for each NIC's module. This directive should not be used in conjunction with MACADDR.

IPADDR=<address>

where <address> is the IP address.

MACADDR=<MAC-address>

where <MAC-address> is the hardware address of the Ethernet device in the form AA:BB:CC:DD:EE:FF. This directive is used to assign a MAC address to an interface, overriding the one assigned to the physical NIC. This directive should not be used in conjunction with HWADDR.

MASTER=<bond-interface>

where <bond-interface> is the channel bonding interface to which the Ethernet interface is linked.

This directive is used in conjunction with the SLAVE directive.

NETMASK=<mask>

where <mask> is the netmask value.

NETWORK=<address>

where <address> is the network address. This directive is deprecated, as the value is calculated automatically with ifcalc.

ONBOOT=<answer>

where <answer> is one of the following:

- yes — This device should be activated at boot-time.
- no — This device should not be activated at boot-time.

PEERDNS=<answer>

where <answer> is one of the following:

- yes — Modify /etc/resolv.conf if the DNS directive is set. If using DHCP, then yes is the default.
- no — Do not modify /etc/resolv.conf.

SLAVE=<bond-interface>

where <bond-interface> is one of the following:

- yes — This device is controlled by the channel bonding interface specified in the MASTER directive.
- no — This device is not controlled by the channel bonding interface specified in the MASTER directive.

This directive is used in conjunction with the MASTER directive.

SRCADDR=<address>

where <address> is the specified source IP address for outgoing packets.

USERCTL=<answer>

where <answer> is one of the following:

- yes — Non-root users are allowed to control this device.
- no — Non-root users are not allowed to control this device.

The interface control scripts

The interface control scripts activate and deactivated system interfaces. There are two primary interface control scripts that call on control scripts located in the /etc/sysconfig/network-scripts/ directory: /sbin/ifdown and /sbin/ifup.

The ifup and ifdown interface scripts are symbolic links to scripts in the /sbin/ directory. When either of these scripts are called, they require the value of the interface to be specified, such as:

ifup eth0

Command we use to open the Network-setting dialog box in text mode **setup.**

Advance Topics in User's , group's and Permissions

Advanced File Permissions in Linux

Here we will discuss about the 3 special attributes other than the common read/write/execute.
Demo:
drwxrwxrwt - Sticky Bits - chmod 1777

drwsrwxrwx - SUID set - chmod 4777

drwxrwsrwx - SGID set - chmod 2777

Sticky bit

Sticky bits are mainly set on directories. If the sticky bit is set for a directory, only the owner of that directory or the owner of a file can delete or rename a file within that directory.

Demo:
Consider you have a directory " test ".

chmod it to " 777 ". This gives permissions for all the users to read, write and execute.
chmod +t test

Demo: ls –al

drwxrwxrwt 2 a1 a1 4096 Jun 13 2008

-rw-rw-r-- 1 a1 a1 0 Jun 11 17:30 1.txt

-rw-rw-r-- 1 b2 b2 0 Jun 11 22:52 2.txt

From the above demo a1 is the owner of the test directory.

a1 can delete or rename the files 1.txt and 2.txt.

b2 can delete or rename the file 2.txt only.

SUID - [Set User ID]

SUID bit is set for files (mainly for scripts). The SUID permission makes a script to run as the user who is the owner of the script, rather than the user who started it.

Demo:
If a1 is the owner of the script and b2 tries to run the same script, the script runs with the ownership of a1.

If the root user wants to give permissions for some scripts to run by different users, he can set the SUID bit for that particular script.

So if any user on the system starts that script, it will run under the root ownership.

SGID - [Set Group ID]

If a file is SGID, it will run with the privileges of the files group owner, instead of the privileges of the person running the program. This permission set also can make a similar impact. Here the script runs under the groups ownership.

You can also set SGID for directories.

Consider you have given 2777 permission for a directory. Any files created by any users under this directory will come as follows.

Demo:
-rw-rw-r-- 1 b2 a1 0 Jun 11 17:30 1.txt

In the above demo you can see that the owner of the file 1.txt is b2 and the group owner is a1.
So both b2 and a1 will have access to the file 1.txt.

Now lets make this more intresting and complicated.

Create a directory "test". Chmod it to 2777. Add sticky bit to it.

Demo:
mkdir test

chmod 2777 test

chmod +t test

ls -al test

drwxrwsrwt 2 a1 a1 4096 Jun 13 2008 test

From the above permission set you can understand that SGID and sticky bit is set for the folder "test".

Now any user can create files under the test directory.

Demo:
drwxrwsrwt 2 a1 a1 4096 Jun 13 2008

-rw-rw-r-- 1 b2 a1 0 Jun 11 17:30 1.txt

-rw-rw-r-- 1 c3 a1 0 Jun 11 17:30 2.txt

-rw-rw-r-- 1 d4 a1 0 Jun 11 17:30 3.txt

So all the a1 user has access to all the files under the test directory. He can edit, rename or remove the file.

b2 user has access to 1.txt only, c3 has access to 2.txt only...

If sticky bit was not set for the test directory, any user can delete any files from the test directory, since the test directory has 777 permissions.

But now it not possible.

Demo:
If d4 tries to remove 1.txt

rm -f 1.txt

rm: cannot remove `1.txt': Operation not permitted

INODE NUMBER'S

The **inode (index node)** is a fundamental concept in the Linux and UNIX filesystem. Each object in the filesystem is represented by an inode. But what are the objects? Let us try to understand it in simple words. Each and every file under Linux (and UNIX) has following attributes:

1. File type (executable, block special etc)
2. Permissions (read, write etc)
3. Owner
4. Group
5. File Size
6. File access, change and modification time (remember UNIX or Linux never stores file creation time,this is favorite question asked in UNIX/Linux sys admin job interview)
7. File deletion time

8. Number of links (soft/hard)

9. Extended attribute such as append only or no one can delete file including root user (immutability)

10. Access Control List (ACLs)

All the above information stored in an inode. In short the inode identifies the file and its attributes (as above) . Each inode is identified by a unique inode number within the file system. Inode is also know as index number.

inode definition

An inode is a data structure on a traditional Unix-style file system such as UFS or ext3. An inode stores basic information about a regular file, directory, or other file system object.

How do I see file inode number?

You can use ls -i command to see inode number of file

$ ls -i /etc/passwd

Sample Output

32820 /etc/passwd

You can also use stat command to find out inode number and its attribute:

$ stat /etc/passwdOutput:

File: `/etc/passwd'
Size: 1988 Blocks: 8 IO Block: 4096 regular file
Device: 341h/833d Inode: 32820 Links: 1
Access: (0644/-rw-r--r--) Uid: (0/ root) Gid: (0/ root)
Access: 2005-11-10 01:26:01.000000000 +0530
Modify: 2005-10-27 13:26:56.000000000 +0530

Change: 2005-10-27 13:26:56.000000000 +0530

Package management

RPM (RedHat Package Management)

The RPM Package Manager (RPM) is a powerful command line driven package management system capable of installing, uninstalling, verifying, querying, and updating computer software packages. Each software package consists of an archive of files along with information about the package like its version, a description, and the like. There is also a library API, permitting advanced developers to manage such transactions from programming languages such as C or Python.

RPM is free software, released under the GNU GPL.

RPM is a core component of many Linux distributions, such as Red Hat Enterprise Linux, the Fedora Project, SUSE Linux Enterprise, openSUSE, CentOS, Meego, Mageia and many others.

It is also used on many other operating systems as well, and the RPM format is part of the Linux Standard Base.

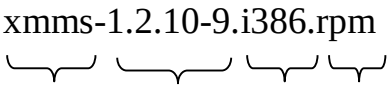
To install RPM Package:

1. Log in as root user at the workstation on which you want to install the software.
2. Download the package you wish to install. The package will be named something like gcc.401.i386.rpm.
3. To install the package, enter the following command at the prompt:

   rpm -i gcc.401.i386.rpm

   If you are upgrading from an earlier version of the software package, run RPM in upgrade mode, as in the following demo:

rpm -U gcc.401.i386.rpm



rpm <options> <package name>
rpm -ivh <package name>

--aid (install package along with dependencies)
--force (forcefully)
--nodebs (to remove package along with dependencies)
rpm -e <package name> (To uninstall package)
-U (upgrade the package)
rpm -q <package name> (Show whether package is install or not)
-qa (queries all installed rpm packages)
-qc (lists only the configuration files stored in the queried package)
-qd (lists only the documentation files stored in the queried rpm)
-qi (displays whole information about the queried rpm)
-qs (displays the states of files in the queried rpm)
-ql (displays all files related to the queried rpm)

SSH (Secure SHell)

There are a couple of ways that you can access a shell (command line) remotely on most Linux/Unix systems. One of the older ways is to use the telnet program, which is available on most network capable operating systems. Accessing a shell account through the telnet method though poses a danger in that everything that you send or receive over that telnet session is visible in plain text on your local network, and the local network of the machine you are connecting to. So anyone who can "sniff" the connection in between can see your username, password, email that you read,

and commands that you run. For these reasons you need a more sophisticated program than telnet to connect to a remote host.

SSH, which is an acronym for Secure SHell, was designed and created to provide the best security when accessing another computer remotely. Not only does it encrypt the session, it also provides better authentication facilities, as well as features like secure file transfer, X session forwarding, port forwarding and more so that you can increase the security of other protocols. It can use different forms of encryption ranging anywhere from 512 bit on up to as high as 32768 bits and includes ciphers like AES (Advanced Encryption Scheme), Triple DES, Blowfish, CAST128 or Arcfour. Of course, the higher the bits, the longer it will take to generate and use keys as well as the longer it will take to pass data over the connection.

Sometimes it may be necessary to identify the SSH client that you are currently running and it's corresponding version number, which can be identified as shown below. Please note that Linux comes with OpenSSH.

$ ssh -V

OpenSSH_3.9p1, OpenSSL 0.9.7a Feb 19 2003

$ ssh -V

ssh: SSH Secure Shell 3.2.9.1 (non-commercial version) on i686-pc-linux-gnu

Login to remote host:

The First time when you login to the remotehost from a localhost, it will display the host key not found message and you can give "yes" to continue. The host key of the remote host will be added under .ssh2/hostkeys directory of your home directory, as shown below.

localhost$ ssh -l demo remotehost.demo.com

Host key not found from database.
Key fingerprint:
xabie-dezbc-manud-bartd-satsy-limit-nexiu-jambl-title-jarde-tuxum
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Are you sure you want to continue connecting (yes/no)? yes
Host key saved to /home/demo/.ssh2/hostkeys/key_22_remotehost.demo.com.pub
host key for remotehost.demo.com, accepted by demo Mon Apr 26 2010 17:05:50 -0700

demo@remotehost.demo.com password:
remotehost.demo.com$

The Second time when you login to the remote host from the localhost, it will prompt only for the password as the remote host key is already added to the known hosts list of the ssh client.

localhost$ **ssh -l demo remotehost.demo.com**
demo@remotehost.demo.com password:
remotehost.demo.com$

For some reason, if the host key of the remote host is changed after you logged in for the first time, you may get a warning message as shown below. This could be because of various reasons such as 1) Sysadmin upgraded/reinstalled the SSH server on the remote host 2) someone is doing malicious activity etc., The best possible action to take before saying "yes" to the message below, is to call your sysadmin and identify why you got the host key changed message and verify whether it is the correct host key or not.

localhost$ **ssh -l demo remotehost.demo.com**

@@@
@@@@@

@ WARNING: HOST IDENTIFICATION HAS CHANGED! @

@@@
@@@@@

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!

