MCA(S5)20

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Housefed Complex, Dispur, Guwahati - 781 006

Master of Computer Applications

LINUX SYSTEM ADMINISTRATION

CONTENTS

Subject Expert

Prof. Anjana Kakati Mahanta, Deptt. of Computer Science, Gauhati University

Prof. Jatindra Kr. Deka, Deptt. of Computer Science and Engineering,
Indian Institute of Technology, Guwahati

Prof. Diganta Goswami, Deptt. of Computer Science and Engineering,
Indian Institute of Technology, Guwahati

Course Coordinator

Tapashi Kashyap Das, Assistant Professor, Computer Science, KKHSOU

Arabinda Saikia, Assistant Professor, Computer Science, KKHSOU

SLM Preparation Team

Units	Contributor
1 , 2	**Pritam Medhi** Research Scholar, Deptt. of Computer Science, Gauhati University
3, 4, 5	**Nanu Alan Kachari** Scientific Officer, Deptt. of Computer Science and Engineering, Indian Institute of Technology Guwahati

July 2013

© Krishna Kanta Handiqui State Open University

No part of this publication which is material protected by this copyright notice may be produced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the KKHSOU.

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

The university acknowledges with thanks the financial support provided by the **Distance Education Council, New Delhi**, for the preparation of this study material.

Housefed Complex, Dispur, Guwahati- 781006; Web: www.kkhsou.net

COURSE INTRODUCTION

This is a course on "***Linux System Administration"***. This course contains five essential units. *Unit 1* is an introductory unit on Linux operating system. With this unit, the learners will be acquainted with some important concepts like open source, free software, various Linux distribution, basic architecture of Linux etc. This unit also discusses various installation process of Linux operating system. *Unit 2* discusses the basics of Linux. This includes the login process, process of creating user account and group, operation with files, and some important Linux commands. The most widely used shell script is introduced in *Unit 3*. *Unit 4* concentrates on System Administration. Unit 5 is the last unit and it discusses Linux Networking. The learners will learn how to install and configure a simple LAN. This unit also discusses installation and congiguration of proxy server, web server, file server, Samba server, SSH server etc.

While going through a unit, you will notice some boxes along-side, which have been included to help you know some of the difficult, unseen terms. Some "ACTIVITY' (s) have been included to help you apply your own thoughts. Again, we have included some relevant concepts in "LET US KNOW" along with the text. And, at the end of each section, you will get "CHECK YOUR PROGRESS" questions. These have been designed to self-check your progress of study. It will be better if you solve the given problems in these boxes immediately, after you finish reading the section in which these questions occur and then match your answers with "ANSWERS TO CHECK YOUR PROGRESS" given at the end of each unit.

MASTER OF COMPUTER APPLICATIONS

Linux System Administration

DETAILED SYLLABUS

Unit 1: Introduction to Linux

Basic idea on Proprietary, Open Source, Free Software etc., Introduction of Various Linux Distribution (Red Hat Enterprise Linux, Cent OS, Fedora Projects, Debian Linux, Ubuntu, etc.); Basic Architecture of Unix/Linux system, Kernel, Shell. Linux File System, Boot block, Super block, Inode table, Data blocks, How Linux access files, storage files, Linux standard directories, LILO, GRUB Boot Loader; Installation of Linux system- Using Live CD, Virtual Machine, Direct Installation; Partitioning the Hard drive for Linux, init and run levels;

Unit 2: Linux Basics

Getting Started: Login process, Creating Users Account and Group, Getting Help. Services and Process, Files and File System (File Types and Permissions, Links, Size and Space, Date and Time) Working with Files: Reading Files, Searching for files, Copying, Moving, Renaming, Deleting, Linking, and Editing Files; Other Commands: ls, rm, rmdir, pwd, more ,less. grep, awt, sort, cat, head, tail, wc, tee, ps, top, tar, unzip, nice, kill, netstat, Disk related commands, checking disk free spaces.

Unit 3: Linux Shell Script

Various types of Shell available in Linux, Comparisons between various Shells, Shell programming in bash, read command, conditional and looping statements, case statements, parameter passing and arguments, Shell variables, system shell variables, shell keywords, creating Shell programs for performing various tasks.

Unit 4: System Administration

System Administration Common administrative tasks, identifying administrative files – configuration and log files, Role of system administrator, Managing user accounts-adding & deleting users, changing permissions and ownerships, Creating and managing groups, modifying group attributes, Temporary disable user's accounts, creating and mounting file system, checking and monitoring system performance file security & Permissions, becoming super user using su; Getting system information with uname, host name, disk partitions & sizes, users, kernel. Backup and restore files, reconfiguration hardware with kudzu, installing and removing packages in Linux. Configure X-windows starting & using X desktop. KDE & Gnome graphical interfaces, changing X windows settings.

Unit 5: Linux Networking

Installation and configuration of a simple LAN; Installation and configuration of: Proxy server(Squid), DNS server(BIND), Mail server, Web server(Apache), File server(Samba), DHCP server; Installation and configuration of a SSH server and cliant; Installation and configuration of FTP server and cliant;

UNIT - 1: INTRODUCTION TO LINUX

UNIT STRUCTURE

1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- know the basics of the linux operating system
- know the different distribution of linux
- study the architecture of the linux operating system
- know the linux file system
- learn about the linux system installation
- study partitioning for the linux system

1.2 INTRODUCTION

Linux is a Unix-like computer operating system assembled under the model of free and open source software development and distribution. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991, by Linus Torvalds. Since the C compiler that builds Linux and the main supporting user space system tools and libraries originated in the GNU Project, initiated in 1983 by Richard Stallman, the Free Software Foundation prefers the name GNU/Linux.

Linux was originally developed as a free operating system for Intel x86-based personal computers. It has since been ported to more computer hardware platforms than any other operating system. It is a leading operating system on servers and other big iron systems such as mainframe computers and supercomputers: more than 90% of today's 500 fastest supercomputers run some variant of Linux, including the 10 fastest. Linux also runs on embedded systems (devices where the operating system is typically built into the firmware and highly tailored to the system) such as mobile phones, tablet computers, network routers, building automation controls, televisions and video game consoles; the Android system in wide use on mobile devices is built on the Linux kernel.

This unit provides the introductory topics to the Linux operating system, its architecture, file system, directory structure, its installation etc.

1.3 PROPRIETARY, OPEN SOURCE, AND FREE SOFTWARE

Proprietary software –

Proprietary software or closed source software is computer software licensed under exclusive legal right of the copyright holder with the intent that the licensee is given the right to use the software only under certain conditions, and restricted from other uses, such as modification, sharing, studying, redistribution, or reverse engineering. The owner of proprietary software exercises certain exclusive rights over the software. The owner can restrict use, inspection of source code, modification of source code, and redistribution. Vendors typically limit the number of computers on which software can be used, and prohibit the user from installing the software on extra computers. Restricted use is sometimes enforced through a technical measure, such as product activation, a product key or serial number, a hardware key, or copy protection. Vendors may also distribute versions that remove particular features, or versions which allow only certain fields of endeavor, such as non-commercial, educational, or non-profit use.

Proprietary software vendors can prohibit users from sharing the software with others. Another unique license is required for another party to use the software. In the case of proprietary software with source code available, the vendor may also prohibit customers from distributing their modifications to the source code. Well known examples of proprietary software include Microsoft Windows, Adobe Flash Player, PS3 OS, iTunes, Adobe Photoshop, Google Earth, Mac OS X, Skype, WinRAR, and some versions of Unix.

Open source software –

Open-source software (OSS) is computer software with its source code made available and licensed with an open-source license in

which the copyright holder provides the rights to study, change and distribute the software for free to anyone and for any purpose. Open-source software is very often developed in a public, collaborative manner. Open-source software is the most prominent example of open-source development and often compared to (technically defined) user-generated content or (legally defined) open-content movements. The Open Source Definition, notably, presents an open-source philosophy, and further defines the terms of usage, modification and redistribution of open-source software. Software licenses grant rights to users which would otherwise be reserved by copyright law to the copyright holder. Several open-source software licenses have qualified within the boundaries of the Open Source Definition. The most prominent and popular example is the GNU General Public License (GPL), which "allows free distribution under the condition that further developments and applications are put under the same license", thus also free. While open-source distribution presents a way to make the source code of a product publicly accessible, the open-source licenses allow the authors to fine tune such access.

Open source software projects are built and maintained by a network of volunteer programmers. Prime examples of open-source products are the Apache HTTP Server, the e-commerce platform osCommerce and the internet browser Mozilla Firefox. One of the most successful open-source products is the GNU/Linux operating system, an open-source Unix-like operating system, and its derivative Android, an operating system for mobile devices.

Free software –

Free software is software provided under terms that guarantee the freedoms of its users (individually and in groups) to run it, adapt it to their needs, and redistribute it with or without changes. These freedoms are protected by granting broad permission to make use of the source code, either alone or in cooperation with other people of the user's choice. Users of free software are free in these activities, because they do not need to ask for any permission; and they are not restricted in activities through restrictive proprietary licenses (e.g. copy-restriction), or requirements of having to agree to restrictive terms of others (e.g. non-disclosure agreements), and they are not already restricted from the outset (e.g. through deliberate non-availability of source code).

The goals of Free Software (control in one's own computing and free cooperation) are reached by granting the following freedom-rights: users are free to run, copy, distribute, study, change and improve the software; these freedoms are explicitly granted and not suppressed (as is the case with proprietary software). Thus, free software is a matter of liberty, not price (users are free – which includes the freedom to redistribute the software, which can be done free or for a fee). Free software guarantees user's freedoms: to study and modify software, by the availability of the source code; as well as freedom to copy, and distribute.

