**Chapter 4: Graphical Interface**

**Learning Objectives**

By the end of this chapter, you should be able to:

* Manage graphical interface sessions.
* Perform basic operations using the graphical interface.

Change the graphical desktop to suit your needs.

**Introduction**

You can use either a Command Line Interface (**CLI**) or a Graphical User Interface (**GUI**) when using Linux. To work at the CLI, you have to remember which programs and commands are used to perform tasks, and how to quickly and accurately obtain more information about their use and options. On the other hand, using the GUI is often quick and easy. It allows you to interact with your system through graphical icons and screens. For repetitive tasks the CLI is often more efficient, while the GUI is easier to navigate if you don't remember all the details or do something only rarely.

In this section you will learn how to manage sessions using the GUI for the three Linux distribution families that we explicitly cover in this course: **CentOS** (**Fedora** family), **OpenSUSE** (**SUSE** family) and **Ubuntu** (**Debian** family).

**GNOME Desktop Environment**



**GNOME** is a popular desktop environment with an easy to use graphical user interface. It is bundled as the default desktop environment for many distributions including **Red Hat Enterprise Linux**, **Fedora**, **CentOS**, **SUSE Linux Enterprise**, and **Debian**. **GNOME** has menu-based navigation and is sometimes an easy transition for at least some **Windows** users. However, as you'll see, the look and feel can be quite different across distributions, even if they are all using **GNOME**.

Another common desktop environment very important in the history of Linux and also widely used is **KDE**, which is used by default in **OpenSUSE**.

Other alternatives for a desktop environment include **Unity** (from **Ubuntu**, based on **GNOME**), **Xfce**, and**LXDE**. Most desktop environments follow a similar structure to **GNOME**.

**GUI Startup**



When you install a desktop environment, the **X** display manager starts at the end of the boot process. This **X** display manager is responsible for starting the graphics system, logging in the user, and starting the user’s desktop environment. You can often select from a choice of desktop environments when logging in to the system.

The default display manager for **GNOME** is called **gdm**. Other popular display managers include **lightdm** (used on **Ubuntu**) and **kdm** (associated with **KDE**).

**Locking the Screen**



It is often a good idea to lock your screen to prevent other people from accessing your session while you are away from your computer. Note this does not suspend the computer; all your applications and processes continue to run while the screen is locked. There are two ways to lock your screen:

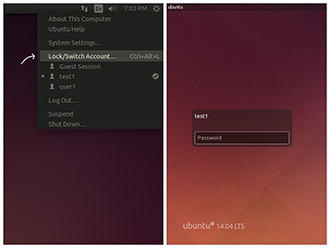
* Using the graphical interface.
* Using the keyboard shortcut **CTRL-ALT-L**.

**Note:** The keyboard shortcut for locking the screen in the three distros can be changed as indicated below:

* **CentOS:** System → Preferences → Keyboard Shortcuts
* **OpenSUSE:** Configure Desktop → Shortcuts and Gestures

**Ubuntu:** System Settings → Keyboard → Shortcuts

**Locking and Unlocking the Screen in Ubuntu**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen17.jpg)

To lock and unlock your screen in **Ubuntu**, perform the following steps:

* On the **Ubuntu** desktop screen, click the power icon with username on the upper-right corner of the screen.
* Click **Lock/Switch Account**… to lock the screen. The login screen is displayed.
* To unlock the screen, enter the password. The desktop screen is displayed.

**Note: When you lock the screen, GNOME will blank the screen or run a screensaver, depending on your settings.**

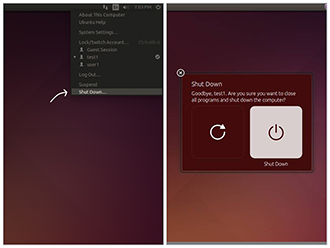
**Shutting Down and Restarting**



Besides normal daily starting and stopping of the computer, a system restart may be required as part of certain major system updates, generally only those involving installing a new Linux **kernel**.

The **init** process is responsible for implementing both restarts and shut downs. On systems using **System V** **init**, run level 0 is usually used for shutting down, and run level 6 is used to reboot the system. (We will discuss system run levels later.)

**Shutting Down and Restarting in Ubuntu**

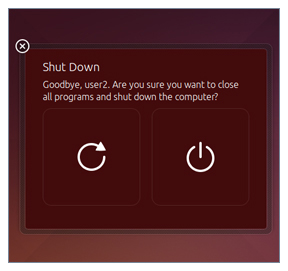
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen29.jpg)

To shut down the computer in **Ubuntu**, perform the following steps:

* On the **Ubuntu** desktop screen, click the power icon on the upper-right corner of the screen.
* Click **Shut Down...**  The **Restart** and **Shutdown** icons are displayed.

Click the shut down icon on the right to shutdown the system and click the restart icon on the left to restart the system.

**Confirmation**



Shutdown, reboot, and logout operations will ask for confirmation before going ahead. This is because many applications will not save their data properly when terminated this way.

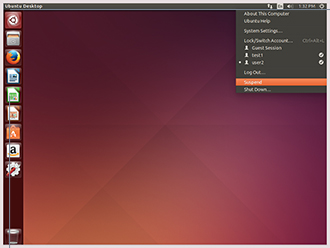
Always save your documents and data before restarting, shutting down, or logging out. The **Ubuntu** screenshot given here is similar to all of the distribution confirmation windows.

**Suspending**



Most modern computers support **suspend mode** or **sleep mode** when you stop using your computer for a short while. Suspend mode saves the current system state and allows you to resume your session more quickly while remaining on but using very little power. It works by keeping your system’s applications, desktop, and so on in system RAM, but turning off all of the other hardware. The suspend mode bypasses the time for a full system start-up and continues to use minimal power.

**Suspending in Ubuntu**

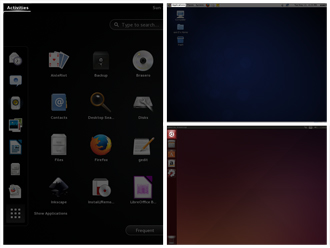
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen33.jpg)

To suspend the system in **Ubuntu**, perform the following steps:

* On the **Ubuntu** desktop screen, click the power icon on the upper-right corner of the screen.
* Click **Suspend** to switch the system to the sleep mode.

**Note: To wake your system and resume your session, move the mouse or press any button on the keyboard. The system will wake up with the screen locked, just as if you had manually locked it; type in your password to resume.**

**Basic Operations**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen37.jpg)

Even experienced users can forget the precise command that launches an application, or exactly what options and arguments it requires. Fortunately, Linux allows you to quickly open applications using the graphical interface.

Applications are found at different places in Linux (and within **GNOME**):

* In **CentOS**, applications can be opened from the **Applications** menu in the upper-left corner of the screen.
* In **OpenSUSE**, applications can be opened from the **Activities** menu in the upper-left corner of the screen.
* In **Ubuntu**, applications can be opened from the **Dash** button in the upper-left corner of the screen.

For **KDE**, and other environments, applications can be opened from the button in the lower-left corner.

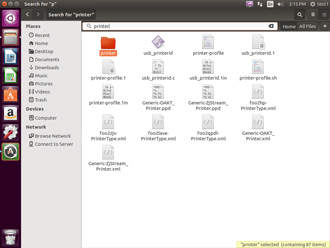
**Submenus**

Submenus for different types of applications include:

* Accessories
* Games
* Graphics
* Internet
* Office
* Sound and Video
* System Tools

On the following screens you will learn how to perform basic operations in Linux using the Graphical Interface.

**Locating Applications**

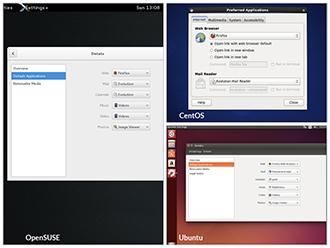
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen39.jpg)

Unlike other operating systems, the initial install of Linux usually comes with a wide range of applications and software archives that contain thousands of programs that enable you to accomplish a wide variety of tasks with your computer. For most key tasks a default application is usually already installed. However, you can always install more applications and try different options.

For example, **Firefox** is popular as the default browser in many Linux distributions, while **Epiphany**, **Konqueror**, and **Chromium** (the open-source base for **Google Chrome**) are usually available for install from software repositories. Proprietary web browsers, such as **Opera** and **Chrome** are also available.

