Chapter 1: The Linux Foundation

INTRODUCTION

This course will teach: * the fundamentals of what Linux is * Teach you how to use the command line

We will talk about some of the basics of the Linux foundation. This intro will also talk through some of the basics of how this course will work. We will also pick out a Linux distribution that is right for you.

SECTION 1: THE LINUX FOUNDATION

Linux started in 1991 and powers everything from the New York Stock Exchange to phones to supercomputers. The Linux foundation was founded in 2000 and serves to promote protect and advance Linux. The Linux foundation is supported by leading technology companies and thousands of individuals. The Linux foundation: * Produces technical events throughout the world * Develops and delivers training programs * Host collaborative projects * Manages kernel.org where the official Linux kernel is released * Runs Linux.com

The Linux foundation hosts conferences throughout the world that: * provides an open forum for developing the next kernel release * brings together developers and sysadmins to solve problems * Hosts workgroups and community groups * Connects end users, sysadmins, and kernel developers * Encourages collaboration among the community

Linux Events are places where the industry and the Linux community meet.

SECTION 2: LINUX FOUNDATION TRAINING

Linux, despite how it is portrayed, doesn't have to be intimidating. We can collaborate to jumpstart our training. Linux Foundation classes help build skills that individuals and organizations need. With these skills, we can continue on with self-directed learning. The Linux foundation training is designed by members of the community. The sysadmin course focuses on enterprise Linux environments. They target: * System administrators * technical support * architects

The developer course features instructors and content from the leaders of the Linux community The Linux foundation training is: * distribution-flexible * technically advanced * created with actual leaders of the Linux development community

Most courses are more than 50% hands-on. The Linux Foundation training options. There are a ton of Linux jobs available on the market today and not enough people to fill that demand. The Linux foundation seeks to make a strong pool of talent to meet that need. Because of the scattered nature of a lot of things in Linux, it is difficult for people to assemble material on their own. The Linux foundation is vendor neutral. Everything is extremely up to date.

The foundation has a strong connection with a lot of leading kernel developers so it is easy to contribute to development. Students will get detailed course manuals that are essentially books. They are easy to reference later on in a few years. After the course, it is easy to find jobs in the industry. Can integrate into the professional community and the development community. The Linux foundation has the copyright for all of these course materials.

SECTION 3: COURSE LINUX REQUIREMENTS

In order to fully benefit from this course, you should have at least one Linux distro installed. In this section, we will learn about some of the available distros. The families and representative distributions are: * Debian family systems (Ubuntu) * SUSE family systems (openSUSE) * Fedora family systems (CentOS)

There are three main Linux distributions, all of them are based around the Linux kernel + the Debian family * Ubuntu * Linux mint + the Fedora family * RHEL + CentOS + OracleLinux + the SUSE family * SLES * openSUSE + other distros

Fedora is the community distro that forms the basis of RHEL, CentOS, Scientific Linux, and Oracle Linux. Fedora contains much more software than RHEL. One reason is that there is a diverse community building fedora. centos is used in this demo mainly because it has a longer dev cycle than fedora. The Linux kernel in RHEL/CentOS 6.x is 2.6.32. It supports x86, x86-64, itanium, powerpc, and ibm system z. fedora uses the RPM-based yum package manager. RHEL is widely used by enterprise which host their own systems.

SUSE, SUSE Linux Enterprise Server (SLES), and openSUSE are similar to the relationship between the fedora-based distros. openSUSE 12.3 is used as an example as this one. Most things in openSUSE can be applied to SLES and SUSE with no problem. SLES is upstream for openSUSE. openSUSE 12.3 uses Linux kernel 3.11. SUSE uses the RPM-based zypper package manager. it includes YaST (Yet another System Tool) application for sysadmin purposes. SUSE is widely used in the retail sector.

The debian distro is upstream for several distributions such as ubuntu, which is upstream for Linux mint. It is commonly used on servers and desktop computers. Debian is a pure open source project and focuses on stability. It also provides the largest and most complete software repository to its users. Ubuntu aims at providing a compromise between stability and ease of use. Ubuntu gets most of its packages from Debian, so it also has a large software repository. Ubuntu 14.04 LTS (long term support) is used in this course. Ubuntu 14.04 uses Linux kernel 3.13. Debian-based systems use the DPKG-based apt-get package manager. Ubuntu has been widely used for cloud deployments. Ubuntu now uses the unity graphical interface and is GNOME-based.

The material provided by the Linux foundation is distribution-flexible. Technical explanations, labs, and procedures should be applicable to most modern distros. When choosing a Linux distro, you will notice that most technical differences are mainly: * package-management systems * software versions * file locations The desktop environment used for this course is GNOME. There are several different environments, but GNOME is the most popular.

SUMMARY

- The Linux foundation is a nonprofit consortium dedicated to fostering the growth of Linux. * the Linux foundation training is for the community by the community. * It is distribution-flexible, technically advanced, and created with the leaders of the Linux community. * The three main distro families are fedora, SUSE, and Debian. # Chapter 2: Linux Philosophy and Concepts

INTRODUCTION

By the end of this chapter you should be able to: * Understand the history and philosophy of Linux * Describe the Linux community * Define the common terms associated with Linux * Understand the components of a Linux distribution

SECTION 1: Linux HISTORY

Linux is a free open-source computer OS initially developed for intel x86 personal computers. It has been ported to many other hardware platforms. Linux Torvalds was a student in Finland in 1991 when started a project to write his own OS kernel. He also collected/developed the other parts to create and entire operation system with his kernel. This became known as the Linux kernel. In 1992, Linux was re-liscenced using the General Public License(GPL) by GNU (a project of the free software foundation (FSF)). This made it possible to build a worldwide community of developers. Numerous other developers created complete systems called Linux distros in the mid-90s. The Linux distros created in the mid-90s provided the basis for fully free computing. This became a driving force in the open-source community. In 1998, major companies like IBM and Oracle announced support for the Linux platform and began their own development efforts. Today, Linux powers more than half the servers on the internet, android, and nearly all of the world's supercomputers.

SECTION 2: Linux PHILOSOPHY

Every organization or project has a philosophy that works as a guide while framing its objectives and creating its growth path. This section talks about the Linux philosophy and its impact on development. Linux is constantly enhanced and maintained by developers from all over the world collaborating over the internet. Linux Torvalds remains the head of all this. Technical skill and desire are the only qualifications for participating and contributing. Linux borrows heavily from the UNIX operation system because it was meant to be an open-source version of UNIX. Files are stored in a heirarchical file system with the top of the system (root) being `/`. Linux always tries to make its components available via files or objects that look like files. Processes, devices, and network sockets are all represented by file-like objects. They can often be worked on using the same utilities used for regular files. Linux is a fully multitasking, multiuser OS with built-in networking and service processes (daemons for the UNIX world).

You use Linux everyday. over 850,000 android phones are activated every day. nearly 700,000 TVs are sold every day, most of which run Linux. Google, Facebook, Twitter, and Amazon are all powered by Linux. Linux is built collaboratively. This is the largest collaborative development project in human history. There are 15 million lines of code. 1.5 million lines were written in the last couple of years. Linux is developed very quickly. A new kernel comes out every 2-3 months. When submitting code to the Linux kernel, patches are submitted. They can: * rework things * improve performance * fix a problem * add new support for a device * add a new feature

Developers post their patches to the relevant mailing list. Patches are accepted by senior developers or maintainers. When the maintainer finishes their review, they will send the patch to Linus, who has ultimate authority on which patches are applied.

SECTION 3: Linux COMMUNITY

Suppose as part of your job you need to configure a Linux file server and you run into some difficulties. If you can't figure out the problem yourself, the Linux community can save the day. There are many ways to engage with the Linux community: * post questions on relevant discussion forums * subscribe to discussion threads * join local Linux groups

In August 1991, Linux Torvalds posted his announcement of Linux. The penguin is Linux's official mascot. Linus decided to use the GPL license created by Richard Stallman. The GPL license gives the user: 1 The freedom to use the software for any purpose 2 The freedom to change the software to suit your needs 3 The freedom to share the software with your friends and neighbors 4 The freedom to share the changes you make

These ideas fueled Linux's spread around the world. It forms the foundation of a thriving industry. In 1999, Red Hat stock tripled as they were the first Linux company to go public. Linux helped fuel the rise of the internet. Whenever something is this disruptive, there is bound to be competitive crossfire. The Linux development community numbers in the thousands. You use Linux literally every time you go on the internet. Linux is in many devices that you use every day. The Linux community has severe ways to connect with each other including: * Linux User Groups (both local and online) * Internet Relay Chat (IRC) channels such as pidgin and xchat * online communities and discussion boards * newsgroups and mailing lists * Community events such as Linuxcon and Apachecon

The official Linux.com website serves over one million unique visitors every month and has active sections on: * news * community discussion threads * free tutorials and user tips

SECTION 4: Linux TERMINOLOGY

There are many unfamiliar terms that you may come across when you start to use Linux. You need to be aware of some basic terms such as: * kernel * distribution * boot loader * service * filesystem * X window system * desktop environment * command line

The kernel is the glue between hardware and applications. considered the brain of the operating system e.g. the Linux kernel. A distribution (distro) is a collection of software making up a Linux-based OS. The bootloader is a program that boots the operating system e.g. GRUB and ISOLinux. A service is a program that runs as a background process e.g. httpd, named. A filesystem is a method for storing and organizing files e.g. ext3, ext4, FAT, XFS, NTFS. The X Window

System provides the standard toolkit to build GUIs. The desktop environment is the GUI on top f the OS. The command line is an interface for typing commands on top of the operation system. The shell is the command line interpreter that interprets the command line input and tells the operating system how to process those inputs.

SECTION 5: Linux DISTRIBUTIONS

Suppose you have been assigned a project building a product for Linux. If we want the project to work on all Linux systems, then we need to learn about different: * components * services * configurations

that are associated with each distribution. A Linux distribution is related to the Linux kernel because it consists of the kernel plus all the other software tools. These tools can be used for: * file-related operations * user management * software package management * generic proprietary applications * Generic free applications * Distro-specific applications * manuals for applications * support services

The current Linux kernel and all past kernels can be found at Linux.org. Various distros can be based on various kernel versions. Other essential tools of Linux distributions include: * C/C++ compiler * gdb debugger * core system libraries needed for programs to run * low-level interface for drawing graphics on the screen * system for installing and updating the various components including the kernel itself

A vast variety of Linux distros cater to different audiences and organizations depending on their specific needs. Large organizations tend to favor the commercially supported distros e.g. RHEL, SUSE, Ubuntu. CentOS is a popular free alternative to RHEL. Ubuntu and Fedora are popular in the education realm. Scientific Linux is favored by the research community. Many commercial distros offer long-term fee-based support for their distros, hardware, and software certification. All major distros provide update services for keeping your system primed with security and bug fixes and online support.

SUMMARY

- The Linux community is the far-reaching ecosystem of envelopers, vendors and users. * They support and advance the Linux OS. * A Linux distro consists of the kernel and other software tools for various tasks. # Chapter 3: Linux Structure and Installation

INTRODUCTION

By the end of this chapter, you should be able to: * identify Linux filesystems * identify the differences between partitions and filesystems * describe the boot process * know how to install Linux on a computer

SECTION 1: LINUX FILESYSTEM BASICS

A filesystem is a method for storing and organizing arbitrary collections of data in a human-usable form. There are several different types of filesystems that Linux can use: * Conventional disk filesystems (ext2, ext3, ext4, XFS, Btrfs, JFS, NTFS, etc.) * Flash storage filesystems (ubifs, JFFS2, YAFFS, etc.) * Database filesystems * Special purpose filesystems (procfs, sysfs, tmpfs, debugfs, etc.)

This section will describe the standard filesystem layout shared by most Linux distributions. A partition is a logical part of the disk, whereas a filesystem is a method of storing/finding those files. The filesystem can be though of as being a family tree that shows descendants and their relationship. A partition can be though of as the different families, each with their own family tree. Windows partitions are Disk1; Linux partitions are /dev/sda1. Windows filesystem types are NTFS/FAT32; Linux filesystem types are usually EXT3/EXT4/XFS…. Windows mounting parameters are drive letters; Linux mounting parameters are mount points. The windows base folder (for the OS) is the C drive; Linux base folder is `/`.

Linux filesystems store their important files in a hierarchy called the filesystem hierarchy standard (FHS). Filesystem standard information can be found here - https://courses.edx.org/asset-v1:LinuxFoundationX+LFS101x.2+1T2015+type@asset+block/LFS101_Ch3_Sec
The original source documents are also on the website here - http://refspecs.Linuxfoundation.org/fhs.shtml. This standard ensures that users can move between distros without having to relearn where everything is. Linux uses the `/` character to separate paths unlike windows which uses `/`. New drives are mounted as directories in a single filesystem (usually under /media). A CD-ROM disc labeled FEDORA might end up being found at `/media/FEDORA`. Linux filesystem names are case-sensitive. Many distributions place files not needed as a core utility under `/usr` (user).

SECTION 2: THE BOOT PROCESS

In this section, we will learn about what happens in the background when the power button is pressed. The boot process is the procedure for initializing the system. The basic process is: 1. power on 2. bios 3. master boot record 4. boot loader 5. kernel 6. initial ram disk 7. /sbin/init 8. command shell 9. x window system

The first step when you power the computer on is Basic Input/Output System (BIOS). This initializes the hardware including the screen and keyboard, and tests the main memory. This process is known as the Power On Self Test (POST). The BIOS software is stored on a ROM chip on the motherboard. After this, the remainder of the boot process is completed entirely by the OS. After BIOS is finished, control is passed to the boot loader. this is usually stored on one of the hard disks in the system, either in the boot sector or the EFI partition. EFI (Extensible Firmware Interface) or UEFI (Unified EFI). Before this stage, the machine doesn't have access to any mass storage media. Afterwards, information on date, time, and peripherals are loaded from CMOS values.

Various boot loaders exist for Linux. The most common ones are GRUB (GRand Unified Boot loader) and ISOLinux (for booting from removable media). Most Linux boot loaders present a user interface for choosing alternative options for booting Linux and other OSes. When booting Linux, the boot loader is responsible for loading the kernel image and the initial RAM disk into memory. The RAM disk contains some critical files and device drivers needed to start the system. The first stage of the boot loader resides at the first sector of the hard disk known as the Master Boot Record (MBR). The MBR is just 512 bytes. In this stage, the boot loader examines the partition table and founds a bootable partition. Upon finding a bootable partition, it looks for the second stage boot loader (e.g. GRUB). this is then loaded into RAM. This is for the BIOS/MBR method. Using the EFI/UEFI method, the UEFI firmware reads it's Boot Manager to determine which UEFI application is to be launched. The firmware launches the UEFI application (e.g. GRUB).

The second stage of the boot loader is located at /boot. A splash screen is displayed to pick the operation system to boot from. After choosing the OS, the boot loader loads the kernel of the selected operation system into RAM and passes control of the system to it. Kernels are almost always compressed, so the first thing it must do is uncompressed itself. Afterwards, it will check and analyze the system hardware and initialize any hardware device drivers built into the kernel. The boot loader will load both the kernel and an initial RAM-based file system called initramfs directly into memory so it can be used by the kernel. When the kernel is loaded into RAM, it initializes and configures the computer's memory and hardware attached to the system. This includes processors, I/O subsystems, storage devices, and various other things. The kernel also loads user space applications. The initramfs filesystem image contains programs and binary files that help mount the root filesystem. They provide kernel functionality for the needed filesystem and device drivers for mass storage controllers. These are loaded with a facility called udev (User DEVice).

After the root filesystem has been found, it is first checked for errors. The mount program instructs the OS that a filesystem is ready for use. It is then associated with a particular point in the overall hierarchy of the filesystem (the mount point). If this is successful, then the initramfs is cleared from RAM and the init program is executed `(/sbin/init)`. init handles the mounting and transferring over the the real root filesystem. If hardware drivers are needed for the real root filesystem, then the must be in the initramfs image. Once the kernel has set up all its hardware, then the /sbin/init program runs. This becomes the initial process. It starts other processes to get the system running. Most other processes trace their origin ultimately to init. The exceptions are the kernel processes for managing the internal OS details. Traditionally, the process startup was done using conventions that date back to System V UNIX.

The system passes through a sequence of runlevels that contain collections of scripts that start and stop services. Each runlevel supports a different mode of running the system. Within each runlevel, individual services can be set to run, or shut down if running. New distributions have begun moving away from System V standard, but usually still support it. Besides starting the system, init is responsible for keeping the system running and for shutting it down cleanly. It acts as a manager of last resort for all non-kenrel processes. It will clean up after them when necessary and restarts user login services as needed when users log in and log out. Near the end of the boot process, init starts a number of text-mode login prompts (through getty). These enable you to type in a username and password and get to a command shell. Usually, the default command shell is bash (gnu Borne Again Shell), but there are other command shells available. The shell prints a text-rompt indicating it is ready to accept commands. The terminals which run the command shells can be accessed using the ALT key plus a function key. Most distributions start six text terminals and one graphics terminal starting with f1 or f2. If the GUI is also started, switching to a text console requires pressing CTRL+ALT+FUNCTION (F7 or F1 is usually the GUI). you may need to run the startx command in order to start or restart the graphical desktop.

Generally, in a Linux desktop system, the X Window System is loaded as the final step in the boot process. A service called the display manager keeps track of the displays being provided and loads the X server. The X server provides graphical services to application sometimes called X clients. The display manager also handles graphical logins and starts the appropriate desktop environment after a user logs in. The desktop environment consists of a session manager and the window manager. The session manager starts and maintains that components of the graphical session. The window manager controls the placement and movement of windows, window title-bars, and controls. Although these can be mixed, usually a set of utilities, session manager, and windows manager are provided together as a unit. If the display manager is not started by default in the default runlevel, you can start X a different way by running startx from the command line.

SECTION 3: Linux DISTRIBUION INSTALLATION

There are many factors that you should think about when choosing a distribution. Choosing a distro requires some planning. Some questions to think about before deciding on a distro: * What is the main function of your system (server or desktop)? * What types of packages are important to your organization e.g. web server, word processing, etc. * How much hard disk space is available (when installing Linux on an embedded device, there will be space limitations) * How often are packages updated * How long is the support cycle for each release i.e. LTS releases have long-term suppor * Do you need kernel customization from the vendor * What hardware are you running the Linux distribution on e.g. x86, ARM, PPC, etc. * Do you need long-term stability or short-term experimental software

A partition layout needs to be decided at the time of installation . Linux systems handle partitions by mounting them at specific points in the filesystem. You can modify the design later. It is easier to try to get it right to begin with. Nearly all installers provide a reasonable filesystem layout by default. They either allocate all space dedicated to normal files on one big partition and a smaller swap partition. Or they separate partitions for some space-sensitive areas like `/home` and `/var`. You can override the defaults and do something different if you have special needs. You can also use more than one disk. All installations include the bare minimum software for running a Linux distribution. Most installers also provide options for adding categories of software. Common applications (e.g. firefox and libreoffice), developer tools (e.g. vi and emacs), and other popular services (e.g. apache and MySQL) are usually included. A desktop environment is also installed by default.

All installers secure the system being installed as part of the installation. This consists of setting the password for the superuser (root) and setting up an initial user. In some cases (e.g. Ubuntu) only an initial user is set up. In this situation, the root user is disabled by default and requires logging in as a normal user and then using sudo. Some distributions will also install more advanced security frameworks such as SELinux or AppArmor. Like other operating systems, Linux distributions can be installed through usb, cd/dvd, and other methods. Most Linux distributions support booting a small image and then downloading the rest of the system over a network. These small images are usable on media or as network boot images, making it possible to install without any local media at all. Many installers can do an installation completely automatically using a configuration file to specify installation options. This file is called a kickstart file for fedora-based systems, an AutoYAST file for SUSE-based systems, and a preseed file for Debian-based systems.

Each distro provides its own documentation and tools for creating and managing these files. The actual install process is very similar for all distributions. After booting from the install media, the installer starts and asks for config settings. These questions can be skipped if an automatic installation file is provided. Then the installation is performed. Finally, the computer reboots to the newly-installed system. Some distros will ask additional questions after the computer reboots. Oftentimes, you can also install updates as part of the installation process. This requires internet access. If this isn't an option, then the system uses the normal update mechanism to retrieve those updates later.

SUMMARY

- A partition is a logical part of the disk.
- A filesystem is a method of storing/finding files on a hard disk.
- Dividing the hard disk into partitions allows data to be grouped and separated as needed.
- When a failure or mistake occurs, only the data in the affected partition will be damaged.
- The data on the other partitions will likely survive.
- The boot process has multiple steps.
- First the BIOS triggers the boot loader, which the starts the Linux kernel, the initramfs filesystem, and the init. program.
- Determining the right distribution requires that you match your requirements with the capabilities of the various distros. # Chapter 4: Graphical Interface

INTRODUCTION

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

SECTION 1: SESSION MANAGEMENT

You can use either a command line interface (CLI) or a graphical user interface (GUI) when using Linux. To work in the CLI, you have to remember which programs and commands are used to perform tasks. You must also remember how to quickly and accurately obtain information about their use and options. 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 more efficient. The GUI is easier to navigate if you don't remember all the details or only do something rarely. In this section, we will learn about managing sessions using the GUI for: * CentOS (fedora-based) * openSUSE (SUSE-based) * Ubuntu (Debian-based)

openSUSE uses KDE instead of GNOME as the default desktop manager. GNOME is a popular desktop environment with an easy to use GUI. It is bundled as the default desktop environment for many distros including RHEL, Fedora, CentOS, SUSE, and Debian. GNOME has menu-based navigation and is sometimes an easy translation for Windows suers. The look and feel can be very different among distro, even if they are all using GNOME. Another common desktop environment is KDE which the the default in openSUSE. Other alternatives include UNITY, Xfce, and LXDE. Most desktop environments follow a similar structure to GNOME.

When you install a desktop environment, the X display manager starts at the end of the boot process. The 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 into the system. The default display manager for GNOME is gdm. Other popular display managers include lightdm (Ubuntu) and kdm (associated with KDE). It is a good idea to lock your screen whenever you step away from your computer to stop them from accessing your computer. This does not suspend your computer: all the applications and processes are still running. There are two ways to lock your screen: * Using the graphical interface * Using the keyboard shortcut `CTRL+ALT+L`

This keyboard shortcut can be changed using the following methods: * CentOS: System->Preferences->Keyboard Shortcuts * openSUSE: Configure Desktop->Shortcuts and Gestures * Ubuntu: System Settings->Keyboard->Shortcuts

Linux is a true multiuser operating system which allows more than one user to be simultaneous logged in. If more than one person uses the system, it is best for each person to have their own user account and password. This allows for individualized settings, home directories, and other files. Users can be logged in simultaneously through the network. Besides normal daily starting and stopping of the computer, a system restart may be required as part of a major system update. These usually are only necessary when 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. Run level 6 is used to reboot the system.

Shutdown, reboot, and logout operations will ask for confirmation before completing. 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. 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. However, suspend mode uses very little power. It works by keeping your system's applications, desktop, and so on in the system RAM. It turns of all other hardware. The suspend mode bypasses the time for a full system start-up and continues to use minimal power.

SECTION 2: BASIC OPERATIONS

Linux allows you to quickly open applications using the GUI. Applications are found in different places in Linux: * CentOS: the Applications menu in the upper-left corner of the screen * openSUSE: the Activities menu in the upper-left corner of the screen * Ubuntu: the Dash button in the upper-left corner of the screen * KDE: the button in the lower-left corner of the screen

There are also several submenus that exist in Linux for different types of applications including: * Accessories * Games * Graphics * Internet * Office * Sound and Video * System Tools

Unlike other OSes, the initial Linux install usually comes with a wide range of applications and software archives. These can contain thousands of programs that enable you to accomplish a wide variety of tasks with your computer. For most key tasks, the default application is usually already installed. You can always install more applications and try different options. e.g. Firefox is popular as the default browser in many distros while Epiphany, Konqueror and Chromium are usually available. Proprietary web browser such as opera and chrome are also available. Locating applications from the GNOME and KDE menu8s is easy as the are organized into submenus. Multiple applications are available to accomplish various tasks and to open a file of a given type. e.g. you can click on a web address when 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. Every user with an account on the system will have a home directory usually created under `/home` e.g. /home/student. By default, saves files will be placed here. Account creation, whether during system installation or at a later time, also induces default directories to be created under the home directory. e.g. Documents, Desktop, and Downloads.

Nautilus (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 forms click view drop-down and select your view. You can also use `CTRL+1`, `CTRL+2`, and `CTRL+3` respectively. In addition, you also arrange the files by name, size, type, or modification date for further sorting. To do so, click view and select arrange items. Another useful option is to show hidden files. These 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 drag and drop. You can alter the size of the icons by selecting zoom in or zoom out under the view menu. Nautilus includes a great search tool inside the file browser. 1. Click search in the toolbar (to bring up a text box). 2. Enter the keyword in the textbox (Nautilus will perform a recursive search from the current directory for any file or directory that contains part of this keyword). 3. To open Nautilus from the command line, simply type nautilus. 4. To open Nautilus in graphical mode, press `ALT+F2` and search for Nautlius.

This 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. 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

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 associated 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. Deleting a file in Nautilus will automatically move the deleted files to .local/share/Trash/files/ directory. This is under the user's HOME directory. There are several ways to delete files using Nautilus: * Select all the files and directories you want to delete * Press `DELETE` (Unity/KDE) or `CTRL+DELETE` (GNOME) on your keyboard, or right click the file * Select move to trash or highlight the file * Click edit and move to trash through the GUI

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, you can select the file or directory you want to delete and press SHIFT+DELETE. As a precaution, you should never delete your HOME directory. Doing so will most likely erase all your GNOME configuration files and possibly prevent you from logging in. Many personal system and configuration files are stored under your `HOME` directory.