1.4 LINUX DISTRIBUTIONS

A Linux distribution is a collection of (usually open source) software on top of a Linux kernel. A distribution can bundle server software, system management tools, documentation and many desktop applications in a central secure software repository. A distribution aims to provide a common look and feel, secure and easy software management and often a specific operational purpose.

Some of the popular Linux distributions are discussed below –

Red Hat Enterprise Linux –

Assembled by the company Red Hat, Red Hat Linux was a popular Linux based operating system until its discontinuation in 2004. Red Hat Linux 1.0 was released on November 3, 1994. It was originally called "Red Hat Commercial Linux". It was the first Linux distribution to use the *RPM Package Manager* as its packaging format, and over time has served as the starting point for several other distributions, such as Mandriva Linux and Yellow Dog Linux. Since 2003, Red Hat discontinued the Red Hat Linux line in favor of *Red Hat Enterprise Linux* (RHEL) for enterprise environments. Fedora, developed by the community-supported Fedora Project and sponsored by Red Hat, is the free version best suited for home use. **Red Hat Enterprise Linux** (**RHEL**) is a Linux-based operating system developed by Red Hat and targeted toward the commercial market. Red Hat Enterprise Linux is released in server versions for x86, x86-64, Itanium, PowerPC and IBM System z, and desktop versions for x86 and x86-64. All of Red Hat's official support and training and the Red Hat Certification Program center around the Red Hat Enterprise Linux platform. Red Hat Enterprise Linux is often abbreviated to RHEL, although this is not an official designation.

The first version of Red Hat Enterprise Linux to bear the name originally came onto the market as "Red Hat Linux Advanced Server". In 2003 Red Hat rebranded Red Hat Linux Advanced Server to "Red Hat Enterprise Linux AS", and added two more variants, Red Hat Enterprise Linux ES and Red Hat Enterprise Linux WS.

While Red Hat uses strict trademark rules to restrict free re-distribution of their officially supported versions of Red Hat Enterprise Linux, Red Hat freely provides the source code for the distribution's software even for software where this is not mandatory. As a result, several distributors have created re-branded and/or community-supported re-builds of Red Hat Enterprise Linux that can legally be made available, without official support from Red Hat. CentOS and Oracle Linux aim to provide 100% binary compatibility with Red Hat Enterprise Linux.

Cent OS –

CentOS (**C**ommunity **ENT**erprise Operating System) is a Linux distribution which attempts to provide a free enterprise class computing platform which has 100% binary compatibility with its upstream source, Red Hat Enterprise Linux (RHEL). In June 2006, David Parsley, the primary developer of Tao Linux, another Red Hat Enterprise Linux (RHEL) clone, announced that it would be retired and rolled into CentOS development. Tao users migrated to the CentOS release via "yum update". In July 2010, CentOS overtook Debian to become the most popular Linux distribution for web servers, with almost 30% of all Linux web servers using it, although Debian retook the lead in January 2012. CentOS version numbers have two parts, a major version and a minor version, which correspond to the major version and update set of Red Hat Enterprise Linux that was used to build that version of CentOS. For example, CentOS 4.7 is built from the source packages from RHEL 4 update 7.

Fedora Projects –

The Fedora Project was created in late 2003, when Red Hat Linux was discontinued. Red Hat Enterprise Linux was to be Red Hat's only officially supported Linux distribution, while Fedora was to be a community distribution. Red Hat Enterprise Linux branches its releases from versions of Fedora. The name of Fedora derives from Fedora Linux, a volunteer project that provided extra software for the Red Hat Linux distribution, and from the characteristic fedora used in Red Hat's "Shadowman" logo. Warren Togami began Fedora Linux in 2002 as an undergraduate project, intended to provide a single repository for well-tested third-party software packages so that non-Red Hat software would be easier to find, develop, and use. The key difference between the approaches of Fedora Linux and Red Hat Linux was that Fedora's repository development would be collaborative with the global volunteer community.../../KKHSOU/Introduction-to-linux/Fedora (operating system) - Wikipedia, the free encyclopedia.htm - cite_note-12 Fedora Linux was eventually absorbed into the Fedora Project, carrying with it this collaborative approach. Fedora is a trademark of Red Hat, and although this had previously been disputed by the creators of the unrelated Fedora repository management software, the issue has now been resolved.

The Fedora Project distributes Fedora in several different ways:

- **Fedora DVD/CD set** – a DVD or CD set of all major Fedora packages at time of shipping.
- **Live images** – CD or DVD sized images that can be used to create a Live CD or boot from a USB flash drive and optionally install to a hard disk
- **Minimal CD** – used for installing over HTTP, FTP or NFS.

The Fedora Project also distributes custom variations of Fedora which are called Fedora spins. These are built from a specific set of software packages and have a combination of software to meet the requirements of a specific kind of end user. Fedora spins are

developed by several Fedora special interest groups. It is also possible to create Live USB versions of Fedora using **Fedora Live USB creator**, **UNetbootin** or **dd**. Extra Packages for Enterprise Linux (EPEL) is a volunteer-based community effort from the Fedora project to create a repository of high-quality add-on packages that complement the Fedora-based Red Hat Enterprise Linux and its compatible spinoffs such as CentOS or Scientific Linux. Software package management is primarily handled by the *yum* utility. Graphical interfaces, such as **pirut** and **pup** are provided, as well as **puplet**, which provides visual notifications in the panel when updates are available. **apt-rpm** is an alternative to yum, and may be more familiar to people used to Debian or Debian-based distributions, where Advanced Packaging Tool is used to manage packages. Additionally, extra repositories can be added to the system, so that packages not available in Fedora can be installed

Fedora comes installed with a wide range of software that includes LibreOffice, Firefox, and Empathy. Additional software that is not installed by default can be downloaded using the package manager. Before Fedora 7, there were two main repositories – Core and Extras. Fedora Core contained all the base packages that were required by the operating system, as well as other packages that were distributed along with the installation CD/DVDs, and was maintained only by Red Hat developers. Fedora Extras, the secondary repository that was included from Fedora Core 3, was community-maintained and not distributed along with the installation CD/DVDs. Also prior to Fedora 7 being released, there was a third repository called Fedora Legacy. This repository was community-maintained and was mainly concerned with extending the life cycle of older Fedora Core distributions and selected Red Hat Linux releases that were no longer officially maintained. Fedora Legacy was shut down in December 2006.

The default desktop in Fedora is the GNOME desktop environment, with Fedora offering the GNOME Shell as its default interface since the release of Fedora 15. Other desktop environments are available from the Fedora package repositories, and can also be installed from the Fedora installer, including the KDE Plasma Workspaces, Xfce, and LXDE desktop environments. In Fedora 18 both the MATE and Cinnamon desktops were made available in the package repositories. In addition, specialized "spins" are available offering these alternative desktops custom configured and offered by default.

Security is one of the most important features in Fedora. One of the security features in Fedora is Security-Enhanced Linux, a Linux feature that implements a variety of security policies, including mandatory access controls, through the use of Linux Security Modules (LSM) in the Linux kernel. Fedora is one of the distributions leading the way with SELinux. SELinux was introduced in Fedora Core 2. It was disabled by default, as it radically altered how the operating system worked, but was enabled by default in Fedora Core 3 and introduced a less strict, targeted policy.

Debian linux –

Debian is an operating system developed by the Debian project which is composed of free software mostly carrying the GNU General Public License. Debian currently uses the Linux kernel or the kernel from FreeBSD operating system, whereas the large part of its most basic tools is made up of free software from the GNU project, hence the names Debian GNU/Linux and Debian GNU/kFreeBSD. Debian includes non-free software, however it is placed outside of its official repositories to comply with its guidelines of providing free software. Debian GNU/Linux is one of the most popular Linux distributions for personal and Internet server machines. The vitality the Debian project plays in open source is demonstrated in pursuit of advancing development and security patches in relation to its strong participation of CVE compatibility efforts.

Debian is one of the most influential open source projects known as a Linux distribution, and maintains repositories with over 37,500 software packages ready for installation. Its repositories host large numbers of software packages for multiple architectures, more in number than any other Linux distribution project. Debian hosts additional repositories called "non-free" but offers its distribution setup without it. Debian includes popular programs such as LibreOffice, Iceweasel (a rebranding of Firefox), Evolution mail, CD/DVD writing programs, music and video players, image viewers and editors, and PDF viewers. Debian's new form of installation-from-USB has been supported since its sixth edition. Debian supports this capability inside it's first iso-file of any of its install media sets (whether CD or DVD) and does not require the help of 'extraction tools' such as unetbootin. This new feature is called Hybrid iso in which an .iso file is dumped to USB. Debian is one of the few Linux distributions offering this feature with their install iso media, and other distributions are starting to adopt this feature.

Debian offers stable and testing CD images specifically built for GNOME (the default), KDE Plasma Workspaces, Xfce and LXDE. Less common window managers such as Enlightenment, Openbox, Fluxbox, GNUstep, IceWM, Window Maker and others can also be installed. It was previously suggested that the default desktop environment of version 7.0 "Wheezy" may be switched to Xfce, because GNOME 3 might not fit on the first CD of the set. The Debian Installer team announced that the first CD includes GNOME thanks to their efforts to minimize the amount of disc space GNOME takes up. A Debian Live system is a version of Debian that can be booted directly from removable media (CDs, DVDs, USB keys) or via network booting without having to install it on the hard drive. This allows the user to try out Debian before installing it or use it as a boot-disk. There are prebuilt Debian Live images for rescue, standard, GNOME, KDE Plasma Workspaces, Xfce and LXDE for several architectures. A hard disk installation can be achieved using the Debian Installer included in the live

image. Most of the live ISO images for the current release no longer fit on a 700 MB CD. Customized CD images can be built using live-build.[30] Live-build can not only generate CD Images, but also bootable DVDs, images for USB thumb drives, or netboot images. Live-magic is a GUI for live-build.