Locating applications from the **GNOME** and **KDE** menus is easy as they are neatly organized in functional submenus.

**Default Applications**

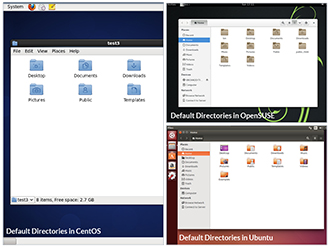
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen40.jpg)

Multiple applications are available to accomplish various tasks and to open a file of a given type. For example, you can click on a web address while reading an email and launch a browser such as **Firefox** or **Chrome**.

The file managing program can be used to set the default application to be used for any  particular file type.

Click the image to view an enlarged version.

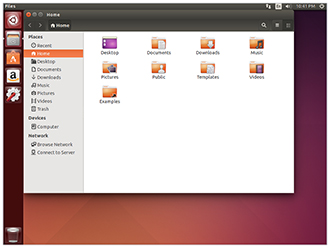
**Default Directories**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen41.jpg)

Every user with an account on the system will have a **home** directory, usually created under /home and named the same as the username (such as /home/student). By default, files the user saves will be placed in a directory tree starting there.  Account creation, whether during system installation or at a later time when a new user is added, also induces default directories to be created under the user's home directory, such as **Documents**, **Desktop**, and **Downloads**.

On the next few screens, you will learn more about the default directories in **CentOS**, **OpenSUSE**, and **Ubuntu**.

**Default Directories in Ubuntu**

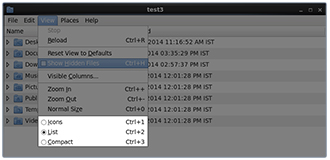
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen44.jpg)

To access your **home** directory from the graphical user interface in **GNOME** on **Ubuntu**, click the **File Cabinet** icon on the left of the screen.

The **File Manager** (**Files** in the case of **Ubuntu**) will open a window with your **Home** directory displayed. The left panel of the **File Manager** window holds a list of commonly used directories, such as **Computer**, **Home**, **Desktop**, **Documents**, **Downloads,** and **Trash**.

You can also click the magnifying glass icon on the top-right of the **File Manager** window to search for files or directories that exist inside your **home** directory.

**Viewing Files**

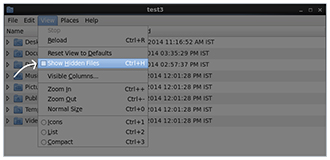
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen46.jpg)

**Nautilus** (the name of the File Manager or file browser) allows you to view files and directories in several different formats.

To view files in the **Icons**, **List**, or **Compact** formats, click the **View** drop-down and select your view, or press **CTRL-1** , **CTRL-2** and **CTRL-3** respectively.

In addition you can also arrange the files and directories by **Name**, **Size**, **Type**, or **Modification Date** for further sorting. To do so, click **View** and select **Arrange Items**.

**More About Viewing Files**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen47.jpg)

Another useful option is to show **hidden** files (sometimes imprecisely called system files), which are usually configuration files that are hidden by default and whose name starts with a dot. To show hidden files, click **View** and select **Show Hidden Files** or press **CTRL- H**.

The file browser provides multiple ways to customize your window view to facilitate easy drag and drop file operations. You can also alter the size of the icons by selecting **Zoom In** and **Zoom Out** under the **View** menu.

**Searching for Files**



**Nautilus** includes a great search tool inside the file browser window.

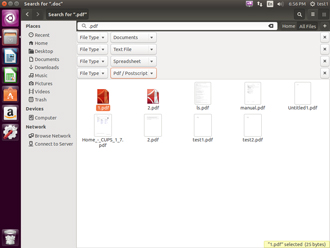
* Click **Search** in the toolbar (to bring up a text box).
* Enter the keyword in the text box. Nautilus will perform a recursive search from the current directory for any file or directory which contains a part of this keyword.
* To open Nautilus from the command line, simply type nautilus
* To open Nautilus in graphical mode, Press **ALT-F2** and search for Nautilus. Click the icon that appears.

**Note: Both the above methods, will open the graphical interface for the program.**

The shortcut key to get to the search text box is **CTRL-F**. You can exit the search text box view by clicking the **Search** button again.

Another quick way to access a specific directory is to press **CTRL-L**, which will give you a **Location** text box to type in a path to a directory.

**More About Searching for Files**

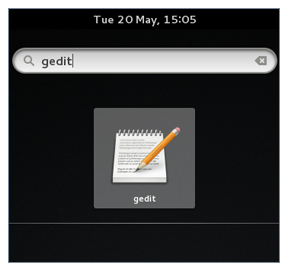
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/ch4_screen50.jpg)

**Nautilus** allows you to refine your search beyond the initial keyword by providing drop-down menus to further filter the search.

* Based on **Location** or **File Type**, select additional criteria from the drop-down.
* To regenerate the search, click the **Reload** button.
* To add multiple search criteria, click the **+** button and select **additional search criteria**.

For example, if you want to find a PDF file containing the word Linux in your **home** directory, navigate to your **home** directory and search for the word “Linux” . You should see that the default search criterion limits the search to your **home** directory already. To finish the job, click the + button to add another search criterion, select **File Type** for the type of criterion, and select **PDF** under the **File Type** drop-down.

**Editing a File**



Editing any text file through the graphical interface is easy in the **GNOME** desktop environment. Simply double-click the file on the desktop or in the **Nautilus** file browser window to open the file with the default text editor.

The default text editor in **GNOME** is **gedit**. It is simple yet powerful, ideal for editing documents, making quick notes, and programming. Although **gedit** is designed as a general purpose text editor, it offers additional features for spell checking, highlighting, file listings, and statistics.

You'll learn much more about using text editors in a later chapter.

**Removing a File**



Deleting a file in **Nautilus** will automatically move the deleted files to the .local/share/Trash/files/ directory (a trash can of sorts) under the user's **HOME** directory. There are several ways to delete files and directories using Nautilus.

* Select all the files and directories that you want to delete.
* Press **Delete** on your keyboard. Or, Right-click the file.
* Select **Move to Trash**. Or, Highlight the file.

Click **Edit** and **Move to Trash** through the graphical interface.

**More About Removing a File**



To **permanently** delete a file:

* On the left panel inside a **Nautilus** file browser window, right-click on the **Trash** directory.
* Select **Empty Trash**.

Alternatively, select the file or directory you want to permanently delete and press **Shift-Delete**.

As a precaution, you should **never delete your home directory** as doing so will most likely erase all your **GNOME** configuration files and possibly prevent you from logging in. Many personal system and program configurations are stored under your **home** directory.

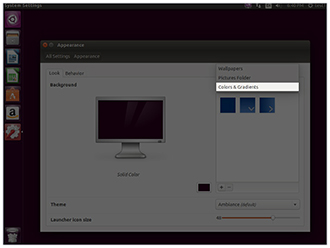
**Graphical Desktop**



Each Linux distribution comes with its own set of desktop backgrounds. You can change the default by choosing a new wallpaper or selecting a custom picture to be set as the desktop background. If you do not want to use an image as the background, you can select a color to be displayed on the desktop instead.

In addition, you can also change the desktop theme, which changes the look and feel of the Linux system. The theme also defines the appearance of application windows.

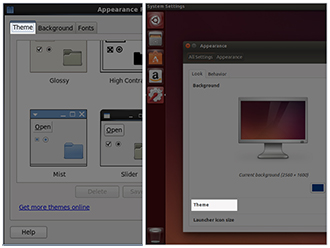
**Desktop Background**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen64.jpg)

If you do not like any of the installed wallpapers, you can use different shades of color as the background using the **Colors and Gradients** drop-down in the **Appearance** window.

There are three types of color: solid, horizontal gradient, and vertical gradient. Click the box at the bottom and pick the effect between solid and the two gradients. In addition, you can also install packages that contain wallpapers by searching for packages using “wallpaper” as a keyword.

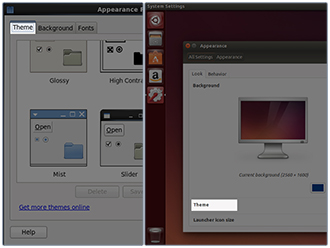
**Changing the Theme**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen72.jpg)

The visual appearances of applications (the buttons, scroll bars, widgets, and other graphical components) are controlled by a theme. **GNOME** comes with a set of different themes which can change the way your applications look. Some additional themes can also be found [here](https://wiki.gnome.org/Personalization).