Someone could be eavesdropping on you right now (man-in-the-middle attack)!

It is also possible that the host key has just been changed.

Please contact your system administrator.

Add correct host key to "/home/demo/.ssh2/hostkeys/key_22_remotehost.demo.com.pub"

to get rid of this message.

Received server key's fingerprint:

xabie-dezbc-manud-bartd-satsy-limit-nexiu-jambl-title-jarde-tuxum

You can get a public key's fingerprint by running

% ssh-keygen -F publickey.pub

on the keyfile.

Agent forwarding is disabled to avoid attacks by corrupted servers.

Are you sure you want to continue connecting (yes/no)? **yes**

Do you want to change the host key on disk (yes/no)? **yes**

Agent forwarding re-enabled.

Host key saved to /home/demo/.ssh2/hostkeys/key_22_remotehost.demo.com.pub

host key for remotehost.demo.com, accepted by demo Mon May 26 2008 16:17:31 -0700

demo @remotehost.demo.com's password:

remotehost$

<u>File transfer to/from remote host</u>:

Another common use of ssh client is to copy files from/to remote host using scp.

Copy file from the remotehost to the localhost:

localhost$ **scp demo@remotehost.demo.com:/home/demo/remotehostfile.txt remotehostfile.txt**

Copy file from the localhost to the remotehost:

localhost$ **scp localhostfile.txt demo@remotehost.demo.com:/home/demo/localhostfile.txt**

Debug SSH Client:

Sometimes it is necessary to view debug messages to troubleshoot any SSH connection issues. For this purpose, pass -v (lowercase v) option to the ssh as shown below.

Demo without debug message:

localhost$ **ssh -l demo remotehost.demo.com**
warning: Connecting to remotehost.demo.com failed: No address associated to the name
localhost$

Demo with debug message:

locaclhost$ **ssh -v -l demo remotehost.demo.com**
debug: SshConfig/sshconfig.c:2838/ssh2_parse_config_ext: Metaconfig parsing stopped at line 3.
debug: SshConfig/sshconfig.c:637/ssh_config_set_param_verbose: Setting variable 'VerboseMode' to 'FALSE'.

debug: SshConfig/sshconfig.c:3130/ssh_config_read_file_ext: Read 17 params from config file.

debug: Ssh2/ssh2.c:1707/main: User config file not found, using defaults. (Looked for '/home/demo/.ssh2/ssh2_config')

debug: Connecting to remotehost.demo.com, port 22... (SOCKS not used)

warning: Connecting to remotehost.demo.com failed: No address associated to

SUDO

sudo allows a permitted user to execute a command as the superuser or another user, as specified in the sudoers file. The real and effective uid and gid are set to match those of the target user as specified in the passwd file (the group vector is also initialized when the target user is not root). By default, **sudo** requires that users authenticate themselves with a password (NOTE: by default this is the user's password, not the root password). Once a user has been authenticated, a timestamp is updated and the user may then use sudo without a password for a short period of time (5 minutes unless overridden in sudoers).

The file /etc/sudoers, has the rules that users have to follow when using sudo command.

Two of the best advantages about using sudo are:

- Restringed privileges
- Logs of the actions done by users

Well but in order to use sudo we first need to configure the sudoers file.

"Do not edit directly the file"

To edit it, use the command

visudo

You will see a file more or less like this.

```
# /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the man page for details on how to write a sudoers file.
#

Defaults        env_reset

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL) ALL
```

As you can see there is basically one line

root ALL=(ALL) ALL

This lines means that the user root can execute from ALL terminals, acting as ALL (any) users, and run ALL (any) command.

So the first part is the user, the second is the terminal from where the user can use sudo, the third is as which user he may act, and the last one, is which commands he may run.

Let's see some other examples.

User ALL= /sbin/poweroff

This makes that users user can from any terminal, run the command poweroff.

You can also create aliases for: users

These are some examples:

User_Alias DEMO = demo

Runas_Alias OP = root, operator

Host_Alias OFNET = 10.1.2.0/255.255.255.0

Cmnd_Alias PRINTING = /usr/sbin/lpc, /usr/bin/lprm

As you can see the alias DEMO include the users demo, the alias OP includes the users root and operator, alias OFNET includes the network 10.1.2.0 (all the C class), and the command alias PRINTING includes the commands lpc and lprm.

So a typical sudoers file may look like this.

```
User_Alias      DEMO = demo
Runas_Alias     OP = root, operator
Host_Alias      OFNET = 10.1.2.0/255.255.255.0
Cmnd_Alias      PRINTING = /usr/sbin/lpc, /usr/bin/lprm

DEMO ALL=ALL

#The users in the DEMO group can run any command from
 any terminal.

linus ALL=(OP) ALL

# The user linus can run any command from any terminal as any
user in the OP group (root or operator).
```

user2 OFNET=(ALL) ALL

user user2 may run any command from any machine in the
OFNET network, as any user.

user3 ALL= PRINTING

user user3 may run lpc and lprm from any machine.

go2linux ALL=(ALL) ALL

YUM (Yellowdog Updater Modified)

The **Yellowdog Updater, Modified** (**YUM**) is an open-source command-line package-management utility for RPM-compatible Linux operating systems and has been released under the GNU General Public License. It was developed by Seth Vidal and a group of volunteer programmers. Though yum has a command-line interface, several other tools provide graphical user interfaces to yum functionality.

The Yellowdog Updater, Modified. Yum is an automatic updater and package installer/remover for rpm-based systems. It automatically computes dependencies and figures out what things should occur in order to safely install, remove, and update rpm packages. Yum also efficiently and easily retrieves information on any package installed or available in a repository to the installer. Yum makes it easier to maintain groups of machines without having to manually update each one using rpm or other tools. Yum can manage package groups, multiple repositories, fallback repositories and more to permit centralized package management performed by just one or two individuals to scale over an entire organization.

Now to create a YUM in your Linux Server

1. Create a directory named **repo**
   [root@localhost ~]# mkdir /repo
2. Insert DVD of rhel6 then Mount CD ROM
   [root@localhost ~]# mount /dev/cdrom /mnt
3. Copy directory named Packages - from DVD of RHEL-6
   [root@localhost ~]# cd /mnt
   [root@localhost mnt# cp -Rvf Server Packages /repo
4. Now create a file named anyname.repo
   # vi /etc/yum.repos.d/base.repo

   [base]
   name= base

baseurl=file:///repo/Packages

enabled=1

gpgcheck=0

(save the file)

5. Import GPG Key from DVD

   # rpm --import /mnt/RPM-GPG*
6. Now we have to group files to create repodata
7. Install standalone service

   # rpm -ivh /mnt/Packages/createrepo*

   It will ask for dependency named deltaparm & python-deltaparm, install both of the packages.
8. Now we have to create repository

   [root@localhost ~]# createrepo -v /repo/Packages
9. **# yum clean all**
10. **# yum list all**
11. Now install package from command

    # yum install <package name>
12. To uninstall package

    # yum remove <package name>
13. Now on graphics you will find add/remove program and all packages are there to install.

Printing in Linux

From the command line, use the **lp** or **lpr** command.

lp file(s)

lpr file(s)

These commands can read from a pipe, so you can print the output of commands using

command | lp

There are many options available to tune the page layout, the number of copies, the printer that you want to print to if you have more than one available, paper size, one-side or double-sided printing if your printer supports this feature, margins and so on.

Status of your print jobs

Once the file is accepted in the print queue, an identification number for the print job is assigned:

```
[root@demo.com ~]# lp /etc/profile
 request id is Printer -253 (1 file(s))
```

To view (query) the print queue, use the **lpq** or **lpstat** command. When entered without arguments, it displays the contents of the default print queue.

```
[root@demo.com ~]# lpq

 Printer  is ready and printing
 Rank      Owner    Job      File(s)              Total Size
 active    root     253      profile              1024 bytes

[root@demo.com ~]# lpstat
 Printer -253          root      1024      Tue 25 Jul 2006 10:20_01 AM IST
```

Status of your printer

```
[root@demo.com ~]#lpstat -d
 system default destination: Printer
```

What is the status of my printer(s)?

```
[root@demo.com ~]# lpstat -p
 printer Printer  now printing Printer -253. enabled since Jan 01 18:01
```

Removing jobs from the print queue

If you don't like what you see from the status commands, use **lprm** or **cancel** to delete jobs.

```
[root@demo.com ~]# lprm 253
```

In the graphical environment, you may see a popup window telling you that the job has been canceled.

In larger environments, **lpc** may be used to control multiple printers. See the Info or man pages on each command.

System Monitoring

Being able to monitor the performance of your system is essential. If system resources become to low it can cause a lot of problems. System resources can be taken up by individual users, or by services your system may host such as email or web pages. The ability to know what is happening can help determine whether system upgrades are needed, or if some services need to be moved to another machine.

The top command.

The most common of these commands is **top**. The **top** will display a continually updating report of system resource usage.

```
# top
 12:10:49  up 1 day,  3:47,  7 users,  load average: 0.23, 0.19, 0.10
125 processes: 105 sleeping, 2 running, 18 zombie, 0 stopped
CPU states:   5.1% user   1.1% system   0.0% nice   0.0% iowait  93.6% idle
Mem:   512716k av,  506176k used,    6540k free,       0k shrd,   21888k buff
Swap: 1044216k av,  161672k used,  882544k free                  199388k cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME CPU COMMAND
 2330 root      15   0  161M  70M  2132 S     4.9 14.0  1000m   0 X
 2605 FIRST     15   0  8240 6340  3804 S     0.3  1.2   1:12   0 kdeinit
 3413 FIRST     15   0  6668 5324  3216 R     0.3  1.0   0:20   0 kdeinit
18734 root      15   0  1192 1192   868 R     0.3  0.2   0:00   0 top
 1619 root      15   0   776  608   504 S     0.1  0.1   0:53   0 dhclient
    1 root      15   0   480  448   424 S     0.0  0.0   0:03   0 init
    2 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 keventd
    3 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 kapmd
    4 root      35  19     0    0     0 SWN   0.0  0.0   0:00   0 ksoftirqd_CPU0
    9 root      25   0     0    0     0 SW    0.0  0.0   0:00   0 bdflush
    5 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 kswapd
   10 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 kupdated
   11 root      25   0     0    0     0 SW    0.0  0.0   0:00   0 mdrecoveryd
   15 root      15   0     0    0     0 SW    0.0  0.0   0:01   0 kjournald
   81 root      25   0     0    0     0 SW    0.0  0.0   0:00   0 khubd
 1188 root      15   0     0    0     0 SW    0.0  0.0   0:00   0 kjournald
 1675 root      15   0   604  572   520 S     0.0  0.1   0:00   0 syslogd
 1679 root      15   0   428  376   372 S     0.0  0.0   0:00   0 klogd
```

```
 1707 rpc       15   0   516  440   436 S     0.0  0.0   0:00   0 portmap
 1776 root      25   0   476  428   424 S     0.0  0.0   0:00   0 apmd
 1813 root      25   0   752  528   524 S     0.0  0.1   0:00   0 sshd
 1828 root      25   0   704  548   544 S     0.0  0.1   0:00   0 xinetd
 1847 ntp       15   0  2396 2396  2160 S     0.0  0.4   0:00   0 ntpd
 1930 root      24   0    76    4     0 S     0.0  0.0   0:00   0 rpc.rquotad
```

The top portion of the report lists information such as the system time, uptime, CPU usage, physical ans swap memory usage, and number of processes. Below that is a list of the processes sorted by CPU utilization.