SECTION 3: GRAPHICAL DESKTOP

Each Linux distro 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 instead. In addition, you can also change the desktop theme. The desktop theme changes the look and feel of the Linux system. The theme also defines the appearance of application windows. If you do not like any of the installed wallpapers, you can use different shades of color as the background. To do this, use the colors and gradients drop-down in the appearance window. There are three types of color: * solid * horizontal gradient * 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. The visual appearance of applications (the buttons, scroll bars, widgets, and other graphical compnents) are set by the theme. GNOME has several different themes that it comes with by default. The method for changing your theme will depend on your linux distro. e.g. Ubuntu: right-click anywhere on the desktop->select a different theme from the theme drop down. e.g. CentOS: System->Preferences->Apppearance. There are other options that you can use to change the them if you download additional packages. You can download and install themes from http://art.gnome.org/themes/. In openSUSE to change themes you will need to install the gnome-weak-tool program.

SUMMARY

- GNOME is a popular desktop environment and graphical user interface that runs on top of the Linux OS.
- The default display manger for GNOME is called gdm.
- The gdm display manager presents the user with the login screen which prompts for a username and password.
- Logging out kills all processes in your current X session and returns to the display manager login screen. # Chapter 5: System Configuration from the Graphical Interface

INTRODUCTION

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 the GUI

SECTION 1: SYSTEM, DISPLAY, DATE AND TIME SETTINGS

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

Accessing System Settings various from distro to distro: * CentOS: System->Preferences * openSUSE: Activities->type Settings into the Search box * Ubuntu: Located in the panel on the left of the screen

The display panel under System Settings (or Display and Monitor panel under Configure Desktop) contains the most common settings for desktop appearance. These settings function independently of the specific display drivers that 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. This configuration program will not be included in System Settings. This program many give more configuration options, but may be more complicated and might require 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 distros, this file is usually present only in unusual circumstances such as when less common graphic drivers are in use. Changing this configuration file directly is usually for the more advanced user.

The system can usually figure out the best resolution for a given screen automatically. However, in some cases, it might get this wrong, or you might want to change the resolution to fit your needs. This can be accomplished through the Displays panel. The new resolution will come into effect when you click Apply and then confirm that the new resolution is working. If the selected resolution didn't work or if you aren't happy with how it looks, the system will switch back to the original resolution after a short time. In most cases, the configuration for multiple displays is set up automatically as one big screen spanning all monitors. This will attempt to find a reasonable guess for the screen layout. If the screen layout is not as desired, then you can change it from this screen.

Linux will always use Coordinated Universal Time (UTC) for its own internal time-keeping. Display 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. You can also right-click Date and Time on the top panel to access the Date and Time Settings window. The Network Time Protocol (NTP) is the most popular and reliable protocol for settings the local time via internet servers. Most Linux distros include a working NTP setup which refers to specific time servers run by the distros. This means no setup beyond on or off is required for network time synchronization. If you want, more detailed configuration is possible by editing the standard NTP config file (/etc/ntp.conf).

SECTION 2: NETWORK MANAGER

All Linux distros have network config files, but file formats and locations can differ from distro to distro. Hand editing of these files can handle complicated setups, but it is not very dynamic or easy to learn and use. The Network Manager utility was developed to make things easier and more uniform across distros. It can: * list all available networks (wired and wireless) * Allow the choice of a wired, wireless, or mobile broadband network * handle passwords * set up Virtual Private Networks (VPNs)

Except for unusual situations, it's generally bet to let the Network Manager establish connections and keep track of settings. In this section we will learn how to manage network connections, including wired and wireless connections, and mobile broadband and VPN connections. Wired connections usually do not require any manual configuration. The hardware and signal presence are automatically detected and the 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 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 ones you are connected to by using Network Manager. You can add, edit, or remove known wireless networks, and also specify which ones you want connected by default when present.. You can set up a mobile broadband connection through the Network Manager. This will launch a wizard to set up the connection details for each connection. Once the config is done, the network is configured automatically each time the broadband network is attached.

Network Manager can also manage VPN connections. It supports VPN technologies such as IPSec, Cisco OpenConnect, Microsoft PPTP, and OpenVPN.

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

SECTION 3: INSTALLING AND UPDATING SOFTWARE

Each package in a Linux distro provides one piece of the system e.g.: * Linux kernel * C compiler * shared software code for USB devices * Firefox web browser

Packages often depend on each other. Firefox communicated using SSL/TLS and depends on the packages for those encryption methods. It won't install unless those packages are also installed. One utility handles the low-level details of installing/unpackaging a package. Most of the time, you will be working with a high-level utility which downloads packages from the internet. This utility can manage dependencies and groups for you. Package Management in Debian is handled through dpkg. It can install, remove, and build packages. It doesn't automatically download and install packages to satisfy dependencies.

The higher-level package management system is the apt (Advanced Package Tool) system of utilities. Each distro within the Debian family will usually use apt. It creates its own user interface on top of it (e.g. apt-get, aptitude, synaptic, Ubuntu Software Center, Update Manager, etc.). Apt repositories are usually compatible with each other, but the software they contain generally isn't. Most apt repositories target a particular distro e.g. Ubuntu. Software often ships with multiple repos to support multiple distros. The Red Hat Package Manager (RPM) is also popular on Linux distros. It was developed by Red Hat and is currently used by distros like openSUSE, Madriva, and CentOS. The high-level package manger differs between distros. Most use the basic format used in yum (Yellowdog Updater, Modified). yum is the package manager used for Fedora and RHEL. they include changes to fit the features they support. The GNOME project has been developing PackageKit as a unified interface. This is now the default interface for Fedora. openSUSE uses YaST (Yet another System Tool). openSUSE used to use apper, but it has been replaced. The YaST Software Manager is similar to other graphical package managers. It is an RPM-based application. You can add, remove, and update packages with this application. 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 distros.

SUMMARY

- You can control basic config options and desktop settings through they System Settings panel.

- Linux 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 wireless or mobile broadband networks, handle passwords, and set up VPNs.
- dpkg and RPM are the most popular package management systems used on Linux distros.
- Debian distros use dpkg and apt-based utilities for package management.
- RPM was developed by Red Hat and adopted by a number of other distros including openSUSE, Mandriva, CentOS, Oracle Linux, and others. # Chapter 6: Finding Linux Documentation

INTRODUCTION

By the end of this chapter you should be able to: * Use different sources of documentation * Use the man pages * access the GNU info system * use the help command and the –help option * use other documentation sources

SECTION 1: DOCUMENTATION SOURCES

Regardless of whether you are an experienced or inexperienced Linux user, you won't always know how to use everything about Linux. Sometimes, you will need to consult the help documentation. Linux-based systems draw from a large variety of sources. There are numerous places for documentation and ways of getting help. Distributors consolidate this material and present it an a comprehensive and easy-to-use manner. Important Linux documentation sources include. * The man pages (short for manual pages) * GNU Info * The help command and –help option * Other documentation sources e.g. https://www.gentoo.org/doc/en/

SECTION 2: THE MAN PAGES

The man pages are the most often-used source of Linux documentation. They provide in-depth documentation about many programs and utilities. They also include documentation on other topics such as configuration files, system calls, library routines, and the kernel. Typing man with a topic name as an argument retrieves the information stored in that topic's manual page. This page will also explain the depth of coverage. The man pages structure was first introduced in the early UNIX version of the early 1970s. The man pages are often converted to: * Web pages * Published books * Graphical help * Other formats

The man program searches, formats, and displays the information associated with that man page. Many topics have a lot of information, sot he output is piped through a terminal pager program such as less to be viewed one page at a time. At the same time, this helps the information be formatted for good visual display. When no options are given, by default one only sees the dedicated page specifically about the topic. You can broaden this to view all man pages containing a string in their name using the `-f` option. You can also view all man pages that discuss a specified subject by using the `-k` option. * `$ man -f` generates the same result as typing `$ whatis`. * `$ man -k` generates the same result as typing `$ apropos`.

The man pages are divided into nine numbered chapter (1 through 9). Sometimes, a letter is appended to the chapter number to identify a specific topic. e.g. many pages describing part of the X Window API are in chapter 3X. The chapter number can be used to force man to display the page from a particular chapter. It is common to have multiple pages across multiple chapters with the same name. This is especially true for the names of library functions or system calls. With the `-a` parameter, man will display all pages with the given name in all chapters one after another. Some possible inputs for man include: * `$ man 3 printf` * `$ man -a printf`

SECTION 3: GNU INFO

Another source of Linux documentation is the GNU Info System. This is the GNU project's standard documentation format (info) which it prefers as an alternative to man. The info system is more free-form and supports linked sub-sections. Functionally, the GNU Info System resembles man in many ways. Topics are connected using links (even though its design predates the Web). Information can be viewed through the CLI, a graphical help utility, printed, or viewed online. Typing info with no arguments in a terminal window displays an index of available topics. You can browse through this topic list using the regular movement keys (e.g. arrows, page up, page down). You can alos look for a particular topic by using info . The system then searches for the topic in all available info files. Some useful keys are q to quit, h for help, and enter to select a menu item. The topic which you view the info page is call a node. Nodes are similar to sections and subsections in written documentation. you can move between nodes or view each node sequentially. Each node may contain menus and linked subtopics (items). Items can be compared to Internet hyperlinks. They are identified by an asterisk (`*`) at the beginning of the item name. Named items (outisde a menu) are identified with double-colons (::) at the end of the item name. Items can refer to other nodes within the file or to other files. Some basic keystrokes for moving between nodes: * `n`: go to the next node * `p`: go to the previous node * `u`: move one node up in the index

SECTION 4: HELP COMMAND

The third source of Linux documentation is the use of the help command. Most commands have an available short description which can be viewed using the `--help` or the `-h` option along with the command. e.g. to learn more about the man command, you can use man `--help .` The `--help` option is useful as a quick reference and displays information faster than the man or info pages. Some popular commands (e.g. `echo`) when run in a bash command shell silently run their own built-in versions of system programs or utilities. This is because it is more efficient to do so with these specific commands. To view a synopsis of these built-in commands, you can simply type help. For these built-in commands, help performs the same basic function as the `-h` and `--help` arguments perform for stand-alone programs.

SECTION 5: OTHER DOCUMENTATION SOURCES

In addition to the man pages, the GNU Info System, and the help command, there are other sources of Linux documentation: * Desktop help system * Package documentation * Online resources

All Linux desktop systems have a graphical help application. This application is usually displayed as a question-mark icon or an images of a ship's life-preserver. These programs will contain custom help for the desktop itself and some of its applications. They will often include graphically rendered info and man pages. You can also start the graphical help system from a graphical terminal using the following commands: * GNOME: gnome-help * KDE: khelpcenter

Linux documentation is also available as part of the package management system. Usually this documentation is pulled directly from the upstream source code, but it can also contain info on how the distro packaged and set up the software. Such information is placed under the /usr/share/doc directory in a subdirectory named after the package. This subdirectory might also include the version number. There are many places to access online Linux documentation and a little bit of searching is usually the best way to find it. There are also several very good sites that you can use to help find things easier. Some offer a free,downloadable command line compendium under a Creative Commons license. http://Linuxcommand.org/tlcl.php. You can also find helpful documentation for each distro. Each distro has its own user-generated forms and wiki sections: * Ubuntu: https://help.ubuntu.com/ * CentOS: https://www.centos.org/docs/ * OpenSUSE: http://en.opensuse.org/Portal:Documentation * GENTOO: http://www.gentoo.org/doc/en

You can also use online search sites to locate helpful resources from all over the internet including: * blog posts * forum and mailing list posts * news articles * etc.

SUMMARY

- The main sources of Linux documentation are the man pages, GNU Info, the help options and commands, and a variety of online documentation resources.
- The man utility searches, formats, and displays man pages.
- The man pages provide in-depth documentation about programs and other topics about the system including config files, system calls, library routines, and the kernel.
- The GNU Info System was created by the GNU project as its standard documentation.
- It is robust and is accessible via command line, web, and graphical tools using info.
- Short descriptions for commands are usually displayed with the `-h` or `--help` argument.
- You can type help at the command line to display a synopsis of built-in commands.
- There are many other help resources both on your system and on the internet. # Chapter 7: Command Line Operations

INTRODUCTION

By the end of this chapter you will be able to: * Use the command line to perform operations in Linux * Search for files * Create and manage files * Install and update software

SECTION 1: COMMAND LINE MODE OPTIONS

Linux sysadmins spend a lot of their time using the CLI. They often automate and troubleshoot tasks in this text environment. A GUI makes tasks easier, while a CLI makes difficult tasks possible. Linux relies heavily on the abundance of command line tools. The CLI provides: * No GUI overhead * Virtually every tasks can be accomplished using the command line * You can script tasks and a series of procedures * You can log on remotely to networked machines anywhere on the internet * You can initiate graphical apps direly from the command line

A terminal emulator program emulates a stand alone terminal within a window on the desktop. This means it behaves essentially as if you were logging into the machine at a pure text terminal with no running GUI. Most terminal emulator programs support multiple terminal sessions by opening additional tabs or windows. By default, GNOME uses the gnome-terminal application to emulate a text-mode terminal in a window. Other programs include: * xterm * rxvt * knosole * terminator

To open a terminal in CentOS: * On the desktop in the upper-left corner, click applications * From the system tools menu, select terminal

To open a terminal in openSUSE: * On the desktop in the upper-left corner, click Activities * From the left pane, click Show Applications * Scroll down and select the required terminal

To open a terminal in Ubuntu: * In the left panel, click the Ubuntu icon * Type Terminal in the Search box

You can always open up a terminal by right clicking anywhere on the desktop and selecting Open in Terminal if nautilus-open-terminal is installed. The customizable nature of Linux allows you to drop the X Window GUI or start it up after the system has been running. Certain distros 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 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 (VTs) are console sessions that use the entire display and keyboard outside of a GUI. Such terminals are considered virtual because only one terminal remains visible at a time even though there can be multiple running at once. A VT is not quite the same as a command line terminal window. you can have as many command line terminal windows visible at once as you want. One virtual terminal (usually number one or seven) is reserved for the GUI and text logins are enabled on the unused VTs. Ubuntu uses VT 7, but CentOS/RHEL and openSUSE use VT 1 for the GUI. An example of a situation where using VTs is helpful is when you run into problems with the graphical desktop.

You can switch to one of the VTs and troubleshoot. To switch between the VTs, press `CTRL+ALT+Fx` key for the VT. e.g. `CTRL+ALT+F6` for VT 6. 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 can be followed by one or more options (or switches) that modify what the command may do. Options usually start with one or two dashes (e.g. `-p` or `--print`). These differentiate them from arguments which represent what the command operates on. Plenty of commands have no options, nor arguments, or neither.

You can also type other things at the command line besides issuing commands, such as setting environment variables. Linux distros can start and stop the graphical desktop in various ways. For Debian-based systems, the Desktop Manager runs as a service which can simply be stopped. For RPM-based systems, the Desktop Manager is run directly by init when set to run level 5. Switching to a different run level stops the desktop. Use the `$ sudo service gdm stop` or `$ sudo service lightdm stop` commands to stop the GUI in Debian-based systems. Use the `$ sudo telinit 3` command on RPM-based systems.

All the demonstrations previously have a user configured with `sudo` capabilities to provide the user with 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 on 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 we will learn about later.

When running on Ubuntu, sudo is already set up for you during installation. If you are running something in the Fedora or openSUSE families, you will likely need to set up `sudo` yourself. If your system does not already have `sudo` set up and enabled, you need to do the following steps: 1. You will need to make modifications as the admin or super user, root * `sudo` is the prefered method of doing this, but we don't have it set up yet so we will use `su` instead * At the command line prompt, type `su` and press Enter * You will then be prompted for the root user password to enter that now * You will notice that nothing is printed so that others can't see the password on screen * You should end up with a different looking prompt, often ending with # e.g. `$ su Password: #` 2. Now you need to create a config file to enable your user account to use `sudo` * This file is usually located in the `/etc/sudoers.d/` directory with the name of the file as your username * e.g. if your name was student, you will create a config file for student using `$ echo student ALL=(ALL) ALL > /etc/sudoers.d/student` 3. Some Linux distros will complain if you don't also change permissions on the file by doing `$ chmod 440 /etc/sudoers.d/student`

Now `sudo` should be set up correctly. When using `sudo` , by default you will be prompted to give a password (your own user's password) at least the first time you do it within a set time. It is possible (but very insecure) to configure `sudo` to not require a password or change the time window in which the password is allowed. This way it doesn't have to be repeated for every `sudo` command.

SECTION 2: BASIC OPERATIONS

To log in and out a text terminal will prompt for a username (with the string login:) and the password. When typing your password, nothing is displayed to prevent others from seeing your password. After you are logged in, you can perform basic operations. Once your session is started, you can also connect and log in to remote systems using the SSH (Secure SHell) utility. e.g. typing `$ ssh username@remote-server.com` would connect you securely to the remote machine and give you a command line terminal window. This lets you use passwords or cryptographic keys to prove your identity. 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 require root access.

When administering a multiuser system, you have the option of notifying all users prior to shutdown. e.g. `$ sudo shutdown -h 10:00 Shutting down for scheduled maintenance`. This will prompt all users with a message "Shutting down for scheduled maintenance" at 10:00. Depending on the specifics of your distro's policy, programs and software packages can be installed in various directories.

In general, executable programs should live under `/bin`, `/usr/bin`, `/sbin`, `/usr/sbin`, or `/opt`. One way to locate programs is to use the which utility. e.g. to find out where the diff program resides on the filesystem use `$ 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. e.g. `$ whereis diff`. 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 $HOME`. Note that some Linux systems actually open new graphical terminals in `$HOME/Desktop`. The following commands are useful for directory navigation. * `$ pwd`: print working directory, displays the current working directory. * `$ cd ~` or `cd`: change your directory to you home directory (short-cut name is `~`). * `$ cd ..`: Changes your working directory to the parent directory * `$ cd -`: Change to the previous directory

There are two ways to identify paths. 1. 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 `/` 2. 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 ignore 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) * `~` (home directory)

e.g.: * Absolute pathname: `$ cd /usr/bin`. * Relative pathname: `$ cd ../../usr/bin`.

In this case, the absolute pathname method is less typing. Traversing up and down the filesystem 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 command can help in exploring the filesystem: * `cd /`: changes the current directory to the root directory (or the path you supply) * `ls`: lists the contents of the present working directory * `ls -a`: lists all files including hidden files and directories (those that start with a `.`) * `tree`: displays a tree view of the filesystem

`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. Suppose that file1 already exists. A hard link called `file2` is created with the command. `$ ln file1 file2`. Note that the two files now appear to exist. However, a close inspection of the file listing shows that this is not quite true. `$ ls -li file1 file2`. The `-i` option is used to to print out 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 only one file is created, but it has more than one name associated with it. This 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 or soft links are created with the `-s` option: * `$ ln -s file1 file4` * `$ ls -li file1 file4`

Notice that `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. In the previous example, if you remove `file1` or `file2`, the inode object (and the remaining file name) will remain. This might lead to suble errors later if your recreate a file of that name. If you edit one of the files, most editors will retain the link by default (inlcluding vi and gedit). However, it is possible to change this default behaviour.

The cd command remembers where you were last and lets you go back with `$ cd -`. For remembering more than just the last directory you visited, use pushd to change the directory instead of cd. This pushes your starting directory to a list. Using `popd` will then send you back to those directories walking in the reverse order. The list of directories can be displayed using the `dirs` command.

SECTION 3: SEARCHING FOR FILES

When commands are executed, by default there are three standard file streams (or descriptors) always open for use: * Standard input (standrd in or `stdin`) * Standard output (standard out or `stdout`) * Standard error (or `stderr`)

Usually, `stdin` is your keyboard and `stdout` and `stderr` are printed on your terminal. Often, `stderr` is redirected to an error logging file or from the output of the 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 file descriptors. File descriptors are representative of the standard file streams: * `stdin` is file descriptor 0 * `stdout` is file descriptor 1 * `stderr` is file descriptor 2

Typically, if other files are opened in addition to these three, they will start at file descriptor 3 and increase from there. Through the command shell, we can redirect the standard filestreams so that we can get input from a file or from another command. We can also write output and error sot files or send them as input for subsequent commands. e.g. if we have a program called do_something that reads from `stdin` and writes to `stdout` and `stderr`, we can change its input by using the `<` followed by the name of the file. `$ do_something < input-file`. If you want to send the output to a file, use `>`. e.g. `$ do_something > output-file`.

`stderr` is not the same as `stdout` and error messages will still be seen on the terminal in the above examples. If you want to redirect `stderr` to a separate file, you need to use `stderr`'s file descriptor number (2) followed by `>` and the name of the file. e.g. `$ 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`. e.g. `$ do_something > all-output-file 2>&1`. bash permits an easier syntax for the above. e.g. `$ do_something >& all-output-file`.

The Linux/UNIX philosophy is to have many simple and short programs (or commands) cooperate together to produce complex results. This is rather than having 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. e.g. `$ command1 command 2 command3`. This represents what we call a pipeline and allows Linux to combine the actions of several commands into one. This is efficient because command2 and command3 do not have to wait for the previous pipeline commands to finish before they start hacking at the data in their input streams.

On multiple CPU-core systems, the computing power is much better utilized and things get done quicker. There is no need to save the output to temporary files between the stages in the pipeline. This saves disk space and reduces reading a writing to the disk which can be the slowest part of getting something done. Being able to quickly find the files your are looking for will make your life much easier. You can search for files in any system directory. The locate utility program performs a search through a previously constructed database of files and directories on your system. It will match 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 only print the lines that contain one or more specified strings. e.g. `$ locate zip grep bin`. This lists all files and directories with both "zip" and "bin" in their name. Note the use of to pipe the commands together. Locate utilizes the database created by another program: updatedb. Most Linux systems run this automatically once a day. You can update it any time by running updatedb form the command line as root user. You can also search for a filename containing specific characters using wildcards: * `?` - Matches any single character * `*` - Matches any string of characters * `[set]` - Matches any character in the set of charter e.g. [adf] will match any 'a', 'd', or 'f' * `[!set]` - Matches any character not listed in the set of characters

To search for files using the `?` wildcard, replace each unknown character with `?`. e.g. if you only know the first to characters are ba of a three-letter filename you can use `$ ls ba?`. find is another often-used utility. It searches recursively down the filesystem tree from any particular directory (or set of directories) and locates files that match the specified conditions. The default pathname is always the present working directory. e.g. sysadmins sometimes scan for large core files (which contain diagnostic info after a program fails) that are several weeks old to remove them. e.g. It is also common to remove files in `/tmp` periodically. Many distros use automated scripts that run periodically to accomplish such house cleaning. 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 the file name) * -type (restricts the results to a specified type such as d for directory, l for symbolic link, f for regular file, etc.)

e.g. * Searching for files and directories named "gcc" `$ find /usr -name gcc` * Searching only for directories named "gcc" `$ find /usr -type d -name gcc` * Searching for only regular files named "test1" `$ find /usr -type f -name test1`

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 that purpose. e.g. To find and remove all files that end with `.swp` `$ find -name *.swp -exec rm ;'`. The {} is a place holder that will be filled with all the file names that result from the find expression and the preceding command will run on each one individually. You also have to end the command with either ';' (including the single quotes) or both are fine. You 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. Sometimes, you might want to find files based on their size, last use, creation, etc. To find based on time e.g. `$ find / -ctime 3`. c-time is when inode meta-data (i.e. file ownership, permissions, etc) last changed. It is often, but not necessarily, when the file was 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 the exact value +n, which is greater than the value, or -n less than. There are similar options of rminutes (-cmin, amin, mmin). e.g. finding based on sizes `$ find / -size 0`. Note that 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 number (n), +n, or -n. For more details, consult the find man page. e.g. to find files greater than `10MB` in size and run a command on those files `$ find / -size +10M -exec command {} \`.

SECTION 4: WORKING WITH FILES

Linux has many commands that help in: * viewing the contents of a file * creating a new file or empty file * changing the timestamp of a file * removing and renaming a file or directory

These command help you in managing your data and files and ensuring the correct data is available at the correct location. You can use the following utilities to help view files: * `cat` - use for viewing files that are not very long; it does not provide any scroll-back * `tac` - used to look at a file backwards, starting with the last line * `less` - used to view larger files because it is a paging program; it pauses at each screenfull of text and provides search and scrollback capabilities Use `/` to search for a patter in the forward direction and ? to search for a pattern in the backwards direction * `tail` - use to print the last 10 lines of a file by default you can also use -n for an n number of the last lines in a file e.g. -n 15 for the last 15 * `head` - the opposite of `tail`

`touch` is a command that can be 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. You can also create an empty file using touch e.g. `$ touch <filename>`. This is normally done to create an empty file as a placeholder for a later purpose. touch provides several options including. -t to set the date and time stamp of the file e.g. `$ touch -t 03201600 myfile`. This sets the file, myfile's timestamp to `4 p.m. March 20th (03 20 1600)`. `mkdir` is used to create a directory e.g.: * `$ mkdir sampdir` * `$ mkdir /usr/sampdir`

Removing a directory is 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`. File commands: * `mv` - rename a file/move a file * `rm` - remove a file * `rm -f` - forcefully remove a file * `rm -i` - interactively remove a file

If you aren't sure about the files you are removing, it is best to use `rm -i`. You can also use some of the same commands or similar ones on directories: * `mv` - rename a directory/move a directory * `rmdir` - remove and empty directory * `rm -rf` - forcefully remove a directory

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 fine in most cases. You can have custom information shown on the command line. e.g. some sysadmins require the user and the host name to show up on the command line (`student@quad32 $`). This can be useful if you are working multiple roles and want to always be reminded of who you are and what machine you are on. The prompt above could be implemented using `\u@\h \$`. Other options: * `$ echo $PS1` * `\$` * `$ PS1=\u@\h \$` * `coop@quad64 $ echo $PS1` * `\u@\h \$` * `coop@quad64 $`

SECTION 5: INSTALLING SOFTWARE

The core parts of a Linux distro 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. e.g. 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 Debain * Those which use RPM

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

Most of the time, users need to 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. It handles the details of finding and installing each dependency for you. Be careful because installing a single package could result in hundreds of dependent packages being installed. The Advanced Packaging Tool (dpt) is the underlying package management system in Debian-based systems. It forms the backend for graphical package managers (e.g. Ubuntu Software Center, synaptic), but its native user interface is the command line. These command line programs include apt-get and apt-cache.