The exact method for changing your theme will depend on your distribution. For example, for **Ubuntu** you can right-click anywhere on the desktop and select a different theme from the **Theme** drop-down.  For **CentOS** you have to click on System → Preferences → Appearance.

**More About Changing the Theme**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen73.jpg)

There are other options to get additional themes beyond the default selection. You can download and install themes from <http://art.gnome.org/themes/>

In **CentOS**, you can change themes from System → Preferences → Appearance

In **Ubuntu**, you can change themes from System Settings → Appearance

In **OpenSUSE** to change themes you will need to install the **gnome-tweak-tool** program.

**Summary (1 of 3)**

You have completed this chapter. Let's summarize the key concepts covered:

* **GNOME** is a popular desktop environment and graphical user interface that runs on top of the Linux operating system.
* The default display manager for **GNOME** is called **gdm**.
* The **gdm** display manager presents the user with the login screen which prompts for the login username and password.

Logging out through the desktop environment kills all processes in your current **X** session and returns to the display manager login screen.

**Summary (2 of 3)**

* 1. Linux enables users to switch between logged in sessions.
  2. Suspending puts the computer into sleep mode.
  3. For each key task, there is generally a default application installed.
  4. Every user created in the system will have a **home** directory.

The **Places** menu contains entries that allow you to access different parts of the computer and the network.

**Summary (3 of 3)**

* **Nautilus** gives three formats to view files.
* Most text editors are located in the **Accessories** submenu.
* Each Linux distribution comes with its own set of desktop backgrounds.

**GNOME** comes with a set of different themes which can change the way your applications look.

**Chapter 5: System Configuration from the Graphical Interface**

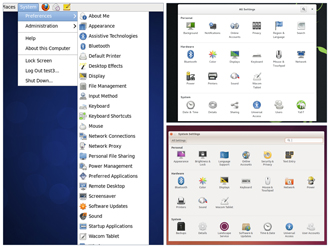
**Learning Objectives**

By the end of this chapter, you should be able to:

* Apply system, display, and date and time settings using the **System Settings** panel.
* Track the network settings and manage connections using **Network Manager** in Linux.

Install and update software in Linux from a graphical interface.

**System Settings**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch05_screen03.jpg)

The **System Settings** panel allows you to control most of the basic configuration options and desktop settings such as specifying the screen resolution, managing network connections, or changing the date and time of the system.

As we mentioned before, **CentOS** and **Ubuntu** use the **GNOME Desktop Manager**, while by default **OpenSUSE** uses the **KDE Desktop Manager**.

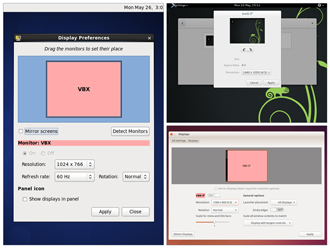
The procedure to access **System Settings** varies according to distribution:

 - **CentOS:** click System → Preferences.

 - **OpenSUSE:** click **Activities**, type **Settings** in the **Search** box.

* **Ubuntu**:they are located in the panel on the left of the screen.

**Display Settings**

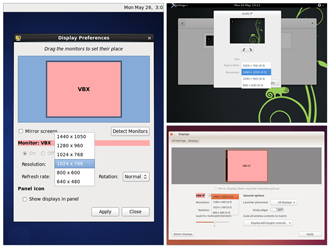
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch05_screen04.jpg)

The **Displays** panel under **System Settings** (or **Display and Monitor** panel under **Configure Desktop**) contains the most common settings for changing  the desktop appearance. These settings function independently of the specific display drivers you are running.

If your system uses a proprietary driver such as those from **nVidia** or **AMD**, you will probably have a configuration program for that driver that is not included in **System Settings**. This program may give more configuration options, but may also be more complicated, and might require sysadmin (root) access. If possible, you should configure the settings in the **Displays** panel rather than the proprietary configuration program.

The **X** server, which actually provides the GUI, uses the /etc/X11/xorg.conf file as its configuration file *if it exists*. In modern Linux distributions, this file is usually present only in unusual circumstances, such as when certain less common graphic drivers are in use. Changing this configuration file directly is usually for more advanced users.

**Setting Resolution and Configuring Multiple Screens**

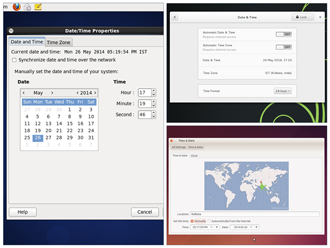
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch05_screen05.jpg)

While your system will usually figure out the best resolution for your screen automatically, it may get this wrong in some cases, or you might want to change the resolution to meet your needs.

You can accomplish this using the **Displays** panel. The switch to the new resolution will be effective when you click **Apply**, and then confirm that the resolution is working. In case the selected resolution fails to work or you are just not happy with the appearance, the system will switch back to the original resolution after a short timeout.

In most cases the configuration for multiple displays is set up automatically as one big screen spanning all monitors, using a reasonable guess for screen layout. If the screen layout is not as desired, a check box can turn on mirrored mode, where the same display is seen on all monitors.

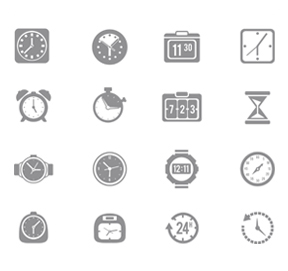
**Date and Time Settings**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch05_screen10.jpg)

Linux always uses **Coordinated Universal Time (UTC)**for its own internal time-keeping. Displayed or stored time values rely on the system time zone setting to get the proper time. UTC is similar to, but more accurate than, Greenwich Mean Time (GMT).

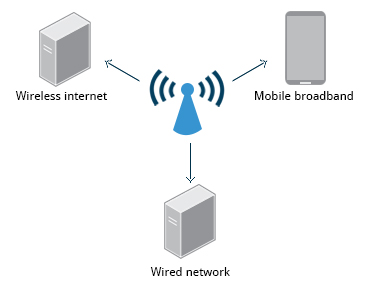
The **Date** **and Time Settings** window can be accessed from the **System Settings** window. Alternatively, you can right-click **Date and Time** on the top panel to access the **Date and Time Settings** window.

**Network Time Protocol**



The **Network Time Protocol (NTP)** is the most popular and reliable protocol for setting the local time via Internet servers. Most Linux distributions include a working NTP setup which refers to specific time servers run by the distribution. This means that no setup, beyond "on or off", is required for network time synchronization. If desired, more detailed configuration is possible by editing the standard NTP configuration file (/etc/ntp.conf) for Linux NTP utilities.

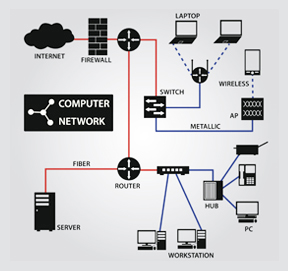
**Network Configuration**



All Linux distributions have network configuration files, but file formats and locations can differ from one distribution to another. Hand editing of these files can handle quite complicated setups, but is not very dynamic or easy to learn and use. The **Network Manager** utility was developed to make things easier and more uniform across distributions. It can list all available networks (both wired and wireless), allow the choice of a wired, wireless or mobile broadband network, handle passwords, and set up **Virtual Private Networks (VPNs)**. Except for unusual situations, it’s generally best to let the **Network Manager** establish your connections and keep track of your settings.

In this section, you will learn how to manage network connections, including wired and wireless connections, and mobile broadband and VPN connections.

**Wired and Wireless Connections**

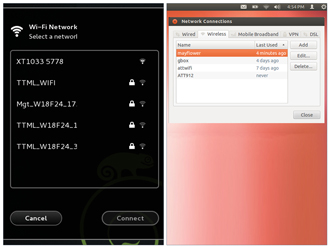


Wired connections usually do not require complicated or manual configuration. The hardware interface and signal presence are automatically detected, and then **Network Manager** sets the actual network settings via **DHCP**(Dynamic Host Control Protocol).