You can modify the output of **top** while is is running. If you hit an i, top will no longer display idle processes. Hit i again to see them again. Hitting M will sort by memory usage, S will sort by how long they processes have been running, and P will sort by CPU usage again.

In addition to viewing options, you can also modify processes from within the **top** command. You can use u to view processes owned by a specific user, k to kill processes, and r to renice them.

For more in-depth information about processes you can look in the /proc filesystem. In the /proc filesystem you will find a series of sub-directories with numeric names. These directories are associated with the processes ids of currently running processes. In each directory you will find a series of files containing information about the process.

The iostat command.

The **iostat** will display the current CPU load average and disk I/O information. This is a great command to monitor your disk I/O usage.

```
# iostat
Linux 2.6.32-24.9 (myhost)       12/23/2003

avg-cpu:  %user   %nice    %sys   %idle
          62.09    0.32    2.97   34.62
```

```
Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
dev3-0            2.22        15.20        47.16    1546846    4799520
```

For 2.4 kernels the devices is names using the device's major and minor number. In this case the device listed is /dev/hda. To have **iostat** print this out for you, use the -x.

```
# iostat –x
Linux 2.6.32-24.9 (myhost)      12/23/2003

avg-cpu:  %user   %nice    %sys   %idle
          62.01    0.32    2.97   34.71

Device:  rrqm/s wrqm/s r/s  w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/hdc   0.00   0.00 .00 0.00   0.00   0.00  0.00  0.00     0.00     2.35  0.00  0.00 14.71
/dev/hda   1.13   4.50 .81 1.39  15.18  47.14  7.59 23.57    28.24     1.99 63.76 70.48 15.56
/dev/hda1  1.08   3.98 .73 1.27  14.49  42.05  7.25 21.02    28.22     0.44 21.82  4.97  1.00
/dev/hda2  0.00   0.51 .07 0.12   0.55   5.07  0.27  2.54    30.35     0.97 52.67 61.73  2.99
/dev/hda3  0.05   0.01 .02 0.00   0.14   0.02  0.07  0.01     8.51     0.00 12.55  2.95  0.01
```

The ps command

The **ps** will provide you a list of processes currently running. There is a wide variety of options that this command gives you.

A common use would be to list all processes currently running. To do this you would use the **ps -ef** command. (Screen output from this command is too large to include, the following is only a partial output.)

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Dec22 ?        00:00:03 init
root         2     1  0 Dec22 ?        00:00:00 [keventd]
```

```
root        3     1  0 Dec22 ?        00:00:00 [kapmd]
root        4     1  0 Dec22 ?        00:00:00 [ksoftirqd_CPU0]
root        9     1  0 Dec22 ?        00:00:00 [bdflush]
root        5     1  0 Dec22 ?        00:00:00 [kswapd]
root        6     1  0 Dec22 ?        00:00:00 [kscand/DMA]
root        7     1  0 Dec22 ?        00:01:28 [kscand/Normal]
root        8     1  0 Dec22 ?        00:00:00 [kscand/HighMem]
root       10     1  0 Dec22 ?        00:00:00 [kupdated]
root       11     1  0 Dec22 ?        00:00:00 [mdrecoveryd]
root       15     1  0 Dec22 ?        00:00:01 [kjournald]
root       81     1  0 Dec22 ?        00:00:00 [khubd]
root     1188     1  0 Dec22 ?        00:00:00 [kjournald]
root     1675     1  0 Dec22 ?        00:00:00 syslogd -m 0
root     1679     1  0 Dec22 ?        00:00:00 klogd -x
rpc      1707     1  0 Dec22 ?        00:00:00 portmap
root     1813     1  0 Dec22 ?        00:00:00 /usr/sbin/sshd
ntp      1847     1  0 Dec22 ?        00:00:00 ntpd -U ntp
root     1930     1  0 Dec22 ?        00:00:00 rpc.rquotad
root     1934     1  0 Dec22 ?        00:00:00 [nfsd]
root     1942     1  0 Dec22 ?        00:00:00 [lockd]
root     1943     1  0 Dec22 ?        00:00:00 [rpciod]
root     1949     1  0 Dec22 ?        00:00:00 rpc.mountd
root     1961     1  0 Dec22 ?        00:00:00 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf
root     2057     1  0 Dec22 ?        00:00:00 /usr/bin/spamd -d -c -a
root     2066     1  0 Dec22 ?        00:00:00 gpm -t ps/2 -m /dev/psaux
bin      2076     1  0 Dec22 ?        00:00:00 /usr/sbin/cannaserver -syslog -u bin
root     2087     1  0 Dec22 ?        00:00:00 crond
daemon   2195     1  0 Dec22 ?        00:00:00 /usr/sbin/atd
root     2215     1  0 Dec22 ?        00:00:11 /usr/sbin/rcd
FIRST    3414  3413  0 Dec22 pts/1    00:00:00 /bin/bash
FIRST    4342  3413  0 Dec22 pts/2    00:00:00 /bin/bash
```

```
FIRST   19121 18668  0 12:58 pts/2    00:00:00 ps –ef
```

The first column shows who owns the process. The second column is the process ID. The Third column is the parent process ID. This is the process that generated, or started, the process. The forth column is the CPU usage (in percent). The fifth column is the start time, of date if the process has been running long enough. The sixth column is the tty associated with the process, if applicable. The seventh column is the cumulitive CPU usage (total amount of CPU time is has used while running). The eighth column is the command itself.

With this information you can see exacly what is running on your system and kill run-away processes, or those that are causing problems.

The vmstat command

The **vmstat** command will provide a report showing statistics for system processes, memory, swap, I/O, and the CPU. These statistics are generated using data from the last time the command was run to the present. In the case of the command never being run, the data will be from the last reboot until the present.

```
# vmstat
   procs                  memory      swap        io     system      cpu
 r  b  w   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id
 0  0  0 181604  17000  26296 201120    0    2     8    24  149     9 61  3 36
```

The following was taken from the **vmstat** man page.

FIELD DESCRIPTIONS

Procs

r: The number of processes waiting for run time.

b: The number of processes in uninterruptable sleep.

w: The number of processes swapped out but otherwise runnable. This field is calculated, but Linux never desperation swaps.

Memory

swpd: the amount of virtual memory used (kB).
free: the amount of idle memory (kB).
buff: the amount of memory used as buffers (kB).

Swap

si: Amount of memory swapped in from disk (kB/s).
so: Amount of memory swapped to disk (kB/s).

IO

bi: Blocks sent to a block device (blocks/s).
bo: Blocks received from a block device (blocks/s).

System

in: The number of interrupts per second, including the clock.
cs: The number of context switches per second.

CPU

These are percentages of total CPU time.
us: user time
sy: system time
id: idle time

The lsof command

The **lsof** command will print out a list of every file that is in use. Since Linux considers everything a file, this list can be very long. However, this command can be useful in diagnosing problems. An example of this is if you wish to unmount a filesystem, but you are being told that it is in use. You could use this command and **grep** for the name of the filesystem to see who is using it.

Or suppose you want to see all files in use by a particular process. To do this you would use **lsof -p -processid-**.

Monitoring Devices

The df command

The **df** is the simplest tool available to view disk usage. Simply type in **df** and you'll be shown disk usage for all your mounted filesystems in 1K blocks

```
[root@demo.com ~]#  df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/hda3             5242904    759692   4483212  15% /
tmpfs                  127876         8    127868   1% /dev/shm
/dev/hda1              127351     33047     87729  28% /boot
/dev/hda9            10485816     33508  10452308   1% /home
/dev/hda8             5242904    932468   4310436  18% /srv
/dev/hda7             3145816     32964   3112852   2% /tmp
/dev/hda5             5160416    474336   4423928  10% /usr
/dev/hda6             3145816    412132   2733684  14% /var
```

You can also use the **-h** to see the output in "human-readable" format. This will be in K, Megs, or Gigs depending on the size of the filesystem. Alternately, you can also use the **-B** to specify block size.

In addition to space usage, you could use the **-i** option to view the number of used and available inodes.

```
[root@demo.com ~]# df –I
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/hda3                  0       0       0    - /
tmpfs                  31969       5   31964    1% /dev/shm
/dev/hda1              32912      47   32865    1% /boot
/dev/hda9                  0       0       0    - /home
/dev/hda8                  0       0       0    - /srv
/dev/hda7                  0       0       0    - /tmp
```

/dev/hda5 656640 26651 629989 5% /usr
/dev/hda6 0 0 0 - /var

The du command

Now that you know how much space has been used on a filesystem how can you find out where that data is? To view usage by a directory or file you can use **du**. Unless you specify a filename **du** will act recursively. For example:

[root@demo.com ~]# du file.txt
1300 file.txt

Or like the **df** I can use the **-h** and get the same output in "human-readable" form.

[root@demo.com ~]# du -h file.txt
1.3M file.txt

Unless you specify a filename **du** will act recursively.

[root@demo.com ~]# du -h /usr/local
4.0K /usr/local/games
16K /usr/local/include/nessus/net
180K /usr/local/include/nessus
208K /usr/local/include
62M /usr/local/lib/nessus/plugins/.desc
97M /usr/local/lib/nessus/plugins
164K /usr/local/lib/nessus/plugins_factory
97M /usr/local/lib/nessus
12K /usr/local/lib/pkgconfig
2.7M /usr/local/lib/ladspa
104M /usr/local/lib
112K /usr/local/man/man1
4.0K /usr/local/man/man2

4.0K /usr/local/man/man3
4.0K /usr/local/man/man4
16K /usr/local/man/man5
4.0K /usr/local/man/man

If you just want a summary of that directory you can use the **-s** option.

[root@demo.com ~]# du -hs /usr/local
210M /usr/local

Monitoring Users

From time to time there are going to be occasions where you will want to know exactly what people are doing on your system. Maybe you notice that a lot of RAM is being used, or a lot of CPU activity. You are going to want to see who is on the system, what they are running, and what kind of resources they are using.

The who command

The easiest way to see who is on the system is to do a **who** or **w**. The --> **who** is a simple tool that lists out who is logged --> on the system and what port or terminal they are logged on at.

[root@demo.com ~]# **who**
demo pts/0 May 23 09:33
user1 pts/3 May 20 11:35
user2 pts/1 May 22 11:03
user2 pts/2 May 23 15:04

The ps command

In the previous section we can see that user user2 is logged onto both pts/1 and pts/2, but what if we want to see what they are doing? We could to a **ps -u user2** and get the following output

[root@demo.com ~]# ps -u user2

```
20876 pts/1    00:00:00 bash
20904 pts/2    00:00:00 bash
20951 pts/2    00:00:00 ssh
21012 pts/1    00:00:00 ps
```

From this we can see that the user is doing a **ps ssh**.

This is a much more consolidated use of the **ps** than discussed previously.

The w command

Even easier than using the **who** and **ps -u** commands is to use the **w**. **w** will print out not only who is on the system, but also the commands they are running.