Yellowdog Update Modified (yum) is the command-line package management utility for RPM Linux systems (e.g. Fedora). yum has both the command line and the GUI. zypper is a package management system for openSUSE also based on RPM. zypper allows you to manage repositories from the command line as well. zypper is fairly straightforward and closely resembles yum. Basic operations for RPM and dpkg based systems:

Operation RPM Deb - - - Install Package rpm -i foo.rpm dpkg –install foo.deb Install package, dependencies yum install foo apt-get install foo Remove package rpm -e foo.rpm dpkg –remove foo.deb Remove package, dependencies yum remove foo apt-get autoremove foo Update package rpm -U foo.rpm dpkg –install foo.deb Update package, dependencies yum update foo apt-get install foo Update entire system yum update apt-get dist-upgrade Show all installed packages rpm -qa/yum list installed dpkg –list Get information on package rpm -qil foo dpkg –listfiles foo Show packages named foo yum list "foo" apt-cache search foo Show all available packages yum list apt-cache dumpavail foo What package is file part of rpm -qf file dpkg –search file

SUMMARY

- Virtual terminals (VT) in Linux are consoles or command line terminals that use the connected monitor and keyboard.
- Different Linux distros 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 what you typed.
- The preferred method to shut down or reboot the system is to use the shutdown command.
- The 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 of the file.
- 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 go 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 distros.
- The zypper package management system is based on RPM and used for openSUSE. # Chapter 8: File Operations

INTRODUCTION

Linux treats almost everything like a file. By the end of this chapter, you should be able to: * Explore the filesystem and its hierarchy * Explain the filesystem architecture * Compare files and identify different file types * Back up and compress data

SECTION 1: FILESYSTEMS

In Linux and all UNIX-like operating systems, it is often said that everything is a file or is treated as such. This means that whether you are dealing with: * Normal data files and documents * Devices such as sound cards and printers

You interact with them through the same kind of input/output (I/O) operations. This is one reason why text editors are so important. On many systems, the filesystem is structured like a tree. The tree is usually portrayed as inverted and starts at what is most often called the root directory (`/`). This marks the beginning of the hierarchical filesystem and is sometimes referred to as the trunk. The root directory is not the same as the root user. The hierarchical filesystem also contains other elements in the path (directory names) which are spearated by forward slashes. e.g. `/usr/bin/awk`. The last element is the actual file name.

The Filesystem Hierarchy Standard (FHS) grew out of historical standards from early versions of UNIX such as the Berkeley Software Distribution (BSD). The FHS provides Linux developers and sysadmins with a standard directory structure for the filesystem. This helps to provide consistency between systems and distributions. You can visit http://www.pathname.com/fhs/ for a list of the main directories and their contents in Linux systems. Linux supports various filesystem type created for Linux along with compatible filesystems from other operating systems such as Windows and MacOS. Legacy filesystems like FAT are also supported. Some examples of common filesystems in Linux are: * `ext3`, `ext4`, `brtfs`, `xfs` (native Linux filesystems) * `vfat`, `ntfs`, `hfs` (filesystems from other OSes)

Each filesystem resides on a hard disk partition. Partitions help to organize the contents of disks according to the kind of data contained. e.g. important program required to run the system are often kept on a separate partition known as root or `/`. e.g. the one that contains files owned by regular users is kept separate on /home. Temporary files created and destroyed during the normal operation of Linux are often located on a separate partition. This way, using all the available space on a particular partition may not fatally affect the normal operation of the system. Before you can start using a filesystem, you need to mount it to the filesystem tree at a mount point. This is a directory (which can be empty) where the filesystem is attached or mounted. Sometimes you may need to create the directory if it doesn't already exist. Note: if you mount a filesystem on a non-empty directory, the form contents of that directory are covered-up and not accessible until the filesystem is unmounted. Because of this, mount points are usually empty directories. The mount command is used to attacha filesystem. This can be local to the computer or on a network. Arguments include the device node and the mount point. e.g. `$ mount /dev/sda5 /home`. e.g. This will attach the filesystem contained in the disk partition associated with the `/dev/sda5` device node into the filesystem tree at the `/home` mount point. Unless the system is otherwise configured, only the root user has permissions to run mount.

If you want it to be automatically available every time the system starts up, you need to edit the file /etc/fstab accordingly (Filesystem Table). Looking at this file will show you the configuration for all pre-configured filesystems. `man fstab` will display how the file is used and how to configure it. Typing mount without any arguments will show all presently mounted filesystems. The command `$ df -Th` (disk-free) will display information about mounted filesystems including usage statistics about currently used and available space. Using NFS (the Network FileSystem) is one of the methods used for sharing data across physical systems.

Many sysadmins mount remote users' home directories on a server in order to give them access to the same files and config files across multiple client systems. This allows the users to log in to different computers and still have access to the same files and resources. On the server machine, NFS daemons (built-in networking and service processes in Linux) and other system servers are typically started with `$ sudo service nfs start`. The text file `/etc/exports` contains the directories and permissions that a host is willing to share with other systems over NFS. An entry in this file might look like `/projects *.example.com(rw)`. This entry allows the directory /projects to be mounted using NFS with read and write (rw) permissions and shared with other hosts in the example.com domain. Every file in Linux has 3 possible permissions: * read (r) * write (w) * execute (x)

After modifying the `/etc/exports file`, you can use the `$ exportfs -av` command. This notifies Linux about the directories you are allowing to be remotely mounted using NFS. You can also restart NFS with `$ sudo service nfs restart`. On a client machine, if it is desired to have the remote filesystem automatically mounted upon boot, the `/etc/fstab` file is modified to accomplish this. e.g. an entry in `/etc/fstab` might look like the following. e.g. `servername:/projects /mnt/nfs/projects nfs defaults 0 0`. You can also mount the remote filesystem without a reboot or as a one-time mount by directly using the mount command. `$ mount servernmae:/projects /mnt/nfs/projects`. If `/etc/fstab` is not modified, this remote mount will not be present the next time the system is restarted.

Certain files like the one mounted at `/proc` are called pseudo filesystems because they have no permanent presence anywhere on disk. The `/proc` filesystem contains virtual files (files that exist only in memory). This permits viewing constantly varying kernel data. This filesystem contains files and directories that mimic kernel structures and configuration information. It doesn't contain any real files, but it does contain runtime information (e.g. system memory, devices mounted, hardware configuration, etc.). Some important files in `/proc` are: * `/proc/cpuinfo` * `/proc/nterrupts` * `/proc/meminfo` * `/proc/mounts` * `/proc/partitions` * `/proc/version`

`/proc` has subdirectories as well including: * `/proc/<Process-ID-#>` *

`/proc/sys`

The first example shows there is a directory for every process running on the system which contains vital information about it. The second example shows a virtual directory that contains a lot of information about the entire system. In particular, it contains information on hardware and its configuration. The `/proc` filesystem is useful because the information it has is gathered only as needed and never stored on disk.

SECTION 2: FILESYSTEM ARCHITECTURE

Each user has a home directory usually under `/home`. The `/root` directory on modern Linux systems is the root user's home directory. The `/home` directory is often mounted in a separate filesystem on its own partition. It is often even exported remotely on a network through NFS. Sometimes you may group users based on their department or function. You can create subdirectories in `/home` for this. e.g. a school may organize `/home` into `/home/faculty/` `/home/staff/` and `/home/students/`. The `/bin` directory contains all executable binaries, essential commands used in single-user mode, and essential commands required by the system including: `ps` - produces a list of precesses along with stats information for the system `ls` - produces a listing of the contents of a directory `cp` - used to copy files

To view a list of programs in `/bin` use `$ ls /bin`. Commands that are not essential for the system in single-user mode are placed in the `/usr/bin` directory. The `/sbin` directory is used for essential binaries related to system administration such as `ifconfig` and `shutdown`. There is also a `/usr/sbin` directory for less essential sysadmin programs. Sometimes `/usr` is a separate fileystem that may not be available/mounted in single-user mode. This was why essential commands were separated from non-essential commands. In some of the most modern Linux systems, this distinction is considered obsolete and the `/usr/bin` and `/bin` directories are linked together as `/usr/sbin` and `/sbin`. The `/dev` directory contains device nodes, a type of pseudo-file used by most hardware and software devices, except for network devices. This directory is: * Empty on the disk partition when it is not mounted * Contains entries which are created by the udev system * The udev system creates and manages device nodes on Linux * It creates them dynamically when devices are found * The `/dev` directory contains items such as: + `/dev/sda1` (the first partition on the first hard disk) + `/dev/lp1` (second printer) + `/dev/dvd1` (first DVD drive)

The `/var` directory contains files that are expected to change in size and content as the system is running (var stands for variable). There are entries like: * System log files: `/var/log` * Package and databse files: `/var/lib` * Print queues: `/var/spool` * Temp files: `/var/tmp`

`/var` may be put it its own filesystem so that growth of the files can be accommodated and the file sizes don't fatally affect the system. Network service directories such as `/var/ftp` (the FTP service) and `/var/www` (the HTTP service) are also found under `/var`. The `/etc` directory is home for system config files. It has no binary programs, although there are some executable scripts. e.g. the file `reslv.conf` tells the system where to go on the network to obtain host name to IP address mappings (DNS). Files like `passwd`, `shadow`, and `group` for managing user accounts are found in the `/etc` directory. System run level scripts are found in subdirectories of `/etc`. e.g. `/etc/rc2.d` contains links to scripts for entering and leaving run level 2. The `rc` directory historically stood for run commands. Some distros extend the contents of `/etc`. e.g. Red Hat adds the sysconfig subdirectory that contains more config files. The `/boot` directory contains the essential files for booting the system. For every alternate kernel installed on the system, there are four files: * `vmlinuz`: the compressed Linux kernel; required for booting * `initramfs`: the initial ram filesystem; required for booting; sometimes called initrd, not initramfs * `config`: the kernel configuration file; only used for debugging and bookkeeping * `System.map`: kernel symbol table; only used for debugging

Each of these files has a kernel version appended to its name. The Grand Unified Bootloader (GRUB) files (such as `/boot/grub/grub.conf` or `/boot/grub2/grub2.cfg`) are also found under `/boot`. `/lib` contains libraries (common code shared by applications and needed for them to run) for the programs in `/bin` and `/sbin`. These library filenames start with `ld` or `lib`. e.g. `/lib/libncurses.so.5.7`… Most of these are known as dynamically loaded libraries (also called shared libraries or Shared Objects (SOs)). On some Linux distros, there exists a `/lib64` directory containing 64-bit libraries while `/lib` contains 32-bit versions. Kernel modules are located in `/lib/modules/<kernel-version-number>`. Kernel modules are kernel code, often device drivers, that can be loaded and unloaded without restarting the system. The `/media` directory is typically located where removable media such as CDs, DVDs, and USB drives are mounted. Unless configuration prohibits it, Linux automatically mounts the removable media in the `/media` directory when they are detected. The following is a list of additional directories and their use: * `/opt` optional application software packages * `/sys` virtual pseudo-filesystem giving info about the system and the hardware can be used to later system parameters and for debugging purposes * `/srv` site-specific data served up by the system; seldom used * `/tmp` temporary files; on some distros erased across a reboot and/or may actually be a ramdisk in memory * `/usr` multi-user applications, utilities, and data

The /usr directory contains non-essential programs in scripts (in the sense that they aren't needed for boot) and has the following subdirectories: * `/usr/include` header files used to compile applications * `/usr/lib` libraries for programs in /usr/bin and /usr/sbin * `/usr/lib64` 64-bit libraries for 64-bit programs in /usr/bin and /usr/sbin * `/usr/sbin` non-essential system binaries such as system daemons * `/usr/share` shared data used by applications, generally architecture-independent * `/usr/src` source code, usually for the Linux kernel * `/usr/` xz `11R6` X Window configuration files; generally obsolete * `/usr/local` data and programs specific to the local machine including subdirectories bin, sbin, lib, share, include, etc * `/usr/bin` this is the primary directory of executable commands on the system

SECTION 3: COMPARING FILES AND FILE TYPES

---

diff is used to compare files and directories. This utility has many options: * `-c` provides a listing of differences that include 3 lines of context before and after the lines differing in context * `-r` used to recursively compare subdirectories as well as the current directory * `-i` ignore the case of letters * `-w` ignore differences in spaces and tabs (white space)

To compare two files use `$ diff <filename1> <filename2>` You can compare 3 files at once with `diff3` which uses one file as a basis for the other two

e.g. two people made modifications to a file at the same time diff three shows the difference with `$ diff3 MY-FILE COMMON-FILE YOUR-FILE`

Many modifications to source code and config files are distributed utilizing patches which are applied with the patch program A patch file contains the deltas required to update the older version of a file to the new one Patch files are actually produced by running diff with the correct options `$ diff -Nur originalfile newfile > patchfile` Distributing just the patch is more concise and efficient than distributing the entire file e.g. if only one line needs to change in a file that contains 1000 lines, the patch will just be a few lines long To apply a patch, you can use either of the following commands: * `$ patch -p1 < patchfile` * `$ patch originalfile patchfile`

The first usage is more common as it is often used to apply changes to an entire directory tree rather than just one file as in the second example You can see the man page for patch to see how the other options work In Linux, a file's extension often doesn't categorize it the way it might in other OSes You can't assume that file named `file.txt` is a text file and not an executable In Linux, a file name is generally more meaningful to the user of the system than to the system itself Most applications directly examine a file's contents to see what kind of object it is rather than relying on an extension This is different from Windows in which .exe represents an executable binary file The real nature of a file can be ascertained using the file utility You can examine the contents and certain characteristics to determine if the files are: * plain text * shared libraries * executable programs * scripts * etc.

SECTION 4: BACKING UP AND COMPRESSING DATA

There are many ways you can back up data or your entire system. You can do a simple copy with cp, or use a more robust system with rsync. Both can be used to synchronize entire directory trees. Types of copies include: * `rsync` is more efficient because it checks if the file being copied already exists and there is no change in size or modification time. * `rsync` will avoid necessary copy and save time. * `rsync` copies only the parts of file that have actually changed. * `cp` can only copy files to and from destinations on the local machine unless you are copying to or from a filesystem mounted using NFS. * `rsync` can also be used to copy files from one machine to another.

Locations are designated in `target:path` form where the target can be in the form of `[user@]host`. The `user@` part is optional and used if the remote user is different from the local user. `rsync` is very efficient when recursively copying one directory tree to another because only the differences are transmitted over the network. You can synchronize the destination directory tree with the origin using the `-r` option to recursively walk down the directory tree. It will copy all files and directories below the one listed as the source. rsync is a very powerful utility. e.g. you can back up a project with `$ rsync -r project-X archive-machine:archives/project-X`.

Note that `rsync` can be very destructive. Accidental misuse can do a lot of harm to data and programs by inadvertently copying changes to where they are not wanted. Take care to specify the correct options and paths. It is highly recommended that you first test `rsync` commands with a dry-run option to ensure that it provides the results you want. To use rsync at the command prompt use `$ rsync sourcefile destinationfile` where either file can be on the local machine or a networked machine. File data is often compressed to save disk space and reduce the time it takes to transmit files over networks. Linux uses a number of methods to perform compression including: * `gzip`: the most frequently used Linux compression utility * `bzip2`: Produces files significantly smaller than those produced by `gzip` * `xz`: The mostspace efficient compression utility used in Linux * zip: Is often required to examine and decompress archives from other operating systems

These techniques vary in the efficiency of the compression (how much space is saved) and in how long they take to compress. gnereally the more efficient techniques take longer. Decompression time doesn't vary as much across different networks. In addition the tar utility is often used to group files in an archive and then compress the whole archive at once. `gzip` is the most commonly used Linux compression utility. It compresses very well and fast. The following table provides some usage examples: * `$ gzip *` compresses all files in the current directory; each file is compressed and renamed with a `.gz` extension * `$ gzip -r projectX` compresses all files in the `projectX` directory along with all files of the directory under `projectX` * `$ gunzip foo` De-compresses foo found in the file foo-gz; under the hood, gunzip command is actually the same as `gzip -d`

`bzip2` has syntax that is similar to `gzip`, but it uses a different compression algorithm and produces significantly smaller files. However, `bzip2` takes more time to work than `gzip`. It is more likely used to compress larger files: * `$ gzip2 *` compresses all files in the current directory and replaces each file with a file renamed with a `.bz2` extension * `$ bunzip2 *.bz2` decompress all of the files with extension .bz2 in the current directory; under the hood, `bunzip2` is the same as calling `bzip2 -d`

`xz` is the most space efficient compression utility in Linux. It is used by www.kernel.org to store archives of the Linux kernel. It trades slower compression for a higher compression ratio: * `$ xz *` compress all files in the current directory and replace each one with a .xz extension * `$ xz foo` compress the file foo into `foo.xz` using the default compression level (-6) and remove foo if the compression succeeds * `$ xz -dk bar.xz` decompress `bar.xz` into bar and don't remove `bar.xz` even if decompression is successful * `$ xz -dcf a.txt b.txt.xz > abcd.txt` decompress a mix of compressed and uncompressed files to standard output using a single command * `$ xz -d *.xz` decompress the files using `xz`

Compressed files are stored with a .xz extension. The zip program is not often used to compress in Linux, but is often required to decompress archives from other OSes. It is only used in Linux when you get a zipped file from a Windows user. It is a legacy program: * `$ zip backup *` compress all files in the current directory and place them in the file `backup.zip` * `$ zip -r backup.zip ~` archives your login directory (~) and all files and directives under it in the file `backup.zip` * `$ unzip backup.zip` extracts all files in the file backup.zip and places them in the current directory

Historically, tar stood for "tape archive" and was used to archive files to a magnetic tape. It allows you to create or extract files from an archive file often called a tarball. You can optionally compress while creating the archive and decompress when extracting the contents: * `$ tar xvf mydir.tar` extract all the files in mydir.tar into the mydir directory * `$ tar zcvf mydir.tar.gz mydir` create the archive and compress with `gzip` * `$ tar jcvf mydir.tar.bz2 mydir` create the archive and compress with bz2 * `$ tar Jcvf mydir.tar. xz mydir` create and compress with `xz` * `$ tar xvf mydir.tar.gz` extract all the files in mydir.tar.gz into the mydir directory; don't have to tell tar it is in `gzip` format

You can separate out the archiving and compression stages as in: * `$ tar mydir.tar mydir ; gzip mydir.tar` * `$ gunzip mydir.tar.gz ; tar xvf mydir.tar` This is slower and wastes space by creating an unneeded intermediary .tar file. The `dd` program is very useful for making copies of raw disk space. e.g. to back up your MBR (Master Boot Record the first 512 byte sector on the disk that contains a table of partitions) you can type: `$ dd if=/dev/sda` `of=sda.mbr bs=512 count=1` To use `dd` to make a copy of one disk onto another (DELETES everything on the second disk). An exact copy of the first disk device is created on the second disk device. `$ dd if=/dev/sda of=/dev/sdb`. Do not experiment with this command as it can erase a hard disk. Exactly what the name `dd` stands for is unknown. data definition is the most popular theory, others include disk destroyer and delete data.

SUMMARY

- The filesystem tree starts at what is often called the root directory (`/`).
- The Filesystem Hierarchy Standard (FHS) provides Linux developers and sysadmins with a standard directory structure for the filesystem.
- Partitions help to segregate files according to usage, ownership, and type.
- Filesystems can be mounted anywhere on the main filesystem tree at a mounting point.
- Automatic filesystem mounting can be set up by editing `/etc/fstab`.
- Filesystems like /proc are called pseudo filesystem because they only exist in memory.
- NFS (Network FileSystem) is a useful method for sharing files and data through network systems.
- `/root` (slash-root) is the home directory for the root user.
- `/var` may be put in its own filesystem so that growth can be contained and not fatally affect the system.
- `/boot` contains the basic files needed to boot the system.
- patch is a very useful tool in Linux.
- Many modifications to source code and configuration files are distributed with patch files as they contain the deltas or changes.
- These deltas go from an old version of a file to the new version of a file.
- File extensions in Linux do not necessarily mean that a file is of a certain type.
- cp is used to copy files on the local machine while `rsync` can also be used to copy files from one machine to another as well as synchronize contents.
- `gzip`, `bzip2`, `xz`, and zip are used to compress files.

- tar allows you to create or extract files from an archive file, often called a tarball.
- You can optionally compress while creating the archive and decompress while extracting its contents.
- `dd` can be used to make large exact copies even of entire disk partitions efficiently. # Chapter 9: User Environment

INTRODUCTION

By the end of this chapter, you should be able to: * Use and configure user accounts and user groups * Use and set environment variables * Use the previous shell command history * Use keyboard shortcuts * Use and define aliases * Use and set file permissions and ownership

SECTION 1: ACCOUNTS

Linux is a multiuser operating system (more than one user can log in at the same time). To list the currently logged-on users, type `$ who`. To identify the current user type `$ whoami`. Use `$who` with the `-a` option to give more detailed information. Linux uses groups to organize users. Groups are collections of accounts that share permissions. Control of group membership is administered through the /etc/group file which shows a list of groups and their members. By default, every user belongs to the default/primary group. Whenever a user logs in, the group membership is set for their primary group and all the members who have the same level of access and privilege. Permissions on various files and directories can be modified at the group level. All Linux users are assigned a unique user ID (uid) which is an integer. They are also assigned one or more group IDs (gid) including a default one which is the same as the user ID. Fedora systems start uids at 500 while other distros start at 1000. The numbers associated with names through the files /etc/passwd and /etc/group. e.g. the first file might contain `george:x:1002:1002:George Metesky:/home/george:/bin/bash`. e.g. and the second `george:x:1002`. Groups are used to make a set of users who have common interests in terms of: * access rights * privileges * security considerations

Access rights to files (and devices) are granted on the basis of the user and the group that they belong to. Distros have straightforward graphical interfaces for creating and removing users and groups and manipulating group membership. It is often useful to do so from the command line or from within shell scripts. Only the root user can add and remove users and groups. Adding a new user is done with `useradd` and removing an existing user is done with `userdel`. An account for the new user turkey would be done with `$ sudo useradd turkey`. Note that in openSUSE, useradd is not in the normal user's PATH so the command would be `$ sudo /usr/sbin/useradd turkey`. this sets the default home directory to `/home/turkey` and populates it with some basic files. It also adds a line to `/etc/passwd` such as. `turkey:x:502:502::/home/turkey:/bin/bash`. This sets the default shell to `/bin/bash`. Removing a user account can be done with `$ userdel turkey`. This will leave the `/home/turkey` directory intact. This can be useful for temporary inactivation.

To remove the home directory also you can add the `-r` option to `userdel`. Typing id with no argument gives information about the current user. e.g. typing `$ id` will give `uid=500(george) gid=500(george) groups=106(fuse),500(george)`. If given the name of another user as an argument, id will report information about that other user. Adding a group can be done with `groupadd`. e.g. `$ sudo /usr/sbin/groupadd anewgroup`. Groups can be removed with `$ sudo /usr/sbin/groupdel anewgroup` Adding a user to an already existing group is done with `usermod`. e.g. 1. You would first look at what groups the user belongs to * `$ groups turkey` * `turkey : turkey` 2. Then add the new group * `$ sudo /usr/sbin/usermod -G anewgroup turkey` * `$ groups turkey` * `turkey : turkey anewgroup`

These utilities update `/etc/group` as necessary. `groupmod` can be used to change group properties such as the Group ID (gid) with the `-g` option or its name with the `-n` option. You can also remove users from groups. The `-G` option to `usermod` must give a complete list of groups. If you do: * `$ sudo /usr/sbin/usermod -G turkey turkey` * `$ groups turkey` * `turkey : turkey`

Only the turkey group will be left. The root account is very powerful and has full access to the system. Other operating systems call this the administrator or admin account. In Linux, it is often called the superuser account. You must be extremely cautious before granting full root access to a user, they rarely need it. External attacks often consist of tricks used to elevate to the root account. You can use the `sudo` feature to assign more limited privileges to user accounts: * on a temporary basis * only for a specific subset of commands

When elevating privileges, you can use the command `su` (switch or substitute user) to launch a new shell running as another user. To do this, you must type in that other users' password. Most often, the other user is root and the new shell allows the use of elevated privileges until exited. It is almost always bad practice to use `su` to become root (both for security and stability). Resulting errors can include deletion of vital files from the system and security breaches. Granting privileges using `sudo` is less dangerous and is preferred. By default, `sudo` must be enabled on a per-user basis. Some distros (such as Ubuntu) enable it by default for at least one main user, or give this as an installation option. To fully become root, one merely types `su` and is then prompted for the root password. To execute just one command with root privilege, type `sudo <command>`. When the command is complete, you will return to being a normal unprivileged user.