Ubuntu linux –

Ubuntu is a computer operating system based on the Debian Linux distribution and distributed as free and open source software, using its own desktop environment. It is named after the Southern African philosophy of ubuntu, which can be translated as "humanity towards others" or "the belief in a universal bond of sharing that connects all humanity". As of 2012, according to online surveys, Ubuntu is the most popular Linux-based operating system on desktop/laptop personal computers, and most Ubuntu coverage focuses on its use in that market. However, it is also popular on servers and for cloud computing.

Development of Ubuntu is led by Canonical Ltd., a UK-based company owned by South African entrepreneur Mark Shuttleworth. Canonical generates revenue through the sale of technical support and services related to Ubuntu. According to Canonical, the Ubuntu project is committed to the principles of open source development; people are encouraged to use free software, improve it, and distribute it. Ubuntu is composed of many software packages, the majority of which are distributed under a free software license. The main license used is the GNU General Public License (GNU GPL) which, along with the GNU Lesser General Public License (GNU LGPL), explicitly declares that users are free to run, copy, distribute, study, change, develop and improve the software. On the other hand, there is also proprietary software available that can run on Ubuntu. The Ubiquity installer allows Ubuntu to be installed to the hard disk from within the Live CD environment, without the need for restarting the computer prior to installation. Beginning with 5.04, UTF-8 became the default character encoding, which allows for support of a variety of non-Roman scripts.

The system requirements vary among Ubuntu products. For the main Ubuntu desktop product, the official Ubuntu Documentation recommends a 1 GHz Pentium 4 processor with 512 megabytes of RAM and 5 gigabytes of hard drive space, or better. For less powerful computers, there are other Ubuntu distributions such as Lubuntu and Xubuntu. As of version 12.04, Ubuntu supports the ARM and x86 (32 bit and 64 bit) architectures. Installation of Ubuntu is generally performed with the Live CD or a Live USB drive. The Ubuntu OS can run directly from the CD (although this is usually slower than running Ubuntu from an HDD), allowing a user to "test-drive" the OS for hardware compatibility and driver support. The CD also contains the Ubiquity installer, which can then guide the user through the permanent installation process. CD images of all current and past versions are available for download at the Ubuntu web site. Installing from the CD requires a minimum of 256 MB of RAM.

Gentoo linux –

Gentoo Linux is a computer operating system built on top of the Linux kernel and based on the **Portage** package management system. It is distributed as free and open source software. Unlike a conventional software distribution, the user compiles the source code locally according to their chosen configuration. Where source code is available, Portage normally supplies no precompiled binaries, continuing in the tradition of the ports collection, although for convenience, some large packages (such as Mozilla Firefox and LibreOffice) are also available as precompiled binaries for various architectures where compiling would otherwise be very time consuming. The development project and its products are named after the fastest-swimming penguin, the Gentoo, to reflect the potential speed improvements of machine-specific optimization. Gentoo package management is designed to be modular, portable, easy to maintain, flexible, and optimized for the user's machine. Gentoo describes itself as a meta-distribution, "because of its near-unlimited adaptability", in that the majority of users have configurations and sets of installed programs which are unique to themselves.

Gentoo Linux was initially created by Daniel Robbins as the *Enoch Linux* distribution. The goal was to create a distribution without precompiled binaries that was tuned to the hardware and only included required programs. Although originally built on the x86 architecture, Gentoo has been ported to many others. Currently it is officially supported and considered stable on x86, x86-64, IA-64, PA-RISC, PowerPC, PowerPC 970, SPARC 64-bit and DEC Alpha architectures.

Gentoo may be installed in several ways. The most common way is to use the Gentoo minimal CD with a stage 3 tarball. As with many Linux distributions, Gentoo may be installed from almost any Linux environment, such as another Linux distribution's LiveCD, LiveUSB or Network Booting using the "Gentoo Alternate Install Guide". A normal install requires a connection to the Internet, but there is also a guide for a network-less install. Previously, Gentoo supported installation from stage 1 and 2 tarballs. However, this is no longer recommended officially by the Gentoo foundation, and is meant only for Gentoo developers. Following the initial install steps, the Gentoo Linux install process requires that all users compile their own Linux kernel. This process is generally not required by other Linux distributions. Although this is widely regarded as a complex task, Gentoo provides documentation and tools such as Genkernel to simplify the process and make it straightforward for novice users. Support for installation is provided on the Gentoo forum and on IRC. Compiling packages from source takes considerably more time than installing pre-built binaries. In some cases (depending on the size of the source code to be compiled and hardware), compilation of large programs can take hours and may also require a few gigabytes of temporary disk space in which to build. Generally, Gentoo users accept long compile times as the cost of being able to apply their own compile-time options and enjoy the flexibility of Portage, but Gentoo

developers have created a number of work-arounds to avoid slow package installation. Pre-compiled binaries are provided for some applications with long build times, such as OpenOffice.org and Mozilla Firefox, provided by upstream maintainers. By using these binaries, installation time is equivalent to other Linux distributions, but users lose the ability to customize optional features. The standard installation process gives users configuration options to reduce compilation times, such as enabling parallel compilation and using pipes instead of temporary files. Other optional features of the Portage system include distributed compiling and using a compiler cache. In addition, the user may be able to mount a large filesystem in RAM to greatly speed up the process of building packages. Some of these approaches have drawbacks, and so are not enabled by default. When installing the same package on multiple computers, the package may be compiled once and a binary package created for quick installation on the other computers, assuming sufficiently similar hardware.

1.5 LINUX/UNIX SYSTEM ARCHITECTURE

Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. It's functionality list is quite similar to that of UNIX.

Basic Features
Following are some of the important features of Linux Operating System.

- **Portable** – Portability means software can work on different type of hardware in same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source** – Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** – Linux is a multiuser system means multiple users can access system resources like memory/ ram/application programs at the same time.
- **Multiprogramming** – Linux is a multiprogramming system meaning multiple applications can run at same time.
- **Hierarchical File System** – Linux provides a standard file structure in which system files/user files are arranged.
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
- **Security** – Linux provides user security using authentication features like password protection/controlled access to specific files/encryption of data.

Components of Linux System

The Linux Operating System has primarily three components:

Kernel – Kernel is the core part of Linux. It is responsible for all major activities of the operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.

System Library – System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implements most of the functionalities of the operating system and do not require kernel module's code access rights.

System Utility – System Utility programs are responsible to do specialized, individual level tasks.

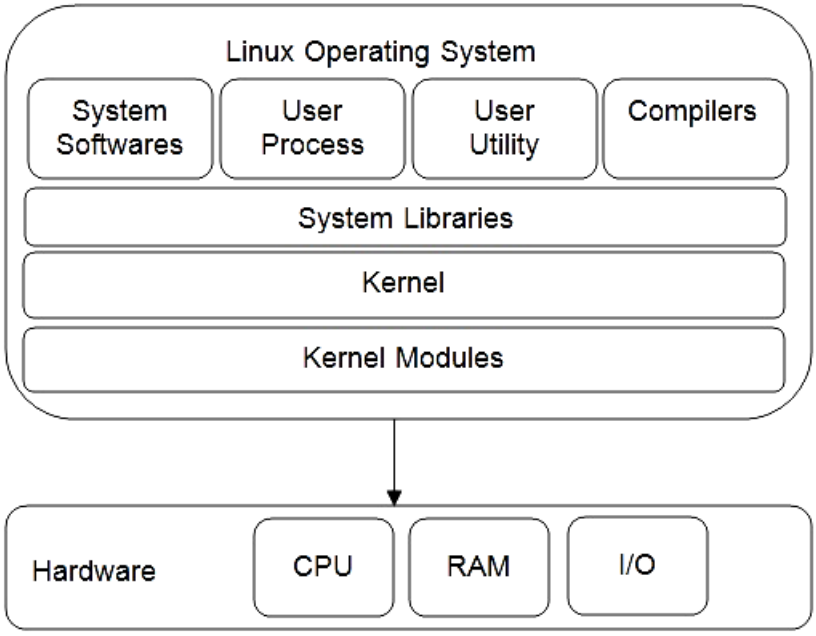


Fig 1.1: Components of Linux

Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called kernel mode with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes. Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in User Mode which has no access to system hardware and kernel code. User programs/utilities use

System libraries to access Kernel functions to get system's low level tasks.