For **static** configurations that don't use **DHCP**, manual setup can also be done easily through **Network Manager**. You can also change the **Ethernet Media Access Control (MAC)** **address** if your hardware supports it. (The MAC address is a unique hexadecimal number of your network card.)

Wireless networks are not connected to the machine by default. You can view the list of available wireless networks and see which one you are connected to by using **Network Manager.**You can then add, edit, or remove known wireless networks, and also specify which ones you want connected by default when present.

**Configuring Wireless Connections in Ubuntu**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch05_screen21.jpg)

To configure **Wireless Network** in **Ubuntu**:

* In top panel, click **Network Manager**.
* Click **Enable Wi-Fi** - to display a list available **Wireless Networks**.
* Click the desired **Wireless Network**.
* For a secured network, enter the password.
* To modify saved wireless network settings, click **Edit Connections**.

**Configuring Wireless Connections in OpenSUSE**

**OpenSUSE** looks different from **CentOS** or **Ubuntu**, but Wired, Wireless, Mobile Broadband, VPN, and DSL are all available from the **Network Connections** dialog box as you will see in the upcoming demonstration.

**Mobile Broadband and VPN Connections**



You can set up a mobile broadband connection with **Network Manager**, which will launch a wizard to set up the connection details for each connection.

Once the configuration is done, the network is configured automatically each time the broadband network is attached.

**Network Manager** can also manage your VPN connections.

It supports many VPN technologies, such as native **IPSec**, **Cisco OpenConnect** (via either the Cisco client or a native open-source client), **Microsoft PPTP**, and **OpenVPN**.

You might get support for VPN as a separate package from your distributor. You need to install this package if your preferred VPN is not supported.

Task to be performed: Set IP address: 10.0.0.1 netmask: 255.255.255.0 to Wired Network Interface.

**Installing and Updating Software**



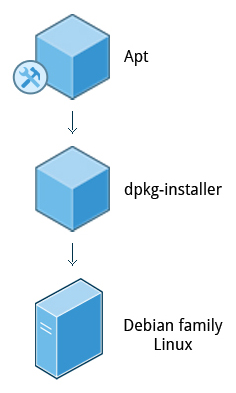
Each **package** in a Linux distribution provides one piece of the system, such as the Linux **kernel,** the **C** compiler, the shared software code for interacting with **USB** devices, or the **Firefox** web browser.

Packages often depend on each other; for example, because **Firefox** can communicate using SSL/TLS, it will depend on a package which provides the ability to encrypt and decrypt SSL and TLS communication, and will not install unless that package is also installed at the same time.

One utility handles the low-level details of unpacking a package and putting the pieces in the right places. Most of the time, you will be working with a higher-level utility which knows how to download packages from the Internet and can manage dependencies and groups for you.

In this section, you will learn how to install and update software in Linux using the **Debian** and **RPM** systems (which is used by both **Fedora** and **SUSE** family systems).

**Debian Family System**

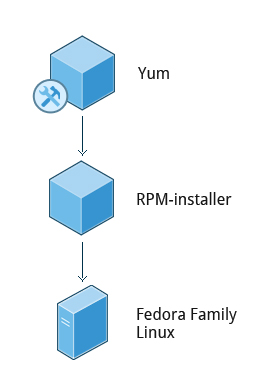


Let’s look at **Package Management** in the **Debian** Family System.

**dpkg** is the underlying package manager for these systems; it can install, remove, and build packages. Unlike higher-level package management systems, it does not automatically download and install packages and satisfy their dependencies.

For Debian-based systems, the higher-level package management system is the **apt** (**A**dvanced **P**ackage **T**ool) system of utilities. Generally, while each distribution within the **Debian** family uses **apt**,it creates its own user interface on top of it (for example, **apt-get**, **aptitude**, **synaptic**, **Ubuntu Software Center**, **Update Manager**, etc). Although **apt** repositories are generally compatible with each other, the software they contain generally isn’t. Therefore, most **apt** repositories target a particular distribution (like **Ubuntu**), and often software distributors ship with multiple repositories to support multiple distributions. The demonstration using the **Ubuntu Software Center** is shown later in this section.

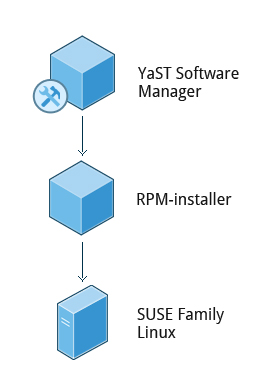
**Red Hat Packet Manager (RPM)**



**Red Hat Package Manager (RPM)** is the other package management system popular on Linux distributions. It was developed by **Red Hat**, and adopted by a number of other distributions, including the **OpenSUSE**, **Mandriva**, **CentOS**, **Oracle Linux**, and others.

The high-level package manager differs between distributions; most use the basic repository format used in **yum** (Yellowdog Updater, Modified - the package manager used by **Fedora** and **Red Hat Enterprise Linux**), but with enhancements and changes to fit the features they support. Recently, the **GNOME** project has been developing **PackageKit** as a unified interface; this is now the default interface for **Fedora**. The demonstration using **PackageKit** is shown later in this section.

**OpenSUSE’s YaST Software Management**



Before **OpenSUSE 13.1**, **Apper** was used for Software Management. Now it has been replaced by the **YaST** (**Y**et **a**nother **S**ystem **T**ool) **Software Manager**.

The **YaST Software Manager** is similar to other graphical package managers. It is an **RPM-**based application. You can add, remove, or update packages using this application very easily. To access the **YaST Software Manager**:

* Click **Activities**
* In the **Search** box type **YaST**
* Click the **YaST** icon
* Click **Software Management**

**OpenSUSE’s YaST Software Management** application is similar to the graphical package managers in other distributions. The demonstration of the **YaST Software Manager** is shown later in this section.

**Summary**

You have completed this chapter. Let's summarize the key concepts covered:

* You can control basic configuration options and desktop settings through the **System Settings** panel
* Linux always uses **Coordinated Universal Time (UTC)**for its own internal time-keeping . You can set **Date** **and Time Settings** from the **System Settings** window.
* The **Network Time Protocol** is the most popular and reliable protocol for setting the local time via Internet servers.
* The **Displays** panel allows you to change the resolution of your display and configure multiple screens.
* **Network Manager** can present available wireless networks, allow the choice of a wireless or mobile broadband network, handle passwords, and set up VPNs.
* **Debian** and **RPM** are the most popular package management systems used on Linux distributions.
* **Debian** distributions use **dpkg** and **apt**-based utilities for package management.

**RPM** was developed by **Red Hat**, and adopted by a number of other  distributions, including the **OpenSUSE**, **Mandriva**, **CentOS**, **Oracle Linux**, and others.

**Chapter 6: Command Line Operations**

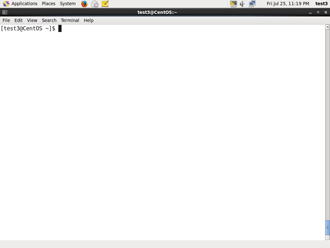
**Learning Objectives**

By the end of this chapter, you should be able to:

* Use the command line to perform operations in Linux.
* Search for files.
* Create and manage files.

Install and update software.

**Introduction to the Command Line**

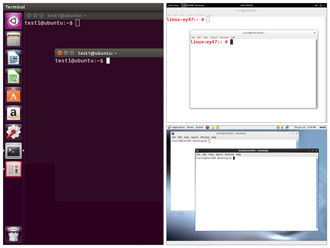
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen03.jpg)

Linux system administrators spend a significant amount of their time at a **command line** prompt. They often automate and troubleshoot tasks in this text environment. There is a saying, "*graphical user interfaces make easy tasks easier, while command line interfaces make difficult tasks possible*." Linux relies heavily on the abundance of command line tools. The command line interface provides the following advantages:

* No GUI overhead.
* Virtually every task can be accomplished using the command line.
* You can script tasks and series of procedures.
* You can log on remotely to networked machines anywhere on the Internet.

You can initiate graphical apps directly from the command line.