`sudo` config files are stored in `/etc/sudoers` file and the `/etc/sudoers.d/` directory. By default, the `sudoers.d` directory is empty. In Linux, the command shell program (generally bash) uses one or more startup files to configure the environment. Files in the `/etc` directory define global settings for all users. Initialization files in the user's home directory can include and/or override the global settings. The startup files can do anything the user would like to do in every command shell such as: * Customizing the user's prompt * Defining command-line shortcuts and aliases * Setting the default text editor * Setting the path for where to find executable programs

When you first login to Linux, /etc/profile is read and evaluated. After this, the following files are searched (if they exist) in the listed order: 1. `~/.bash_profile` 2. `~/.bash_login` 3. `~/.profile`

The Linux login shell evaluates whatever startup file it comes across first and ignores the rest. If it finds `~/.bash_profile`, it ignores `~/.bash_login` and `~/.profile`. Different distros may use different startup files. Every time you create a new shell, terminal window, etc. you do not perform a full system login. Only the `~/.bashrc` file is read and evaulated. Although this file is not read and evaluated with the login shell, most distros/users include the `~/.bashrc` file from within one of the three user-owned startup files. In Ubuntu, openSUSE, and CentOS distros, the user must make changes in the `~/.bash_profile` file to include the `~/.bashrc` file. The `.bash_profile` will have certain extra lines which collect the required customization parameters from `.bashrc`.

SECTION 2: ENVIRONMENT VARIABLES

Environment variables are named quantities that have specific values and are understood by the command shell (e.g. bash). Some of these are built-in to the system and others can be set by the users at the command line or through startup scripts. An environment variable is a character string that contains information used by an application. There are a number of ways to view the value of currently set environment variables. e.g. `set`, `env`, and `export`. Depending on the state of your system, `set` may print out many more lines than the other two methods.

- `$ set`
- `BASH=/bin/bash`
- `BASHOPTS=checkwinsize:cmdhist:Expand_aliases:extglbo:extquote:force_fignore`
- `BASH_ALIASES=()`

- `...`
- `$ env`
- `SSH_AGENT_PID=1892`
- `GPG_AGENT_INFO=/run/user/me/keyring-Ilf3vt/gpg:0:1`
- `TERM=xterm`
- `SHELL=/bin/bash`
- `...`
- `$ export`
- `declare -x COLORTERM=gnome-terminal`
- `declare -x COMPIZ_BIN_PATH=/usr/bin /`
- `declare -x COMPIZ_CONIFG_PROFILE=ubuntu`
- `...`

By default, environment variables are created within a script are only available to the current shell. Child processes will not have access to values that have been set or modified. Allowing child processes to see the value requires the uses of `export`.

Task Command - - Show the value of a specific variable `echo $SHELL` Export a new variable value `export VARIABLE=value` or `VARAIBLE=value; export VARIABLE` Add a variable permanently 1. Edit `~/.bashrc` and add the line `export VARIABLE=value` 2. Type `source ~/.bashrc` or just `. ~/.bashrc` or just start a new shell by typing `bash` The `HOME` variable is the variable that represents the home (or login) directory of the user. `cd` without arguments will change the current working directory to the value of `HOME`. Note that the tilde character (`~`) is often used as an abbreviation for `$HOME`. Thus `cd $HOME` and `cd ~` are equivalent statements.

Command Explanation - - `$ echo $HOME` Show the value of the `HOME` variable (which will return `/home/me`) `$ cd /bin` Change the directory to `/bin` `$ pwd` Prints the current working directory (which will return `/bin`) `$ cd` Changes the directory to the one set by `$HOME` `$ pwd` Calling `pwd` again will return `/home/me`

`PATH` is an order4ed list of directories (the path) which is scanned when a command is given to find the appropriate program or script to run. Each directory in the path is separated by colons (`:`). A null (empty) directory name (or `./`) indicates the current directory and any given time. * `:path1:path2` * `path1::path2`

In the example `:path1:path2`, there is a null directory before the first colon. In the second example, there is a null directory between `path1` and `path2` To prefix a private `bin` directory to your path: 1. `$ export PATH=$HOME/bin:$PATH` 2. `$ echo $PATH` 3. `/home/me/bin:/usr/local/bin:/usr/bin:/bin/usr`

Prompt Statement (`PS`) is used to customize your prompt string to display the information that you want. `PS1` is the primary prompt variable which controls what your command line prompt looks like. The following special characters can be included in `PS1`: * `\u` - User name * `\h` - Host name * `\w` - Current working directory * `\!` - History number of this command * `\d` - Date

They must be surrounded in single quotes when the are used as in the following example: * `$ echo $PS1` * `$` * `export PS1='\u@\h:\w$` * `me@example.com:~$ # new prompt` * `me@example.com:$`

To revert the changes: * `me@example.com:~$ export PS1='$ '` * `$`

Even better practice would be to save the old prompt first and then restore it as in: * `$ OLD_PS1=$PS1`

This allows you to change the prompt and then eventually change it back with: * `$ PS1=$OLD_PS1` * `$`

The environment variable `SHELL` points to the user's default command shell. The command shell handles whatever you type in a command window and usually bash. `SHELL` contains the full pathname to the shell: * `$ echo $SHELL` * `/bin/bash` * `$`

SECTION 3: RECALLING PREVIOUS COMMANDS

Bash will keep track of previously entered commands and statements in a history buffer. You can recall previously used commands simply by using the `up` and `down` arrow keys. To view the list of previously executed commands, you can type history at the command line. The list of commands is displayed with the most recent item appearing last in the list. This information is stored under `~/.bash_history`. Several associated environment variables can be used to get information about the history file. * `HISTFILE` stores the location of the history file. * `HISTFILESIZE` stores the maximum number of lines in the history file. * `HISTSIZE` stores the maximum number of lines in the history file for the current session.

Specific keys can perform various tasks.

Key Usage - - `Up`/`Down` arrow keys Browse through the list of commands previously executed `! !` (pronounced bang-bang) Execute the previous command `CTRL+R` Search previously used commands

If you want to recall a command in the history list, but don't want to press the arrow key, you can press `CTRL+R` to do a reverse intelligent search. As you start typing, the search goes back in reverse order to the first command that matches the letters you've typed. You can type more letters to match more specific commands. The following is an example of how you can use `CTRL+R` to search through the command history. * `$ ^R` # This all happens on 1 line * `(reverse-i-search)'s': sleep 1000` # searched for 's'; matched "sleep" * `$ sleep 1000` # Pressed enter to execute the searched command

The table below describes the syntax used to execute previously used commands:

Syntax Task - - `!` Start a histroy substitution `!$` Refer to the last argument in a line `!n` Refer to the nth command line `!string` Refer to the most recent command starting with string

All history substitutions start with `!`. In the line `$ ls -l /bin /etc /var`, `!$` refers to `/var` which is the last argument in the line. Below is another example: 1. `$ echo $SHELL` 2. `$ echo $HOME` 3. `$ echo $PS1` 4. `$ ls -a` 5. `$ ls -l /etc/ passwd` 6. `$ sleep 1000` 7. `$ history`

- `$ !1` # execute command #1 above
- `echo $SHELL`
- `/bin/bash`
- `$ !sl` # Execute the command beginning with "sl"
- `sleep 1000`

There are also keyboard shortcuts that you can use to preform different tasks quickly. The table below lists some of the keybord shortcuts and their uses:

Keyboard Shortcut Task - - `CTRL+L` Clears the screen `CTRL+D` Exits the current shell `CTRL+Z` Puts the current process into suspended background `CTRL+C` Kills the current process `CTRL+H` Works the same as backspace `CTRL+A` Goes to the beginning of the line `CTRL+W` Deletes the word before the cursor `CTRL+U` Deletes from the beginning of the line to the cursor position `CTRL+E` Goes the end of the line `TAB` Auto-completes files, directories, and binaries

SECTION 4: COMMAND ALIASES

You can create custom commands or modify the behaviour of existing commands by creating aliases. Most often, these aliases are placed in your ~/.bashrc file so they are available to any command shells you create. Typing alias with no arguments will list currently defined aliases. Note that there should not be any spaces on either side of the equal sign and the alias definition needs to be placed within either single or double quotes if it contains any spaces.

SECTION 5: FILE PERMISSIONS

In Linux and other UNIX-based systems, every file is associated with a user who is the owner. Every file is also associated with a group (a subset of all users) which has an interest in the file and certain rights or permissions: read, write, and execute. The following utility programs involve user and group ownership and permission setting:

Command Usage - - `chown` Change user ownership of a file or directory `chgrp` Change group ownership `chmod` Change the permissions on the file which can be done separately for owner, group, and the rest of the system (often named as other)

Files have three kinds of permission: * read (r) * write (w) * execute (x)

These permissions affect three groups of owners: * user/owner (u) * group (g) * others (o)

As a result, you have the following three groups of permissions * rwx: rwx: rwx * u: g: o

There are different ways to use chmod. To give the owner and others execute permission and remove the group write permission: * `$ ls -l test1` * `-rw-rw-r-- 1 coop coop 1601 Mar 9 15:04 test1` * `$ chmod uo+x, g-w test1` * `ls -l test1` * `-rwxr--r-x 1 coop coop 1601 Mar 9 15:04 test1`

`u` stands for the user (ownser), `o` stands for other (world), and `g` stands for group. This kind of syntax can be difficult to type and remember, so there is a shorthand which lets you set all the permissions in one step. This is done with a simple algorithm and a single digit specifies all three permission bits for each entity. This digit is the sum of: * 4 for read permission * 2 for write permission * 1 for execute permission

Using this scheme, 7 means read/write/execute, 6 means read/write, and 5 means read/execute. When you apply this the chmod command, you have to give three digits for each degree of freedom such as in: * `$ chmod 755 test1` * `$ ls -l test1` * `-rwxr-xr-x 1 coop coop 1601 Mar 9 15:04 test1`

We can also change file ownership using `chown`: * `$ ls -l` * `total 4` * `-rw-rw-r--. 1 bob bob 0 Mar 16 19:04 file-1` * `-rw-rw-r--. 1 bob bob 0 Mar 16 19:04 file-2` * `drwxrwxr-x. 2bob bob 4096 Mar 16 19:04 temp`

- `$ sudo chown root file-1`
- [sudo] password for bob:
- `$ ls -l`

- `total 4`
- `-rw-rw-r--.  1 root bob 0 Mar 16 19:04 file-1`
- `-rw-rw-r--.  1 bob bob 0 Mar 16 19:04 file-2`
- `drwxrwxr-x.  2 bob bob 4096 Mar 16 19:04 temp`

Wee can also change file ownership using `chgrp`: * `$ sudo shgrp bin file-2` * `$ ls -l` * `total 4` * `-rw-rw-r--.  1 root bob 0 Mar 16 19:04 file-1` * `-rw-rw-r--.  1 bob bin 0 Mar 16 19:04 file-2` * `drwxrwxr-x.  1 bob bob 4096 Mar 16 19:04 temp`

SUMMARY

- Linux is a multiuser system.
- To find the currently logged on users, you can use the `who` command.
- To find the current user ID, you can use the `whoami` command.
- The `root` account has full access to the system.
- It is never sensible to grant full root access to a user.
- You can assign root privileges to regular user acounts on a temporary basis using the `sudo` command.
- The shell program (bash) uses multiple startup files to create the user environment.
- Each startup file affects the interactive environment in a different way.
- `/etc/profile` provides the global settings startup file.
- Advantages of startup files include that they customize the user's prompt, set the user's terminal type, set the command-line shortcuts and aliases, and set the default text editor.
- An environment variable is a character string that contains data used by one or more applications.
- The built-in shell variables can be customized to suit your requirements.
- The `histroy` command recalls a list of previous commands which can be edited and recycled.

- In Linux, various keyboard shortcuts can be used at the command prompt instead of the long actual commands.
- You can customize commands by creating aliases.
- Adding an alias to `~/.bashrc` will make it available for other shells.
- File permission can be changed by typing `chmod permissions filename`.
- File ownership is changed by typing `chown owner filename`.
- File group ownerships is changed by typing `chgrp gropup filename`. # Chapter 10: Text Editors

INTRODUCTION

By the end of this chapter, you should be familiar with: * How to create and edit files using the available Linux text editors * nano, a simple text-based editor * gedit, a simple graphical editor * vi and emacs, two advanced editors with both text-based and graphical interfaces

SECTION 1: BASIC EDITORS NANO AND GEDIT

At some point, you will need to manually edit text files. You might be: * Composing an email off-line * Writing a script to be used for bash or other command interpreters * Altering a system or application configuration file * Developing source code for a programming language such as `C` of `Java`

Linux admins often sidestep text editors by using graphical utilities for modifying config files. However, this is much more laborious than using a text editor. Note that word processing applications such as Notepad or the applications that are part of office suites are not really basic text editors because they have a lot of extra formatting information. This formatting information will render sysadmin config files unusable for their intended purpose. Using true text editors is essential for Linux. There are many choices when it comes to using text editors for Linux including: * nano * gedit * vi * emacs

In this section, we will cover nano and gedit. Both of these editors are relatively simple and easy to learn. Sometimes, you might want to create a short file and don't want to bother invoking a text editor. in addition, doing so can be quite useful when used from within scripts, even when creating longer files. You'll find yourself using this method when you start on the later chapters that cover bash scripting.

If you want to create a file without using an editor, there are two standard ways to do so from the command line and fill it with content. The first is to use echo repeatedly: * `$ echo line one > myfile` * `$ echo line two >> myfile` * `$ echo line three >> myfile`

The single `>` will send the output of a command to a file. The double `>>` will append the output to an existing file. The second way is to use cat combined with redirection: * `$ cat << EOF > myfile` * `> line one` * `> line two` * `> line three` * `> EOF`

Both the above techniques will produce the same output file. They can be extremely useful when employed by scripts.

There are some text editors that are extremely easy to use and obvious in how they work. These don't take that much work to learn, but they aren't very robust. One particularly easy one to use is the text-terminal based editor nano. Just invoke nano by giving a file name as an argument. All the help you need is displayed at the bottom of the screen. As a graphical editor, gedit is part of the GNOME desktop system (kwrite is associated with KDE.) The gedit and kwrite editors are also very easy to use and extremely capable. They are also very configurable. They will look a lot like Notepad in Windows. Other variants such as kedit and kate are also supported by KDE.

nano is easy to use and requires very little effort to learn. To open a file in nano, type `$ nano <filename>` and press `ENTER`. If the file doesn't exist, it will be created. nano provides a two line "shortcut bar" at the bottom of the screen that lists the available commands. Some of these commands include: * `CTRL+G`: Display the help screen * `CTRL+O`: Write to a file * `CTRL+X`: Exit a file * `CTRL+R`: Insert contents from another file to the current buffer * `CTRL+C`: Cancels the previous commands

gedit is a simple to use graphical editor that can only be run within a graphical desktop environment. It is visually similar to Notepad in Windows, but is far more capable and very configurable. it also has a wealth of plugins available to further extend its capabilities. To open a new file in gedit, find the program in your desktop's menu system, or from the command line type `$ gedit <filename>`. If the file doesn't exist, it will be created. Using gedit is also very straightforward and doesn't require much training. Its interface is composed of very familiar elements.

SECTION 2: MORE ADVANCED EDITORS VI AND EMACS ____

Developers and admins experienced with UNIX-like systems almost always use one of two venerable editors: vi and emacs. Both are present or easily available on all distros and are have versions compatible with other OSes. Both vi and emacs have a purely text-based form that can run in a non-GUI environment. They also have one or more X-based graphical forms with extended capabilities. The graphical forms can be friendlier for the less experienced user. vi and emacs have a very steep learning curve, but they are extremely efficient once they are learned. Note: there are often fights among users over which editor is best that are often described as a holy war.

First we will cover vi. The actual program installed on your system usually isn't vi, it's vim which stands for vi improved. This is still referred to as vi. Even if you don't want to use vi, it is good to gain some familiarity with it. vi is a standard tool that is installed on virtually all Linux distros. There are many times when no other editor is available on the system. GNOME extends vi with a very graphical interface known as gvim. KDE offers kvim. Either of these may be easier to use at first. When using vi, all commands are entered through the keyboard. This is because you don't want to keep moving your hands to use a pointer devices such as a mouse or touchpad.

Typing vimtutor launches a short but very comprehensive tutorial for those that want to learn their first vi commands. The tutorial is a great place to start learning vi. Even though it only provides an introduction and seven lessons, it has enough material to make someone a proficient vi user because it covers a large number of commands. After learning the basic ones, you can look up new tricks to incorporate into your list of vi commands. There are always more optimal ways to do things in vi with less typing.

Vi provides three modes as described in the table below. it is vital not to lose track of which mode you are in. Many keystrokes and commands behave differently in different modes.

Mode Feature - - Command

By default, vi starts in command mode

Each key is an editor command

Keybaord strokes are interpreted as commands that can modify file contents

Insert

Type `i` to switch to insert mode from command mode

Insert mode is used to enter (insert) text into a file

Insert mode is indicated by an `? INSERT ?` indicator at the bottom of the screen

Press `ESC` to exit insert mode and return to command mode

Line

Type `:` to switch to the line mode from command mode

Each key is an external command, including operations such as writing the file contents to disk or exiting

vi uses line editing commands inherited from older line editors

Most of these commands are no longer used

Some line editing commands are very powerful

Press `ESC` to exit line mode and return to command mode

The table below describes the most important commands used to start, exit, read, and write files in vi. The `ENTER` key needs to be pressed after all of these commands.

Command Usage - - `vi myfile` Start the vi editor and edit the myfile file `vi -r myfile` Start vi and edit myfile in recovery mode from a system crash `:r file2` Read in file2 and insert at current position `:w` Write to file (save) `:w myfile` Write out the file to myfile `:w! file2` Overwrite file2 `:x` or `:wq` Exit vi and rite out modified file `:q` Quit vi `:q!` Quit vi even though modifications have not been saved

The table below describes the most important keystrokes used when changing cursor position in vi. Line mode commands (those following colon) require the `ENTER` key to be pressed after the command is typed.

Key Usage - - arrow keys To move up, down, left, and right `j` or `<ret>` To move one line down `k` To move one line up `h` or `BACKSPACE` To move one character left `l` or `SPACE` To move one character right `0` To move to the beginning of the line `$` To move to the end of the line `w` To move to the beginning of the next word `:0` or `1G` To move to the beginning of the file `:n` or `nG` To move to line n `:$` or `G` To move to the last line in the file `CTRL+f` or `PAGE DOWN` To move forward one page `CTRL+b` or `PAGE UP` To move backward one page `^l` To refresh and center the screen

The following table describes the most important commands when searching for text in vi. The `ENTER` key should be pressed after typing the search pattern.

Command Usage - - `/pattern` Search forward for pattern `?pattern` Search backward for pattern

The following table describes the most important keystrokes used when searched for text in vi

Key Usage - - `n` Move to the next occurrence of the search pattern `N` Move to the previous occurrence of the search pattern

The table below describes the most important keystrokes used when changing, adding, and deleting text vi.

Key Usage - - `a` Append text after cursor; stop upon `ESCAPE` key `A` Append text at the end of the current line; stop upon `ESCAPE` key `i` Insert text before the cursor: stop upon `ESCAPE` key `I` Insert text at the beginning of the current line; stop upon `ESCAPE` key `o` Start a new line below current text, insert text there; stop upon `ESCAPE` key `O` Start a new line above the current line; insert text there; stop upon `ESCAPE` key `r` Replace character at current position `R` Replace text starting with current position; stop upon `ESCAPE` key `x` Delete character at current position `Nx` Delete N characters, starting at current position `dw` Delete the word at the current position `D` Delete the rest of the current line `dd` Delete the current line `Ndd` or `dNd` Delete N lines `u` Undo the previous operation `yy` Yank (copy) the current line and put it in the buffer `Nyy` or `yNy` Yank N lines and put them in the buffer `p` Paste at the current position the yanked lines or lines from the buffer

Typing `:sh command` opens an external command shell. When you exit the shell, you will resume your vi editing session. Typing `:!` executes a command from within vi. The command follows the exclamation point. This technique is best suited for non-interactive commands such as `:! wc %` Typing this will run the `wc` (word count) command on the file; the character `%` represents the file currently being edited. The `fmt` command does simple formatting of text. If you are editing a file and want the fie to look nice, you can run the file through `fmt`. One way to to do this while editing is by using `: %!fmt,` which runs the entire file (the `%` part) through fmt and replaces the file with the results.

Now we will go over emacs. The emacs editor is a popular competitor for vi Unlike vi, it does not work with modes. emacs is highly customizable and includes a large number of features. It was initially designed for the console, but a GUI version was soon released too. emacs has many other capabilities other than text editing. It can be used for email, debugging and many other things. Rather than having different modes for command and insert, emacs uses the `CTRL` and meta (`ALT` or `ESC`) keys for special commands. The following table lists some of the most important key combinations that are used when starting, exiting, reading, and writing files in emacs.

Key Usage - - `emacs myfile` Start emacs and edit myfile `CTRLx i` Insert prompted for file at current position `CTRL+x s` Save all files `CTRL+x CTRL+w` Write to the file giving a new name when prompted `CTLR+x CTRL+s` Saves the current file `CTRL+x CTRL+c` Exit after being prompted to save any modified files

The emacs tutorial is a good place to start learning basic emacs commands. It is available any time when in emacs by simply typing `CTLR+h` (for help) and then the letter `t` for tutorial.

The following table lists some of the keys and key combinations that are used for changing cursor position in emacs.

Key Usage - - arrow keys Use the arrow keys for up, down, left, and right `CTRL+n` One line down `CTRL+p` One line up `CTRL+f` One character forward/right `CTRL+b` One character back/left `CTRL+a` Move to the beginning of the line `CTRL+e` Move to the end of the line `CTRL+f` Move to the beginning of the next word `CTRL+b` Move to the beginning of the preceding word `META+>` Move to the end of the file `CTRL+v` or `PAGE DOWN` Move forward one page `META+v` or `PAGE UP` Move backward one page `CTRL+l` Refresh and center screen

The following table lists key combinations that are used for searching text in emacs.

Key Usage - - `CTRL+s` Search forward for prompted pattern, or for next pattern `CTRL+r` Search backward for prompted pattern, or for next pattern

The following table lists some of the key combinations used for changing, adding, and deleting text in emacs.

Key Usage - - `CTRL+o` Insert a blank line `CTRL+d` Delete character at current position `CTRL+k` Delete the rest of the current line `CTRL+_` Undo the previous command `CTRL+SPACE` or `CTRL+@` Mark the beginning of the selected region. The end will be at the cursor position `CTRL+w` Delete the current marked text and write it to the buffer `CTRL+y` Insert at the current cursor location whatever was most recently deleted

SUMMARY

- Text editors (rather than word processing programs) are used often in Linux for tasks such as creating or modifying system config files, writing scripts, developing source code, etc.
- nano is an easy-to-use text-based editor that utilizes on-screen prompts.
- gedit is a graphical editor very similar to Notepad in Windows.
- The vi editor is available on all Linux systems and is very widely used.
- Graphical extension versions of vi are also widely available.
- emacs is available on all Linux systems as well as a popular alternative to vi.
- emacs can support both a GUI and a text mode interface.
- To access the vi tutorial, type vimtutor at the command line window.
- To access the emacs tutorial, type `CTRL+h` and then `t` from within emacs.
- vi has three modes: Command, Insert, and Line; emacs has only one, but requires use of special keys such as `CTRL` and `ESC`.
- Both editors use various combinations of keystrokes t accomplish tasks.
- The learning curve to master these can be long, but once mastered, using either editor is extremely efficient. # Chapter 11: Local Security Principles

INTRODUCTION

By the end of this chapter, you should: * Have a good grasp of best practices and tools for making Linux systems as secure as possible * Understand the powers and dangers of using the root (superuser) account * Know how to use thee `sudo` command to perform privileged operations while restricting enhanced powers as much as feasible * Be able to explain the importance of process isolation and hardware access * Know how to work with passwords, including how to set and change them * Describe how to secure the boot process and hardware resources

SECTION 1: UNDERSTANDING LINUX SECURITY

The Linux kernel allows properly authenticated users to access files and applications. Each user is identified by a unique integer (the user id or UID.) A separate database associates a username with each UID. When an account is created, new user information is added to the user database and the user's home directory is created and populated with some essential files. You can use command line programs like `useradd` and `userdel` along with GUI tools to create and remove accounts. For each user, the following table lists the seven fields that are maintained in the /etc/passwd file.

Field Name Details Remarks - - - Username User login name Should be between 1 and 32 characters long Password User password (or the character 'x' if the password is stored in the /etc/shadow file) in encrypted format Is never show in Linux when it is being typed User ID (UID) Every user must have a user id (UID)

UID 0 is reserved for the root user

UIDs ranging from 1-99 are reserved for other predefined accounts

UIDs ranging from 100-999 are reserved for system accounts and groups (except for RHEL which reserves only up to 499)

Normal users have UIDs of 1000 or greater, except on RHEL where they start at 500

Gropu ID (GID) The primary Group ID (GID) is stored in the /etc/group file Will be covered ein further detail in the chapter on processes User Info This field is optional and allows insertion of extra information about the user such as their name e.g. Rufus T. Firefly Home Directory The absolute path location of the user's home directory e.g. /home/rtfirefly Shell The absolute location of the user's default shell e.g. /bin/bash

By default, Linux distinguishes between several account types to isolate processes and workloads. Linux has four types of accounts: * root * System * Normal * Network

For a safe working environment, it is advised to grant the minimum privileges possible and necessary to the accounts and remove inactive accounts. The last utility, which shows the last time each user has logged into the system, can be used to help identify potentially inactive accounts should be considered for removal. Keep in mind that you should be more strict on multi-user systems than you are on personal desktop systems that only affect the casual user. The practices in this chapter are meant for use on enterprise servers that you can use on all systems, but they can be relaxed on personal systems.

root is the most privileged account on a Linux/UNIX system. This account has the ability to carry out all facets of system administration, including adding accounts, changing user passwords, examining log files, installing software, etc. Utmost care must be taken when using this account; it has no security restrictions on it. When you are signed in as or acting as root, the shell prompt displays `#` (If you are using bash and haven't customized the prompt.) This convention is intended to serve as a warning of the absolute power of this account.