The Architecture of Linux consist of five parts –

1. Kernel
2. Shell
3. System Utilities
4. User Applications
5. Hardware Platform

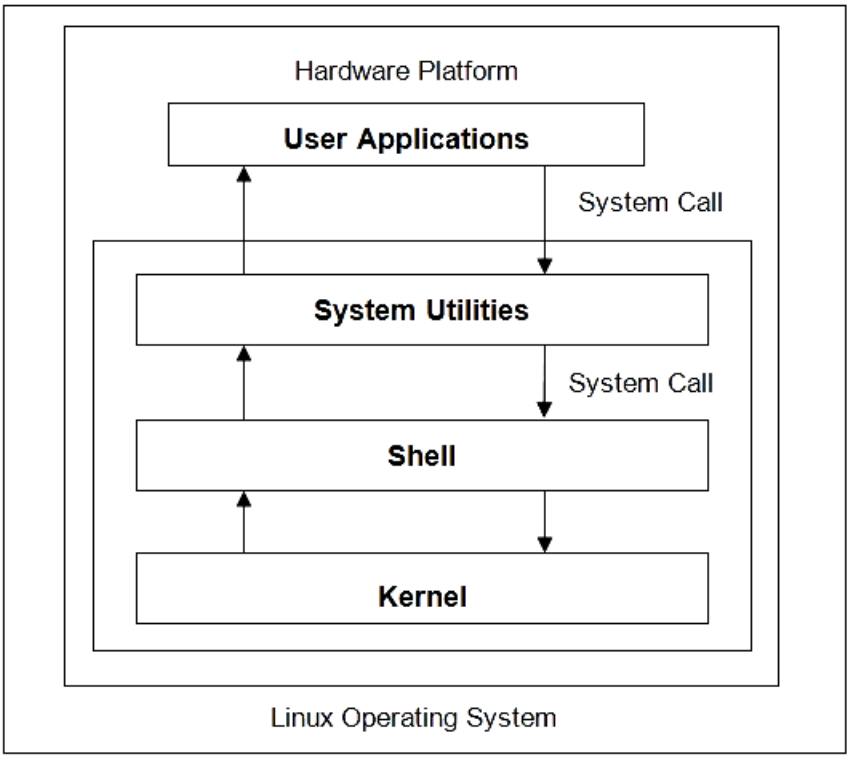


Fig 1.2: Linux System architecture

1. **The KERNEL** –

On a purely technical level, the kernel is an intermediary layer between the hardware and the software. Its purpose is to pass application requests to the hardware and to act as a low-level driver to address the devices and components of the system. Nevertheless, there are other interesting ways of viewing the kernel. The **kernel** is the core of the Linux operating system: it manages communication between devices and software, manages the system resources (like CPU time, memory, network, ...) and shields off the complexity of device programming from the developer as it provides an interface for the programmer to manipulate hardware.

- The kernel can be regarded as an enhanced machine that, in the view of the application, abstracts the computer on a

high level. For example, when the kernel addresses a hard disk, it must decide which path to use to copy data from disk to memory, where the data reside, which commands must be sent to the disk via which path, and so on. Applications, on the other hand, need only issue the command that data are to be transferred. How this is done is irrelevant to the application — the details are abstracted by the kernel. Application programs have no contact with the hardware itself, only with the kernel, which, for them, represents the lowest level in the hierarchy they know — and is therefore an enhanced machine.

- Viewing the kernel as a resource manager is justified when several programs are run concurrently on a system. In this case, the kernel is an instance that shares available resources — CPU time, disk space, network connections, and so on — between the various system processes while at the same time ensuring system integrity.

- Another view of the kernel is as a library providing a range of system-oriented commands. As is generally known, system calls are used to send requests to the computer; with the help of the C standard library, these appear to the application programs as normal functions that are invoked in the same way as any other function.

The Linux kernel is composed of five main subsystems:

a) The **Process Scheduler** (SCHED) is responsible for controlling process access to the CPU. The scheduler enforces a policy that ensures that processes will have fair access to the CPU, while ensuring that necessary hardware actions are performed by the kernel on time.

b) The **Memory Manager** (MM) permits multiple processes to securely share the machine's main memory system. In addition, the memory manager supports virtual memory that allows Linux to support processes that use more memory than is available in the system. Unused memory is swapped out to persistent storage using the file system then swapped back in when it is needed.

c) The **Virtual File System** (VFS) abstracts the details of the variety of hardware devices by presenting a common file interface to all devices. In addition, the VFS supports several file system formats that are compatible with other operating systems.

d) The **Network Interface** (NET) provides access to several networking standards and a variety of network hardware.

e) The **Inter-Process Communication** (IPC) subsystem supports several mechanisms for process-to-process communication on a single Linux system.

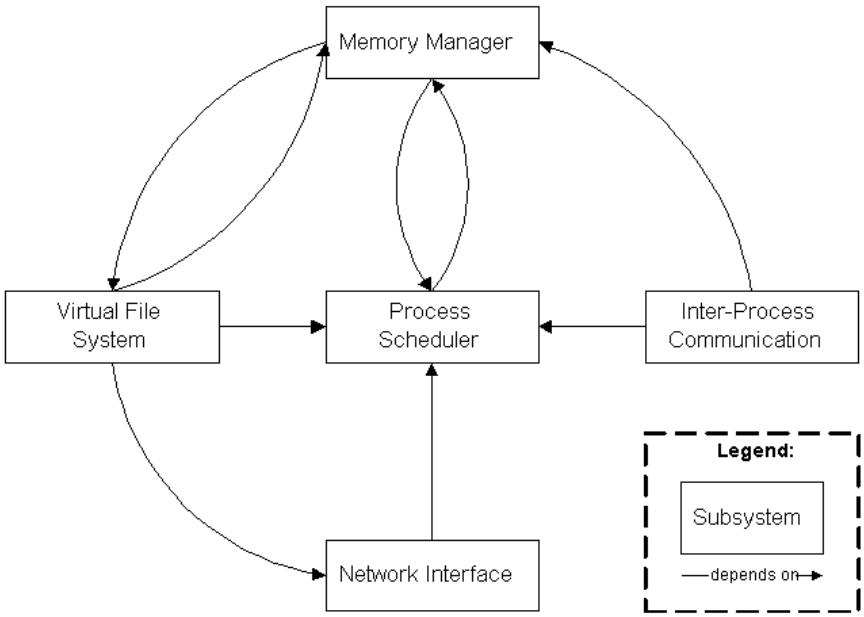


Fig 1.3: Kernel Subsystem overview

This diagram emphasizes that the most central subsystem is the process scheduler: all other subsystems depend on the process scheduler since all subsystems need to suspend and resume processes. Usually a subsystem will suspend a process that is waiting for a hardware operation to complete, and resume the process when the operation is finished. For example, when a process attempts to send a message across the network, the network interface may need to suspend the process until the hardware has completed sending the message successfully. After the message has been sent (or the hardware returns a failure), the network interface then resumes the process with a return code indicating the success or failure of the operation. The other subsystems (memory manager, virtual file system, and inter-process communication) all depend on the process scheduler for similar reasons.

The other dependencies are somewhat less obvious, but equally important:

- The **process-scheduler** subsystem uses the memory manager to adjust the hardware memory map for a specific process when that process is resumed.
- The **inter-process communication** subsystem depends on the memory manager to support a shared-memory communication mechanism. This mechanism allows two processes to access an area of common memory in addition to their usual private memory.
- The **virtual file system** uses the network interface to support a network file system (NFS), and also uses the memory manager to provide a ramdisk device.
- The **memory manager** uses the virtual file system to support swapping; this is the only reason that the memory manager depends on the process scheduler. When a

process accesses memory that is currently swapped out, the memory manager makes a request to the file system to fetch the memory from persistent storage, and suspends the process.

In addition to the dependencies that are shown explicitly, all subsystems in the kernel rely on some common resources that are not shown in any subsystem. These include procedures that all kernel subsystems use to allocate and free memory for the kernel's use, procedures to print warning or error messages, and system debugging routines. Each of the depicted subsystems contains state information that is accessed using a procedural interface, and the subsystems are each responsible for maintaining the integrity of their managed resources.

2. **The Shell** –

It acts as an interface between user and the operating system. It is the software that provides an interface for the user of an operation system which needs services of a kernel. An operating system shell is divided into two parts:-

- Command line
- GUI

Command line Shell

It is the part of the operating system which receives and executes the operating system command by the user. The commands are then sent to the kernel for execution. If the command is valid the kernel starts the execution else an error results.

GUI (Graphical User Interface)

This provides a user-friendly environment. Users cannot remember the syntax of all the command and thus it helps to simply point toward the object by the mouse or some other pointing device which a user required for its execution.

Interconnection between Kernel and Shell

When user gives any command for performing any operation the request goes to the SHELL. The Shell then translates these human-readable programs to machine language and then transfers the request to the kernel. The kernel receives the request from the shell, processes the request and then displays the result on the screen. All these functions are performed by the kernel in a transparent manner.

3. **System Utilities –**

The System Utilities consist of various system interrupts and system calls which is to transfer the control for the user mode to the kernel mode containing the kernel and the shell for further execution of the commands. The control can be transfer using System calls.

System Call

System call is an interface between a process and the operating system. In simple words a system call is the request for running any program and for performing any operation on the system that the user has requested. eg. A user has requested MS paint and this request will be sent to the operating system, which will then generate some activity to support MS paint and run on the system.

System calls are of different types –

a) File Management System calls – For performing open, close, read, write etc operations.
b) Process Control System calls – For performing Load, execute, create etc operations.
c) Device Management System calls – For performing request device, write device, and release device etc operations.
d) Communication System calls – For performing Send message, transfer status etc operations etc.

4. User Applications –

These are the applications which a user requires to perform some basic tasks. Linux and other operating systems come up with various different applications in them like g++, gcc, office suits etc. Kernel is used to generate processes to support these applications.

5. Hardware Platform –

The resource of the system such as keyboard, monitor, printer etc with which the user can input/output the request.

CHECK YOUR PROGRESS

1. Fill in the blanks

(a) The defining component of Linux is the Linux _____.

(b) _____ software is computer software licensed under exclusive legal right of the copyright holder.

(c) _____ software is very often developed in a public, collaborative manner.

(d) _____ software is software provided under terms that guarantee the freedoms of its users to run it.

(e) _____ version numbers have two parts, a _____ version and a _____ version.