```
[root@demo.com ~]#  w
user2   :0       09:32   ?xdm?  30:09   0.02s -:0
user2   pts/0    09:33    5:49m  0.00s  0.82s kdeinit: kded
user2   pts/2    09:35    8.00s  0.55s  0.36s vi sag-0.9.sgml
user2   pts/1    15:03   59.00s  0.03s  0.03s /bin/bash
```

Kernel Monitoring

uname is one of the most useful commands when it comes to gathering basic information about your system. You can use it to find out the hostname of the system you're on, the hardware architectures supported by the currently used kernel and the exact release of your system.

Basic uname usage

uname -n

This command shows you the node (host) name of your system:

```
[root@demo.com ~]# uname -n
demo.com
```

uname -i

If you're interested in confirming the hardware platform of your system, this is the command to use.

For Linux, it will return **i386** for 32-bit processors or **x86_64** for 64-bit ones. For Solaris, it will confirm the actual server type used:

[root@demo.com ~]# uname -i

i386

Release and version of the Unix kernel

To find out the release and version of your Unix kernel, you need to use **uname -r** and **uname -v**.

uname -r

This allows you to confirm the release of Unix kernel used in your OS.

On Linux, it looks like this:

[root@demo.com ~]# uname -r

2.6.32-8.el6

For the version of Unix kernel, use **uname -v**:

Typical Linux output:

[root@demo.com ~]# uname -v

#1 SMP Fri Jan 26 14:15:14 EST 2007

Common uname usage

uname -a

Most usually, you simply use uname to output everything it knows about your system.

Mounting External Devices

Unix systems have a single directory tree. All accessible storage must have an associated location in this single directory tree. This is unlike Windows where (in the most common syntax for file paths) there is one directory tree per storage component (drive).

Mounting is the act of associating a storage device to a particular location in the directory tree. For example, when the system boots, a particular storage device (commonly called the root partition) is associated with the root of the directory tree, i.e., that storage device is mounted on / (the root directory).

Let's say you now want to access files on a CD-ROM. You must mount the CD-ROM on a location in the directory tree (this may be done automatically when you insert the CD). Let's say the CD-ROM device is /dev/cdrom and the chosen mount point is /media/cdrom. The corresponding command is

[root@demo.com ~]# mount /dev/cdrom /media

After that command is run, a file whose location on the CD-ROM is /dir/file is now accessible on your system as /media/cdrom/dir/file. When you've finished using the CD, you run the commandumount /dev/cdrom or umount /media/cdrom (both will work; typical desktop environments will do this when you click on the “eject” or ”safely remove” button).
Mounting applies to anything that is made accessible as files, not just actual storage devices. For example, all Linux systems have a special filesystem mounted under /proc. That filesystem (called proc) does not have underlying storage: the files in it give information about running processes and various other system information; the information is provided directly by the kernel from its in-memory data structures.

To mount your hard drive you need to use the mount command. (To unmount the drive when you are ready to unplug it and take it home, use the umount command.) Here’s an example:

To check the id of harddrive or flash drive which we want to mount command is:

[root@demo.com ~]# blkid

[root@demo.com ~]# mkdir /mnt/usbdrive

[root@demo.com ~]# mount /dev/sda1 /mnt/usbdrive

The above command first creates a directory in your /mnt folder, typically the folder will be descriptive of the device, so we use /mnt/usbdrive. This only has to be done once, as you won't be deleting this folder. This folder will be your mount point for the external hard drive.

The second line mounts the device at the /dev/sda1 location to the /mnt/usbdrive directory.

Now you can reference your hard drive from the command line using the folder /mnt/usbdrive.

Creating New Partition

Mounted File Systems vs. Logical Volumes

There are two ways to configure a new disk drive into a RHEL system. One very simple method is to create one or more Linux partitions on the new drive, create Linux file systems on those partitions and then mount them at specific mount points so that they can be accessed. This is the approach that will be covered in this chapter.

Another approach is to add the new space to an existing volume group or create a new volume group. When RHEL is installed using the default disk configuration layout, a volume group is created and called VolGroup00. Within this volume group are two logical volumes named LogVol00 and LogVol01 that are used to store the / file system and swap partition respectively. By configuring the new disk as part of a volume group we are able to increase the disk space available to the existing logical volumes. Using this approach we are able, therefore, to increase the size of the / file system by allocating some or all of the space on the new disk to LogVol00.

Let's assumes that the new physical hard drive has been installed on the system and is visible to the RHEL operating system. The best way to do this is to enter the system BIOS during the boot process and ensuring that the BIOS sees the disk drive. Sometimes the BIOS will provide a menu option to scan for new drives. If the BIOS does not see the disk drive double check the connectors and jumper settings (if any) on the drive.

Finding the New Hard Drive in RHEL 6

Assuming the drive is visible to the BIOS it should automatically be detected by the operating system. Typically, the disk drives in a system are assigned device names beginning hd or sd followed by a letter to indicate the device number. Typically hd devices are attached to an IDE disk controller and sd devices are attached to a SATA controller. For example, the first device on a SATA controller might be /dev/sda, the second /dev/sdb and so on.

The following is output from a system with only one physical disk drive:

[root@demo.com ~]# ls /dev/sd*

/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4 /dev/sda5

This shows that the disk drive represented by /dev/sda is itself divided into 5 partitions, represented by /dev/sda1 through /dev/sda5. In all likelihood, when a second disk drive is detected by the system it will be assigned to /dev/sdb.

The following output is from the same system after a second hard disk drive has been installed:

[root@demo.com ~]# ls /dev/sd*

/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4 /dev/sda5 /dev/sdb

As shown above, the new hard drive has been assigned to the device file /dev/sdb. At this point the drive has no partitions shown (because we have yet to create any).

At this point we have a choice of creating partitions and file systems on the new drive and mounting them for access or adding the disk as a physical volume as part of a volume group.

Creating Linux Partitions

The next step is to create one or more Linux partitions on the new disk drive. This is achieved using the fdisk utility which takes as a command-line argument the device to be partitioned:

[root@demo.com ~]# fdisk /dev/sdb

Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel. Changes will remain in memory only,

until you decide to write them. After that, of course, the previous content won't be recoverable.

The number of cylinders for this disk is set to 4177.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
(e.g., DOS FDISK, OS/2 FDISK)
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help):
In order to view the current partitions on the disk enter the p command:
Command (m for help): p

Disk /dev/sdb: 34.3 GB, 34359738368 bytes
255 heads, 63 sectors/track, 4177 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot Start End Blocks Id System

As we can see from the above fdisk output the disk currently has no partitions because it is a previously unused disk. The next step is to create a new partition on the disk, a task which is performed by entering n (for new partition) and p (for primary partition):

Command (m for help): n

Command action
e extended
p primary partition (1-4)
p
Partition number (1-4):

In this example we only plan to create one partition which will be partition 1. Next we need to specify where the partition will begin and end. Since this is the first partition we need it to start at cylinder 1 and since we want to use the entire disk we specify the last cylinder as the end. Note that if you wish to create multiple partitions you can specify the size of each partition by cylinders, bytes, kilobytes or megabytes.

Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-4177, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-4177, default 4177):
Using default value 4177
Now that we have specified the partition we need to write it to the disk using the w command:

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.

If we now look at the devices again we will see that the new partition is visible as /dev/hdb1:

[root@demo.com ~]# partprobe

[root@demo.com ~]# ls /dev/sd*

/dev/sda /dev/sda2 /dev/sda4 /dev/sdb

/dev/sda1 /dev/sda3 /dev/sda5 /dev/sdb1

The next step is to create a file system on our new partition.

Creating a File System on an RHEL Disk Partition

We now have a new disk installed, it is visible to RHEL and we have configured a Linux partition on the disk. The next step is to create a Linux file system on the partition so that the operating system can use it to store files and data. The easiest way to create a file system on a partition is to use the mkfs.ext4 utility which takes as arguments the label and the partition device:

[root@demo.com ~]# mkfs.ext4 –L /userdata /dev/sdb1
Filesystem label=/userdata
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
4194304 inodes, 8387930 blocks
419396 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
256 block groups
32768 blocks per group, 32768 fragments per group
16384 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208, 4096000, 7962624

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 34 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.

Mounting a File System

Now that we have created a new file system on the Linux partition of our new disk drive we need to mount it so that it is accessible to the RHEL system and its users. In order to do this we need to create a mount point. A mount point is simply a directory or folder into which the file system will be mounted. For the purposes of this example we will create a /userdata directory to match our file system label (although it is not necessary that these values match):

[root@demo.com ~]# mkdir /userdata

The file system may then be manually mounted using the mount command:

[root@demo.com ~]# mount /dev/sdb1 /userdata

Running the mount command with no arguments shows us all currently mounted file systems (including our new file system):

[root@demo.com ~]# mount

/dev/mapper/VolGroup00-LogVol00 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda3 on /boot type ext4 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/sdb1 on /userdata type ext4 (rw)

Configuring RHEL 6 to Automatically Mount a File System

In order to set up the system so that the new file system is automatically mounted at boot time an entry needs to be added to the

/etc/fstab file.

The following example shows an fstab file configured to automount our /userdata partition:

```
/dev/VolGroup00/LogVol00  /              ext4    defaults       1 1
LABEL=/boot               /boot          ext4    defaults       1 2
tmpfs                     /dev/shm       tmpfs   defaults       0 0
ysfs                      /sys           sysfs   defaults       0 0
proc                      /proc          proc    defaults       0 0
/dev/VolGroup00/LogVol01 swap            swap    defaults       0 0
LABEL=/userdata           /userdata      ext4    defaults       1 2
```

With the appropriate configuration line added to the fstab file, the file system will automatically mount on the next system restart.

Creating Swap Partition

1. Firstly if there is swap partition in your system, stop swap

   [root@demo.com ~]# free -m (to check swap is working)

   [root@demo.com ~]# swapon -s (to check swap is on which partition)
2. First remove the entry of swap from fstab
3. **[root@demo.com ~]# fdisk /dev/hda**

   (delete swap and reboot)
4. Now create

   :n

   :l (for seeing the swap code)

   :t (to give code)

   Partition no: 5

   Hex code :82 (for swap)

   :w (save)
5. **[root@demo.com ~]# partprobe**
6. **[root@demo.com ~]# mkswap /dev/hda5** (to make partition swap)
7. **[root@demo.com ~]# swapon -a /dev/hda5**
8. **[root@demo.com ~]# e2label /dev/hda5 hda5-swap** (to give label to partition)

9. Now in fstab
   LABEL=hda5-swap swap swap defaults 0 0
   :wq (save)
10. Now reboot your system and you will find a new swap partition.

SHELL SCRIPTING

What is Linux Shell ?

Shell is a user program or it's environment provided for user interaction. Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

Several shell available with Linux including:

Shell Name	**Developed by**	**Where**	**Remark**
BASH (Bourne-Again SHell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	--
TCSH	See the man page. Type $ man tcsh	--	TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

To find all available shells in your system type following command:

$ cat /etc/shells

To find your current shell type following command

$ echo $SHELL

What is Shell Script ?

Normally shells are interactive. It means shell accept command from you (via keyboard) and execute them. But if you use command one by one (sequence of 'n' number of commands) , the you can store this sequence of command to text file and tell the shell to execute this text file instead of entering the commands. This is know as ***shell script***.

Shell script defined as:

"*Shell Script is **series of command** written **in plain text file**. Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file.*"

Why use Shell Script:

- Shell script can take input from user, file and output them on screen.
- Useful to create our own commands.
- Save lots of time.
- To automate some task of day today life.
- System Administration part can be also automated.

Execute your script as

syntax:
bash your-script-name

sh your-script-name

./your-script-name

Examples:
$ sh first.sh

$ ./first.sh

Variables in Shell

To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data. Programmer can give a unique name to this memory location/address called memory variable or variable (Its a named storage location that may take different values, but only one at a time).

In Linux (Shell), there are two types of variable:

(1) **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.

(2) **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

You can see system variables by giving command like **$ set**, some of the important System variables are:

System Variable	Meaning

BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/User	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=demo	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/demo/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=demo	User name who is currently login to this PC

To define UDV use following syntax

Syntax:
variable name=value

'**value**' is assigned to given '**variable name**' and Value must be on right side = sign.

Example:
$ no=22# this is ok

$ 22=no# Error, NOT Ok, Value must be on right side of = sign.

To define variable called 'abc' having value name

$ abc=name

To define variable called no having value 1

$ no=1

ECHO COMMAND

The **echo** command displays a message on the screen and is primarily useful for programmers writing shell scripts. But anyone can use **echo** to show the value of environment variables. Here are some examples:

echo [options] [string, variables...]

Options

-n Do not output the trailing new line.

-e Enable interpretation of the following backslash escaped characters in the strings:

\a alert (bell)

\b backspace

\c suppress trailing new line

\n new line

\r carriage return

\t horizontal tab

\\ backslash

$ echo -e "Hello \a\t\tFreinds\n"

echo $PATH

/usr/bin:/usr/local/bin

echo My name is $USER - Home directory=$HOME.

My name is demo - Home directory=/home/demo.

Example:

$ vi first.sh