SECTION 2: UNDERSTANDING THE USAGE OF THE ROOT ACCOUNT

You must have root privileges in order to perform operations such as: * Creating, removing, and managing users accounts * Managing software packages * Removing or modifying system files * Restarting system services

Regular account users of Linux distros may be allowed to install software packages, update some settings, and apply various changes to the system. However, root privilege is required for performing some administration tasks such as restarting services, manually installing packages, and managing parts of the filesystem that are outside of the normal user's directories.

To create a new user account in Linux: 1. At the command prompt, as root, type `$ useradd <username>` 2. To set the inital password, type `$ passwd <username>`: the new password prompt is displayed 3. You must enter and confirm the new password 4. The message `passwd: all authentication tokens updated succesfully` is displayed

A regular account user can perform some operations requiring special permissions. The system configuration must allow such abilities to be exercised. SUID (Set owner User ID upon execution - similar to the Windows "run as" feature) is a special kind of file permission given to a file. SUID provides temporary permissions to a user to run a program with the permissions of the file owner instead of the permissions held by the user. The table below provides examples of operations that don't require root:

Operations that do not require Root privilege Examples of this operation - - Running a network client Sharing a file over the network Using devices such as printers Printing over the network Operations on files that the user has proper permissions to access Accessing files that you have access to or sharing data over the network Running SUID-root applications Executing programs such as `passwd`

SECTION 3: USING SUDO; THE IMPORTANCE OF PROCESS ISOLATION LIMITING HARDWARE ACCESS AND KEEPING SYSTEMS CURRENT

In Linux, you can use either `su` or `sudo` to temporarily grant root access to a normal user. The two methods are very different. The table below illustrates some of the differences between the two.

`su` `sudo` - - When elevating privilege, you need to enter the root password. Giving the root password to a normal user should never be done. When elevating privilege, you need to enter the user's password and not the root password. Once a user elevates to the root account using `su`, the user can do anything that the root user can do for as long as the user wants without being asked again for a password. Offers more features and is considered more secure and more configurable. Exactly what the user is allowed to do can be precisely controlled and limited. By default, the user will either always have to keep giving their password to do further operations with `sudo`, or can avoid doing so for a configurable time interval. The command has limited logging features. The command has detailed logging features.

`sudo` has the ability to keep track of unsuccessful attempts at gaining root access. User's authorization for using `sudo` is based on configuration information stored in the `/etc/sudoers` file and in the `/etc/sudoers.d` directory. A messsage such as the following would appear in a system log file (usually `/var/log/secure`) when trying to execute `sudo bash` without successfully authenticating the user:

- `authntication failure; logname=op uid=0 euid=0 tty=/dev/pts/6 ruser=op rhost= user=op`
- `conversation failed`
- `auth could not identify password fo [op]`
- `op : 1 incorrect password attempt` ;
- `TTY=pts/6 ; PWD=/var/log ; USER=root ; COMMAND=/bin/bash`

Whenever `sudo` is invoked, a trigger will look at `/etc/sudoers` and the files in `/etc/sudoers.d` to determine if the user has the right to use `sudo` and what the scope of their privilege is. Unknown user requests and requests to do operations not allowed to the user even with `sudo` are reported. You can edit the `sudoers` file by using visudo. This ensures that only one person is editing the file at a time, has the proper permissions, and refuses to write out the file and exit if there is an error in the changes made. The basic structure of an entry is `who where = (as_whom) what`. The file has a lot of documentation in it about how to customize. Most Linux distros now prefer that you add a file in the directory `/etc/sudoers.d` with the same name as the user. This file contains the individual user's `sudo` configuration and one should leave the master configuration file untouched except for changes that affect all users. By default, `sudo` commands and any failures are logged in `/var/log/auth.log` under the Debian distro family and in `/var/log/messages` or `/var/log/secure` on other systems. This is an important safeguard to allow for tracking the accountability of `sudo` use. A typical entry of the message contains: * Calling username * Terminal info * Working directory * User account invoked * Command with arguments

Running a command such as `$ sudo whoami` results ina log file entry such as `Dec 8 14:20:47 server1 sudo:  op :  TTYP=pts/6 PWD=/var/log USER=root COMMAND=/usr/bin/whoami` Linux is considered to be more secure than many other operating systems because processes are naturally isolated from each other. One process cannot normally access the resources of another process, even when that process is running with the same user privileges. Additional security mechanisms that have been recently introduced in order to make risks even smaller are: * Control Gropus (cgroups): Allows sysadmins to group processes and associate finite resources to each group * Linux Containers (LXC): Makes it possible to run multiple isolated Linux systems (containers) on a single system by relying on cgroups * Virtualization: Hardware is emulated in such a way that not can processes be isolated, but entire systems can be run simultaneously as isolated and insulated guests (virtual machines) on one physical host

Linux limits user access to non-networking hardware devices in a manner that is extremely siilar to regular file access. Applications interact by engagin nthe filesystem layer (which is independent of the actual device or hadrware the file resides on.) This layer will then open a device special file (often called a device node) under the `dev` directory that corresponds to the device being accessed. Each device special file has standard owner, group, and world permission fields. Security is naturally enforced just as it is when standard files are accessed. Hard disks, for example, are represented as `/dev/sd*`. While a root user can read and write to the disk in a raw fashion by doing something like `$ echo hello world > /dev/sda1`, the standard permissions as show in the figure make it impossible for regular users to do so. Writing to a devices in this fashion can easily obliterate the filesystem stored on it in a way that cannot be repaired without great effort, if at all. The normal reading an writing of files on the hard disk by applications is done at a higher level through the filesystem and never through direct aces to the device node.

When security problems in either the Linux kernel or applicaions and libraries are discovered, Linux distributions have a good record of reacting quickly and pushing out fixes to all systems by updating their software repositories and sending notifications to update immediately. The same thing is true with bug fixes and performance improvements that are not security related. However, it is well known that many systems do not get update frequently enough and problems which have already been cured are allowed to remain on computers for a long time. This is particularly true with proprietary operating systems where users are either uninformed or distrustful of the vendor's patching policy as sometimes updates can cause new problems and break existing operations. Many of the most successful attack vectors come from exploiting security holes for which fixes are already known but not universally deployed. The best practice is to take advantage of your Linux distro's mechanism for automatic updates and never postpone them. It is extremely rare that such an update will cause new problems.

SECTION 4: WORKING WITH PASSWORDS

Linux verifies authenticity and identity using user credentials. Originally, encrypted passwords were stored in the `/etc/passwd` file, which was readable by everyone. This made it easy for passwords to be cracked. On modern systems, passwords are actually stored in an encrypted format in a secondary file name `/etc/shadow`. Only those with root access can modify/read this file. Protecting passwords has become a crucial element of security. Most Linux distros rely on a modern password algorithm called SHA-512 (Secure Hashing Algorithm 512 bits) developed by the US National Security Agency (NSA) to encrypt passwords. The SHA-512 algorithm is widely used for security applications and protocols. These security applications and protocols include TLS, SSL, PHP, SSH, S/MIME, and IPSec. SHA-512 is one of the most tested hashing algorithms. For example, (if you want to experement with SHA-512 encoding) the word "test" can be encoded using the program sha512sum to produce the SHA-512 form.

IT professionals follow several good practices for securing the data and password of every user. 1. Password aging is a method to ensure that users get prompts that remind them to create a new password after a specific period. This can ensure that passwords, if cracked, will only be usable for a limited amount of time. This feature is implemented using `chage` which configures the password expiry information for a user. 2. Another method is to force users to set strong passwords using Pluggable Authentication Modules (PAM). PAM can be configured to automatically verify that a password created or modified using the passwd utility is sufficiently strong. PAM configuration is implemented using a library called pam_cracklib.so, which can be replaced by pam_passwdqcso for more options. 3. One can also install password cracking programs such as John The Ripper to secure the password file and detect weak password entries. It is recommended that written authorization be obtained before installing such tools on any system that you do not own.

SECTION 5: SECURING THE BOOT PROCESS AND HARDWARE RESOURCES

You can secure the boot process with a secure password to prevent someone from bypassing the authentication step. For systems using the GRUB boot loader, for the older GRUB version 1, you can invoke grub-md5-crypt which will prompt you for a password and then encrypt as shown on the adjoining screen. You then must edit `/boot/grub/grub.conf` by adding the following line below the timeout entry: `password --md5 $1$Wnvo.1$qz781HRVG4jUnJXmdSCZ30`. You can also force passwords for only certain boot choices rather than all. For the now more common GRUB version 2, things are more complicated, but you have more flexibility and can do things like use user-specific passwords which can be their normal login password. Also, you never edit the configuration file `/boot/grub/grub.cfg` directly, rather you edit your system configuration files in `/etc/grub.d /` and thne run update-grub. One explanation can be found at https://help.ubuntu.com/community/Grub2/Passwords.

When hardware is physically accessible, security can be compromised by: * Key Logging: Recording the real time activity of a computer user including the keys they press. The captured data can either be stored locally or transmitted to remote machines. * Network sniffing: Capturing and viewing the network packet level dat on your network. * Booting with a live or rescue disk. * Remounting and modifying content.

Your IT security policy should start with requirements on how to properly secure physical access to server and workstations. Physical access to a system makes it possible for attackers to easily leverage several attack vector, in a way that makes all operating system level recommendations irrelevant. The guidelines of security are: * Lock down workstations and servers. * Protect your network links such that it cannot be accessed by people you don't trust. * Protect your keyboards where passwords are entered to ensure that keyboards can't be tampered with. * Ensure a password protects the BIOS in such a way that the system cannot be booted with a live or rescue DVD or USB key.

For single user computer and those in a home environment, some of the above features (like preventing booting from removable media) can be excessive, and you can avoid implementing them. However, if sensitive information is on your system that requires careful protection, either it shouldn't be there, or it should be better protected by following the above guidelines. Like all software, hackers occasionally find weaknesses in the Linux ecosystem. The strength of Linux (and the open source community in general) is the speed with which such vulnerabilities are exposed and remedied.

SUMMARY

- The root account has authority over the entire system.
- root privileges may be required for tasks, such as restarting services, manually installing packages, and managing parts of the filesystem that are outside your home directory.
- In order to perform any privileged operations such as system-wide changes, you need to use either `su` or `sudo`.
- Calls to `sudo` trigger a lookup in the `/etc/sudoers` file or in the `/etc/sudoers.d` directory which first validates that the calling user is allowed to use `sudo` and that it is being used within the permitted scope.
- One of the most powerful features of `sudo` is the ability to log unsuccessful attempts at gaining root access. By default, `sudo`commands and failures are logged in `/var/log/auth.log` under the Debian family and `/var/log/messages` in other distros.
- One process cannot access another process' resources, even when that process is running the the same user privileges.
- Using the user credentials, the system verifies the authenticity and identity.
- The SHA-512 algorithm is typically used to encode passwords. They can be encrypted, but not decrypted.
- Pluggable Authentication Modules (PAM) can be configured to automatically verify that passwords created or modified using the passwd utility are strong enough (what is considered strong enough can also be configured.)
- Your IT security policy should start with requirements on how to properly secure physical access to servers and workstations.
- Keeping your systems updated is an important step in avoiding security attacks. # Chapter 12: Network Operations

Introduction

By the end of this chapter you should be able to: * Explain many basic networking concepts including types of networks and addressing issues. * Know how to configure network interfaces and use basic networking utilites such as `ifconfig`, `ip`, `ping`, `route`, and `traceroute`. * Use graphical and non-graphical browsers such as Lynx, w3m, Firefox, Chrome, and Epiphany. * Transfer files to and from clients and servers using both graphical and text mode applications such as Filezilla, ftp, sftp, curl, and wget.

SECTION 1: INTRODUCTION TO NETWORKING

A network is a group of computers and computing devices connected together through communication channels such as cables or wireless media. The computers connected over a network may be located in the same geographical area or spread across the world. A network is used to: * Allow the connected devices to communicate with each other. * Enable multiple users to share devices over the network such as printers and scanners. * Share and manage information across computers easily.

Most organizations have both an internal network and an Internet connection for users to communicate with machines and people outside the organization. The Internet is the largest network in the world and is often called "the network of networks".

Devices attached to a network must have at least one unique network address identifier know as the IP (Internet Protocl) address. The address is essential for routing packets of information through the network. Exchanging information across the network requires using streams of bite-size packets, each of which contains a piece of the information going from one machine to another. These packets contain data buffers together with headers which contain information about where the packet is going to and coming from, and where it fits in the sequence of packets that constitute the stream. Networking protocols and software are rather complicated due to the diversity of machines and operating system they must deal with, as well as the fact that even very old standards must be supported.

There are two different types of IP addresses available: IPv4 (version 4) and IPv6 (version 6). IPv4 is older and by far the more widely used, while IPv6 is newer and is designed to get past the limitations of the older standard and furnish many more possible addresses. IPv4 uses 32-bits for addresses; there are only 4.3 billion unique addresses available. Furthermore, many addresses are allotted and reserved, but not actually used. IPv4 is becoming inadequate because the number of devices available on the global network has significantly increased over the past years. IPv6 uses 128-bits for addresses; this allows for 3.4x10ˆ38 unique addresses. If you have a larger network of computers and want to add more, you may want to move to IPv6, because it provides more unique addresses. However, it is difficult to move to IPv6 as the two protocols do not inter-operate. Due to this, migrating equipment and addresses to IPv6 requires significant effort and hasn't been as fast as was originally intended.

A 32-bit IPv4 address is divided into four 8-bit sections called octets. Example: IP address 172.16.31.46 - - Bit format 10101100.00010000.00011111.00101110

Network addresses are divided into five classes: A, B, C, D, and E. Classes A, B, and C are classified into two parts: Network Addresses (Net ID) and Host addresses (Host ID). The Net ID is used to identify the network, while the host ID is used to identify a host in the network. Class D is used for special multicast applications (information is broadcast to multiple computers simultaneously) and Class E is reserved for future use. in this section, we will learn about classes A, B, and C.

Class A addresses use the first octet of an IP address as their Net ID and use the other three octets as the Host ID. The first bit of the first octet is always set to zero. Thus you can only use 7-bits for unique network numbers. As a result, there are a maximum of 126 Class A networks available (the addresses 00000000 and 11111111 are reserved.) Not surprisingly, this was only feasible when there were very few unique networks with large numbers of hosts. As the use of the Internet expanded, Classes B and C were added in order to accommodate the growing demand for independent networks. Each Class A network can have up to 16.7 million unique hosts on its network. The range of host addresses is from 1.0.0.0 to 127.255.255.255 The value of an octet, or 8-bits, can range from 0 to 255.

Class B addresses use the first two octets of the IP address as their Net ID and the last two octets as the host ID. The first two bits of the first octet are always set to binary 10, so there are a maximum of 16,384 (14-bits) Class B networks. The first octet of a Class B address has values form 128 to 191. The introduction of Class B networks expanded the number of networks, but it soon became clear that a further level would be needed. Each Class B network can support a maximum of 65,536 unique hosts on its network. The range of host addresses is from 128.0.0.0 to 191.255.255.255

Class C addresses use the first three octets of the IP address as their Net ID and the last octet as their Host ID. The first three bits of the first octet are set to binary 110, so almost 2.1 million (21-bits) Class C networks are available. The first octet of Class C addresses has values from 192 to 223. These are most common for smaller networks which don't have many unique hosts. Each Class C network can support up to 256 (8-bits) unique hosts. The range of host addresses is from 192.0.0.0 to 223.255.255.255.

Typically, a range of IP addresses are requested from your Internet Service Provider (ISP) by your organization's network administrator. Often, your choice of which class of IP address you are given depends on the size of your network and expected growth needs. You can assign IP addresses to computers over a network manually or dynamically. When you assign IP addresses manually, you add static (never changing) addresses to the network. When you assign IP addresses dynamically (they can change every time you reboot or even more often), the Dynamic Host Configuration Protocol (DHCP) is used to assign IP addresses.

Before an IP address can be allocated manually, one must identify the size of the network by determining the host range; this determines which network class (A, B, or C) can be used. The ipcalc program can b used to ascertain the host range. Note that the version of ipcalc supplied in the Fedora family of distros doesn't behave as described below. It is an entirely different program. Assume that you have a Class C network. The first three octets of the IP address are 192.168.0. As it uses 3 octets (i.e. 24 bits) for the netmask, the shorthand for this type of address is 192.168.0.0/24. To determine the hst range of the addresses you can use for this new host, at the command prompt, type: `$ ipcalc 192.168.0.0/24`. From this result, you check the HostMin and HostMax values to manually assign a static address available from 1 to 254 (192.168.0.1 to 192.168.0.254).

Name resolution is used to convert numerical IP address values into a human-readable format known as the hostname. For example, 140.211.169.4 is the numerical IP address that refers to the linuxfoundation.org hostname. Hostanmes are easier to remember the an IP address. Given an IP address, you can obtain its corresponding hostname as well. Accessing the machine over the network becomes easier when you type the hostname instead of the IP address. You can view your system's hostname by simply typing hostname with no argument. Note that if you give an argument, the system will try to change its hostname to match it. However, only root users can do that. The special hostname localhost is associated with the IP address of 127.0.0.1 and describes the machine you are currently on (which normally has additional network-related IP addresses.)

Network interfaces are a connection channel between a device and a network. Physically, network interfaces can proceed through a network interface card (NIC) or can be more abstractly implemented as software. You can have multiple network interfaces operating at once. Specific interfaces can be brought up (activated) or brought down (de-activated) at any time. A list of currently active network interfaces is reported by the ifconfig utility which you may have to run as the superuser or at least give a full path i.e. `/sbin/ifconfig` on some distros.

Network config files are essential to ensure that interfaces function correctly. For Debian family configuration, the basic network config ile is `/etc/network/interfaces`. You can type `$ /etc/init.d/networking start` to start the networking configuration. For Fedora family system configuration, the routing and host information is contained in `/etc/sysconfig/network`. The network interface config script is located at

`/etc/sysconfig/network-scripts/ifcfg-eth0`. For SUSE family system configuration, the routing and host information and network interface config scripts are contained in the `/etc/sysconfig/network` directory. You can type `$ /etc/init.d/network start` to start the networking configuration for Fedora and SUSE families.

To view the IP address, type `$ /sbin/ip addr show`. To view the routing information type `$ /sbin/ip route show`. ip is a very powerful program that can do many things. Older (and more specific) utilities such as ifconfig and route are often used to accomplish similar tasks. A look at the relevant man pages can tell you much more about these utilities.

ping is used to check whether or not a machine attached to the network can receive and send data. In other words, it confirms that the remote host is online and is responding. To check he status of a remote host at the command prompt, type `$ ping <hostname>`. ping is frequently used for network testing and management. However, it's usage can increase network load unacceptably. Hence, you can abort the execution of ping by typing `CTRL+C` or using the `-c` option which limits the number of packets that ping will sent before it quites. When execution stops, a summary is displayed.

A network requires the connection of many nodes. Data moves from source to destination by passing through a series of routers and potentially across multiple networks. Servers maintain routing tables containing the addresses for each node in the network. The IP Routing Protocols enable routers to build up a forwarding table that correlates final destination with the next hop address. route is used to view or change the IP routing table. You may want to change the IP routing table to add, delete, or modify static routes to specific hosts or networks. The table below explains some commands that can be used to manage IP routing:

Task Command - - Show current routing table `$ route -n` Add static route `$ route add -net address` Delete static route `$ route del -net address`

traceroute is used to inspect the route which the data packet takes to reach the destination host which makes it quite useful for troubleshooting network delays and errors. By using traceroute, you can isolate connectivity issues between hops which helps resolve them faster. To print the route taken by the packet to reach the host network, at the terminal, type `$ traceroute <domain>`.

There are several other helpful networking tools. Networking tools are very useful for monitoring and debugging network problems such as network connectivity and network traffic.

Networking Tool Description - - ethtool Queries network interfaces and can also set various parameters such as the speed. netstat Displays all active connections and routing tables. Useful for monitoring performance and troubleshooting. nmap Scans open ports on a network. Important for security analysis. tcpdump Dumps network traffic for analysis. iptraf Monitors network traffic in text mode.

SECTION 2: BROWSERS

Browsers are used to retrieve, transmit, and explore information resources, usually on the World Wide Web. Linux users commonly use both graphical and non-graphical browser applications. The common graphical browsers used in Linux are: * Firefox * Google Chrome * Chromium * Epiphany * Opera

Sometimes, you either do not have a graphical environment to work in (or have reasons not too use it) but still need to access web resources. in such a case, you can still use non-graphical browsers such as the following:

Non-Graphical Browser Description lynx Configurable text-based web browser; the earliest such browser and still in use. links or elinks Based on lynx. It can display tables and frames. w3m Newer text-based web browser with many features.

Sometimes you need to download files and information but a browser is not the best choice, either because you want to download multiple files and/or directories, or you want to perform the action from a command line or a script. wget is a command line utility that can capably handle the following types of downloads: * Large file downloads * Recursive downloads where a web page refers to other web pages and all are downloaded at once * Password required downloads * Multiple file downloads

To download a webpage, you can simply type `$ wget <urL>` and then you can read the downloaded page as a local file using a graphical or non-graphical browser. Besides downloading you may want to obtain information about a URL, such as the source code being used. curl can be used from the command line or from a script to read such information. curl also allows you to save the contents of a webpage to a file like wget. you can read a URL using `$ curl <URL>`. For example, if you want to read http://www.linuxfoundation.org, type `$ curl http://www.linuxfoundation.org`. To get the contents of a webpage and store it to a file, type `$ curl -o saved.html http://www.mysite.com`. The contents of the main index file t the website will be saved in `saved.html`.

SECTION 3: TRANSFERRING FILES

When you are connected to a network, you may need to transfer files from one machine to another. File Transfer Protocol (FTP) is a well-known and popular method for transferring files between computers using the Internet. This method is built on a client-server model. FTP can be used within a browser or with standalone client programs.

FTP client enable you to transfer files with remote computers using the FTP protocol. These clients can be either graphical or command line tools. Filezilla, for example, allows use of the drag-and-drop approach to transfer files between hosts. All web browsers support FTP, all you have to do is give a URL like: ftp://ftp.kernel.org where the usual http:// becomes fpt://. Some command line FTP clients are: * ftp * sftp * ncftp * yafc (Yet Another FTP Client)

sftp is a very secure mode of connection which uses the Secure Shell (ssh) protocol. sftp encrypts its data and thus sensitive information is transmitted more securely. However, it does not work with so-called anonymous FTP (gues user credentials.) Both ncftp and yafc are also powerful FTP clients which work on a wide variety of operating systems including Windows and Linux.

Secure Shell (ssh) is a cryptographic network protocol used for secure data communication. It is also used for remote services and other secure services between two devices on the network and is very useful for administering systems which are not easily available to physically work on, but to which you have remote access. To run `my_command` on a remote system via SSh, at the terminal type: `$ ssh <remotesystem> my_command`. ssh then prompts you for the remote password. You can also configure ssh to securely allow your remot access without typing a password each time.

We can also move files securely using Secure Copy (scp) between two networked hosts. scp uses the ssh protocol for transferring data. To copy a local file to a remote system, at the command prompt, type `$ scp <localfile> <user@remotesystem>:/home/user/`. You will receive a prompt for the remote password. You can also configure scp so that it doesn't prompt fo a password for each transfer.

SUMMARY

- The IP (Internet Protocol) address is a unique logical network address that is assigned to a device on a network.
- IPv4 uses 32-bits for addresses and IPv6 uses 128-bits for addresses.
- Every IP address contains both a network and a host address field.
- There are five classes of network addresses available: A-E.
- DNS (Domain Name System) is used for converting internet domain names and host names to IP addresses.
- The ifconfig program is used to display current active network interfraces.

- The commands `ip addr show` and `ip route show` can be used to view IP addresses and routing information.
- You can use ping to check if the remote host is alive and responding.
- You can use the route utility program to manage IP routing.
- You can monitor and ebug network problems using networking tools.
- Firefox, Google Chrome, Chromium, and Epiphany are the main graphical browsers used in Linux.
- Non-graphical browsers or text browsers used in Linux are Lynx, Links, and w3m.
- You can use curl to obtain information about URLs.
- FTP (File Transfer Protocol) is used to transfer files over a network.
- ftp, sftp, ncftp, and yafc are command line FTp clients used in Linux.
- You can use ssh to run commands on remote systems. # Chapter 13: Manipulating Text

INTRODUCTION

By the end of this chapter you should be able to: * Display and append to file contents using cat and echo. * Edit and print file contents using sed and awk. * Search for patterns using grep. * Use multiple other utilities for file and text manipulation.