**Using a Text Terminal on the Graphical Desktop**

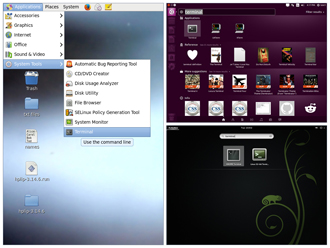
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen04.jpg)

A **terminal emulator** program emulates (simulates) a stand alone terminal within a window on the desktop. By this we mean it behaves essentially as if you were logging into the machine at a pure text terminal with no running graphical interface. Most terminal emulator programs support multiple terminal sessions by opening additional tabs or windows.

By default, on **GNOME** desktop environments, the**gnome-terminal** application is used to emulate a text-mode terminal in a window. Other available terminal programs include:

* **xterm**
* **rxvt**
* **console**
* **terminator**

**Launching Terminal Windows**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen05.jpg)

To open a terminal in **CentOS**:

* 1. On the **CentOS** desktop, in the upper-left corner, click **Applications**.
  2. From the **System** **Tools** menu, select **Terminal**.

To open a terminal in **OpenSUSE**:

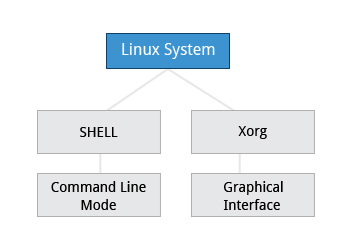
* 1. On the **OpenSUSE** desktop, in the upper-left corner of the screen, click **Activities**.
  2. From the left pane, click **Show** **Applications**.
  3. Scroll-down and select the required terminal.

To open a terminal in **Ubuntu**:

* 1. In the left panel, click the **Ubuntu** icon.
  2. Type **terminal** in the **Search** box.

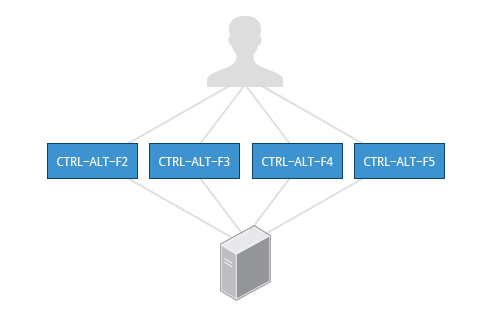
If the **nautilus-open-terminal** package is installed on any of these distributions, you can always open a terminal by right clicking anywhere on the desktop background and selecting **Open in Terminal.**

**The X Window System**



The customizable nature of Linux allows you to drop (temporarily or permanently) the **X Window** graphical interface, or to start it up after the system has been running. Certain Linux distributions distinguish versions of the install media between desktop (with **X**) and server (usually without **X**); Linux production servers are usually installed without **X** and even if it is installed, usually do not launch it during system start up. Removing **X** from a production server can be very helpful in maintaining a lean system which can be easier to support and keep secure.

**Virtual Terminals**



**Virtual Terminals (VT)** are **console** sessions that use the entire display and keyboard outside of a graphical environment. Such terminals are considered "virtual" because although there can be multiple active terminals, only one terminal remains visible at a time. A VT is not quite the same as a command line terminal window; you can have many of those visible at once on a graphical desktop.

One virtual terminal (usually number one or seven) is reserved for the graphical environment, and text logins are enabled on the unused VTs. **Ubuntu** uses VT 7, but **CentOS/RHEL** and **OpenSUSE** use VT 1 for the graphical display.

An example of a situation where using the VTs is helpful when you run into problems with the graphical desktop. In this situation, you can switch to one of the text VTs and troubleshoot.

To switch between the VTs, press **CTRL-ALT-corresponding function key** for the VT. For example, press **CTRL-ALT-F6** for VT 6. (Actually you only have to press **ALT-F6** key combination if you are ina VT not running **X** and want to switch to another VT.)

**The Command Line**

Most input lines entered at the shell prompt have three basic elements:

* Command
* Options
* Arguments

The **command** is the name of the program you are executing. It may be followed by one or more **options** (or switches) that modify what the command may do. Options usually start with one or two dashes, for example,-p or--print, in order to differentiate them from **arguments**, which represent what the command operates on.

However, plenty of commands have no options, no arguments, or neither. You can also type other things at the command line besides issuing commands, such as setting environment variables.

**Turning off the Graphical Desktop**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen08.jpg)

Linux distributions can start and stop the graphical desktop in various ways. For **Debian**-based systems, the **Desktop Manager** runs as a service which can be simply stopped. For RPM-based systems, the **Desktop** **Manager** is run directly by **init** when set to run level 5; switching to a different runlevel stops the desktop.

Use the sudo service gdm stop or sudo service lightdm stop commands, to stop the graphical user interface in **Debian**-based systems. On **RPM**-based systems typing sudo telinit 3 may have the same effect of killing the GUI.

**sudo**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch04_screen12.jpg)

All the demonstrations created have a user configured with **sudo**capabilities to provide the user with administrative (admin) privileges when required. **sudo**allows users to run programs using the security privileges of another user, generally root (superuser). The functionality of **sudo** is similar to that of **run as** in **Windows**.

On your own systems, you may need to set up and enable**sudo** to work correctly. To do this, you need to follow some steps that we won’t explain in much detail now, but you will learn about later in this course. When running on **Ubuntu**,**sudo** is already always  set up for you during installation. If you are running something in the **Fedora** or **OpenSUSE** families of distributions, you will likely need to set up **sudo**to work properly for you after initial installation.

Next, you will learn the steps to setup and run **sudo**on your system.

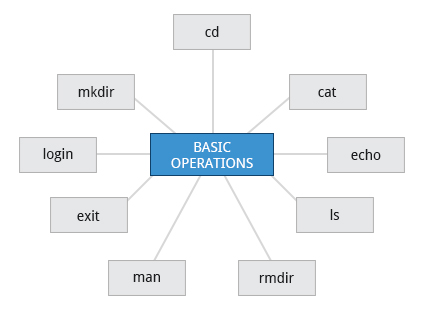
**Steps for Setting up and Running sudo**

If your system does not already have **sudo**set up and enabled, you need to do the following steps:

* You will need to make modifications as the administrative or super user, root. While **sudo** will become the preferred method of doing this, we don’t have it set up yet, so we will use **su** (which we will discuss later in detail) instead. At the command line prompt, type **su**and press **Enter.** You will then be prompted for the root password, so enter it and press **Enter**. You will notice that nothing is printed; this is so others cannot see the password on the screen. You should end up with a different looking prompt, often ending with ‘#’. For example: $ su Password: #
* Now you need to create a configuration file to enable your user account to use **sudo**. Typically, this file is created in the /etc/sudoers.d/ directory with the name of the file the same as your username. For example, for this demo, let’s say your username is “student”. After doing step 1, you would then create the configuration file for “student” by doing this: # echo "student ALL=(ALL) ALL" > /etc/sudoers.d/student
* Finally, some Linux distributions will complain if you don’t also change permissions on the file by doing: # chmod 440 /etc/sudoers.d/student

That should be it. For the rest of this course, if you use **sudo**you should be properly set up. When using **sudo,** by default you will be prompted to give a password (your own user password) at least the first time you do it within a specificed time interval.  It is possible (though very insecure) to configure **sudo** to not require a password or change the time window in which the password does not have to be repeated with every **sudo** command.

**Basic Operations**



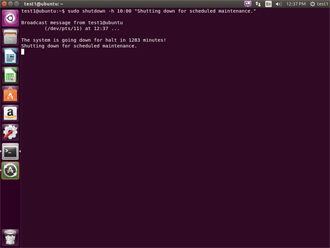
In this section we will discuss how to accomplish basic operations from the command line. These include how to log in and log out from the system, restart or shutdown the system, locate applications, access directories, identify the absolute and relative paths, and explore the filesystem.

**Logging In and Out**

An available **text terminal** will prompt for a username (with the string login:) and password. When typing your password, nothing is displayed on the terminal (not even a \* to indicate that you typed in something) to prevent others from seeing your password. After you have logged in to the system, you can perform basic operations.

Once your session is started (either by logging in to a text terminal or via a graphical terminal program) you can also connect and log in to remote systems via the **Secure** **Shell (SSH)** utility. For example, by typing ssh username@remote-server.com, **SSH** would connect securely to the remote machine and give you a command line terminal window, using passwords (as with regular logins) or cryptographic keys (a topic we won't discuss) to prove your identity.