(f) _____ Linux branches its releases from versions of Fedora.

(g) _____ is a computer operating system based on the Debian Linux distribution.

(h) _____ Linux is built on top of the Linux kernel and based on the _____ system.

(i) _____ are special functions or programs using which application programs or system utilities accesses _____ features.

(j) The kernel is an intermediary layer between the _____ and the _____.

(k) The Process Scheduler is responsible for controlling _____ to the CPU.

(l) The Shell acts as an _____ between user and the operating system.

1.6 LINUX FILE SYSTEM

On a LINUX system, everything is a file; if something is not a file, it is a process. This statement is true because there are special files that are more than just files (named pipes and sockets, for instance), but to keep things simple, saying that everything is a file is an acceptable generalization. A Linux system makes no difference between a file and a directory, since a directory is just a file containing names of other files. Programs, services, texts, images, and so forth, are all files. Input and output devices, and generally all devices, are considered to be files, according to the system.

The filesystem includes the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Thus, one might say “I have two filesystems'' meaning one has two partitions on which one stores files, or that one is using the “extended filesystem'', meaning the type of the filesystem. The difference between a disk or partition and the filesystem it contains is important. A few programs (including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if there is an existing file system there it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore will not work on a partition that does not contain one (or that contains one of the wrong types). Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called making a filesystem. Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are superblock, inode, data block, directory block, and indirection block. The superblock contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

UNIX filesystems usually allow one to create a hole in a file (this is done with the lseek() system call), which means that the filesystem just pretends that at a particular place in the file there is just zero bytes, but no actual disk sectors are reserved for that place in the file (this means that the file will use a bit less disk space). This happens especially often for small binaries, Linux shared libraries, some databases, and a few other special cases. (Holes are implemented by storing a special value as the address of the data

block in the indirect block or inode. This special address means that no data block is allocated for that part of the file, ergo, there is a hole in the file.)

Linux supports several types of filesystems. As of this writing the most important ones are:

minix –

The oldest, presumed to be the most reliable, but quite limited in features (some time stamps are missing, at most 30 character filenames) and restricted in capabilities (at most 64 MB per filesystem).

xia –

A modified version of the minix filesystem that lifts the limits on the filenames and filesystem sizes, but does not otherwise introduce new features. It is not very popular, but is reported to work very well.

ext3 –

The ext3 filesystem has all the features of the ext2 filesystem. The difference is, journaling has been added. This improves performance and recovery time in case of a system crash. This has become more popular than ext2.

ext2 –

This is the most featureful of the native Linux filesystems. It is designed to be easily upwards compatible, so that new versions of the filesystem code do not require re-making the existing filesystems.

ext –

An older version of ext2 that was not upwards compatible. It is hardly ever used in new installations any more, and most people have converted to ext2.

reiserfs –

This is a more robust filesystem where Journaling is used which makes data loss less likely. Journaling is a mechanism whereby a record is kept of transaction which is to be performed, or which have been performed. This allows the filesystem to reconstruct itself fairly easily after damage caused by, for example, improper shutdowns.

jfs –

JFS is a journaled filesystem designed by IBM to to work in high performance environments>

xfs –

XFS was originally designed by Silicon Graphics to work as a 64-bit journaled filesystem. XFS was also designed to maintain high performance with large files and filesystems. In addition, support

for several foreign filesystems exists, to make it easier to exchange files with other operating systems. These foreign filesystems work just like native ones, except that they may be lacking in some usual UNIX features, or have curious limitations, or other oddities.

msdos –

Compatibility with MS-DOS (and OS/2 and Windows NT) FAT filesystems.

umsdos –

Extends the msdos filesystem driver under Linux to get long filenames, owners, permissions, links, and device files. This allows a normal msdos filesystem to be used as if it were a Linux one, thus removing the need for a separate partition for Linux.

vfat –

This is an extension of the FAT filesystem known as FAT32. It supports larger disk sizes than FAT. Most MS Windows disks are vfat.

iso9660 –

The standard CD-ROM filesystem; the popular Rock Ridge extension to the CD-ROM standard that allows longer file names is supported automatically.

nfs –

A networked filesystem that allows sharing a filesystem between many computers to allow easy access to the files from all of them.

smbfs –

A networks filesystem which allows sharing of a filesystem with an MS Windows computer. It is compatible with the Windows file sharing protocols.

hpfs –

The OS/2 filesystem.

sysv –

SystemV/386, Coherent, and Xenix filesystems.

NTFS –

The most advanced Microsoft journaled filesystem providing faster file access and stability over previous Microsoft filesystems.

The choice of filesystem to use depends on the situation. If compatibility or other reasons make one of the non-native filesystems necessary, then that one must be used. If one can choose freely, then it is probably wisest to use ext3, since it has all the features of ext2, and is a journaled filesystem. There is also the There is also the proc filesystem, usually accessible as the /proc

proc filesystem, usually accessible as the /proc directory, which is not really a filesystem at all, even though it looks like one. The proc filesystem makes it easy to access certain kernel data structures, such as the process list (hence the name). It makes these data structures look like a filesystem, and that filesystem can be manipulated with all the usual file tools. For example, to get a listing of all processes one might use the command:

```
$ ls -l /proc
total 0
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 1
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 63
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 94
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 95
dr-xr-xr-x   4 root     users           0 Jan 31 20:37 98
dr-xr-xr-x   4 liw      users           0 Jan 31 20:37 99
-r--r--r--   1 root     root            0 Jan 31 20:37 devices
-r--r--r--   1 root     root            0 Jan 31 20:37 dma
-r--r--r--   1 root     root            0 Jan 31 20:37 filesystems
-r--r--r--   1 root     root            0 Jan 31 20:37 interrupts
-r--------   1 root     root      8654848 Jan 31 20:37 kcore
-r--r--r--   1 root     root            0 Jan 31 11:50 kmsg
-r--r--r--   1 root     root            0 Jan 31 20:37 ksyms
-r--r--r--   1 root     root            0 Jan 31 11:51 loadavg
-r--r--r--   1 root     root            0 Jan 31 20:37 meminfo
-r--r--r--   1 root     root            0 Jan 31 20:37 modules
dr-xr-xr-x   2 root     root            0 Jan 31 20:37 net
dr-xr-xr-x   4 root     root            0 Jan 31 20:37 self
-r--r--r--   1 root     root            0 Jan 31 20:37 stat
-r--r--r--   1 root     root            0 Jan 31 20:37 uptime
-r--r--r--   1 root     root            0 Jan 31 20:37
version
$
```

(There will be a few extra files that do not correspond to processes, though. The above example has been shortened.)

Note that even though it is called a filesystem, no part of the proc filesystem touches any disk. It exists only in the kernel's imagination. Whenever anyone tries to look at any part of the proc filesystem, the kernel makes it look as if the part existed somewhere, even though it does not. So, even though there is a multi-megabyte/proc/kcore file, it does not take any disk space.

Bootblock –

A program at some fixed location on a hard disk, floppy disk or other media, which is loaded when the computer is turned on or rebooted and which controls the next phase of loading the actual operating system. The loading and execution of the boot block is usually controlled by firmware in ROM or PROM. It is a dedicated block usually at the beginning (first block on first track) of a storage medium that holds special data used to start a system. Some systems use a boot block of several physical sectors, while some use only one boot sector. Other manufacturers use the terms boot block and boot sector interchangeably.

Superblock –

A *superblock* is a record of the characteristics of a *filesystem*, including its size, the *block* size, the empty and the filled blocks and their respective counts, the size and location of the *inode* tables, the disk block map and usage information, and the size of the *block groups*. A request to access any file requires access to the filesystem's superblock. If its superblock cannot be accessed, a filesystem cannot be mounted (i.e., logically attached to the main filesystem) and thus files cannot be accessed. Any attempt to mount a filesystem with a corrupted or otherwise damaged superblock will likely fail (and usually generate an error message such as cannot read superblock).

Because of the importance of the superblock and because damage to it (for example, from physical damage to the magnetic recording medium on the disk) could erase crucial data, backup copies are created automatically at intervals on the filesystem (e.g., at the beginning of each block group). For each mounted filesystem, Linux also maintains a copy of its superblock in memory. Thus there are backup copies of the superblock at block offsets 8193, 16385, 24577, etc. If the ext2 filesystem is used, then the filesystem has block groups each comprised of 8192 blocks can be confirmed with the dumpe2fs command as follows:

dumpe2fs device_name | less

device_name is the name of the partition on which the filesystem resides. The output of dumpe2fs is piped (i.e., sent) to the less command because it can be long and thus in order to read it one screenful at a time. It can be seen that dumpe2fs also provides a great deal of additional information about the filesystem, including the block size.

For example, the following will provide the location of the primary and backup superblocks on the first partition of the first HDD:

/dumpe2fs /dev/hda1 | less

If a filesystem cannot be mounted because of superblock problems, it is likely that e2fsck, and the related fsck command, which are used to check and repair the filesystem, will fail as well, at least initially. Fortunately, however, e2fsck can be instructed to use one of the superblock copies instead by issuing a command similar to the following:

e2fsck -f -b block_offset device

block_offset is the offset to a superblock copy, and it is usually 8193. The -f option is used to force e2fsck to check the filesystem. When using superblock backup copies, the filesystem may appear to be clean, in which case no check is needed, but -f overrides this. For example, to check and repair the filesystem on /dev/hda2 (i.e., the second partition of the first HDD) if it has a defective superblock, the following can be used:

e2fsck -f -b 8193 /dev/hda2

This command can be executed from an appropriate emergency floppy disk, and it is possible that it will allow the designated filesystem to be mounted again.