SECTION 1: CAT AND ECHO

Irrespective of the role you play with Linux (sysadmin, developer, or user), you often need to browse through and parse text files and/or extract data from them. These are file manipulation operations. Thus, it is essential for the Linux user to become adept at performing certain operations on files. Most of the time such file manipulation is done at the command line which allows users to perform tasks more efficiently than while using a GUI. Furthermore, the command line is more suitable for automating often executed tasks. Experienced sysadmins write customized scripts to accomplish such repetitive tasks, standardized for each particular environment. In this section, we will concentrate on command line file and text manipulation related utilities.

cat is short for concatenate and is one of the most frequently used Linux command line utilities. It is often used to read and print files as well as for simply viewing file contents. To view a file, use `$ cat <filename>`. For example, `$ cat readme.txt` will display the contents of `readme.txt` on the terminal. Often the main purpose of cat is to combine (concatenate) multiple files together. You can perform the actions listed on the following table using cat:

Command Usage - - `cat file1 file2` Concatenate multiple files and display the output. i.e. the entire content of the first file followed by that of the second file. `cat file1 file2 > newfile` Combine multiple files and save the output to a new file. `cat file >> existingfile` Append a file to the end of an existing file. `cat > file` Any subsequent lines typed will go into the file until `CTRL+D` is typed. `cat >> file` Any subsequent lines typed are appended to the file until `CTLR+D` is typed.

The tac command (cat spelled backwards) prints the lines of a file in reverse order. Each line remains the same, but the order of the lines is reversed. The syntax of tac is exactly the same at for cat as in: * `$ tac file` * `$ tac file1 file2 > newfile`

cat can be used to read from standard input (such as the terminal window) if no other files are specified. you can use the `>` operator to create and add lines into a new file and the `>>` operator to append lines (or files) to an existing file. To create a new file at the command prompt, type `$ cat > <filename>`. The command creates a new file and waits for the user to edit/enter the text. After you finish tying the required text, press `CTRL+D` at the beginning of the next line to save and exit the editing. Another way to create a file at the terminal is `$ cat > <filename> << EOF`. A new file is created and you can type the required input. To exit, enter `EOF` at the beginning of a line. Note that `EOF` is case-sensitive. You can also use another word such as `STOP`.

echo simply displays (echos) text. It is run by typing `$ echo string`. echo can be used to display a string on standard output (i.e. the terminal) or to place in a new file (using the `>` operator) or append to an already existing file (using the `>>` operator.) The `-e` option along with the following switches is used to enable special character sequences such as the newline character or horizontal tab. * `\n` represents newline. * `\t` represents tab.

echo is particularly useful for viewing the values of environment variables (built-in shell variables.) For example, `$ echo $USERNAME` will print the name of the user who has logged into the current terminal. The following table lists echo commands and their usage:

Command Usage - - `echo string > newfile` The specified string is placed in a new file. `echo string >> existingfile` The specified string is appended to the end of an already existing file. `echo $variable` The contents of the specified environment variable are displayed.

SECTION 2: SED AND AWK

It is very common to create and then repeatedly edit and/or extract contents from a file. This section will cover how to use sed and awk to easily perform such operations. Note that many Linux users and administrators will write scripts using more comprehensive language utilities such as python and perl rather than use sed and awk (and some other utilities discussed later.) Using such utilities is fine under most circumstances; one should always feel free to use the tools one is experienced with. However, the utilities that are described here are much lighter; i.e. they use fewer system resources and execute faster. There are times (such as during system booting) where a lot of time would be wasted using the more complicated tools and the system may not even be able to run them. Thus the simpler tools will always be needed.

sed is a powerful text processing tool and is one of the oldest, earliest, and most popular Linux utilities. It is used to modify the contents of a file, usually placing the contents into a new file. Its name is an abbreviation for stream editor. sed can filter text as well as perform substitutions in data streams working like a churn-mill. Data from an input source/file (or stream) is taken and moved to a working space. The entire list of operations/modifications is applied over the data in the working space, and the final contents are moved to the standard output space (or stream.)

You can invoke sed using commands like those listed in the following table:

Command Usage - - `sed -e command <filename>` Specify editing commands at the command line, operate on file, and put the output on standard out (i.e. the terminal.) `sed -f scriptfile <filename>` Specify a scriptfile containing sed commands, operate on file, and put output on standard out.

The `-e` command allows you to specify multiple editing commands simultaneously at the command line.

The following table lists some basic operations that can be used to perform multiple editing and filtering operations with sed where `pattern` is the current string and `replace_string` is the new string:

Command Usage - - `sed s/pattern/replace_string file` Substitute the first string occurrence in a line. `sed s/pattern/replace_string/g file` Substitute all string occurrences in a line. `sed 1, 3s/pattern/replace_string/g file` Substitute all string occurrences in a range of lines. `sed -i s/pattern/replace_string/g file` Save changes for string substituion in the same file.

You must use the -i option with care because the action is not reversible. It is always safer to use sed without the -i option and then replace the file yourself as in `$ sed s/pattern/replace_string/g file1 > file2`. This command will replace all occurrences of pattern with replace_string in file1 and then move the contents to `file2`. The contents of `file2` can be viewed with cat `file2`. If you approve, you can then overwrite the original file with `mv file1 file2`. Example: to convert 01/02/... to JAN/FEB/...: *

```
$ sed -e 's/01/JAN/' -e 's/02/FEB/' -e 's/03/MAR' -e 's/04/APR/'
-e 's/05/MAY/' -e 's/06/JUN/' -e 's/07/JUL/' -e 's/08/AUG/' -e
's/09/SEP' -e 's/10/OCT' -e 's/11/NOV/' -e 's/12/DEC/'
```

awk is used to extract and then print specific contents of a file and is often used to construct reports. It was created at Bell Labs in the 1970s and derived its name from the last names of its authors: Alfred Aho, Peter Weinberger, and Brian Kernighan. awk has the following features: * It is a powerful utility and interpreted programming language. * It is used to manipulate data files, retrieving and processing text. * It works well with fields (containing a single piece of data, essentially a column) and records (a collection of fields, essentially a line in a file.)

awk is invoked as shown in the following:

Command Usage - - `awk 'command' var=value file` Specify a command directly at the command line. `awk -f scriptfile var=value file` Specify a file that contains the script to be executed along with `f`.

As with sed, short awk commands can be specified directly at the command line, but a more complex script can be saved in a file that you can specify using the -f option.

The following table explains the basic tasks the can be performed using awk. The input file is read one line at a time and for each line, awk matches the given pattern in the given order and preforms the requested action. The `-F` option allows you to specify a particular field sparator character. For example, the `etc/passwd` file uses `:` to separate the fields, so the `-F:` option is used with the `/etc/passwd` file. The command/action in awk needs to be surrounded with apostrophes (or single-quote (')). awk can be used as follows:

Command Usage - - `awk '{ print $0 }' /etc/passwod` Print the entire file. `awk -F: '{ print $1 }' /etc/passwd` Print the first field (column of every line, separated by a space.) `awk -F: '{ print $1 $6 }' /etc/passwd` Print the first and sixth field of every line.

SECTION 3: FILE MANIPULATION UTILITIES

In managing your files, you may need to perform many tasks such as sorting data and copying data from one location to another. Linux provides several file manipulation utilities that you can use while working with text files. In this section, you will learn about the following file manipulation programs: * sort * uniq * paste * join * split

You will also learn about regular expressions and search patterns.

sort is used to rearrange the lines of a text file either in ascending or descending order, according to a sort key. You can also sort by particular fields of a file. The default sort key is the order of the ASCII characters (i.e. essentially alphabetically.) sort can be used as follows:

Syntax Usage -- `sort <filename>` Sort the lines in the specified file. `cat file1 file2 sort` Append the two files, then sort the lines and display the output on the terminal. `sort -r <filename>` Sort the lines in reverse order

When used with the `-u` option, sort checks for unique values after sorting the records (lines). It is equivalent to running `uniq` on the output of sort.

uniq is used to remove duplicate lines in a text file and is useful for simplifying text display. uniq requires that the duplicate entries to be removed are consecutive. Therefore, one often runs sort first and then pipes the output into uniq; if sort passed the `-u` option, it can do all this in one step. in the example shown, the file is called `names` and was originaly ted, Bob, Alice, bob, Carol, Alice. to remove the duplicate entries from some files, you can do `$ sort file1 file2 uniq > file3` or `sort -u file1 file2 > file 3` To count the number of duplicate entries, you can use `$ uniq -c filename`.

Suppose that you have a file that contains the full name of all employees and another file that lists their phone numbers and Employee IDs. You want to create a new file that contains all the data listed in three columns: name, employee ID, and phone number. How can you do this effectively without investing too much time? paste can be used to create a single file containing all three columns. The different columns are identified based on delimiters (spacing used to separate two fields.) For example, delimiters can be a blank space, a tab, or an enter. paste accepts the following options: * `-d` delimiters which specify a list of delimiters to be used instead of tabs for separating consecutive values on a single line. Each delimiter is used in turn; when the list has been exhausted, paste begins again at the next delimiter. * `-s` which causes paste to append the data in series rather than in parallel; that is, in a horizontal rather than a vertical fashion.

paste can be used to combine fields (such as name or phone number) from different files as well as combine lines from multiple files. For example, line one from `file` can be combined with line one of `file 2`, line two from `file1` can be combined with line two of `file2` and so on. To paste contents from two files, you can do `$ paste file1 file2`. The syntax to use a different delimiter is `$ paste -d, file1 file2` Common delimiters are 'space', 'tab', '', 'comma', etc.

Suppose that you have two files with similar columns. You have saved employee's phone numbers in two files: one with their first name and the other with their last name. You want to combine the files without repeating the data of common columns. you can achieve this using join, which is essentially an enhanced version of paste. It first checks whether the files share common fields such as names or phone numbers, and then joins the lines in two files based on a common field. To combibne two files on a common field, at the command prompt, type `$ join file1 file2`. For example, the common field (i.e. the field containing the same values) among the phonebook and directory files is the pohne number, as shown by the output of the following cat commands. * `$ cat phonebook` * `555-12-3456 Bob` * `555-231-3325 Carol` * `555-240-5678 Ted` * `555-290-6193 Alice`

- `$ cat directory`
- `555-123-4567 Anytown`
- `555-231-3325 Mytown`
- `555-340-5678 Yourtown`
- `555-290-6193 Youngstown`

The result of joining these two files is shown as the output of the following command: * `$ join phonebook directory` * `555-123-4567 Bob Anytown` * `555-231-3325 Carol Mytown` * `555-340-5678 Ted Yourtown` * `555-290-6193 Alice Youngstown`

split is used to break up (or split) a file into equal-sized segments for easier viewing and manipulation, and is generally used only on relatively large files. By default, split breaks up a file into 1,000-line segments. The original file remains unchanged, and a set of new files with the same name plus an added prefix is created. By default, the x prefix is added. to split a file into segments, use the command `$ split infle`. To split a file into segments using a different prefix, use the command `$ split infile <prefix>`

Regular expressions and text strings are used for matching a specific pattern or to search for a specific location such as the start or end of a line or word. Regular expressions can contain both normal characters or so-called metacharacters such as * and $. Many text editors and utilities such as vi, sed, awk, find, and grep work extensively with regular expressions. Some of the popular computer languages that use regular expressions include Perl, Python, and Ruby. It can get rather complicated and there are whole books written about regular expressions. These regular expressions are different from the wildcards (or "metacharacters") used in filename matching in command shells such as bash. The table below lists search patterns and their usage.

Search Patterns Usage `- -` `.(dot)` Match any single character az Match a or z `$` Match end of string `*` match preceding item 0 or more times.

For example, consider the expression the quick brown fox jumped over the lazy dog. Some of the patterns that can be applied to this sentence are:

Command Result `- -` `a..` matches azy b.j. matches br and ju `..$` matches og `l.*` matches lazy dog `l.*y` matches lazy `the.*` matches the whole sentence

SECTION 4: GREP

---

grep is extensively used as a primary text searching tool. It scans for specified patterns and can be used with regular expressions as well as simple strings as shown in the table below: Command Usage - - `grep [pattern] <filename>` Search for a pattern in a file and print all matching lines. `grep -v [pattern] <filename>` Print all lines that do not match the pattern. `grep [0-9] <filename>` Print the lines that contain the numbers 0 through 9. `grep -C 3 [pattern] <filename>` Print context of lines (specified number of lines above and below the pattern) for. matching the pattern. Here the number of lines is specified as 3.

SECTION 5: MISCELLANEOOUS TEXT UTILITIES

In this section, we will learn about some additional text utilities that you can use for performing various actions on your Linux files, such as changing the case of letters or determining the count of words, lines, and characters in a file. The tr utility is used to translate specified characters into other characters or to delete them. The general syntax is `$ tr [options] set1 [set2]` The items in the square brackets are optional. tr requires at least on argument and accepts a maximum of two. The first, designated `set1` in the example, lists the characters in the etext to be replaced ore moved. The second, `set2`, lists the characters that are to be substituted for the characters listed in the first argument. Sometimes these sets need to be surrounded by apostrophes (or single-quotes) in order to have the shell ignore that they means something special to the shell. It is usually safe (and may be required) to use the single-quotes around each of the sets as you will see in the following example. Suppose you have a file named `city` containing several lines of text in mixed case. To translate all lower case characters to upper case, at the command prompt, type `$ cat city tr a-z A-Z`. The following table lists additional flags you may specify.

Command Usage - - `$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLNOPQRSTUVWXYZ` Convert lower case to upper case. `$ tr '{}' '()' < inputfile > outputfile` Translates braces into parenthesis. $ echo “This is for testing” tr [:space:] ‘’ Translate white-space into tabs. $ echo “This is for testing” tr -s [:space:] Squeeze repetition of characters using `-s`. $ echo “the geek stuff” tr -d ‘t’ Delete specified characters using the `-d` option. $ echo “my username is 432234” tr -cd [:digit:] Complements the sets using `-c` option. `$ tr -cd [:print:] < file.txt` Remove all non-printable characters from a file. `$ tr -s '\n' ' ' < file.txt` Join all the lines in a file into a single file.

tee takes the output from any command and, while sending it to standard output, also saves it to a file. In other words, it “tees” the output stream from the command: one stream is displayed on the standard output and the other is saved to a file. For example, to list the contents of a directory on the screen and save the output to a file, use `$ ls -l tee newfile`. Typing `$ cat newfile` will then display the output of `ls -l`.

wc (word counts) counts the number of lines, words, and characters in a file or list of files. By default, all three of these options are active, but you can also toggle one or all of them manually. For example, to print the number of lines contained in a file, use `wc -l filename`.

Option Description - - `-l` Displays the number of lines `-c` Displays the number of bytes `-w` Displays the number of words

cut is used for manipulating column-based files and is designed to extract specific columns. The default column separator is the tab character. A different delimiter can be given as a command option. For example, to display the third column delimited by a blank space, use `ls -l cut -d -f3`.

SECTION 6: DEALING WITH LARGE FILES AND TEXT RELATED UTILITIES

Sysadmins need to work with configuration files, text files, documentation files, and log files. Some of these files may be large or become quite large as they accumulate data with time. These files will require both viewing and administrative updating. in this section, we will learn how to manage large files. For example, a banking system might maintain one simple large log file to record details of all of one day's ATM transactions. Due to a security attack or a malfunction, the administrator might be forced to check for some data by navigating within the file. In such cases, directly opening the file in an editor will cause issues due to the huge memory utilization, as an editor will usually try to read the whole file into memory first. However, one can use less to view the contents of such a large file, scrolling up and won page by page without the system having to place the entire file in memory before starting. This is much faster than using a text editor. Viewing the file can be done by uwing `$ less <filename>` or `$ cat <filename> less`. By default, manual (the man command) pages are sent through the less command.

head reads the first few lines of each named file (10 by default) and displays it on standard output. You can give a different number of lines as an option as well. For example, if you want to print the first 5 lines from `atmtrans.txt` you would use `$ head -n 5 atmtrans.txt`. You can also just use `head -5 atmtrans.txt`.

tail prints the last few lines of each named file and displays it on standard output. By default it displays the last 10 lines. Just as with head, you can give a different number of lines as an option. tail is especially useful when you are troubleshooting any issue using log files as you probably want to see only the most recent lines of output. For example, to display the last 15 lines of `atmtrans.txt`, use the command `$ tail -n 15 atmtrans.txt`. You can also just say `tail -15 atmtrans.txt`. To continually monitor new output in a growing log file, use `$ tail -f atmtrans.txt`. This command will continuously display any new lines of output in atmtrans.txt as soon as they appear. Thus it enables you to monitor any current activity that is being reported and recorded.

strings is used to extract all printable character strings found in a file or files given as arguments. It is useful in locating human readable content embedded in binary files: for text files you can just use grep. For example, to search for the string my_string in a spreadsheet, use `$ strings book1.xls grep my_string`. When working with compressed files many standard commands can't be used directly. For many commonly-used file and text manipulation programs, there is also a version especially designed to work directly with compressed files.

When working with compressed files, many standard commands cannot be used directly. For many commonly-used file and text manipulation programs, there is also a version especially designed to work directly with compressed files. These associated utilities have the letter z prefixed to their name. For example, we have utility programs such as zcat, zless, zdiff, and zgrep. The following table lists some z family commands:

Command Description `- - $ zcat compressed-file.txt.gz` To view a compressed file. `$ zless <filename.gz` or `$ zmore <filename>.gz` To page through a compressed file. `$ zgrep -i less test-file.txt.gz` To search inside a compressed file. `$ zdiff filename1.txt.gz filename2.txt.gz` To compare two compressed files.

Note that if you run zless on an uncompressed file, it will still work and ignore the decompression stage. There are also equivalent utility programs for other compression methods besides gzip; i.e. we have bzcat and bzless associated with bazip2 and xzcat and xzless associated with xz.

SUMMARY

- The command line often allows the user to perform tasks more efficiently than the GUI.
- cat, short for concatenate, is used to read, print, and combine files.
- echo displays a line of text either on standard output or to place in a file.
- sed is a popular stream editor often used to filter and perform substitutions on files and text data streams.

- awk is an interpreted programming language typically used as a data extraction and reporting tool.
- sort is used to sort text files and output stream in either ascending or descending order.
- uniq eliminates duplicate entries in a text file.
- paste combines fields from different files and can also extract and combine lines from multiple sources.
- join combines lines from two files based on a common field. It works only if files share a common field.
- split breaks up a large file into equal-sized segments.
- Regular expressions are text strings used for pattern matching. The pattern can be used to search for a specific location, such as the start or end of a line or word.
- grep searches text files and data streams for patterns and can be used with regular expressions.
- tr translates characters, copies standard input to standard output, and handles special characters.
- tee saves a copy of standard output to a file while still displaying it at the terminal.
- wc (word count) displays the number of lines, words, and characters in a file or a group of files.
- cut extracts columns from a file.
- less views files a page at a time and allows scrolling in both directions.
- head displays the first few lines of a file or data stream on standard output. By default it displays 10 lines.
- tail displays the last few lines of a file or data stream or standard output. By default, it displays 10 lines.
- strings extracts printable character strings from binary files.
- The z command family is used to read and work with compressed files. # Chapter 14: Printing

INTRODUCTION

By the end of this chapter you should know how to: * Configure a printer on a Linux machine. * Print documents. * Manipulate postscript and pdf files using command line utilities.

SECTION 1: CONFIGURATION

To manage printers and print directly from a computer or across a networked environment, you need to know hot configure and install a printer. Printing itself requires software that converts information from the application you are using to a language your printer can understand. The standard for printing software is the Common UNIX Printing System (CUPS). CUPS is the software that is used behind the scenes to print from applications like a web browser or LibreOffice. It converts page descriptions produced by your applications (put a paragraph here, draw a line here, and so forth) and then sends the information to the printer. it acts as a print server for local as well as network printers.

Printers manufactured by different companies may use their own particular print languages and formats. CUPS uses a modular printing system which accommodates a wide variety of printers and aso processes various data formats. This make the printing process simpler; you can concentrate more on printing and less on how to print. Generally, the only time you should need to configure your printer is when you use it for the first time. in fact, CUPS often figures things out on its own by detecting and configuring any printers it locates. CUPS carries out the printing process with the help of various components: * Configuration files * Scheduler * Job files * Log files * Filter * Printer drivers * Backend

CUPS is designed around a print scheduler that manages print jobs, handles admin commands, allows users to query the printer status, and manages the flow of data through all CUPS components. CUPS has a browser-based interface which allows you to view and manipulate the order and status of pending print jobs.

The print scheduler reads server settings from several configuration files, the two most important of which are `cupsd.conf` and `printers.conf`. These and all other CUPS related config files are stored under the `/etc/cups` directory. `cupsd.conf` is where most system-wide settings are located; it doesn't contain any printer-specific details. Most of the settings available in this file relate to network security. For example which systems can access CUPS network capabilities, how printers are advertised on the local network, what management features are offered, and so on. `printers.conf` is where you will find the printer-specific settings. For every printer connected to the system, a corresponding section describes the rpinter's status and capabilities. This file is generated only after adding a printer to the system and should not be modified by hand. You can view the full list of config files by typing `$ ls -l /etc/cups/`.

CUPS stores print requests as files under the `/var/spool/cups` (these can actually be accessed before a document is sent to the printer.) Data files are prefixed with the letter 'd' while control files are prefixed with the letter 'c'. After a printer successfully handles a job, data files are automatically removed. These data files belong to what is commonly known as the print queue.

Log files are places in `/var/log/cups` and are used by the scheduler to record activities that have taken place. These files include access, error, and page records. to view what log files exist, type `sudo ls -l /var/log/cups`. Note that on some distros, permissions are set such that you don't need the sudo.) You can view the log files with the usual tools.

CUPS uses filters to convert job file formats to printable formats. Printer drivers contain descriptions for currently connected and configured printers and are usually stored under `/etc/cups/ppd/`. The print data is then sent to the printer through a filter and vie a backend that helps to locate devices connected to the system. When you execute a print command, the scheduler validates the command and processes the print job creating job files according to the settings specified in the config files. Simultaneously, the scheduler records activities in the log files. Job files are processes with the help of the filter, printer driver, and backend, and then sent to the printer.

Due to printing being a relatively important and fundamental feature of any Linux distro, most Linux systems come with CUPS preinstalled. In some cases, especially for Linux server setups, CUPS may have been left uninstalled. This may be fixed by installing the corresponding package manually. To install CUPS, ensure that your system in connected to the internet.

After installing CUPS, you'll need to start and manage the CUPS daemon so that CUPS is ready for configuring a printer. Managing the CUPS daemon is simple; all management features are wrapped around the cups init script, which can easily be started, stopped, and restarted.

Each Linux distro has a GUI application that lets you add, remove, and configure local or remote printers. Using this application, you can easily set up the system to use both local and network printers. The following screens show how to find and use the appropriate application in each of the distro families covered in this course. When configuring a printer, make sure the device is currently turned on and connected to the system; if so, it should show up in the printer selection menu. If the printer is not visible, you may want to troubleshoot using tools that will determine if the printer is connected. For common USB printers, for example, the lsusb utility will show a line for the printer. Some printer manufacturers also require some extra software to be installed in order to make the printer visible to CUPS, however, due to the standardization these days, this is rarely required.

CUPS also comes with its own web server, which makes a configurable interface available via a set of CGI scripts. The web interface allows you to: * Add and remove local/remote printers * Configure printers: - Local/remote printers - Share a printer as a CUPS server * Control print jobs: - Monitor jobs - Show completed or pending jobs - Cancel or move jobs

SECTION 2: PRINTING OPERATIONS

Many graphical applications allow users to access printing features using the `CTRL+P` shortcut. To print a file, you first need to specify the printer (or a file name and location if youare printing to a file instead) you want to use; then select the page setup, quality, and color options. After selecting the required options, you can submit the document for printing. The document is then submitted to CUPS. You can use your browser to access the CUPS web interface at http://localhost:631/ to monitor the status of a printing job. Now that you have configured your printer, you can print using either the Graphical or Command Lin interfaces.

CUPS provides two command-line interfaces, descended from the System V and BSd flavors of UNIX. You can use either lp (System V) or lpr (BSD) to print. You can use these commands to print text, PostScript, PDF, and image files. These commands are useful in cases where printing operations must be automated (from shell scripts, for instance, which contain multiple commands in one file.) lp is just a command line front-end to use the lpr utility that passes input to lpr. We will only discuss lp in detail.

lp and lpr accept command line options that help you perform all operations that the GUI can accomplish. lp is typically used with a file name as an argument. Some lp commands and other printing utilities are listed in the table below:

Command Usage `-` `-` `lp <filename>` To print the file to the default printer `lp -d printer <filename>` To print to a specific printer (useful if multiple printers are available) program lp echo string lp To print the output of a program `lp \n number <filename>` To print multiple copies `lpoptions -d printer` to set the default printer `lpq -a` To show the queue status `lpadmin` To configure printer queues

The lpoptions utility can be used to set printer options and defaults. Each printer has a set of tags associated with it, such as the default number of copies and authentication requirements. You can execute the command `lpoptions help` to obtain a list of supported options. lpoptions can also be used to set system-wide values such as the default printer.