**Rebooting and Shutting Down**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen14.jpg)

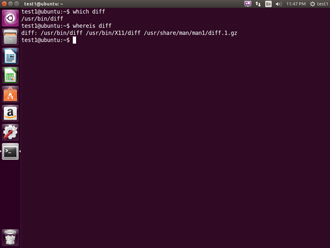
The preferred method to shut down or reboot the system is to use the **shutdown** command. This sends a warning message and then prevents further users from logging in. The **init** process will then control shutting down or rebooting the system. It is important to always shut down properly; failure to do so can result in damage to the system and/or loss of data.

The **halt** and **poweroff**commands issue shutdown -h to halt the system; **reboot**issues shutdown -r and causes the machine to reboot instead of just shutting down. Both rebooting and shutting down from the command line requires superuser (root) access.

When administering a multiuser system, you have the option of notifying all users prior to shutdown as in:

$ sudo shutdown -h 10:00 "Shutting down for scheduled maintenance."

**Locating Applications**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen15.jpg)

Depending on the specifics of your particular distribution's policy, programs and software packages can be installed in various directories. In general, executable programs should live in the /bin, /usr/bin,/sbin,/usr/sbin directories or under /opt.

One way to locate programs is to employ the **which** utility. For example, to find out exactly where the **diff** program resides on the filesystem:

$ which diff

If **which** does not find the program, **whereis** is a good alternative because it looks for packages in a broader range of system directories:

$ whereis diff

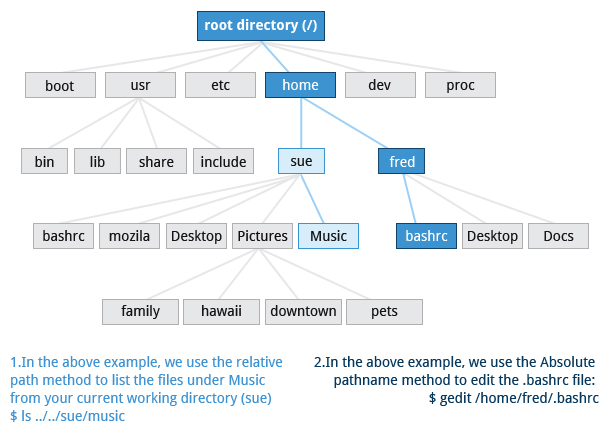
Click the image to view an enlarged version.

**Accessing Directories**

When you first log into a system or open a terminal, the default directory should be your **home directory**; you can print the exact path of this by typing echo $HOME. (Note that some Linux distributions actually open new **graphical** terminals in $HOME/Desktop.)  The following commands are useful for directory navigation:

|  |  |
| --- | --- |
| **Command** | **Result** |
| pwd | Displays the present working directory |
| cd ~ or cd | Change to your home directory (short-cut name is ~ (tilde)) |
| cd .. | Change to parent directory (..) |
| cd - | Change to previous directory (- (minus)) |

**Understanding Absolute and Relative Paths**



There are two ways to identify paths:

* **Absolute pathname**: An absolute pathname begins with the root directory and follows the tree, branch by branch, until it reaches the desired directory or file. Absolute paths always start with /.
* **Relative pathname**: A relative pathname starts from the present working directory. Relative paths never start with /.

Multiple slashes (/) between directories and files are allowed, but all but one slash between elements in the pathname is ignored by the system. ////usr//bin is valid, but seen as /usr/bin by the system.

Most of the time it is most convenient to use relative paths, which require less typing. Usually you take advantage of the shortcuts provided by: . (present directory), .. (parent directory) and ~ (your home directory).

For example, suppose you are currently working in your home directory and wish to move to the /usr/bin directory. The following two ways will bring you to the same directory from your home directory:

* Absolute pathname method: $ cd /usr/bin
* Relative pathname method:   $ cd ../../usr/bin

In this case, the absolute pathname method is less typing.

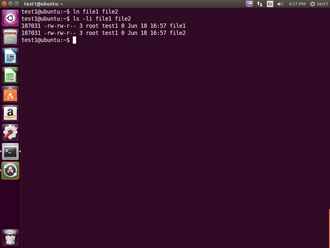
**Exploring the Filesystem**

Traversing up and down the filesystem tree can get tedious. The **tree** command is a good way to get a bird’s-eye view of the filesystem tree. Use tree -d to view just the directories and to suppress listing file names.

The following commands can help in exploring the filesystem:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| cd / | **C**hanges your current **d**irectory to the root (/) directory (or path you supply) |
| ls | **L**i**s**t the contents of the present working directory |
| ls –a | **L**i**s**t **all** files including **hidden** files and directories (those whose name start with . ) |
| tree | Displays a **tree** view of the filesystem |

**Hard and Soft (Symbolic) Links**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen21a.jpg)

**ln** can be used to create **hard links** and (with the -s option) **soft links**, also known as **symbolic links** or **symlinks**. These two kinds of links are very useful in UNIX-based operating systems. The advantages of symbolic links are discusssed on the following screen.

Suppose that file1 already exists. A **hard** link, called file2, is created with the command:

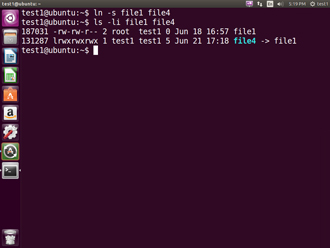
$ ln file1 file2

Note that two files now appear to exist. However, a closer inspection of the file listing shows that this is not quite true.

$ ls -li file1 file2

The -i option to **ls** prints out in the first column the **inode** number, which is a unique quantity for each file object. This field is the same for both of these files; what is really going on here is that it is only **one** file but it has more than one nameassociated with it,  as is indicated by the **3** that appears in the **ls** output.  Thus, there already was another object linked to file1 before the command was executed.

**Symbolic Links**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen23a.jpg)

**Symbolic** (or **Soft**) links are created with the -s option as in:

$ ln -s file1 file4 $ ls -li file1 file4

Notice file4 no longer appears to be a regular file, and it clearly points to file1 and has a different inode number.

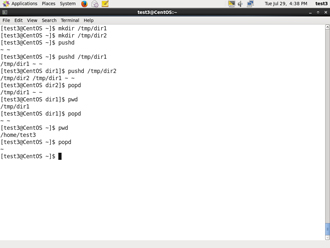
Symbolic links take no extra space on the filesystem (unless their names are very long). They are extremely convenient as they can easily be modified to point to different places. An easy way to create a shortcut from your **home** directory to long pathnames is to create a symbolic link.

Unlike hard links, soft links can point to objects even on different filesystems (or partitions) which may or may not be currently available or even exist. In the case where the link does not point to a currently available or existing object, you obtain a **dangling** link.

Hard links are very useful and they save space, but you have to be careful with their use, sometimes in subtle ways. For one thing if you remove either file1 or file2 in the example on the previous screen, the **inode object** (and the remaining file name) will remain, which might be undesirable as it may lead to subtle errors later if you recreate a file of that name.

If youedit one of the files, exactly what happens depends on your editor; most editors including **vi** and **gedit** will retain the link by default but it is possible that modifying one of the names may break the link and result in the creation of two objects.

**Navigating the Directory History**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen26.jpg)

The **cd**command remembers where you were last, and lets you get back there with cd -. For remembering more than just the last directory visited, use **pushd** to change the directory instead of **cd**; this pushes your starting directory onto a list. Using **popd**will then send you back to those directories, walking in reverse order (the most recent directory will be the first one retrieved with **popd**). The list of directories is displayed with the **dirs**command.

**Standard File Streams**

When commands are executed, by default there are three standard **file streams** (or **descriptors**) always open for use: **standard input** (standard in or **stdin**), **standard output** (standard out or **stdout**) and **standard error** (or **stderr**). Usually, **stdin** is your keyboard, **stdout** and **stderr** are printed on your terminal; often **stderr** is redirected to an error logging file. **stdin** is often supplied by directing input to come from a file or from the output of a previous command through a **pipe**. **stdout** is also often redirected into a file. Since **stderr** is where error messages are written, often nothing will go there.

In Linux, all open files are represented internally by what are called **file descriptors**. Simply put, these are represented by numbers starting at zero. **stdin** is file descriptor 0, **stdout** is file descriptor 1, and **stderr** is file descriptor 2. Typically, if other files are opened in addition to these three, which are opened by default, they will start at file descriptor 3 and increase from there.