The equivalent to the superblock on Microsoft Windows filesystem is the file allocation table (FAT), which records which disk blocks hold the topmost directory. On Unix-like operating systems the superblock is virtually always held in memory, whereas it is not for older operating systems such as MS-DOS. The superblock acquired its name from the fact that the first data block of a disk or of a partition was used to hold the meta-data (i.e., data about data) about the partition itself. Superblock are now independent of the concept of the data block, but it remains the data structure that holds information about each mounted filesystem.

Inodes & Inode table –

For most users and for most common system administration tasks, it is enough to accept that files and directories are ordered in a tree-like structure. The computer, however, does not understand a thing about trees or tree-structures.

Every partition has its own file system. By imagining all those file systems together, we can form an idea of the tree-structure of the entire system, but it is not as simple as that. In a file system, a file is represented by an inode, a kind of serial number containing information about the actual data that makes up the file: to whom this file belongs, and where is it located on the hard disk. Every partition has its own set of inodes; throughout a system with multiple partitions, files with the same inode number can exist. Each inode describes a data structure on the hard disk, storing the properties of a file, including the physical location of the file data. When a hard disk is initialized to accept data storage, usually during the initial system installation process or when adding extra disks to an existing system, a fixed number of inodes per partition is created. This number will be the maximum amount of files, of all types (including directories, special files, links etc.) that can exist at the same time on the partition. We typically count on having 1 inode per 2 to 8 kilobytes of storage.

At the time a new file is created, it gets a free inode. In that inode is the following information:

- Owner and group owner of the file.
- File type (regular, directory, ...)
- Permissions on the file
- Date and time of creation, last read and change.
- Date and time this information has been changed in the inode.
- Number of links to this file
- File size
- An address defining the actual location of the file data.

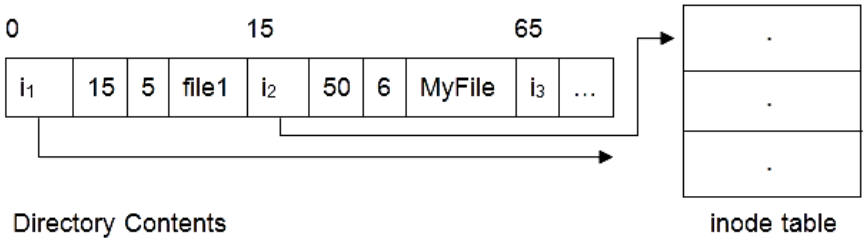
The only information not included in an inode, is the file name and directory. These are stored in the special directory files. By comparing file names and inode numbers, the system can make up a tree-structure that the user understands. Users can display

inode numbers using the -i option to ls. The inodes have their own separate space on the disk.

Inode table –

Each time we create a file in a directory, the system simply allocates a free I-number from the file system and uses an empty slot in the correspondent directory to record this I-number and the name of the file we created. When we issue a command to delete a file, the system simply replaces the file's I-number by 0, and the slot is still occupied by its file name, although we are no longer able to access that file anymore. Therefore, if we frequently create and delete files within a directory, the (file) size of the directory becomes bigger and bigger and the I-numbers are no longer consecutive, this will slow down the file searching operation (and hence, the system performance, too) considerably. This is one of the main reasons why the system performance is gradually deteriorating. We can prevent this from happening by grouping the files which are changing constantly in a fixed file system, say /tmp, and periodically repairing the file system. The following diagram depicts the directory contents of some ext2 directory.

```
/* include/linux/dirent.h */
struct dirent {
        long            d_ino;
        __kernel_off_t d_off;
        unsigned short          d_reclen;
        char            d_name[256]; /* We must not include limits.h! */
};
```



An ordinary file is just a sequence of data bytes stored in some physical device without any name attached to it. The administrative information of this file, such as owner, permissions, size, times, etc., is stored in the inode structure of the file. All of the file system's inodes are collected together to form an inode table. Each file system occupies a logical disk. Starting from the 2nd block of a logical disk, the kernel stores the inode table of the file system in a consecutive disk blocks. Each inode, an entry in the inode table, is a data structure which the system uses to store the following information about a file:

1. Type of file (ordinary, directory or special file).
2. Access permissions for the file owner, the owner's group members and others (i.e. the general public).
3. Number of links.
4. File owner's user and group IDs
5. File size in bytes.
6. The disk addresses of the data blocks where the contents of the file are actually stored.
7. Time of last access (read or executed), time of last modification (i.e. written) and time which the inode itself was last changed.

How linux accesses files –

With all these locations, it might be difficult to locate a particular file. Most of the time, the file we want to locate is inside the home directory. However, in some cases we want to locate a particular file somewhere on our entire system.

locate –

locate, a Unix utility first created in 1983, serves to find files on filesystems. It searches through a prebuilt database of files generated by **updatedb** or by a daemon and compressed using incremental encoding. It operates significantly faster than **find**, but requires regular updating of the database. This sacrifices overall efficiency (because of the regular interrogation of filesystems even when no user needs information) and absolute accuracy (since the database does not update in real time) for significant speed improvements (particularly on very large filesystems). On fast systems with small drives, locate is neither necessary nor desirable.

find –

The **find** command is a very important and powerful command. Unlike **locate**, it only returns live information (so it does not use a database). This makes searches with **find** somewhat slow, but **find**'s power is not speed, but the options one can give to find a particular file.

Regular find patterns

The most simple **find** construct is to locate a particular file inside one or more directories. For instance, to find files or directories inside /etc whose name is dhcpd.conf (exact matches):

```
$ find /etc -name dhcpd.conf
/etc/dhcp/dhcpd.conf
```

To find files (not directories) where dhcpd is in the filename, also inside /etc directory:

```
$ find /etc -type f -name '*dhcpd*'
/etc/conf.d/dhcpd
/etc/init.d/dhcpd
/etc/udhcpd.conf
```

```
/etc/dhcp/dhcpd.conf
```

To find files in the /etc directory who have been modified within the last 7 days (read: "less than 7 days ago"):

```
$ find /etc -type f -mtime -7
/etc/mtab
/etc/adjtime
/etc/wifi-radar.conf
/etc/genkernel.conf
```

We can even find files based on their ownership. For instance, to find the files in /etc that do not belong to the root user:

```
$ find /etc -type f -not -user root
```

Combining find patterns

We can also combine find patterns. For instance, to find files modified within the last 7 days but whose name does not contain .conf:

```
$ find /etc -type f -mtime -7 -not -name '*.conf'
/etc/mtab
/etc/adjtime
```

Or, to find the same files, but the name should also not be mtab:

```
$ find /etc -type f -mtime -7 -not \( -name '*.conf' -
or -name mtab)
/etc/adjtime
```

1.7 LINUX STANDARD DIRECTORIES

In order to manage all those files in an orderly fashion, we think of them in an ordered tree-like structure on the hard disk, as we know from MS-DOS (Disk Operating System) for instance. The large branches contain more branches, and the branches at the end contain the tree's leaves or normal files.

Types of Files –

Most files are just files, called regular files; they contain normal data, for example text files, executable files or programs, input for or output from a program and so on. While it is reasonably safe to suppose that everything we encounter on a Linux system is a file, there are some exceptions.

- Directories: files that are lists of other files.
- Special files: the mechanism used for input and output. Most special files are in /dev.
- Links: a system to make a file or directory visible in multiple parts of the system's file tree.
- (Domain) sockets: a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control.

- Named pipes: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

The -l option to ls displays the file type, using the first character of each input line:

```
jaime:~/Documents> ls -l
total 80
-rw-rw-r--   1 jaime   jaime   31744 Feb 21 17:56 intro Linux.doc
-rw-rw-r--   1 jaime   jaime   41472 Feb 21 17:56 Linux.doc
drwxrwxr-x   2 jaime   jaime    4096 Feb 25 11:50 course
```

This table gives an overview of the characters determining the file type:

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

In order not to always have to perform a long listing for seeing the file type, a lot of systems by default do not issue just ls, but ls -F, which suffixes file names with one of the characters "/=*|@" to indicate the file type. To make it extra easy on the beginning user, both the -F and --color options are usually combined. For convenience, the Linux directory system is usually thought of in a tree structure. On a standard Linux system we will find the layout generally follows the scheme presented below.