```
#
#
# Script to print user information who currently login , current date & time
#
clear
echo "Hello $USER"
echo "Today is \c ";date
echo "Number of user login : \c" ; who | wc –l
echo "Calendar"
cal
exit 0
```

Arithmatic operations

EXPR Function

expr evaluates each numeric_expr command line argument as a separate numeric expression. Thus a single expression cannot contain unescaped whitespaces or needs to be placed in a quoted string (i.e. between "{" and "}"). Arithmetic is performed on signed integers (currently numbers in the range from -2,147,483,648 to 2,147,483,647). Successful calculations cause no output (to either standard out/error or environment variables). So each useful numeric_expr needs to include an assignment (or op-assignment). Each numeric_expr argument supplied is evaluated in the order given (i.e. left to right) until they all evaluate successfully (returning a true status). If evaluating a numeric_expr fails (usually due to a syntax error) then the expr command fails with "error" as the exit status and the error message is written to the environment variable "error".

Syntax:
expr op1 math-operator op2

Examples:
$ expr 1 + 3

$ expr 2 – 1

$ expr 10 / 2

$ expr 20 % 3

$ expr 10 * 3

$ echo `expr 6 + 3`

Note:
expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2.

expr 10 * 3 - Multiplication use * and not * since its wild card.

For the last statement not the following points

(1) First, before expr keyword we used ` (back quote) sign not the (single quote i.e. ') sign. Back quote is generally found on the key under tilde (~) on PC keyboard OR to the above of TAB key.

(2) Second, expr is also end with ` i.e. back quote.

(3) Here expr 6 + 3 is evaluated to 9, then echo command prints 9 as sum

(4) Here if you use double quote or single quote, it will NOT work For e.g.

$ echo "expr 6 + 3" # It will print expr 6 + 3

$ echo 'expr 6 + 3' # It will print expr 6 + 3

There are three types of quotes

Quotes	**Name**	**Meaning**
"	Double Quotes	"Double Quotes" - Anything enclose in double quotes removed meaning of that characters (except \ and $).
'	Single quotes	'Single quotes' - Enclosed in single quotes remains unchanged.
`	Back quote	`Back quote` - To execute command

Example:

$ echo "Today is date"

Can't print message with today's date.

$ echo "Today is `date`".

It will print today's date as, Today is Tue Jan ....,Can you see that the `date` statement uses back quote?

Exit Status

By default in Linux if particular command/shell script is executed, it return two type of values which is used to see whether command or shell script executed is successful or not.

(1) If return *value is zero* (0), command is successful.

(2) If return *value is nonzero*, command is not successful or some sort of error executing command/shell script.

This value is know as ***Exit Status***.

But how to find out exit status of command or shell script?

Simple, to determine this exit Status you can use **$?** special variable of shell.

For e.g. (This example assumes that **abc file** doest not exist on your hard drive)

$ rm abc

It will show error as follows

rm: cannot remove `abc`: No such file or directory and after that if you give command

$ echo $?

it will print nonzero value to indicate error. Now give command

$ ls

$ echo $?

It will print 0 to indicate command is successful.

Command line arguments

$ vi arg.sh

#!/bin/sh
#
Script that demos, command line args

```
#
echo "Total number of command line argument are $#"

echo "$0 is script name"

echo "$1 is first argument"

echo "$2 is second argument"

echo "All of them are :- $* or $@"
```

Run the script:

$ sh arg.sh we are Indians .

Output:

Total number of command line argument are 4

arg.sh is script name

we is first argument

are is second argument

All of them are :- we are Indians .

In above example $# gives the total no. of arguments given on command line, $0 gives the name of the script, $1 $2 ….. $n specifies the 1st 2nd and so on arguments, $@ and $* prints all the agruments.

The READ Statement

Use to get input (data from user) from keyboard and store (data) to variable.

Syntax:
read variable1, variable2,...variableN

Following script first ask user, name and then waits to enter name from the user via keyboard. Then user enters name from keyboard (after giving name you have to press ENTER key) and entered name through keyboard is stored (assigned) to variable myname.

```
$ vi second.sh
#
#Script to read your name from key-board
#
echo –n "Your first name please:"
read myname
echo "Hello $myname, Let’s be friend!"
```

Run it as follows:

$ ./second.sh

Your first name please: **demo**

Hello demo, Let’s be friend!

BASH STRUCTURED LANGUAGE CONSTRUCTS

Making decision is important part in ONCE life as well as in computers logical driven program. In fact logic is not LOGIC until you use decision making. This topic introduces to the bashs structured language constricts such as:

- Decision making
- Loops

if condition

if condition which is used for decision making in shell script, If given condition is true then command1 is executed.

Syntax:

if condition
then
command1 if condition is true or if exit status
of condition is 0 (zero)
...
...
fi

Condition is defined as:

"*Condition is nothing but comparison between two values.*"
For compression you can use test or [expr] statements or even exist status can be also used.
Expreession is defined as:

"*An expression is nothing but combination of values, relational operator (such as >,<, <> etc) and mathematical operators (such as +, -, / etc).*"

Following are all examples of expression:

5 > 2

3 + 6

3 * 65

a < b

c > 5

c > 5 + 30 -1

Type following commands (assumes you have file called **foo**)

$ cat abc

$ echo $?

The cat command return zero(0) i.e. exit status, on successful, this can be used, in if condition as follows,

Write shell script as

```
$ cat > third.sh
!/bin/sh
#
#Script to print file
#
if cat $1
then
echo -e "\n\nFile $1, found and successfully echoed"
fi
```

Run above script as:

$./third.sh abc

Shell script name is third.sh ($0) and abc is argument (which is $1).Then shell compare it as follows:
if cat $1 which is expanded to if cat abc.

test command or [expr]

test command or [expr] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.

Syntax:
test expression OR [expression]

Example:
Following script determine whether given argument number is positive.

```
$ cat > positive.sh

#!/bin/sh
#
# Script to see whether argument is positive

#
if test $1 -gt 0

then
echo "$1 number is positive"

fi
```

Run it as follows

$ ./positive.sh 5

5 number is positive

$./positive.sh -45

Nothing is printed

$./positive.sh

./ispostive: test: -gt: unary operator expected

Detailed explanation

The line, if test $1 -gt 0 , test to see if first command line argument($1) is greater than 0. If it is true(0) then test will return 0 and output will printed as 5 number is positive but for -45 argument there is no output because our condition is not true(0) (no -45 is not greater than 0) hence echo statement is skipped. And for last statement we have not supplied any argument hence error ./positive.sh: test: -gt: unary operator expected, is generated by shell , to avoid such error we can test whether command line argument is supplied or not.

For Mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	**Meaning**	**Normal Arithmetical/ Mathematical Statements**	**But in Shell**	
			For test statement with if command	**For [expr] statement with if command**
-eq	is equal to	3 == 7	if test 3 -eq 7	if [3 -eq 7]
-ne	is not equal to	3 != 7	if test 3 -ne 7	if [3 -ne 7]
-lt	is less than	3 < 7	if test 3 -lt 7	if [3 -lt 7]
-le	is less than or equal to	3 <= 7	if test 3 -le 7	if [3 -le 7]

-gt	is greater than	3 > 7	if test 3 -gt 7	if [3 -gt 7]
-ge	is greater than or equal to	3 >= 7	if test 3 -ge 7	if [3 -ge 7]

NOTE: == is equal, != is not equal.

For string Comparisons use

Operator	**Meaning**
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Shell also test for file and directory types

Test	**Meaning**
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

Logical Operators

Logical operators are used to combine two or more condition at a time

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

if...else...fi

If given condition is true then command1 is executed otherwise command2 is executed.

Syntax:

```
if condition
then
        condition is zero (true - 0)
        execute all commands up to else statement

else
        if condition is not true then
        execute all commands up to fi
fi
```

Write Script as follows:

```
$ vi numbers.sh

#!/bin/sh
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
echo "$0 : You must give/supply one integers"
exit 1
fi

if test $1 -gt 0
then
echo "$1 number is positive"
else
echo "$1 number is negative"
fi
```

Try it as follows:

$ numbers.sh 5

5 number is positive

$ numbers.sh -45

-45 number is negative

$ numbers.sh

./ numbers.sh *: You must give/supply one integers*

$ numbers.sh 0

0 number is negative

Multilevel if-then-else

Syntax:

```
if condition
then
        condition is zero (true - 0)
        execute all commands up to elif statement
elif condition1
then
        condition1 is zero (true - 0)
        execute all commands up to elif statement
elif condition2
then
        condition2 is zero (true - 0)
        execute all commands up to elif statement
else
        None of the above condtion,condtion1,condtion2 are true (i.e.
        all of the above nonzero or false)
        execute all commands up to fi
fi
```

For multilevel if-then-else statement try the following script:

```
$ cat > fourth.sh

#
#!/bin/sh
# Script to test if..elif...else

#
if [ $1 -gt 0 ]; then

  echo "$1 is positive"

elif [ $1 -lt 0 ]

then
  echo "$1 is negative"

elif [ $1 -eq 0 ]

then
  echo "$1 is zero"

else
  echo "Opps! $1 is not number, give number"

fi
```

Try above script as follows:

$ ./ fourth.sh

$ ./ fourth.sh -2

$ ./ fourth.sh 0

$ ./ fourth.sh a

Here o/p for last sample run:

./ fourth.sh: [: -gt: unary operator expected

./ fourth.sh: [: -lt: unary operator expected

./ fourth.sh: [: -eq: unary operator expected

Opps! a is not number, give number

Above program gives error for last run, here integer comparison is expected therefore error like "*./ fourth.sh: [: -gt: unary operator expected*" occurs, but still our program notify this error to user by providing message "*Opps! a is not number, give number*".

Loops in Shell Scripts

Loop defined as:

"*Computer can repeat particular instruction again and again, until particular condition satisfies. A group of instruction that is executed repeatedly is called a loop.*"

Bash supports:

- for loop
- while loop

Note that in each and every loop,

(a) First, the variable used in loop condition must be initialized, then execution of the loop begins.

(b) A test (condition) is made at the beginning of each iteration.

(c) The body of loop ends with a statement that modifies the value of the test (condition) variable.

for Loop

Syntax:

for { variable name } in { list }
do
execute one for each item in the list until the list is
not finished (And repeat all statement between do and done)
done

Before try to understand above syntax try the following script:

```
$ cat > for.sh

for i in 1 2 3 4 5

do
echo "Welcome $i times"

done
```

Run it above script as follows:

$ ./for.sh

The for loop first creates i variable and assigned a number to i from the list of number from 1 to 5, The shell execute echo statement for each assignment of i. (This is usually know as iteration) This process will continue until all the items in the list were not finished, because of this it will repeat 5 echo statements. To make you idea more clear try following script:

```
$ cat > table.sh
```

```
#!/bin/sh
#
#Script to test for loop
#
#
if [ $# -eq 0 ]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
echo "Use to print multiplication table for given number"
exit 1
fi
n=$1
for i in 1 2 3 4 5 6 7 8 9 10
do
echo "$n * $i = `expr $i \* $n`"
done
```

Save above script and run it as:

$ ./table.sh 6

$ ./table.sh

For first run, above script print multiplication table of given number where i = 1,2 ... 10 is multiply by given n (here command line argument 6) in order to produce multiplication table as 6* 1 = 6

6 * 2 = 12

...

..

6 * 10 = 60

And for second test run, it will print message –

Error - Number missing form command line argument

Syntax : ./table.sh number

Use to print multiplication table for given number

This happened because we have not supplied given number for which we want multiplication table, Hence script is showing Error message, Syntax and usage of our script. This is good idea if our program takes some argument, let the user know what is use of the script and how to used the script.

Note that to terminate our script we used 'exit 1' command which takes 1 as argument (1 indicates error and therefore script is terminated)

Even you can use following syntax:

Syntax:

```
for (( expr1; expr2; expr3 ))
do
    .....
                        ...
    repeat all statements between do and
    done until expr2 is TRUE
Done
```

In above syntax BEFORE the first iteration, ***expr1*** is evaluated. This is usually used to initialize variables for the loop.

All the statements between do and done is executed repeatedly UNTIL the value of ***expr2*** is TRUE. AFTER each iteration of the loop, ***expr3*** is evaluated. This is usually use to increment a loop counter.

```
$ cat > for1.sh

for ((  i = 0 ;  i <= 5;  i++  ))

do
  echo "Welcome $i times"

done
```

Run the above script as follows:

$ ./for1.sh

Welcome 0 times

Welcome 1 times

Welcome 2 times

Welcome 3 times

Welcome 4 times

Welcome 5 times

In above example, first expression (i = 0), is used to set the value variable **i** to zero. Second expression is condition i.e. all statements between do and done executed as long as

expression 2 (i.e continue as long as the value of variable **i** is less than or equel to 5) is TRUE. Last expression **i++** increments the value of **i** by 1 i.e. it's equivalent to i = i + 1 statement.

while loop

Syntax:

```
while [ condition ]
do
    command1
    command2
    command3
    ..
    ....
 done
```

Loop is executed as long as given condition is true. For e.g.. Above for loop program (shown in last section of for loop) can be written using while loop as:

```
$cat > while.sh

#!/bin/sh
#
#Script to test while statement
```

```
#
#
if [ $# -eq 0 ]
then
   echo "Error - Number missing form command line argument"
   echo "Syntax : $0 number"
   echo " Use to print multiplication table for given number"
exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n`"
  i=`expr $i + 1`
done
```

Save it and try as

$./while.sh 6

Above loop can be explained as follows:

n=$1	Set the value of command line argument to variable n. (Here it's set to 6)
i=1	Set variable i to 1
while [$i -le 10]	This is our loop condition, here if value of i is less than 10 then, shell execute all statements between do and done
do	Start loop
echo "$n * $i = `expr $i \* $n`"	Print multiplication table as 6 * 1 = 6 6 * 2 = 12 .... 6 * 10 = 60, Here each time value of variable n is multiply be i.
i=`expr $i + 1`	Increment i by 1 and store result to i. (i.e. i=i+1) <u>Caution:</u> If you ignore (remove) this statement than our loop become infinite loop because value of variable i always remain less than 10 and program will only output 6 * 1 = 6 ... ... E (infinite times)
done	Loop stops here if i is not less than 10 i.e. condition of loop is not true. Hence loop is terminated.

<u>Note:</u> For infinite loop with while you have to write **while :**

The case Statement

The case statement is good alternative to Multilevel if-then-else-fi statement. It enable you to match several values against one variable. Its easier to read and write.

Syntax:

```
case $variable-name in
   pattern1)  command
              ...
              ..
              command;;
   pattern2)  command
              ...
              ..
              command;;
   patternN)  command
              ...
              ..
              command;;
   *)         command
              ...
              ..
              command;;
esac
```

The *$variable-name* is compared against the patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is *) and its executed if no match is found. For e.g. write script as follows:

```
$ cat > case.sh

#
# if no name is given
```

```
# i.e. -z $1 is defined and it is NULL

#
# if no command line arg

if [ -z $1 ]

then
  rental="*** I don’t know the person ***"

elif [ -n $1 ]

then
# otherwise make first arg as name

                                                                    name=$1

fi

case $name in

   "abc") echo "Hello abc, How are you?";;

   "def") echo "Hello def, How are you?";;

   "ghi") echo "Hello ghi, How are you?";;

   "jkl") echo "Hello jkl, How are you?";;

   *) echo "Sorry, I don’t know who are you";;

 Esac
```

Save it by pressing CTRL+D and run it as follows:

$ ./case.sh abc

$ ./case.sh def

$ ./case.sh ghi

First script will check, that if $1(first command line argument) is given or not, if NOT given set value of name variable to "*** I don't know the person ***",if command line arg is supplied/given set value of name variable to given value (command line arg). The $name is compared against the patterns until a match is found.
For first test run its match with abc and it will show output " Hello abc, How are you?" For second test run it print, " Hello def, How are you?".

And for last run, there is no match for xyz, hence default i.e. *) is executed and it prints, " Sorry, I don't know who are you ".

Note that esac is always required to indicate end of case statement.

ADVANCED SHELL SCRIPTING

Functions

When program gets complex we need to use divide and conquer technique. It means whenever programs gets complicated, we divide it into small chunks/entities which is know as *function*.

Function is series of instruction/commands. Function performs particular activity in shell i.e. it had specific work to do or simply say task. To define function use following syntax:
Syntax:

```
function-name ( )
{
    command1
    command2
    .....
    ...
    commandN
    return
}
```

Where function-name is name of you function, that executes series of commands. A return statement will terminate the function. *Example:*

Type Hello() at $ prompt as follows

$ Hello()

{
echo "Hello $LOGNAME, Have nice computing"

return
}

To execute this Hello() function just type it name as follows:

$ Hello

Hello demo, Have nice computing.

This way you can call function. Note that after restarting your computer you will loss this Hello() function, since its created for current session only. To overcome this problem and to add you own function to automat some of the day today life task, add your function to */etc/bashrc* file. To add function to this file you must logon as root.

vi /etc/bashrc

At the end of file add following in /etc/bashrc file

#

today() to print formatted date

```
#
# To run this function type today at the $ prompt

#
today()
{
echo This is a `date +"%A %d in %B of %Y (%r)"`

return
}
```

DIALOG UTILITY

Before programming using dialog utility you need to install the dialog utility, since dialog utility in not installed by default.

For Red Hat Linux user install the dialog utility as follows

```
# mount /mnt/cdrom

# cd /mnt/cdrom/RHEL-6…./Packages

# rpm -ivh dialog-0.6-16.i686.rpm
```

After installation you can start to use dialog utility. Before understanding the syntax of dialog utility try the following script:

```
$ cat > dialog.sh
dialog --title "Linux Dialog Utility Infobox" --backtitle "Linux Shell Script\
Tutorial" --infobox "This is dialog box called infobox, which is used \
```

to show some information on screen, Thanks to Savio Lam and\
Stuart Herbert to give us this utility. Press any key. . . " 7 50 ; read

Save the shell script and run it as:

$ sh dialog.sh

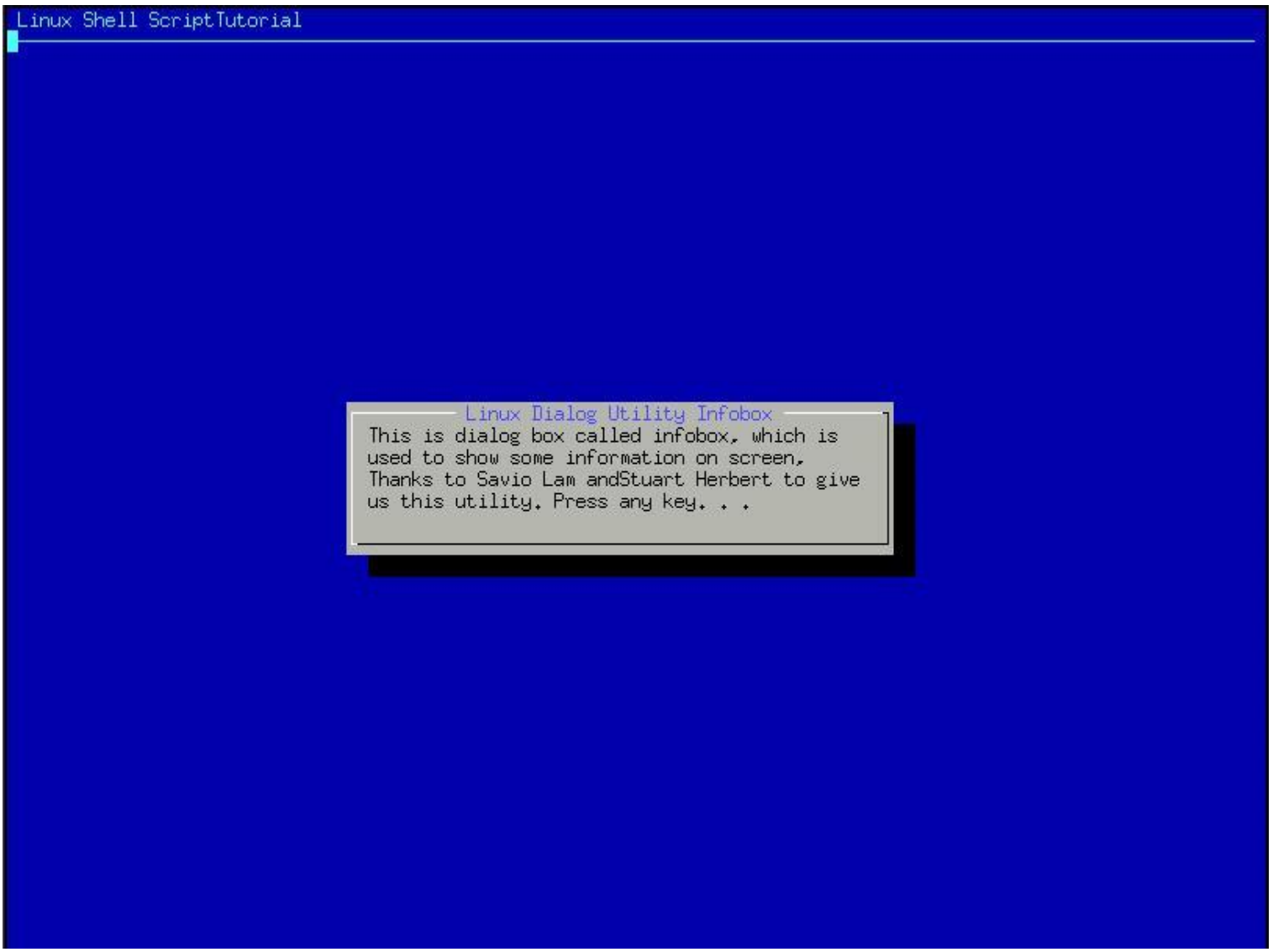


Figure 30 Linux Dialog Utility Infobox

After executing this dialog statement you will see box on screen with titled as "Welcome to Linux Dialog Utility" and message "This is dialog....Press any key. . ." inside this box. The title of box is specified by --title option and infobox with --infobox "Message" with this option. Here 7 and 50 are height-of-box and width-of-box respectively. "Linux Shell Script Tutorial" is the backtitle of dialog show on upper left side of screen and below that line is drawn. Use dialog utility to Display dialog boxes from shell scripts.

Syntax:

dialog --title {title} --backtitle {backtitle} {Box options}

where Box options can be any one of following

--yesno {text} {height} {width}

--msgbox {text} {height} {width}

--infobox {text} {height} {width}

--inputbox {text} {height} {width} [{init}]

--textbox {file} {height} {width}

--menu {text} {height} {width} {menu} {height} {tag1} item1}...

Message box (msgbox) using dialog utility

$cat > **dialog1.sh**