**I/O Redirection**

Through the command **shell** we can **redirect** the three standard filestreams so that we can get input from either a file or another command instead of from our keyboard, and we can write output and errors to files or send them as input for subsequent commands.

For example, if we have a program called **do\_something** that reads from **stdin** and writes to **stdout** and **stderr**, we can change its input source by using the less-than sign ( < ) followed by the name of the file to be consumed for input data:

$ do\_something < input-file

If you want to send the output to a file, use the greater-than sign (>) as in: $ do\_something > output-file

Because **stderr** is **not** the same as **stdout**, error messages will still be seen on the terminal windows in the above example.

If you want to redirect **stderr** to a separate file, you use **stderr’s** file descriptor number (2), the greater-than sign (>), followed by the name of the file you want to hold everything the running command writes to **stderr**: $ do\_something 2> error-file

A special shorthand notation can be used to put anything written to file descriptor 2 (**stderr**) in the same place as file descriptor 1 (**stdout**): 2>&1 $ do\_something > all-output-file 2>&1

**bash**permits an easier syntax for the above:

$ do\_something >& all-output-file

**Pipes**

The UNIX/Linux philosophy is to have many simple and short programs (or commands) cooperate together to produce quite complex results, rather than have one complex program with many possible options and modes of operation. In order to accomplish this, extensive use of **pipes** is made; you can pipe the output of one command or program into another as its input.

In order to do this we use the vertical-bar, |, (pipe symbol) between commands as in:  $ command1 | command2 | command3

The above represents what we often call a **pipeline** and allows Linux to combine the actions of several commands into one. This is extraordinarily efficient because **command2**and **command3** do not have to wait for the previous pipeline commands to complete before they can begin hacking at the data in their input streams; on multiple CPU or core systems the available computing power is much better utilized and things get done quicker. In addition there is no need to save output in (temporary) files between the stages in the pipeline, which saves diskspace and reduces reading and writing from disk, which is often the slowest bottleneck in getting something done.

**Searching for Files**



Being able to quickly find the files you are looking for will make you a much happier Linux user! You can search for files in your parent directory or any other directory on the system as needed.

In this section, you will learn how to use the **locate** and **find**utilities, and how to use **wildcards** in **bash**.

**locate**



The **locate** utility program performs a search through a previously constructed database of files and directories on your system, matching all entries that contain a specified character string. This can sometimes result in a very long list.

To get a shorter more relevant list we can use the **grep** program as a filter; **grep** will print only the lines that contain one or more specified strings as in:

$ locate zip | grep bin

which will list all files and directories with both "zip" and "bin" in their name . (We will cover **grep**in much more detail later.) Notice the use of **|** to pipe the two commands together.

**locate** utiliizes the database created by another program, **updatedb.** Most Linux systems run this automatically once a day. However, you can update it at any time by just running **updatedb** from the command line as the root user.

**Try-It-Yourself: Locating Files**

Tasks to be performed:

* Find all files with the extension .doc. [locate .doc]
* Copy the file named **pdb.doc** from /usr/lib/python2.7 to current working directory as **Myfile.doc**. [cp /usr/lib/python2.7/pdb.doc Myfile.doc]
* Update database using updatedb command. [sudo updated]

Locate the file **Myfile.doc** (remember that filenames are case sensitive!). [locate Myfile.doc]

**Wildcards and Matching File Names**

You can search for a filename containing specific characters using **wildcards**.

|  |  |
| --- | --- |
| **Wildcard** | **Result** |
| ? | Matches any single character |
| \* | Matches any string of characters |
| [set] | Matches any character in the set of characters, for example [adf] will match any occurrence of "a", "d", or "f" |
| [!set] | Matches any character not in the set of characters |

To search for files using the ? wildcard, replace each unknown **character** with ?, e.g. if you know only the first 2 letters are 'ba' of a 3-letter filename with an extension of .out, type ls ba?.out.

To search for files using the \* wildcard, replace the unknown **string** with \*, e.g. if you remember only that the extension was .out, type ls \*.out

**Try-It-Yourself: Locating Files using bash Wildcards**

Tasks to be performed:

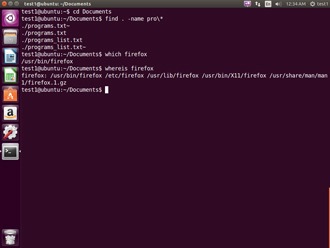
* View all the files in the current directory using ls command with -a option. [ls –a]
* List (using ls) files with names starting with g and containing five letters.

[ls g????]

* List (using ls) files whose names begin with mk and end with any characters. [ls mk\*]
* List (using ls) files having five letter names starting with g and second character between a-n. [ls g[a-n]???]

List (using ls) five letter named files starting with g and not having the second character between a-m. [ls g[!a-m]???]

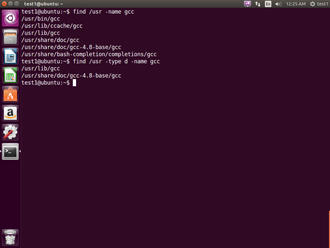
**Finding Files In a Directory**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen39.jpg)

**find**is extremely useful and often-used utility program in the daily life of a Linux system administrator. It recurses down the filesystem tree from any particular directory (or set of directories) and locates files that match specified conditions. The default pathname is always the present working directory.

For example, administrators sometimes scan for large **core files** (which contain diagnostic information after a program fails) that are more than several weeks old in order to remove them. It is also common to remove files in **/tmp** (and other temporary directories, such as those containing cached files) that have not been accessed recently. Many distros use automated scripts that run periodically to accomplish such house cleaning.

Using **find**

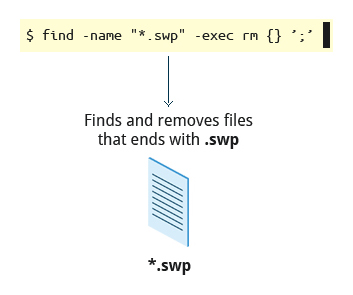
[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen40.jpg)

When no arguments are given, **find** lists all files in the current directory and all of its subdirectories. Commonly used options to shorten the list include -name (only list files with a certain pattern in their name), -iname (also ignore the case of file names), and -type (which will restrict the results to files of a certain specified type, such as **d** for directory or **l** for symbolic link, etc).

Searching for files and directories named "gcc": $ find /usr -name gcc

Searching only for directories named "gcc": $ find /usr -type d -name gcc

**Using Advanced find Options**



Another good use of **find** is being able to run commands on the files that match your search criteria. The -exec option is used for this purpose.

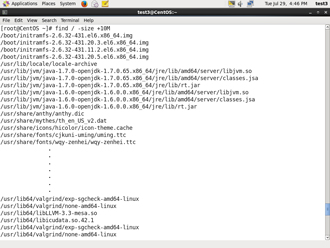
To find and remove all files that end with .swp: $ find -name "\*.swp" -exec rm {} ’;’

The {} (squiggly brackets) is a place holder that will be filled with all the file names that result from the **find** expression, and the preceding command will be run on each one individually.

Note that you have to end the command with either ‘;’ (including the single-quotes) or \; Both forms are fine.

One can also use the -ok option which behaves the same as -exec except that **find**will prompt you for permission before executing the command. This makes it a good way to test your results before blindly executing any potentially dangerous commands.

**Finding Files Based on Time and Size**

[](https://courses.edx.org/c4x/LinuxFoundationX/LFS101x/asset/LFS01_ch06_screen42.jpg)

It is sometimes the case that you wish to find files according to attributes such as when they were created, last used, etc, or based on their size. Both are easy to accomplish.

Finding based on time: $ find / -ctime 3

Here, -ctime is when the inode meta-data (i.e., file ownership, permissions, etc) last changed; it is often, but not necessarily when the file was first created. You can also search for accessed/last read (-atime) or modified/last written (-mtime) times. The number is the number of days and can be expressed as either a number (n) that means exactly that value, +n which means greater than that number, or -n which means less than that number. There are similar options for times in minutes (as in -cmin, -amin, and -mmin).

Finding based on sizes:

$ find / -size 0

Note the size here is in 512-byte blocks, by default; you can also specify bytes (**c**), kilobytes (**k**), megabytes (**M**), gigabytes (**G**), etc. As with the time numbers above, file sizes can also be exact numbers (n), +n or -n. For details consult the **man**page for **find**.