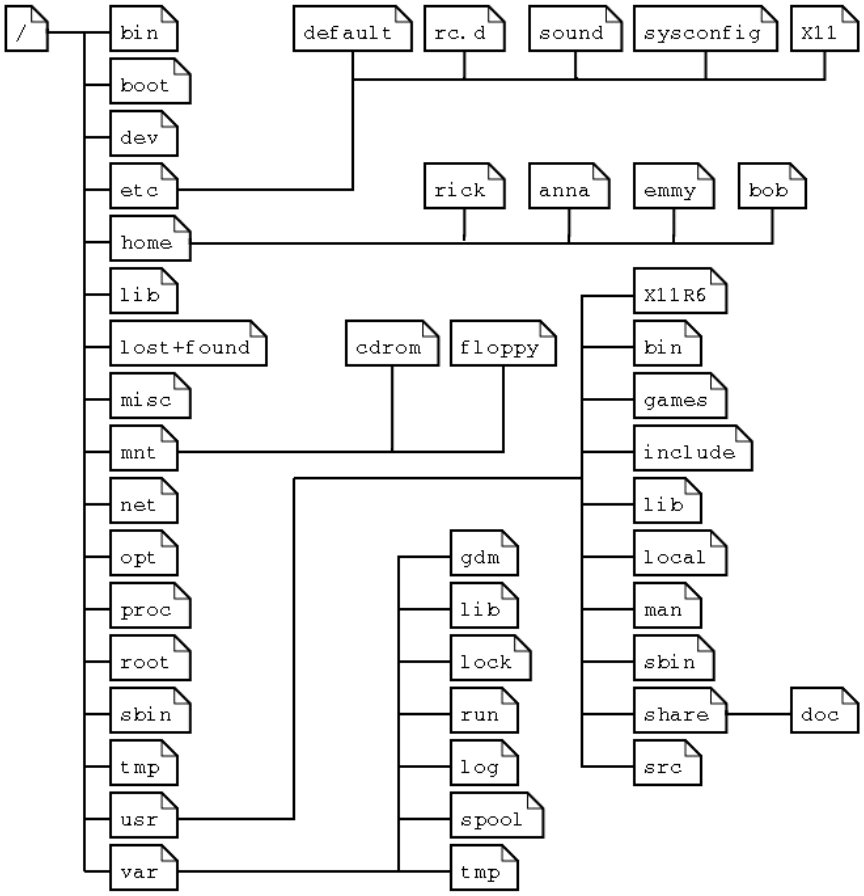


Fig 1.4: Linux directory system layout

This is a layout from a RedHat system. Depending on the system admin, the operating system and the mission of the UNIX machine, the structure may vary, and directories may be left out or added at will. The names are not even required; they are only a convention. The tree of the file system starts at the trunk or slash, indicated by a forward slash (/). This directory, containing all underlying directories and files, is also called the root directory or "the root" of the file system. Directories that are only one level below the root directory are often preceded by a slash, to indicate their position and prevent confusion with other directories that could have the same name. When starting with a new system, it is always a good idea to take a look in the root directory.

Directory	Content
/bin	Common programs, shared by the system, the system administrator and the users.
/boot	The startup files and the kernel, vmlinuz. In some recent distributions also grub data. Grub is the Grand Unified Boot loader and is an attempt to get rid of the many different boot-loaders we know today.
/dev	Contains references to all the CPU peripheral hardware, which are represented as files with special

	properties.
/etc	Most important system configuration files are in /etc, this directory contains data similar to those in the Control Panel in Windows
/home	Home directories of the common users.
/initrd	(on some distributions) Information for booting.
/lib	Library files, includes files for all kinds of programs needed by the system and the users.
/lost+found	Every partition has a lost+found in its upper directory. Files that were saved during failures are here.
/misc	For miscellaneous purposes
/mnt	Standard mount point for external file systems, e.g. a CD-ROM or a digital camera.
/net	Standard mount point for entire remote file systems.
/opt	Typically contains extra and third party software.
/proc	A virtual file system containing information about system resources. More information about the meaning of the files in proc is obtained by entering the command man proc in a terminal window. The file proc.txt discusses the virtual file system in detail.
/root	The administrative user's home directory. Mind the difference between /, the root directory and /root, the home directory of the *root* user.
/sbin	Programs for use by the system and the system administrator
/tmp	Temporary space for use by the system, cleaned upon reboot, so don't use this for saving any work
/usr	Programs, libraries, documentation etc. for all user-related programs.
/var	Storage for all variable files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files downloaded from the Internet, or to keep an image of a CD before burning it.

LILO bootloader –

LILO (LInux LOader) is a boot loader for Linux and was the default boot loader for most Linux distributions in the years after the popularity of loadlin. Today, most distributions use GRUB as the

default boot loader. LILO does not depend on a specific file system, and can boot an operating system (e.g., Linux kernel images) from floppy disks and hard disks. One of up to sixteen different images can be selected at boot time. Various parameters, such as the root device, can be set independently for each kernel. LILO can be placed in the master boot record (MBR) or the boot sector of a partition. In the latter case, the MBR must contain code to load LILO. At system start, only the BIOS drivers are available for LILO to access hard disks. For this reason, a very old BIOS access area is limited to cylinders 0 to 1023 of the first two hard disks. For later BIOS, LILO can use 32-bit "logical block addressing" (LBA) to access the entire capacity of the hard disks the BIOS has access to.

GRUB bootloader –

GNU GRUB is a bootloader capable of loading a variety of free and proprietary operating systems. GRUB will work well with Linux, DOS, Windows, or BSD. GRUB stands for GRand Unified Bootloader. GRUB is dynamically configurable. This means that the user can make changes during the boot time, which include altering existing boot entries, adding new, custom entries, selecting different kernels, or modifying initrd. GRUB also supports Logical Block Address mode. This means that for a computer having a fairly modern BIOS that can access more than 8GB (first 1024 cylinders) of hard disk space, GRUB will automatically be able to access all of it. GRUB can be run from or be installed to any device (floppy disk, hard disk, CD-ROM, USB drive, network drive) and can load operating systems from just as many locations, including network drives. It can also decompress operating system images before booting them.

Working –

When a computer boots, the BIOS transfers control to the first boot device, which can be a hard disk, a floppy disk, a CD-ROM, or any other BIOS-recognized device. The first sector on a hard is called the Master Boot Record (MBR). This sector is only 512 bytes long and contains a small piece of code (446 bytes) called the primary boot loader and the partition table (64 bytes) describing the primary and extended partitions. By default, MBR code looks for the partition marked as active and once such a partition is found, it loads its boot sector into memory and passes control to it. GRUB replaces the default MBR with its own code.

Furthermore, GRUB works in stages –

Stage 1 is located in the MBR and mainly points to Stage 2, since the MBR is too small to contain all of the needed data.

Stage 2 points to its configuration file, which contains the entire complex user interface and options we are normally familiar with when talking about GRUB. Stage 2 can be located anywhere on the disk. If Stage 2 cannot find its configuration table, GRUB will cease the boot sequence and present the user with a command line for manual configuration.

Stage 1.5 also exists and might be used if the boot information is small enough to fit in the area immediately after MBR. The Stage architecture allows GRUB to be large (~20-30K) and therefore fairly complex and highly configurable, compared to most bootloaders, which are sparse and simple to fit within the limitations of the Partition Table.

1.8 INSTALLING THE LINUX SYSTEM

One of the things that makes Linux special is that it can play nice with other operating systems. We can run Linux alongside of other operating systems quite easily. The most popular installation process for installing Linux is to install a Fresh Installation of Linux with no other operating system in place. This allows the computer to dedicated 100% of its resources to running Linux. However, it is quite easy to install Linux as a one of a series of operating systems that a computer has available to it.

Live CD/DVD Booting Linux – If one is just looking to try Linux out to see if one likes it, but does not want to commit to wiping out the main operating system, one may want to consider trying Linux from a **Live CD/DVD**. Many Linux installations provide the option of downloading and running Linux as a **Live CD**, which means that Linux runs as a completely bootable operating system from the CD/DVD. The files are loaded into the computer's memory, rather than being run for a hard disk drive. In layman's terms, this means that one can run Linux from a CD/DVD, and then when the PC is rebooted, and the CD/DVD is removed, it will boot back into its old operating system without any creating any difference. This gives us an easy way to try out several distributions of Linux until we find the one that we like. Using a **Live CD/DVD** is also a popular method of rescuing files from a corrupted operating system.

Linux as a VM inside another Operating System – If one likes a (non-linux) desktop operating system, but would like an easy way to access a Linux desktop or run ones favorite open source software, one may want to consider running Linux as a VM inside another operating system. There are a number of ways to do this, but one simple one would be to download and install a Virtual Server application, and then install the Linux distribution under that host software. Everything that one can do with our other operating system can be done with Linux. That means word processing, databases, spreadsheets, Internet browsers, e-mail, photo touch-ups, MP3, CD Players, cameras and then there are a lot of things that Linux has to offer on top of all that that other operating systems do not.

F**resh Install of Linux** – This method is by far the most popular installation method available. In this approach, one has to format the computer's hard drive and install Linux from a CD/DVD. Linux then runs as the only operating system on the computer.

1.9 HARD DISK PARTITIONING FOR LINUX

Most people have a vague knowledge of what partitions are, since every operating system has the ability to create or remove them. It may seem strange that Linux uses more than one partition on the same disk, even when using the standard installation procedure, so some explanation is called for. One of the goals of having different partitions is to achieve higher data security in case of disaster. By dividing the hard disk in partitions, data can be grouped and separated. When an accident occurs, only the data in the partition that got the hit will be damaged, while the data on the other partitions will most likely survive. This principle dates from the days when Linux did not have journaled file systems and power failures might have lead to disaster. The use of partitions remains for security and robustness reasons, so a breach on one part of the system does not automatically mean that the whole computer is in danger. This is currently the most important reason for partitioning. A simple example: a user creates a script, a program or a web application that starts filling up the disk. If the disk contains only one big partition, the entire system will stop functioning if the disk is full. If the user stores the data on a separate partition, then only that (data) partition will be affected, while the system partitions and possible other data partitions keep functioning. Mind that having a journaled file system only provides data security in case of power failure and sudden disconnection of storage devices. This does not protect the data against bad blocks and logical errors in the file system. In those cases, we should use a RAID (Redundant Array of Inexpensive Disks) solution.

There are two kinds of major partitions on a Linux system:

- **data partition**: normal Linux system data, including the root partition containing all the data to start up and run the system; and
- **swap partition**: expansion of the computer's physical memory, extra memory on hard disk.