```
dialog --title "Linux Dialog Utility Msgbox" --backtitle "Linux Shell Script\
Tutorial" --msgbox "This is dialog box called msgbox, which is used\
to show some information on screen which has also Ok button, Thanks to Savio Lam\
and Stuart Herbert to give us this utility. Press any key. . . " 9 50
```

Save it and run as

$ sh dialog1.sh

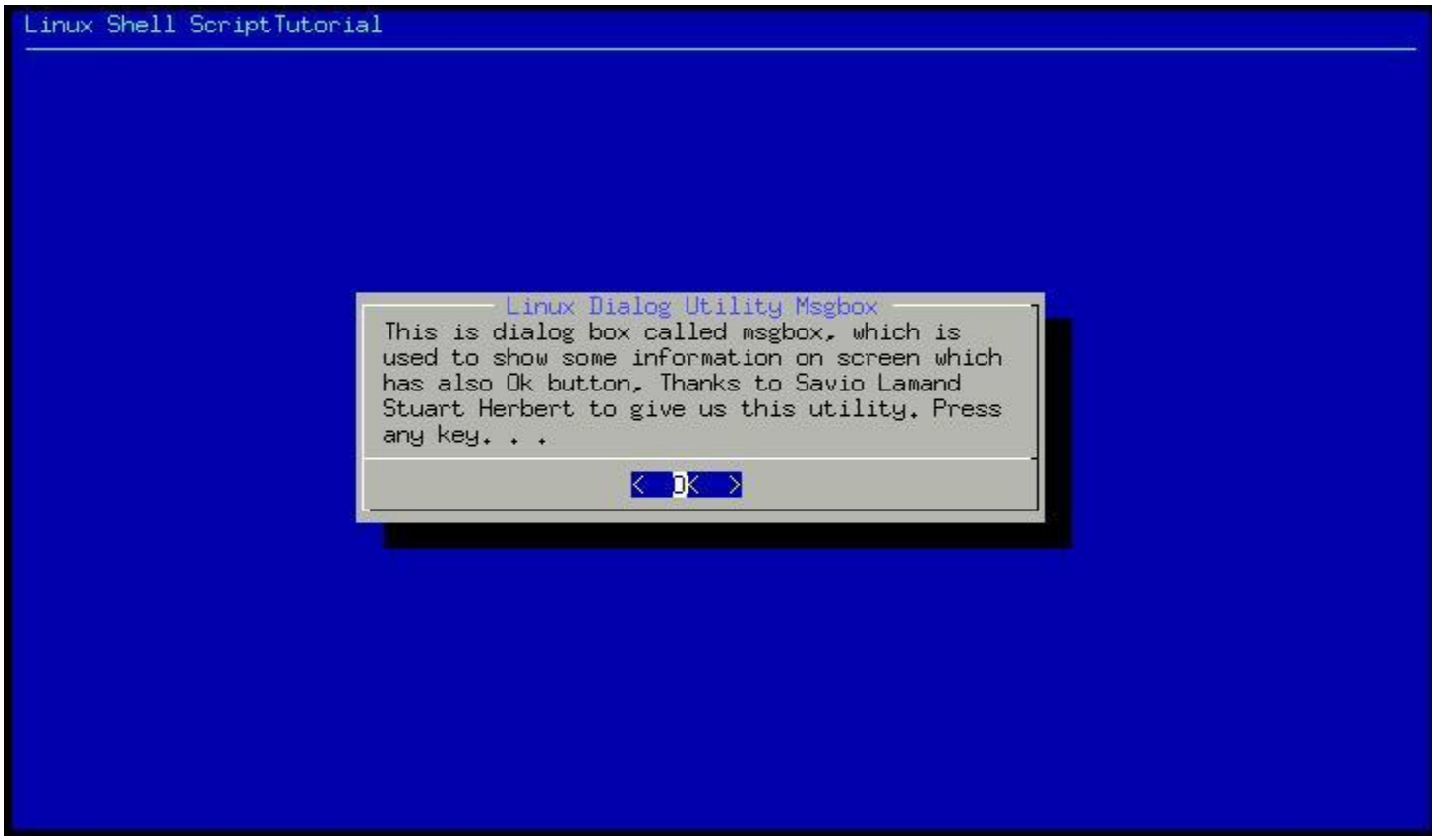


Figure 31 Linux Dialog Utility Msgbox

yesno box using dialog utility

$ cat > **dialog2.sh**

dialog --title "Alert : Delete File" --backtitle "Linux Shell Script\

Tutorial" --yesno "\nDo you want to delete '/usr/letters/jobapplication'\

file" 7 60

```
sel=$?
case $sel in
        0)  echo "User select to delete file";;
        1) echo "User select not to delete file";;
        255) echo "Canceled by user by pressing [ESC] key";;
Esac
```

Save the script and run it as:

$ chmod +x dialog2.sh

$ ./dialog2.sh

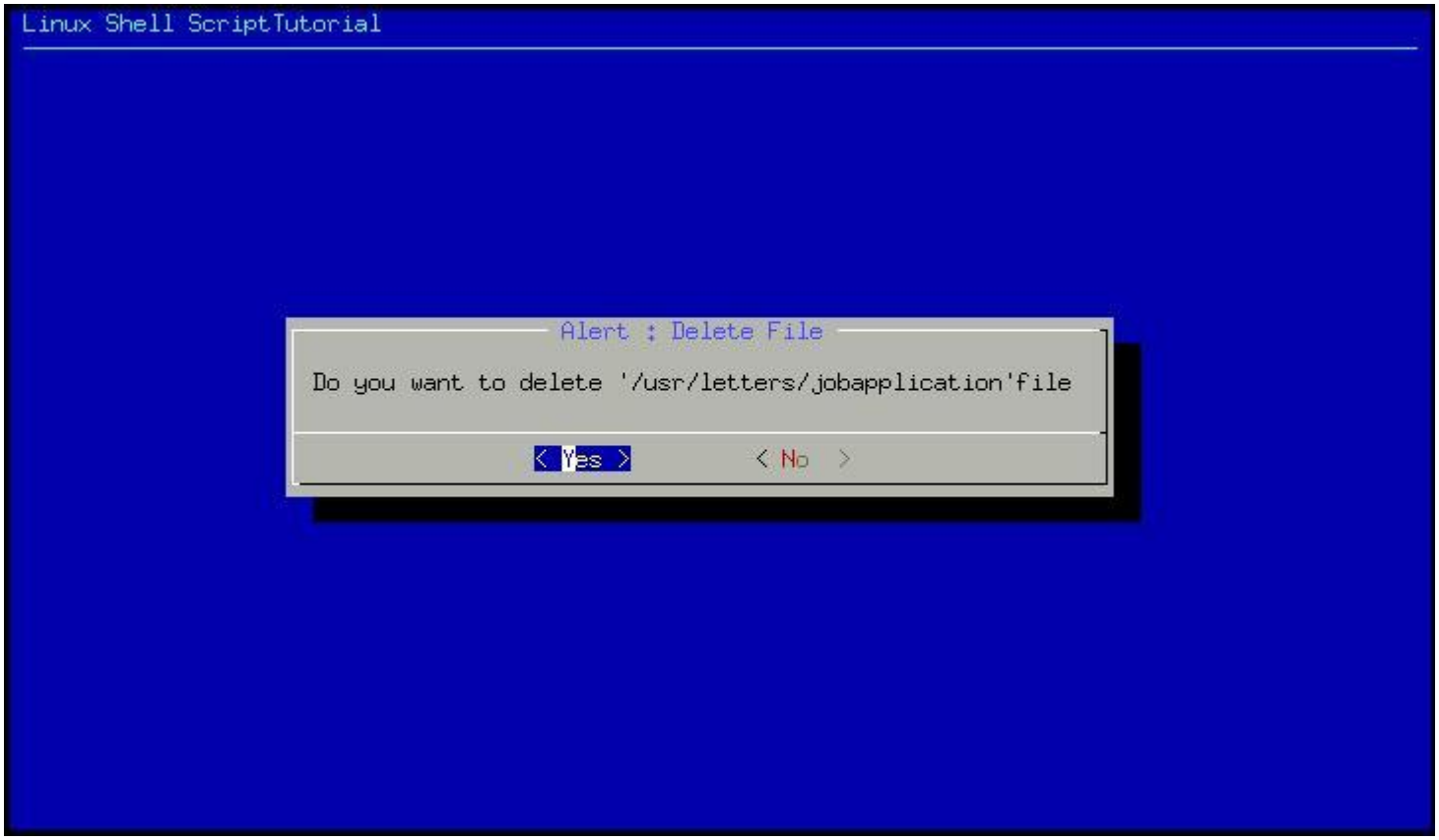


Figure 32 Alert Dialog box

Above script creates yesno type dialog box, which is used to ask some questions to the user , and answer to those question either yes or no. After asking question how do we know, whether user has press yes or no button ? The answer is exit status, if user press yes button exit status will be zero, if user press no button exit status will be one and if user press Escape key to cancel dialog box exit status will be one 255. That is what we have tested in our above shell script as

Statement	**Meaning**
sel=$?	Get exit status of dialog utility
case $sel in **0) echo "You select to delete file";;** **1) echo "You select not to delete file";;** **255) echo "Canceled by you by pressing [Escape] key";;** **esac**	Now take action according to exit status of dialog utility, if exit status is 0 , delete file, if exit status is 1 do not delete file and if exit status is 255, means Escape key is pressed.

Input Box (inputbox) using dialog utility

```
$ cat > dialog3.sh
dialog --title "Inputbox - To take input from you" --backtitle "Linux Shell\
Script Tutorial" --inputbox "Enter your name please" 8 60 2>/tmp/input.$$
sel=$?