In Linux, command line print job management commands allow you to monitor the jab state as well as managing the listing of all printers and checking their status, and cancelling or moving print jobs to another printer. Some of the commands that can be used are listed in the following table:

Command Usage - - `lpstat -p -d` To get a list of available printers, along with their status `lpstat -a` To check the status of all connected printers, including job numbers `cancel job-id` or `lprm job-id` to cancel a print job `lpmove job-id newprinter` To move a print job to a new printer

SECTION 3: MANIPULATING POSTSCRIPT AND PDF FILES

Postscript is a standard page description language. It effectively manages scaling of fonts and vector graphics to provide quality printouts. It is purely a text format that contains the data fed to a PostScript interpreter. The format itself is a language that was developed by Adobe in the early 1980s to enable the transfer of data to printers. Features of PostScript are: * It can be used on any printer that is PostScript-compatible; i.e. any modern printer. * Any program that understands the PostScript specification can print to it. * Information about page appearance etc. is embedded in the page.

enscript is a tool that is used to convert a text file to PostScript and other formats. It also supports Rich Text Format (RTF) and HyperText Markup Language (HTML). For example, you can convert a text file to two column (-2) formatted PostScript using the command: `enscript -2 -r -p psfile.ps textfile.txt`. This command will also rotate 9-r) the output to print so the width of the paper is greater than the height (landscape mode) reducing the number of pages required for printing. The commands that are used with enscript are listed in the following table (for a file called `textfile.txt`):

Command Usage - - `enscript -p psfile.ps textfile.txt` Convert a text file to PostScript (saved to psfile.ps) `enscript -n -p psfile.ps textfile.txt` Convert a text file to n columns where n = 1 - 9 (saved in psfile.ps) `enscript textfile.txt` Print a text file directly to the default printer

Linux has many standard programs that can read PDF files as well as many applications that can easily create them including all available office suites such as LibreOffice. The most common Linux PDF readers are: 1. Evince is available on virtually all distros and is the most widely used program 2. Okular is based on the older kpdf and available on any distro that provides the KDE environment 3. GhostView is on of the first open source PDF readers and is universally available 4. Xpdf is one of the oldest open source PDF readers and still has a good user base

All of these open source PDF readers support and can read files following the PostScript standard unlike the proprietary Adobe Acrobat Reader which was once widely used on Linux systems, but with the growth of these excellent programs, few Linux users use it today.

At times, you may want to merge, split or roate PDf files; not all of these operations can be achieved while using a PDF viewer. A great way to do this is to use the "PDF Toolkit" pdftk, to perform a very large variety of sophisticated tasks. Some of these operations include: * Merging/splitting/rotating PDF documents * Repairing corrupted PDF pages * Pulling single pages from a file * Encrypting and decrypting PDF files * Adding, updating, and exporting a PDF's metadata * Exporting bookmarks to a text file * Filling out PDF forms

There's very little pdftk cannot do when it comes to working with PDF files; it is the Swiss Army knife of PDF tools.

To install pdftk on Ubuntu, use the command `$ sudo apt-get install pdftk`. On CentOS, use `$ sudo yuim install pdftk`. On openSUSE use `$ sudo zypper install pdftk`. You may find that CentOS (and RHEL) don't have pdftk in their packaging system, but you can obtain the PDF Toolkit directly from the PDF Lab's vwebsite by downloading from http://www.pdflabs.com/docs/install-pdftk-on-redhat-or-centos/

You can accomplish a wide variety of tasks using pdftk including those listed on the following table:

Command Usage - - `pdftk 1.pdf 2.pdf cat output 12.pdf` Merge the two documents `1.pdf` and `2.pdf`. The output will be saved to `12.pdf`. `pdftk A=1.pdf cat A1-2 output new.pdf` Write only pages 1 and 2 of `1.pdf`. The output will be saved to `new.pdf`. `pdftk A=1.pdf cat A1-endright output new.pdf` Rotate all pages of `1.pdf` 90 degrees clockwise and save result in `new.pdf`.

If you're working with PDF files that contain confidential information and you want to ensure that only certain people can view the PDF file, you can apply a password to it using the user_pw option. You can do this using a command like `$ pdftk public.pdf output private.pdf user_pw PROMPT`. When you run this command, you will receive a prompt to set the required password, which can have a maximum of 32 characters. A new file, `private.pdf` will be created with the identical content as `pulic.pdf`, but anyone will need to type the password to be able to view it.

You can also use other tools such as pdfinfo, flpsed, and pdfmod to work with PDF files. pdfinfo can extract information about PDF files, especially when the files are very large or when a graphical interface is not available. flpsed can add data to a PostScript document. This tool is especially useful for filling in forms or adding short comments into the document. pdfmod is a simple application that provides a graphical interface for modifying PDF documents. Using this tool, you can reorder, rotate, and remove pages; export images from a document; edit the title, subject, and author; add keywords; and combine documents using drag-and-drop action. For example, to collect the details of a document, you can use the command `$ pdfinfo /usr/share/doc/readme.pdf`. Most users today are far more accustomed to working with files in PDF format, viewing them easily either on the Internet through their browser or locally on their machine. The PostScript format is still important for various technical reasons that the general user will rarely have to deal with. From time to time, you may need to convert files from one format to the other, and there are very simple utilities for accomplishing that. ps2pdf and pdf2ps are part of the ghostscript package installed on or available on all Linux distros. As an alternative, there are pstopdf and pdftops which are usually part of the poppler packagte which may need to be added through your package manager. Unless you are doing a lot of conversions or need some of the fancier options which you can read about in the man pages for these utilities) it doesn't really matter which one you use. The following table lists some usage examples for these utilities:

Command Usage `- -` `pdf2ps file.pdf` Converts `file.pdf` to `file.ps` `ps2pdf file.ps` Converts `file.ps` to `file.pdf` `pstopdf input.ps output.pdf` Converts `input.ps` to `output.pdf` `pdftops input.pdf output.ps` Converts `input.pdf` to `output.ps`

SUMMARY

- CUPS provides two command-line interfaces: the System V and BSD interfaces.
- The CUPS interface is available at http://localhost:631.
- lp and lpr are used to submit a document to CUPS directly from the command line.
- lpoptions can be used to set printer options and defaults.
- PostScript effectively manages scaling of fonts and vector graphics to provide quality prints.
- enscript is used to convert a text file to PostScript and other formats.
- Portable Document Format (PDF) is the standard format used to exchange documents while ensure a certain level of consistency in the way the documents are viewed.
- pdftk joins and splits PDFs; pulls single pages from a file; encrypts and decrypts PDF files; adds, updates, and exports a PDF's metadata; exports bookmarks to a text file; adds or removes attachments to a PDF; fixes a damaged PDf; and fills out PDF forms.
- pdfinfo can extract information about PDF documents.

- flpsed can add data to a PostScript document.
- pdfmod is a simple application with a graphical interface that you can use to modify PDF documents. # Chapter 15: Bash Shell Scripting

INTRODUCTION

By the end of this chapter, you should be able to: * Explain the features and capabilities of bash shell scripting. * Know the basic syntax of scripting statements. * Be familiar with various methods and constructs used. * Know how to test for properties and existence of files and other objects. * Use conditional statements such as if-then-else blocks. * Perform arithmetic operations using scripting language.

SECTION 1: FEATURES AND CAPABILITIES

Suppose you want to look up a filename, check if the associated file exists, and then respond accordingly, displaying an error message confirming or denying the file's existence. If you only need to do it once, then you can just type a sequence of commands at a terminal. However, if you need to do this multiple times, automation is the way to go. In order to automate sets of commands, you'll need to learn how to write shell scripts, the most common of which are used with bash.

A shell is a command line interpreter which provides the user interface for terminal windows. It can also be used to run scripts, even in non-interactive sessions without a terminal window, as if the commands were being directly typed in. For example, typing `$ FIND . -NAME *.c -ls` at the command line accomplishes the same things as executing a script file containing the lines

[]

The `#!/bin/bash` in the first line should be recognized by anyone who has developed any kind of script in UNIX environments. The first line of the scripts that starts with `#!` contains the full path to the command interpreter (in this case `/bin/bash`) that is to be used on the file. You have a few choices depending on which scripting language you use.

The command interpreter is tasked with executing statements that follow it in the script. Commonly used interpreters include: * `/usr/bin/perl` * `/bin/bash` * `/bin/csh` * `/usr/bin/python` * `/bin/sh`

Typing a long sequence of commands at a terminal window can be complicated, time consuming, and error prone. By deploying shell scripts, using the command-line becomes an efficient and quick way to launch complex sequences of steps. The fact that shell scripts are saved in a file also makes it easy to use them to create new script variations and share standard procedures with several users. Linux provides a wide choice of shells; exactly what is available on the system is listed in `/etc/shells`. Typical choices include: * `/bin/sh` * `/bin/bash` * `/bin/tcsh` * `/bin/csh` * `/bin/ksh`

Most Linux users use the default bash shell, but those with long UNIX backgrounds with other shells may want to override the default.

For the first exercise, we will write a simple bash script that displays a two-line message on the screen. You can type:

[]

then press `ENTER` and `CTRL+D` to save the file, or just create `exscript.sh` in your favorite text editor. Then, type `$ chmod +x exscript.sh` to make the file executable. The `chmod +x` makes the file executable for all users. You can then run it by simply typing `$ ./exscript.sh` or doing:

[]

Note that if you use the second form you won't have to make the file executable.

We can also create more interactive examples using a bash script. In this example, the user will be prompted to enter a value, which is then displayed on the screen. The value is stored in a temporary variable `sname`. We can reference the value of a shell variable by using a $''infrontofthevariablesuchas$‘sname `. To create this script, you need to create a file named` ioscript.sh `in your favorite editor with the following content: "‘bash #!/bin/bash # Interactive reading of variables echo ENTER YOUR NAME read sname # DISPLAY of variable values echo $sname "‘ Once again, your file needs to be made executable with` $ chmod +x ioscript.sh `. In the above example, when the script` ./ioscript.sh `is executed, the user will receive a prompt` ENTER YOUR NAME `. The user then needs to enter a value and press the` ENTER ‘ key. The value will then be printed out. The pound-sign(#) is used to start comments in the script and can be placed anywhere in the line (the rest of the line is considered a comment.)

All shell scripts generate a return value upon finishing execution. The value can be set with the exit statement. Return values permit a process to monitor the exit state of another process often in a parent-child relationship. This helps to determine how this process terminated and take any appropriate steps necessary, contingent on success or failure.

As a script executes, one can check for a specific value or condition and return success or failure as the result. By convention, success is return as 0, and failure is returned as a non-zero value. An easy way to demonstrate success and failure completion is to execute `ls` on a file that exists and one that doesn't, as shown in the following example, where the return value is stored by the environment variable represented by `$?`:

[]

In this example, the system is able to locate the file `/etc/passwd` and returns a value of 0 to indicate success; the return value is always stored in the `$?` environment variable. Applications often translate these return values into meaningful messages that can be easily understood by the user.

SECTION 2: SYNTAX

Scripts require you to follow a standard language syntax. Rules deliniate how to define variables and how to construct and format allowed statements, etc. The table below lists some special character usages within bash scripts.

Character Description - - `#` Used to add a comment, except when used as `\#` or as `#!` when starting a script. `\` Used at the end of a line to indicate continuation on the next line. `;` Used to interpret what follows as a new command. `$` Indicates what follows is a variable.

Note that when `#` is inserted at the beginning of a line of commentary, the whole line is ignored.

[]

Users sometimes need to combine several commands and statements and even execute them based on the behaviour of operators used in between them. This method is called chaining of commands. The concatenation operator (\) is used to concatenate large commands over several lines in the shell. For example, you want to copy the file `/var/ftp/pub/userdata/custdata/read` from server1.linux.com to the `/opt/oradba/master/abc` directory on server3.linux.co.in. To perform this action, you can write the command using the \ operator as:

[]

The command is divided into multiple lines to make it look readable and easier to understand. The \ operator at the end of each line combines the commands from multiple lines and executes it as one single command.

Sometimes, you may want to group multiple commands on a single line. The `;` (semicolon) character is used to separate these commands and execute them sequentially as if they had been typed on separate lines. The 3 commands in the following example will all execute even if the ones preceding them fail:

[]

However, you may want to abort subsequent commands if one fails. You can do this using the `&&` operator as in:

[]

If the first command fails, the second one will never be executed. A final refinement is to use the (or) operator as in:

[]

In this case, the script proceeds until something succeeds and then stops executing any further steps.

A function is a code block that implements a set of operations. Functions are useful for executing procedures multiple times perhaps with varying input variables. Functions are also often called subroutines. using functions in scripts requires two steps: 1. Declaring a function 2. Calling a function

The function declaration requires a name which is used to invoke it. The proper syntax is:

[]

For example, the following function is named `display`:

[]

The function can be as long as desired and have many statements. Once defined, the function can be called later as many times as necessary. You can also pass an argument into a function. These can be referred to as `$1`, `$2`, etc. Shell scripts are used to execute sequences of commands and other types of statements. Commands can be divided into the following categories: * Compiled applications * Built-in bash commands * Other scripts

Compiles applications are binary executable files that you can find on the filesystem. The shell script always has access to compiled applications such as rm, ls, df, vi, and gzip. bash has many built-in commands which can only be used to display the output within a terminal shell or shell script. Sometimes these commands have the same name as executable programs on the system such as echo, which can lead to subtle problems. bash built-in commands include cd, pwd, echo, read, logout, printf, let, and ulimit. A complete list of bash built-in commands can be found in the bash man page or by simply typing `help`. At times, you may need to substitute the result of a command as a portion of another command. This can be done in two ways: 1. By enclosing the inner command with backticks(') 2. By enclosing the inner command with `$()`.

No matter the method, the innermost command will be execute in a newly launched shell environment and the standard output of the shell will be inserted where the command substitution has been done. Virtually any command can be execute this way. Both of these methods enable command substitution; however, the `$()` method allow command nesting. New scripts should always use this more modern method. For example:

[]

In the above example, the output of the command `uname -r` becomes the argument for the `cd` command. Almost all scripts use variables containing a value which can be used anywhere in the script. These variables can either be user or system defined. Many applications use such environment variables for supplying inputs, validation, and controlling behaviour. Some examples of standard environment variables are `HOME`, `PATH`, and `HOST`. When referenced, environment variables must be prefixed with the `$` symbol as in `$HOME`. You can view and set the value of environment variables. For example, the following command displays the value stored in the `PATH` variable.

[]

However, no prefix is required when setting or modifying the variable value. For example, the following command sets the value of `MYCOLOR` to blue:

[]

You can get a list of environment variables with the env, set, or printenv commands.

By default, the variables created within a script are available only tot he subsequent steps of that script. Any child processes (sub-shells) do not have automatic access to the values of these variables. To make them available to child processes, they must be promoted to environment variables using the export statement as in:

[]

or

[]

While child processes are allowed to modify the value of exported variables the parent will not see any changes; exported variables are not shared, only copied.

Users often need to pass parameter values to a script such as a filename, date, etc. Scripts will take different paths or arrive at different values according to the parameters (command arguments) that are passed to them. These values can be text or numbers as in:

[]

Within a script, the parameter or an argument is represented with a `$` and a number. The table below lists some of the parameters: Parameter Meaning `- -` `$0` Script name `$1` First parameter `$2`, `$3`, etc. Second parameter, third parameter, etc. `$` All parameters `$#` Number of arguments

The following is an example script that demonstrates usage of parameters in a script.

[]

You can make the script executable with `chmod +x`. Now run the script giving it three arguments as in `script3.sh one two three`. The script will be processed as follows: * `$0` prints the script name: `script3.sh` * `$1` prints the first parameter: `one` * `$2` prints the second parameter: `two` * `$3` prints the third parameter: `three` * `$` prints all parameters: `one two three` * The final statement becomes: `All done with script3.sh`

Most operating system accept input from the keyboard and display the output on the terminal. However, in shell scripting, you can send the output to a file. the process of diverting the output to a file is called redirection. The `>` character is used to write output to a file. For example, the following command sends the output of free to the file `/tmp/free.out`:

[]

To check the contents of the `/tmp/free.out` file, you can use `$ cat /tmp/free.out`. Two `>` characters (`>>`) will append output to a file if it exists and act just like `>` if the file doesn't already exist.

Just as the output can be redirected to a file, the input of a a file can be read from a file. The process of reading input from a file is called input redirection and uses the `>` character. If you create a file called `script8.sh` with the following contents:

[]

and then execute it with `$ chmod +x script8.sh ; ./script8.sh`, it will count the number of lines from the `/temp/free.out` file and display the results.

SECTION 3: CONSTRUCTS

Conditional decision making using an if statement is a basic construct that any useful programming or scripting language must have. When an if statement is used, the ensuing action depend on the evaluation of specified conditions such as: * Numerical or sting comparisons * Return value of a command (0 for success) * File existence or permissions

In compact form, the syntax of an if statement is:

[]

A more general definition is:

[]

In this first example, we will accept a file name from the command prompt and then display its existence. In order to this this we need to: 1. Accept the command line argument and store in 'file1' 2. Check for its existence and if found true 1. Display the "File exists" 3. Else 1. Display the "File doesn't exist"

This can be accomplished with the following code:

[]

You can also nest if statements. In this example, we will accept 2 numbers, 1 operator, and display the calculated value based on the operator. The steps to do this are: 1. Display a message to enter the first number 2. Read the first number 3. Display a message to get the second number 4. Read the second number 5. Display a message to enter the option for the operation 6. Read the option number 7. If the operator is 1 1. Display the sum of 2 numbers 8. Else if the operator is 2 1. Display the difference of 2 numbers 9. Else if the operator is 3 1. Display the product of 2 numbers 10. Else 1. Display invalid input

This can be accomplished with the following code:

[]

Another statement that you can use is the elif statement. This works the same way as the nested if statement in the above example allowing you to check for multiple values. In this example, we will accept a number and display if it is equal to, greater than, or less than 100. The steps to accomplish this are: 1. Display the message to enter a number 2. Read the number 3. If give number = 100 1. Display "Count is 100" 4. Else if number > 100 1. Display "Count is greater than 100" 5. Else 1. Display "Count is less than 100"

You can use the following code to accomplish this.

[]

The following if statement checks for the `/etc/passwd` file and if the file is found, displays `/etc/passwd exists`.

[]

Notice the use of the square brackets (`[]`) to delineate the test condition. There are many other kinds of test you can perform such as checking whether two numbers are equal to, greater than, or less than each other and make a decision accordingly. In modern scripts, you may see double brackets as in `[[ -f /etc/passwd ]]`. This is not an error. It is never wrong to do so and it avoid subtle problems such as referring to an empty environment variable without surrounding it in double quotes.

The if statement can be used to test for file attributes such as: * File or directory existence * Read or write permissions * Executable permission

The if statement in the following example checks if the file `/etc/passwd` is a regular file.

[]

Note the very common practice of putting `; then` on the same line as the if statement. bash provides a set of file conditionals that can be used with the if statement including those listed in the following table.

Condition Meaning - - -`-e file` Check if the file exists. `-d file` Check if the file is a directory. `-f file` Check if the file is a regular file (i.e. not a symbolic link, device node, directory, etc.) `-s file` Check if the file is of non-zero size. `-g file` Check if the file has `sgid` set. `-u file` Check if the file has `suid` set. `-r file` Check if the file is readable. `-w file` Check if the file is writable. `-x file` check if the file is executable.

You can view the full list of file conditions using the command `man 1 test`.

You can use the if statement to compare strings using the `==` operator. The syntax is as follows:

[]

You can also use specially defined operators with the if statement to compare numbers. The various operators that are available are listed in the following table:

Operator Meaning - - -`-eq` Equal to `-ne` Not equal to `-gt` Greater than `-lt` Less than `-ge` Greater than or equal to `-le` Less than or equal to

The syntax for comparing numbers is `exp1 -op exp2`.

In the following example, we will display if two numbers are zeroes and if they are less than, equal to, or greater than compared to each other. To accomplish this we will use the following steps: 1. Ask the user to enter the first number 2. Read the first number 3. Ask the user to enter the second number 4. Read the second number 5. Compare the first number and second number to zero. If found true 1. Display both numbers are zero 6. If not, then compare whether the first number is equal to the second number. If found true 1. Display both numbers are equal 7. If not, then compare whether the first number if greater than the second number. If found true 1. Display the first number is greater than the second number 8. If not 1. Display the first number is less than the second number

This can be accomplished with the following code:

[]

Arithmetic experssion can be evaluated in the following three ways (spaces are important) * Using the expr utility: expr is a standard but somewhat deprecated program. The syntax is as follows: `bash expr 8 + 8 echo $(expr 8 + 8)` * Using the `$((...))` syntax: this is the built-in shell format. The syntax is as follows: `bash echo $((x+1))` * Using the built in shell command `let`. The syntax is as follows: `bash let x=( 1 + 2 ); echo $x`

In modern shell scripts, the use of expr is better replaced with `var=$((...))`.

SUMMARY

- Scripts are a sequence of statements and commands stored in a file that can be executed by a shell. The most commonly used shell in Linux is bash.
- Command substitution allows you to substitute the result of a command as a portion of another command.
- Functions or routines are a group of commands that are used for execution.
- Environment variables are quantities either pre-assigned by the shell, or defined and modified by the user. To make environment variables visible to child processes, they need to be exported.
- Scripts can behave differently based on the parameters (values) passed to them.
- The process of writing the output to a file is called output redirection.
- The process of reading input from a file is called input redirection.
- The if statement is used to select an action based on a condition.
- Arithmetic expressions consist of numbers and arithmetic operators such as `+`, `-`, and `*`. # Chapter 16: Advanced Bash Scripting

INTRODUCTION

By the end of this chapter, you should be able to: * Manipulate strings to perform actions such as comparison and sorting. * Use boolean expressions when working with multiple data types including strings or numbers, as well as files. * Use case statements to handle command line options. * Use looping constructs to execute one or more lines of code repetitively. * Debug scripts using set `-x` and set `+x`. * Create temporary files and directories. * Create and use random numbers.

SECTION 1: STRING MANIPULATION

A string variable contains a sequence of text characters. It can include letters, numbers, symbols, and punctuation marks. Some examples include: * "abcde" * "123" * "abcde 123" * "abcde-123" * "&abcde=%123"

String operators include those that do comparison, sorting, and finding the length. The following table demonstrates the use of some basic string operations.

Operator Meaning - - `[[ strin1 > string2 ]]` Compares the sorting order of string1 and string2 `[[ string1 == string2 ]]` compares the characters in string1 with the characters in string2 `myLen1=${#string1}` Saves the length of string1 in the variable myLen1.

At times, you may not need to compare or use an entire string. To extract the first character of a string, we can specify `${string:0:1}`. Here, 0 is the offset of the stri;ng (i.e. which character to begin from) where the extraction needs to start and 1 is the number of characters to be extracted. To extract all characters in a string after a dot (`.`), use the expression `${string#.}`.

SECTION 2: BOOLEAN EXPRESSIONS

Boolean expressions evaluate to either true or false. Results are obtained using the various boolean operators listed in the following table.

Operator Operation Meaning - - - `&&` AND The action will be performed only if both the conditions evaluate to true. OR The action will be performed if any one of the conditions evaluates to true. `!` NOT The action will be performed only if the condition evaluates to false.

Note that if you have multiple conditions string together with the `&&` operator, processing stops as soon as the condition evaluates to false. For example, if you have `A && B && C` and A is true but B is false, then C will never be evaluated. Likewise, if you use the operator, processing stops as soon as anything is true. For example, if you have `A B C` and A is false and B is true, you will never execute C.

Boolean expressions return either true or false. We can use such expressions when working with multiple data types including strings or numbers as well as within files. For example, to check if a file exists, use the conditional test `[ -e <filename> ]`. Similarly, to check if the value of `number1` is greater than the value of `number2`, use the conditional test `[ $number1 -gt $number2 ]`. The operator `-gt` returns true if `number1` is greater than `number2`.

SECTION 3: THE CASE STATEMENT

The case statement is used in scenarios where the actual value of a variable can lead to different execution paths. case statements are often used to handle command-line options. The following are some of the advantages of use the case statement: * It is easier to read and write. * It is a good alternative to nested, multi-level if-then-else-fi code blocks. * It enables you to compare a variable against several values at once. * It reduces the complexity of a program.

The following is the basic structure of a case statement:

[]

Another example of a case statement is the following which prompts the user to type in a letter and identifies if it is a vowel or consonant:

[]

SECTION 4: LOOPING CONSTRUCTS ____

By using looping constructs, you can execute one or more lines of code repetitively. Usually, you do this until a conditional test return either true or false as is required. Three types of loops are often used in most programming languages: * `for` * `while` * `until`

All of these loops are easily used for repeating a set of statements until the exit condition is true.

The for loop operates on each element of a list of items. The syntax for the for loop is:

[]

In this case, `variable-name` and `list` are substituted by you as appropriate. As with other looping constructs, the statements that are repeated should be enclosed by `do` and `done`. The following example demonstrates how to count the numbers from 1 to 4 in order and can be extrapolated to any length of numbers.

[]

The while loop repeats a set of statements as long as the control command returns true. The syntax is:

[]

The set of commands that need to be repeated should be enclosed between `do` and `done`. You can use any command or operator as the condition. Often, it is enclosed within square brackets(`[]`). The following example calculates the factorial of a number:

[]

The until loop repeats a set of statements as long as the control command is false. Thus it is essentially the opposite of the while loop. The syntax is:

[]

Similar to the while loop, the set of commands that need to be repeated should be enclosed between do and done. you can use any command or operator as the condition. The following example demonstrates how to display the odd numbers from 1 to 10.

[]

This will print out:

```
NUMBER
1
3
5
7
9
```

SECTION 5: SCRIPT DEBUGGING ____

While working with scripts and commands, you may run into errors. These may be due to an error in the script, such as incorrect syntax, or other ingredients such as a missing file or insufficient permission to do an operation. These errors may be reported with a specific error code, but often just yield incorrect or confusing output. How do you got about identifying and fixing an error. Debugging helps you troubleshoot and resolve such errors and is one of the most important tasks a syadmin performs.

Before fixing a bug, it is vital to know its source. In bash shell scripting, you can run a script in debug mode by using `bash -x ./script_file`. Debug mode helps identify the error because: * It traces and prefixes each command with the `+` character. * It displays each command before executing it. * It can debug only selected parts of the script (if desired) with: `bash set -x # turns on debugging ... set +x # turns off debugging`

In UNIX/Linux, all programs that run are given three open file streams when they are started as listed in the followin g table: File stream Description File Descriptor - - - stdin Standard input, by default the keybaord/terminal for programs run from the command line 0 stdout Standard output, by default the screen for programs run from the command line 1 stderr Standard error where output messages are shown or saved 2 using redirection we can save the stdout and stderr output streams to one file or two separate files for later analysis after a program or command is executed. The following is an example of a buggy shell script:

[]

The following is an example output from the script being executed and having the errors redirected to the file `error.txt`.