For example, to find files greater than 10 MB in size and running a command on those files: $ find / -size +10M -exec command {} ’;’

**Working with Files**



Linux provides many commands that help you in viewing the contents of a file, creating a new file or an empty file, changing the **timestamp** of a file, and removing and renaming a file or directory. These commands help you in managing your data and files and in ensuring that the correct data is available at the correct location.

**Viewing Files**

You can use the following utilities to view files:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| **cat** | Used for viewing files that are not very long; it does not provide any scroll-back. |
| **tac** | Used to look at a file backwards, one line at a time. |
| **less** | Used to view larger files because it is a paging program; it pauses at each screenful of text, provides scroll-back capabilities, and lets you are search and navigate within the file. Note: Use / to search for a pattern in the forward direction and ? for a pattern in the backward direction. |
| **tail** | Used to print the last 10 lines of a file by default. You can change the number of lines by doing -n 15 or just -15 if you wanted to look at the last 15 lines instead of the default. |
| **head** | The opposite of **tail**; by default it prints the first 10 lines of a file. |

**touch and mkdir**



**touch** is often used to set or update the access, change, and modify times of files. By default it resets a file's time stamp to match the current time.

However, you can also create an **empty** file using touch: $ touch <filename>

This is normally done to create an empty file as a placeholder for a later purpose.

**touch**provides the following options:

* The -t option allows you to set the date and time stamp of the file.
* The -c option allows you to test (or check) the directory permission (to see if you can create files in the directory).

To set the time stamp to a specific time: $ touch -t 03201600 myfile

This sets the file, myfile's, time stamp to 4 p.m., March 20th (03 20 1600).

**mkdir** is used to create a directory.

* To create a sample directory named sampdir under the current directory, type mkdir sampdir.
* To create a sample directory called sampdir under /usr, type mkdir /usr/sampdir.

Removing a directory is simply done with **rmdir.**The directory must be empty or it will fail. To remove a directory and all of its contents you have to do rm -rf as we shall discuss.

**Removing a File**

|  |  |
| --- | --- |
| **Command** | **Usage** |
| mv | Rename a file |
| rm | Remove a file |
| rm –f | Forcefully remove a file |
| rm –i | Interactively remove a file |

If you are not certain about removing files that match a pattern you supply, it is always good to run **rm** interactively (rm –i) to prompt before every removal.

**Renaming or Removing a Directory**

**rmdir**works only on empty directories; otherwise you get an error.

While typing rm –rf is a fast and easy way to remove a whole filesystem tree recursively, it is extremely dangerous and should be used with the utmost care, especially when used by root (recall that recursive means drilling down through all sub-directories, all the way down a tree). Below are the commands used to rename or remove a directory:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| mv | Rename a directory |
| rmdir | Remove an empty directory |
| rm -rf | Forcefully remove a directory recursively |

**Modifying the Command Line Prompt**

The **PS1** variable is the character string that is displayed as the prompt on the command line. Most distributions set **PS1** to a known default value, which is suitable in most cases. However, users may want custom information to show on the command line. For example, some system administrators require the user and the host system name to show up on the command line as in:

student@quad32 $

This could prove useful if you are working in multiple roles and want to be always reminded of who you are and what machine you are on. The prompt above could be implemented by setting the PS1 variable to: \u@\h \$

For example:

$ echo $PS1 \$ $ PS1="\u@\h \$ " coop@quad64 $ echo $PS1 \u@\h \$  coop@quad64 $

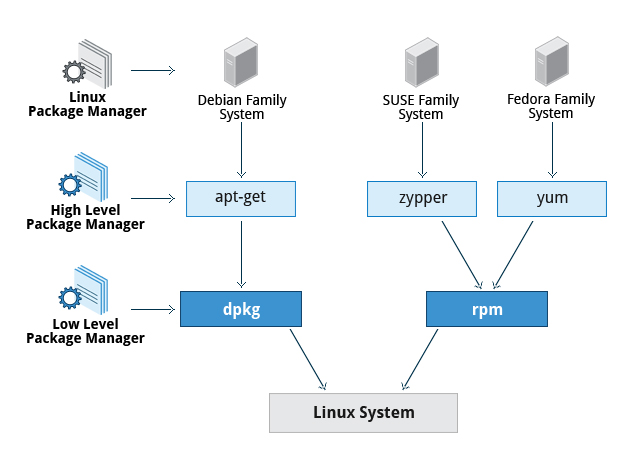
**Package Management Systems on Linux**



The core parts of a Linux distribution and most of its add-on software are installed via the **Package Management System**. Each package contains the files and other instructions needed to make one software component work on the system. Packages can depend on each other. For example, a package for a Web-based application written in PHP can depend on the PHP package.

There are two broad families of package managers: those based on **Debian** and those which use **RPM** as their low-level package manager. The two systems are incompatible, but provide the same features at a broad level.

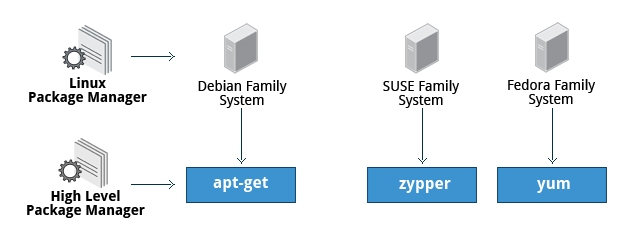
**Package Managers: Two Levels**



Both package management systems provide two tool levels: a low-level tool (such as **dpkg** or **rpm**), takes care of the details of unpacking individual packages, running scripts, getting the software installed correctly, while a high-level tool (such as **apt-get**, **yum**, or **zypper**) works with groups of packages, downloads packages from the vendor, and figures out dependencies.

Most of the time users need work only with the high-level tool, which will take care of calling the low-level tool as needed. Dependency tracking is a particularly important feature of the high-level tool, as it handles the details of finding and installing each dependency for you. Be careful, however, as installing a single package could result in many dozens or even hundreds of dependent packages being installed.

**Working With Different Package Management Systems**



* The **Advanced Packaging Tool** (apt) is the underlying package management system that manages software on Debian-based systems. While it forms the backend for graphical package managers, such as the **Ubuntu Software Center** and **synaptic**, its native user interface is at the command line, with programs that include apt-get and apt-cache.
* **Yellowdog Updater Modified** (**yum**) is an open-source command-line package-management utility for RPM-compatible Linux systems, basically what we have called the **Fedora** family. **yum** has both command line and graphical user interfaces.

**zypper** is a package management system for **openSUSE** that is based on RPM. **zypper** also allows you to manage repositories from the command line. **zypper** is fairly straightforward to use and resembles **yum**

**Summary (1 of 2)**



You have completed this chapter. Let’s summarize the key concepts covered.

* Virtual terminals (VT) in Linux are consoles, or command line terminals that use the connected monitor and keyboard.
* Different Linux distributions start and stop the graphical desktop in different ways.
* A terminal emulator program on the graphical desktop works by emulating a terminal within a window on the desktop.
* The Linux system allows you to either log in via text terminal or remotely via the console.
* When typing your password, nothing is printed to the terminal, not even a generic symbol to indicate that you typed.
* The preferred method to shut down or reboot the system is to use the **shutdown** command.
* There are two types of **pathnames:** absolute and relative.

An absolute pathname begins with the root directory and follows the tree, branch by branch, until it reaches the desired directory or file.

**Summary (2 of 2)**



* A relative pathname starts from the present working directory.
* Using**hard** and**soft** (**symbolic**) links is extremely useful in Linux.
* **cd** remembers where you were last, and lets you get back there with cd -.
* **locate**performs a database search to find all file names that match a given pattern.
* **find**locates files recursively from a given directory or set of directories.
* **find** is able to run commands on the files that it lists, when used with the -exec option.
* **touch**is used to set the access, change, and edit times of files as well as to create empty files.
* The **Advanced Packaging Tool** (apt) package management system is used to manage installed software on Debian-based systems.
* You can use the **Yellowdog Updater Modified** (yum) open-source command-line package-management utility for **RPM**-compatible Linux operating systems.

The **zypper** package management system is based on RPM and used for openSUSE.