na=`cat /tmp/input.$$`
case $sel in
        0) echo "Hello $na" ;;
```

```
1)  echo "Cancel is Press" ;;
 255) echo "[ESCAPE] key pressed" ;;
esac

rm -f /tmp/input.$$
```

Run it as follows:

$ sh dialog3.sh

Inputbox is used to take input from user, In this example we are taking Name of user as input. But where we are going to store inputted name, the answer is to redirect inputted name to file via statement *2>/tmp/input.$$* at the end of dialog command, which means send screen output to file called */tmp/input.$$*, letter we can retrieve this inputted name and store to variable as follows *na=`cat /tmp/input.$$`*.

For input box's exit status refer the following table:

Exit Status for Input box	**Meaning**
0	Command is successful
1	Cancel button is pressed by user
255	Escape key is pressed by user

Its time to write script to create menus using dialog utility, following are menu items
Date/time
Calendar

Editor

and action for each menu-item is follows :

MENU-ITEM	**ACTION**
Date/time	Show current date/time
Calendar	Show calendar
Editor	Start vi Editor

```
$ cat > dialogmenu.sh
#
#How to create small menu using dialog
#
dialog --backtitle "Linux Shell Script Tutorial " --title "Main\
Menu" --menu "Move using [UP] [DOWN],[Enter] to\
Select" 15 50 3\
Date/time "Shows Date and Time"\
Calendar "To see calendar "\
Editor "To start vi editor " 2>/tmp/choice.$$

choice=`cat /tmp/choice.$$`
opt=$?

case $choice in
Date/time) date;;
Calendar) cal;;
Editor) vi;;
Esac
```

Save it and run as:

$ rm -f /tmp/choice.$$

$ sh dialogmenu.sh

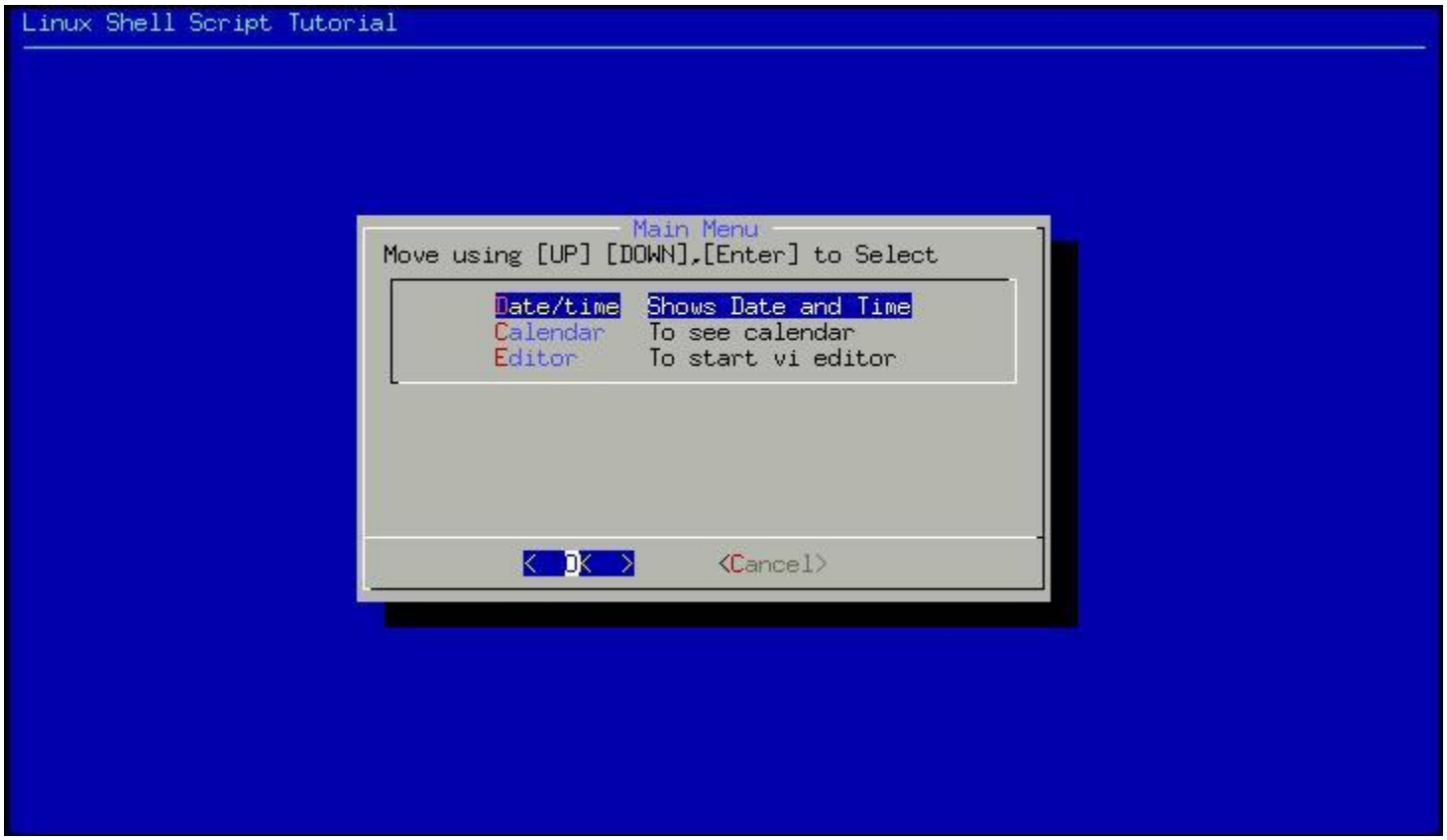


Figure 33 Menu Dialog Box

--menu option is used of dialog utility to create menus, menu option take

--menu options	**Meaning**
"Move using [UP] [DOWN],[Enter] to Select"	This is text show before menu
15	Height of box
50	Width of box
3	Height of menu

Date/time "Shows Date and Time"	First menu item called as *tag1* (i.e. **Date/time**) and description for menu item called as *item1* (i.e. **"Shows Date and Time")**
Calendar "To see calendar "	First menu item called as *tag2* (i.e. **Calendar**) and description for menu item called as *item2* (i.e. **"To see calendar")**
Editor "To start vi editor "	First menu item called as *tag3* (i.e. **Editor**) and description for menu item called as *item3* (i.e.**"To start vi editor")**
2>/tmp/choice.$$	Send selected menu item (tag) to this temporary file

After creating menus, user selects menu-item by pressing the ENTER key, selected choice is redirected to temporary file, Next this menu-item is retrieved from temporary file and following case statement compare the menu-item and takes appropriate step according to selected menu item. As you see, dialog utility allows more powerful user interaction then the older read and echo statement. The only problem with dialog utility is it work slowly.

TRAP COMMAND

Controls the action to be taken by the shell when a signal is received.T rap command is used to catch a signal that is sent to a process. An action is taken based on the signal by using the action which is defined in the trap command instead of taking the default effect on the process.

Trap [OPTIONS] [[arg] signspec..]

Arg is the action to be taken or executed on receiving a signal specified in signspec.

e.g. trap “rm $FILE; exit” // exit (signal) and remove file (action)

Signal Number	When occurs
0	shell exit
1	Hangup
2	interrupt (CTRL+C)

3	quit
9	kill (cannot be caught)

Example:

```
$ vi trap.sh

#
# signal is trapped to delete temporary file , version 2

#
del_file()
{
  echo "* * * CTRL + C Trap Occurs (removing temporary file)* * *"

  rm -f /tmp/input0.$$

  exit 1

}


Take_input1()
{
recno=0
clear
echo "Appointment Note keeper Application for Linux"

echo -n "Enter your database file name : "

read filename

if [ ! -f $filename ]; then
```

```
  echo "Sorry, $filename does not exit, Creating $filename database"

  echo "Appointment Note keeper Application database file" > $filename

fi
echo "Data entry start data: `date`" >/tmp/input0.$$

#
# Set a infinite loop

#
while                                                                                    :
do
  echo -n "Appointment Title:"

  read na

  echo -n "time :"

  read ti

  echo -n "Any Remark :"

  read remark

  echo -n "Is data okay (y/n) ?"

  read ans

  if [ $ans = y -o $ans = Y ]; then

   recno=`expr $recno + 1`

   echo "$recno. $na $ti $remark" >> /tmp/input0.$$
```

```
                                                                    fi
  echo -n "Add next appointment (y/n)?"
  read isnext
  if [ $isnext = n -o $isnext = N ]; then
    cat /tmp/input0.$$ >> $filename
    rm -f /tmp/input0.$$
    return # terminate loop
  fi
done # end_while
}
#
# Set trap to for CTRL+C interrupt i.e. Install our error handler
# When occurs it first it calls del_file() and then exit
#
trap del_file 2
#
# Call our user define function : Take_input1
#
Take_input1
```

Run the script as:

$ ./trap.sh

After giving database file name and after giving appointment title press CTRL+C, Here we have already captured this CTRL + C signal (interrupt), so first our function del_file() is called, in which it gives message as "* * * CTRL + C Trap Occurs (removing temporary file)* * * " and then it remove our temporary file and then exit with exit status 1. Now check /tmp directory as follows

The shift Command

The shift command moves the current values stored in the positional parameters (command line args) to the left one position. For example, if the values of the current positional parameters are:

$1 = -a $2 = abc $3 = xyz

and you executed the shift command the resulting positional parameters would be as follows:

$1 = abc $2 = xyz

e.g. Write the following shell script to clear you idea:

```
$ vi shift.sh

echo "Current command line args are: \$1=$1, \$2=$2, \$3=$3"

shift
echo "After shift command the args are: \$1=$1, \$2=$2, \$3=$3"
```

Excute above script as follows:

$ ./shift.sh -a abc xyz

Current command line args are: $1=-a, $2=abc, $3=xyz

After shift command the args are: $1=abc, $2=xyz, $3=

You can also move the positional parameters over more than one place by specifying a number with the shift command. The following command would shift the positional parameters two places:

shift 2

Troubleshooting

Password Recovery

Breaking root password:

When system boots it ask to select OS,select linux OS and then press ‘e’.

Choose the second option ‘Kernel/vmlinuz-2.6.i8, again press ‘e’.

Write ‘1’ at the last of the line,

Then press esc key and then ‘b’.

Now system will reboot and stops on single user mode

sh#

here type passwd , then enter

type new password

now reboot & give root and new password.

Password Security (adding password on grub after installation):

{on graphical terminal}

grub-md5-crypt

Password:

Confirm password:

(Here you see some digits, copy those digits)

Now edit file grub.conf

vi /boot/grub/grub.conf

Here you see a line where written hidden menu, come under that line and type

password --md5 {paste those digits}

Save file.

Now your password cannot be broken on startup.

Grub recovery

Booting partition problem, when grub is destroyed:

Now when you boot your system, it does'nt boot and came to grub>

Then we have to do the following step

grub> root (hd0, #use tab see all partitions

grub> root (hd0,1)

grub> kernel /vmlinuz-2.6.262.8.el6 root=/dev/sda3

grub> initrd /initrd-2.6.262-8.el6.img

grub> boot

Now it will boot your linux PC .

Grub installation:

Now if grub of our linux pc is destroyed and cannot be recovered, then we have to install grub

For installing grub boot from your linux DVD and write linux rescue to boot your PC in rescue mode

chroot /mnt/sysimage

grub-install /dev/sda,hda

#exit

Finally it installs your grub and now your system will boot with linux OS.

INITTAB REPAIR

If your inittab file is missed or moved and now your PC is unable to boot and it stops on grub, then

first check the from where you have to copy the inittab file here we are taking pendrive (/dev/sdb) drive as source.

grub> kernel /vmlinuz-2.6.18-8.el5 rw root=lable=/ init=/bin/sh

sh-3.1# cd /etc/rc.d

sh-3.1# ./rc.sysinit

sh-3.1# mkdir /dir

sh-3.1# mount /dev/sdb /dir

sh-3.1# cp inittab /etc

now reboot your PC.