[]

Using `cat` to display the contents of `error.txt` shows the errors of executing the buggy shell script (for further debugging.)

SECTION 6: SOME ADDITIONAL USEFUL TECHNIQUES

Consider a situation where you want to retrieve 100 records from a file with 10,000 records. You will need a place to store the extracted information, perhaps in a temporary file, while you do further processing on it. Temporary files (and directories) are meant to store data for a short time. Usually one arranges it so that these files disappear when the program using them terminates. While you can also use touch to create a temporary file, this may make it easy for hackers to gain access to your data. The best practice is to create random and unpredictable filenames for temporary storage. One way to do this is with the mktemp utility as in the following examples. The `XXXXXXXX` is replaced by mktemp with random characters to ensure the name of the file cannot be easily predicted and is only known within your program.

Command Usage - - `TEMP=$(mktemp /tmp/tempfile.XXXXXXXX)` To create a temporary file `TEMPDIR=$(mktempt -d /temp/tempdir.XXXXXXXX)` To create a temporary directory

There is also a danger with using symbolic links that can be negated by using mktemp. If someone creates a symbolic link from a known temporary file used by root the the /etc/passwd file like `$ ln -s /etc/passwd /tmp/tmpfile`, there could be a big problem if the script has a line in it like `echo $VAR > /tmp/tempfile`. The password file will be overwritten by the temporary file contents. To prevent such a situation, make sure you randomize your temporary filenames by replacing the above line with the following lines:

[]

Certain commands like find will produce voluminous amounts of output which can overwhelm the console. To avoid this, we can redirect the large output to a special file (a device node) called `/dev/null`. This file is also called the bit bucket or black hole. It discards all data that gets written to it and never returns a failure on write operations. Using the proper redirection operators, it can make the output disappear from commands that would normally generate output to stdout and/or stderr like `$ find / > /dev/null`. In the previous command, the entire standard output stream is ignored, but any errors will still appear on the console.

It is often useful to generate random numbers and other random data when performing tasks such as: * Performing security-related tasks. * Reinitializing storage devices. * Erasing and/or obscuring existing data. * Generating meaningless data to be used for tests.

Such random numbers can be generated by using the `$RANDOM` environment variable which is derived from the Linux kernel's built-in random number generator, or by the OpenSSL library function which uses the FIPS140 algorithm to generate random numbers for encryption. To read more about FIPS140, see http://en.wikipedia.org/wiki/FIPS_140-2. The following example shows you how to easily use the environmental variable method to generate random numbers:

[]

Some servers have hardware random number generators that take as input different types of noise signals, such as thermal noise and photoelectric effect. A transducer converts this noise into an electric signal, which is again converted into a digital number by an A-D converter. This number is considered random. However, most common computers do not contain such specialized hardware and instead rely on events created during booting to create the raw data needed. Regardless of which of these two sources is used, the system maintains a so-called entropy pool of these digital numbers/random bits. Random numbers are created from this entropy pool. The Linux kernel offers the `/dev/random` and `/dev/urandom` device nodes which draw on the entropy pool to provide random numbers which are drawn from the estimated number of bits of noise in the entropy pool. `/dev/random` is used where very high quality randomness is required such as one-time pad or key generation, but it is relatively slow to provide values. `/dev/urandom` is faster and suitable (good enough) for most cryptographic purposes. Furthermore, when the entropy pool is empty, `/dev/random` is blocked and does not generate any number until additional environmental noise (network traffic, mouse movement, etc.) is gathered, whereas `/dev/urandom` reuses the internal pool to produce pseudo-random bits.

SUMMARY

- You can manipulate strings to perform actions such as comparison, sorting, and finding length.
- You can use boolean expressions when working with multiple data types including strings or numbers, as well as files.
- The output of a boolean expression is either true or false.
- Operators used in boolean expressions include `&&` (AND), (OR), and `!` (NOT) operators.
- We looked at the advantages of using the case statement in scenarios where the value of a variable can lead to different execution paths.
- Script debugging methods help troubleshoot and resolve errors.
- The standard and error outputs form a script or shell commands can easily be redirected into the same file or separate files to aid in debugging and saving results.

- Linux allows you to create temporary files and directories which store data for a short duration, both saving space and increasing security.
- Linux provides several different ways of generating random numbers, which are widely used. # Chapter 17: Processes

INTRODUCTION

By the end of this chapter, you should be able to: * Describe what a process is and distinguish between types of processes. * Enumerate process attributes * Manage processes using ps and top * Understand the use of load averages and other process metrics * Manipulate processes by putting them in background and restoring them to foreground * Use at, cron, and sleep to schedule processes for the future or pause them.

SECTION 1: INTRODUCTION TO PROCESSES AND PROCESS ATTRIBUTES

A process is simply an instance of one or more related tasks (threads) executing on your computer. It is not the same as a program or a command; a single program may actually start several processes simultaneously. Some processes are independent of each other and others are related. A failure of one process may or may not affect the others running on the system. Processes use many system resources such as memory, CPU cycles, and peripheral devices such as printers and displays. The operating system (especially the kernel) is responsible for allocating a proper share of these resources to each process and ensuring overall optimum utilization.

A terminal window (one kind of command shell) is a process that runs as long as needed. It allows users to execute programs and access resources in an interactive environment. You can also run programs in the background which means they become detached from the shell. Processes can be different types according to the task being performed. The following table lists some different process types along with their descriptions and examples.

Process Type Description Example - - - Interactive Processes Need to be started by a user, either at a command line or through a graphical interface such as an icon or a menu selection. bash firefox, top Batch processes Automatic processes which are scheduled from and then disconnected from the terminal. These tasks are queued and work on a FIFO (first in first out) basis. updatedb Daemons Server processes that run continuously. Many are launched during system startup and then wait for a user or system request indicating that their service is required. httpd, xinetd, sshd Threads Lightweight processes. These are tasks that run under the umbrella of a main process, sharing memory and other resources, but are scheduled and run by the system on an individual basis. An individual thread can end without terminating the whole process and a process can create new threads at any time. Many non-trivial programs are multi-threaded. gnome-terminal, firefox Kernel Threads Kernel tasks that users neither start nor terminate and have little control over. These may perform actions like moving a thread from one CPU to another or making sure input/output operations to the disk are completed kswapd0, migration, ksoftirqd

When a process is in a so-called running state, it means that it is either currently executing instructions on a CPU or is waiting for a share (or time slice) so it can run. A critical kernel routing called the scheduler constantly shifts processes in and out of the CPU sharing time according to relative priority, how much time is needed, and how much has already been granted to a task. All processes in this state reside on what is called a run queue and on a computer with multiple CPUs or cores, there is a run queue for each. However, sometimes processes go into what is called a sleep state, generally when they are waiting for something to happen before they can resume, perhaps for the user to type something. In this condition, a process is sitting in the wait queue. There are some other less frequent process states, especially when a process is terminating. Sometime, a child process completes but its parent process has not asked about its state. Such a processes is said to be in a zombie state; it is not really alive, but still shows up in the system's list of processes.

At any given time, there are always multiple processes being executed. The operating system keeps track of them by assigning each a unique process ID (PID) number. The PID is used to track process state, CPU usage, memory use, precisely where resources are located in memory, and other characteristics. New PIDs are usually assigned in ascending order as processes are born. PID 1 denotes the init process (initialization process) and succeeding processes are gradually assigned higher numbers. The table below explains the PID types and their descriptions.

ID Type Description - - Process ID (PID) Unique Process ID number. Parent Process ID (PPID) Process (Parent) that started this process. Thread ID (TID) Thread ID number. This is the same as the PID for a single-threaded process. For a multi-threaded process, each thread shares the same PID but has a unique TID.

Many users can access a system simultaneously and each user can run multiple processes. The operating system identifies the user who starts the process by the Real User ID (RUID) assigned to the user. The user who determines the access rights for the users is identified by the Effective UID (EUID). The EUID may or may not be the same as the RUID. Users can be categorized into various groups. Each group is identified by the Real Group ID (RGID). The access rights of the group are determined by the Effective Group ID (EGID). Each user can be a member of one or more groups. Most of the time, we will just talk about the UID.

At any given time, many processes are running (i.e. in the run queue) on the system. However, a CPU can actually accommodate only one task at a time. Some processes are more important than others so Linux allows you to set and manipulate process priority. Higher priority processes are granted more time on the CPU. The priority for a process can be set by specifying a nice value or niceness for the processes. The lower the nice value, the higher the priority. Low values are assigned to important processes while high values are assigned to processes that can wait longer. A process with a high nice value simply allows other processes to be executed first. In Linux, a nice value of -20 represents the highest priority and 19 represents the lowest. (This sounds backwards, but this convention goes back to the earliest days of UNIX.) You can also assign a so-called real-time priority to time-sensitive tasks such as controlling machines through a computer or collecting incoming data. This is just a very high priority and is not to be confused with what is called hard real time which is conceptually different and has more to do with making sure a job gets completed within a very well-defined time window.

SECTION 2: LISTING PROCESSES

---

ps provides information about currently running processes keyed by PID. If you want a repetitive update of this status, you can use top or commonly installed variants such as htop or atop from th command line, or you can invoke your distro's graphical system monitor application. ps has many options for specifying exactly which tasks to examine, what information to display about them, and precisely what output format should be used. Without options, ps will display all processes running under the current shell. you can use the `-u` option to display information of processes for a specified username. The command `ps -ef` displays all the processes in the system in full detail. The command `ps -eLf` goes one step further and displays one line of information for every thread (remember that every process can contain multiple threads.)

ps has another style of option specification which stems from the BSD variety of UNIX where options are specified without preceding dashes. For example, the command `ps aux` displays all processes of all users. The command `ps axo` allows you tot specify with attributes you want to view. The following is sample output of ps with the aux qualifier:

[]

The following is sample output of ps with the axo qualifier:

[]

At some point, one of your applications may stop working properly. pstree displays the processes running on the system in the form of a tree diagram showing the relationship between a process and its parent process and any other processes that it created. Repeated entries of a process are not displayed, and threads are displayed in curly braces. To terminate a process you can use `$ kill -SIGKILL <pid>` or `kill -9 <pid>`. Note that you can only kill your own processes: those belonging to another user are off limits unless you are root.

While a static view of what the system is doing is useful, monitoring the system performance live over time is also valuable. One option would be to run ps at regular intervals. A better alternative is to use the top command to get constant real-time updates (every 2 seconds by default) until you exit by typing 'q'. top clearly highlights which processes are consuming the most CPU cycles and memory (using appropriate commands from within top.)

The first line of the top output displays a quick summary of what is happening in the system including: * How long the system has been up * How many users are logged in * What the load average is

The load average determines how busy the system is. A full load average of 1.00 per CPU indicates a fully subscribed, but not overloaded, system. If the load average goes above this value, it indicates that processes are competing for CPU time. If the load average is very high, it might indicate that the system is having a problem, such as a runaway process (a process in a non-responding state.)

The second line of the top output displays the total number of processes, the number of running, sleeping, stopped, and zombie processes. Comparing the number of running processes with the load average helps determine if the system has reached its capacity or perhaps a particular user is running too many processes. The stopped processes should be examined to see if everything is running correctly.

The third line of output indicates how the CPU time is being divided between the users (us) and the kernel (sy) by displaying the percentage of CPU time used for each. The percentage of user jobs running at a lower priority (niceness -ni) is then listed. Idle mode (id) should be low if the load average is high, and vice versa. The percentage of jobs waiting (wa) for I/O is listed. Interrupts include the percentage of hardware (hi) vs software (si) interrupts. Steal time (st) is generally used with virtual machines, which has some of its idle CPU time taken for other uses. The fourth and fifth lines of the top output indicate memory usage which is divided into two categories: * Physical memory (RAM) - displayed on line 4. * Swap space - displayed on line 5.

Both categories display total memory, used memory, and free space. You need to monitor memory usage very carefully to ensure good system performance. Once the physical memory is exhausted, the system starts using swap space (temporary storage space on the hard drive) as an extended memory pool, and since accessing disk is much slower than accessing memory, this will negatively affect system performance. If the system starts using swap often, you can add more swap space. However, adding more physical memory should also be considered.

Each line in the process list of the top output displays information about a process. By default, processes are ordered by highest CPU usage. The following information about each process is displayed: * Process Identification Number (PID) * Process owner (USER) * Priority (PR) and nice values (NI) * Virtual (VIRT), physical (RES), and shared memory (SHR) * Status (S) * Percentage of CPU (%CPU) and memory (%MEM) used * Execution time (TIME+) * Command (COMMAND)

Besides reporting information, top can be utilized interactively for monitoring and controlling processes. While top is running in a terminal window, you can enter single-letter commands to change its behaviour. For example, you can view the top-ranked processes based on CPU or memory usage. If needed, you can alter the priorities of running processes or you can stop/kill a process. The following table lists what happens when pressing various keys when running top:

Command output `- -` `t` Display or hide information (rows 2 and 3) `m` Display or hide memory information (rows 4 and 5) `A` Sort the process list by top resource consumers `r` Renice (change the priority of) a specific process `k` Kill a specific process `f` Enter the top configuration screen `o` Interactively select a new sort order in the process list

SECTION 3: PROCESS METRICS AND PROCESS CONTROL

Load average is the average of the load number for a given period of time. It takes into account processes that are: * Actively running on a CPU. * Considered runnable, but waiting for a CPU to become available. * Sleeping: i.e. waiting for some kind of resource (typically I/O) to become available.

The load average can obtained by running w, top, or uptime.

The load average is displayed using three sets of numbers. The last piece of information is the average load of the system. Assuming our system is a single-CPU, the 0.25 means that for the past minute, on average, the system has been 25% utilized. 0.12 in the next position means that over the past 5 minutes, on average, the system has been 12% utilized. 0.15 in the final position means that over the past 15 minutes, on average, the system has been 15% utilized. If we saw a value of 1.00 in the second position, that would imply that the single-CPU system was 100% utilized, on average, over the past 5 minutes; this is good if want to fully use a system. A value over 1.00 for a single-CPU system implies that the system was over-utilized: there were more processes needing CPU than CPU was available. If we had more than one CPU, say a quad-CPU system, we would divide the load average numbers by the number of CPUs. In this case, for example, seeing a 1 minute load average of 4.00 implies that the whole system was 100% (4.00/4) utilized during the last minute. Short term increases are not usually a problem. A high peak you see is likely a burst of activity, not a new level. For example, at start up, many processes start and then activity settles down. If a high peak is seen in the 5 and 15 minute load averages, it may be cause for concern.

Linux supports background and foreground job processing. A job in this context is just a command launched from a terminal window. Foreground jobs run directly from the shell, and when one foreground job is running, other jobs need to wait for shell access (at least in that terminal window if using a GUI) until it is completed. This is fine when jobs complete quickly, but this can have an adverse effect if the current job is going to take a long time (even several hours) to complete.

In such cases, you can run the job in the background and free the shell for other tasks. The background job will be executed at lower priority which, in turn, will allow smooth execution of the interactive task and allow you to type other commands in the terminal window while the background job is running. By default, all jobs are executed in the foreground. You can put a job in the background by suffixing `&` to the command like `$ updatedb &`. You can either use `CTRL+Z` to suspend a foreground job, or `CTRL+C` to terminate a foreground job, and you can always use the bg and fg commands to run a process in the background and foreground respectively.

The jobs utility displays all jobs running in the background. The display shows the job ID, state, and command name as shown in the following:

[]

`jobs -l` provides the same information as `jobs` but also includes the PID of the background jobs. The background jobs are connected to the terminal window, so if you log off, the jobs utility will not show the ones started from the window.

SECTION 4: STARTING PROCESSES IN THE FUTURE ____

Suppose you need to perform a task on a specific day sometime in the future but you know you will be away from the machine that day. You can use the at utility program to execute any non-interactive command at a specified time as evidenced in the following example:

[]

cron is a time-based scheduling utility program. It can launch routine background jobs at specific times and/or days on an on-going basis. cron is driven by a configuration file called `/etc/crontab` (cron table) which contains the various shell commands that need to be run at the properly scheduled times. There are both system-wide crontab files and individual user-based ones. Each line of a crontab file represents a job and is composed of a so-called CRON expression followed by a shell command to execute. The crontab -e command will open the crontab editor to edit existing jobs or to create new jobs. Each line of the crontab file will contain 6 fields as detailed in the following table:

Field Description Values - - - MIN Minutes 0 to 59 HOUR Hours 0 to 23 DOM Day of Month 1 to 31 MON Month 1 to 12 DOW Days of Week 0 to 6 (6 is Sunday) CMD Command Any command to be executed

Examples: * The entry “” * * * * * /usr/local/bin/execute/this/script.sh " will schedule a job to execute “script.sh” every minutes of every hour of every day of the month, and every month and every day of the week. * The entry “30 08 10 06 * /home/sysadmin/full-backup” will schedule a full-backup at 8:30AM on the 10th of June irrespective of the day of the week.

Sometimes a command or job must be delayed or suspended. Suppose an application has read and processed the contents of a data file and then needs to save a report on a backup system. If the backup system is currently busy or not available, the application can be made to sleep (wait) until it can complete its work. Such a light delay might be to mount the backup device and prepare it for writing. sleep suspends execution for at least the specified period of time which can be given as the number of seconds (default), minutes, hours, or days. After that time has passed (or an interrupting signal has bee received) execution will resume. The syntax is `sleep NUMBER[SUFFIX] . . .` where SUFFIX may be: * `s` for seconds (default) * `m` for minutes * `h` for hours * `d` for days

sleep and at very different; sleep delays execution for a specific period while at starts execution at a later time.

SUMMARY

- Processes are used to perform various tasks on the system.
- Processes can be single-threaded or multi-threaded.
- Processes can be of different types such as interactive and non-interactive.
- Every process has a unique identifier (PID) to enable the operating system to keep track of it.
- The nice value or niceness can be used to set priority.
- ps provides information about the currently running processes.
- You can use top to get constant real-time updates about overall system performance as well as information about the processes running on the system.
- Load average indicates the amount of utilization the system is under at particular times.

- Linux supports background and foreground processing for a job.
- at executes any non-interactive command at a specified time.
- cron is used to schedule tasks that need to be performed at regular intervals. # Chapter 18: Common Applications

INTRODUCTION

By the end of this chapter, you should be familiar with common Linux applications including: * Internet applications such as browsers and email programs. * Office Productivity Suites such as LibreOffice. * Developer tools such as compilers, debuggers, etc. * Multimedia applications such as those for audio and video. * Graphics editors such as GIMP and other graphics utilities.

SECTION 1: INTERNET APPLICATIONS

The internet is a global network that allows users around the world to perform multiple tasks such as searching for data, communication through emails, and online shopping. Obviously, you need to use network-aware applications to take advantage of the intert. These include: * Web browsers * Email clients * Online media applications * Other applications

Linux offers a wide variety of web browsers, both graphical and text based including: * Firefox * Google Chrome * Chromium * Epiphany * Konqueror * w3m * lynx

Email applications allow for sending, receiving, and reading messages over the internet. Linux systems offer a wide number of email clients, both graphical and text-based. In addition, many users simply use their browsers to access their email accounts. Most email clients use the Internet Message Access Protocol (IMAP) or the older Post Office Protocol (POP) to access emails stored on a remote mail server. Most email applications also display HyperText Markup Language (HTML) formatted emails that display objects such as pictures and hyperlinks. the features of advanced email applications include the ability of importing address book/contatct lists, configuration information, and emails form other applications. Linux supports the following types of email applications: * Graphical email clients such as Thunderbird (produced by Mozilla), Evolution, and Claws Mail. * Text mode email clients such as mutt and mail.

Linux systems provide many other applications for performing internet-related tasks. These include the applications on the following table:

Application Use - - FileZilla Intuitive graphical FTP client that supports FTP, Secure File Transfer Protocol (SFTP) and FTP Secured (FTPS). Used to transfer files to/from FTP servers. Pidgin To access GTalk, AIM, ICQ, MSN, IRC, and other messaging networks. Ekiga To connect to Voice over Internet Protocol (VoIP) networks. Hexchat To access Internet Relay Chat (IRC) networks.

SECTION 2: PRODUCTIVITY AND DEVELOPMENT APPLICATIONS

Most day-to-day computer systems have productivity applications (sometimes called office suites) available or installed. Each suite is a collection of closely coupled programs used to create and edit different kinds of file such as: * Text (articles, book, reports, etc.) * Spreadsheets * Presentations * Graphical objects

Most Linux distros offer LibreOffice, an open source office suit that started in 2010 and has evolved from OpenOffice.org. While other office suites are available as listed, LibreOffice is the most mature, widely used, and intensely developed. The component applications included in LibreOffice are:

Component of LibreOffice Usage - - Writer Word processing Calc Spreadsheets Impress Presentations Draw Create and edit graphics and diagrams

Linux distros come with a complete set of applications and tools that are needed by those developing or maintaining both user applications and the kernel itself. These tools are tightly integrated and include: * Advanced editors customized for programmers' needs such as vi and emacs. * Compilers (such as gcc for programs in C and C++) for every computer language * Debuggers such as gdb and various graphical front ends to it and many other debugging tools (such as valgrind.) * Performance measuring and monitoring programs, some with easy to use graphical interfaces, others more arcane and meant to be used only by serious experienced development engineers. * Complete Integrated Development Environments (IDEs) such as Eclipse that put all these tools together.

On other operating systems, these tool have to be maintained and installed separately, often at high cost, while on Linux they are all available at no cost through the standard package installation system.

SECTION 3: MULTIMEDIA APPLICATIONS

Multimedia applications are used to listen to music, view videos, etc., as well as to present and view text and graphics. Linux systems offer a number of sound player applications including:

Application Use - - Amarok Mature MP3 player with a graphical interface that plays audio and video files and streams (online audio files.) It allows you to create a playlist that contains a group of songs and uses a database to store information about the music collection. Audacity Used to record and edit sounds and can be quickly installed through a packager manager. Audacity has a simple interface to get started. Rhythmbox Supports a large variety of digital music sources including streaming Internet audio and podcasts. The application also enables search of particular audio in a library. It supports "smart playlists" with an automatic update feature which can revise playlists based on specified selection criteria.

Of course, Linux systems also can connect with commercial online music streaming services such as Pandora and Spotify through web browsers.

Movie players can portray input from many different sources either local to the machine or on the internet. Linux systems offer a number of movie players including: * VLC * MPlayer * Xine * Totem

Movie editors are used to edit videos or movies. Linux offers a number of movie editors including:

Application Use - - Kino Acquire and edit camera streams. Kino can merge and separate video clips. Cinepaint Frame-by-frame retouching. Cinepaint is used for editing images in a video. Blender Create 3D animation and design. Blender is a professional tool that uses modeling as a starting point. there are complex and powerful tools for camera capture, recording, editing, enhancing, and creating video, each having its own focus. Cinelerra Capture, compose, and edit audio/video. FFmpeg Record, convert, and stream audio/video. FFmpeg is a format converter, among other things, and has other tools such as ffplay and ffserver.

SECTION 4: GRAPHICS EDITORS AND UTILITIES

Graphics editors allow you to create, edit, view, and organize images of various formats like Joint Photographic Experts Group (JPEG or JPG), Portable Network Graphics (PNG), Graphics Interchange Format (GIF), and Tagged Image File Format (TIFF). GIMP (GNU Image Manipulation Program) is a feature-rich image retouching and editing tool similar to Adobe Photoshop and is available on all Linux distros. Some features of GIMP are: * It can handle any image file format. * It has many special purpose plugins and filters. * It provides extensive information about the image, such as layers, channels, and histograms.

In addition to GIMP, there are other graphics utilities that help perform various image-relted tasks including:

Graphics Utility Use - - eog Eye of Gnome (eog) is an image viewer that provides slid show capability and a few image editing tools such as rotate and resize. It can also step through the images in a directory with just a click. Inkscape Inkscape is an image editor with lots of editing features. It works with layers and transformations of the image. It is sometimes compared to Adobe Illustrator convert convert is a command line tool (part of the ImageMagick set of applications) that can modify image files in many ways. The options include file format conversion and numerous image modification options such as blur, resize, despeckle, etc. Scribus Scribus is used for creating documents used for publishing and providing a What You See Is What You Get (WYSIWYG) environment. It also provides numerous editing tools.

SUMMARY

- Linux offers a wide variety of Internet applications such as web browsers, email clients, online media applications, and others.
- Web browsers supported by Linux can either be graphical or text-based such as Firefox, Google Chrome, Epiphany, w3m, lynx, and others.
- Linux supports graphical email clients such as Thunderbird, Evolution, and Claws Mail along with text mode email clients such as mutt and mail.
- Linux systems provide many other applications for performing Internet-related tasks such as FileZilla XChat, Pidgin, and others.
- Most Linux distros offer LibreOffice to create and edit different kinds of documents.
- Linux systems offer entire suites of development applications and tools including compilers and debuggers.
- Linux systems offer a number of sound players including Amarok, Audacity, and Rhythmbox.
- Linux systems offer a number of movie players including VLC, MPlayer, Xine, and Totem.
- Linux systems offer a number of movie editors including Kino, Cinepaint, and Blender among others.
- The GIMP (GNU Image Manipulation Program) utility is a feature-rich image retouching and editing tool available on all Linux distros.
- Other grahpics utilities that help perform various image-related tasks are eog, Inkscape, convert, and Scribus.