Most systems contain a root partition, one or more data partitions and one or more swap partitions. Systems in mixed environments may contain partitions for other system data, such as a partition with a FAT or VFAT file system for MS Windows data. Most Linux systems use fdisk at installation time to set the partition type. The standard Linux partitions have number 82 for swap and 83 for data, which can be journaled (ext3) or normal (ext2, on older systems). The fdisk utility has built-in help, should one forget these values. Apart from these two, Linux supports a variety of other file system types, such as the relatively new Reiser file system, JFS, NFS, FATxx and many other file systems natively available on other (proprietary) operating systems. The standard root partition (indicated with a single forward slash, /) is about 100-500 MB, and contains the system configuration files, most basic commands and server programs, system libraries, some temporary space and the home directory of the administrative user. A standard installation requires about 250 MB for the root partition.

Swap space (indicated with swap) is only accessible for the system itself, and is hidden from view during normal operation. Swap is the system that ensures, like on normal UNIX systems, that we can keep on working, whatever happens. On Linux, we will virtually never see irritating messages like “Out of memory, please close some applications first and try again”, because of this extra memory. The swap or virtual memory procedure has long been adopted by operating systems outside the UNIX world by now. Using memory on a hard disk is naturally slower than using the real memory chips of a computer, but having this little extra is a great comfort.

Linux generally counts on having twice the amount of physical memory in the form of swap space on the hard disk. An example on a system with 512 MB of RAM:

- 1st possibility: one swap partition of 1 GB
- 2nd possibility: two swap partitions of 512 MB
- 3rd possibility: with two hard disks: 1 partition of 512 MB on each disk.

The last option will give the best results when a lot of I/O is to be expected. Some applications, such as databases, might require more swap space. Others, such as some handheld systems, might not have any swap at all by lack of a hard disk. Swap space may also depend on the kernel version. The kernel is on a separate partition as well in many distributions, because it is the most important file of the system. If such cases, there shall also be a /boot partition, holding the kernel(s) and accompanying data files. The rest of the hard disk(s) is generally divided in data partitions, although it may be that all of the non-system critical data resides on one partition, for example when we perform a standard workstation installation. When non-critical data is separated on different partitions, it usually happens following a set pattern:

- a partition for user programs (/usr)
- a partition containing the users' personal data (/home)
- a partition to store temporary data like print- and mail-queues (/var)
- a partition for third party and extra software (/opt)

Once the partitions are made, we can only add more. Changing sizes or properties of existing partitions is possible but not advisable. The division of hard disks into partitions is determined by the system administrator. On larger systems, he/she may even spread one partition over several hard disks, using the appropriate software. Most distributions allow for standard setups optimized for workstations (average users) and for general server purposes, but also accept customized partitions. During the installation process we can define our own partition layout using either the distribution specific tool, which is usually a straight forward graphical interface, or fdisk, a text-based tool for creating partitions and setting their properties.

A workstation or client installation is for use by mainly one and the same person. The selected software for installation reflects this and the stress is on common user packages, such as nice desktop themes, development tools, client programs for E-mail, multimedia software, web and other services. Everything is put together on one large partition, swap space twice the amount of RAM is added and the generic workstation is complete, providing the largest amount of disk space possible for personal use, but with the disadvantage of possible data integrity loss during problem situations. On a server, system data tends to be separate from user data. Programs that offer services are kept in a different place than the data handled by this service. Different partitions will be created on such systems:

- a partition with all data necessary to boot the machine
- a partition with configuration data and server programs
- one or more partitions containing the server data such as database tables, user mails, an ftp archive etc.
- a partition with user programs and applications
- one or more partitions for the user specific files (home directories)
- one or more swap partitions (virtual memory)

Servers usually have more memory and thus more swap space. Certain server processes, such as databases, may require more swap space than usual, For better performance, swap is often divided into different swap partitions.

Creating a New Partition in Linux:

In most Linux systems, we can use the **fdisk** utility to create a new partition and to do other disk management operations. As a tool with a text interface, **fdisk** requires typing the commands on the **fdisk** command line. The following fdisk commands may be helpful:

Options	Description
m	Displays the available commands.
p	Displays the list of existing partitions on your hda drive. Unpartitioned space is not listed.
n	Creates a new partition.
q	Exits fdisk without saving the changes.
l	Lists partition types.
w	Writes changes to the partition table.

To create a new partition on Linux:

1. We first start a terminal.
2. Then start fdisk using the following command:

   /sbin/fdisk /dev/hda

   where /dev/hda stands for the hard drive that we want to partition.
3. In fdisk, to create a new partition, we type the following command:

   n

   - When prompted to specify the Partition type, we type p to create a primary partition or e to create an extended one. There may be up to four primary partitions. In case we want to create more than four partitions, we make the last partition extended, and it will be a container for other logical partitions.
   - When prompted for the Number, in most cases, we type 3 because a typical Linux virtual machine has two partitions by default.
   - When prompted for the Start cylinder, we type a starting cylinder number or press **Return** to use the first cylinder available.
   - When prompted for the Last cylinder, we press **Return** to allocate all the available space or specify the size of a new partition in cylinders if we do not want to use all the available space.

   By default, **fdisk** creates a partition with a System ID of 83. If we are unsure of the partition's System ID, we use the

   l

   command to check it.
4. Then use the

   w

   command to write the changes to the partition table.
5. Then we restart the virtual machine by entering the reboot command.
6. When restarted, we create a file system on the new partition. In most cases it will be either the ext3 or reiserFS file system. For example, to create the Ext3 file system, we enter the following command:

   /sbin/mkfs -t ext3 /dev/hda3
7. We then create a directory that will be a mount point for the new partition. For example, to name it data, we enter:

   mkdir /data
8. Then we mount the new partition to the directory we just created by using the following command:

mount /dev/hda3 /data

9. We then make changes in our static file system information by editing the /etc/fstab file in any of the available text editors. For example, we add the following string to this file:

   /dev/hda3 /data ext3 defaults 0 0

   In this string /dev/hda3 is the partition we just created, /data is a mount point for the new partition, ext3 is the file type of the new partition.

10. Finally we save the /etc/fstab file.

init –

The **init** process is the first user level process started by the kernel. **init** has many important duties, such as starting **getty** (so that users can log in), implementing run levels, and taking care of orphaned processes. init is one of those programs that are absolutely essential to the operation of a Linux system, but that we still can mostly ignore. A good Linux distribution will come with a configuration for init that will work for most systems, and on these systems there is nothing we need to do about init. Usually, we only need to worry about init if we hook up serial terminals, dial-in (not dial-out) modems, or if we want to change the default run level. When the kernel has started itself (has been loaded into memory, has started running, and has initialized all device drivers and data structures and such), it finishes its own part of the boot process by starting a user level program, init. Thus, init is always the first process (its process number is always 1). The kernel looks for init in a few locations that have been historically used for it, but the proper location for it (on a Linux system) is /sbin/init. If the kernel cannot find init, it tries to run /bin/sh, and if that also fails, the startup of the system fails.

When init starts, it finishes the boot process by doing a number of administrative tasks, such as checking filesystems, cleaning up /tmp, starting various services, and starting a getty for each terminal and virtual console where users should be able to log in. After the system is properly up, init restarts getty for each terminal after a user has logged out (so that the next user can log in). init also adopts orphan processes: when a process starts a child process and dies before its child, the child immediately becomes a child of init. This is important for various technical reasons, but it is good to know it, since it makes it easier to understand process lists and process tree graphs. There are a few variants of init available.

Run levels –

A run level is a state of init and the whole system that defines what system services are operating. Run levels are identified by numbers. Some system administrators use run levels to define which subsystems are working, e.g., whether X is running, whether the network is operational, and so on. Others have all subsystems always running or start and stop them individually, without

changing run levels, since run levels are too coarse for controlling their systems.

The following table defines how most Linux Distributions define the different run levels. However, run-levels 2 through 5 can be modified to suit ones tastes.

0	Halt the system
1	Single-user mode (for special administration)
2	Local Multiuser with Networking but without network service (like NFS)
3	Full Multiuser with Networking
4	Not Used
5	Full Multiuser with Networking and X Windows (GUI)
6	Reboot

Services that get started at a certain runtime are determined by the contents of the various rcN.d directories. Most distributions locate these directories either at /etc/init.d/rcN.d or /etc/rcN.d. (Replace the N with the run-level number.) In each run-level we find a series of if links pointing to start-up scripts located in /etc/init.d. The names of these links all start as either K or S, followed by a number. If the name of the link starts with an S, then that indicates the service will be started when we go into that run level. If the name of the link starts with a K, the service will be killed (if running). The number following the K or S indicates the order the scripts will be run.

How run levels start are configured in /etc/inittab by lines like the following:

```
12:2:wait:/etc/init.d/rc 2
```

The first field is an arbitrary label, the second one means that this applies for run level 2. The third field means that init should run the command in the fourth field once, when the run level is entered, and that init should wait for it to complete. The /etc/init.d/rc command runs whatever commands are necessary to start and stop services to enter run level 2. The command in the fourth field does all the hard work of setting up a run level. It starts services that are not already running, and stops services that should not be running in the new run level any more. Exactly what the command is, and how run levels are configured, depends on the Linux distribution. When init starts, it looks for a line in /etc/inittab that specifies the default run level:

`id:2:initdefault:`

We can ask init to go to a non-default run level at startup by giving the kernel a command line argument of single or emergency. Kernel command line arguments can be given via LILO, for example. This allows us to choose the single user mode (run level 1). While the system is running, the telinit command can change the run level. When the run level is changed, init runs the relevant command from /etc/inittab.

CHECK YOUR PROGRESS

2. Fill in the blanks

(a) The _____ contains information about the filesystem as a whole.

(b) ___ was originally designed by Silicon Graphics to work as a 64-bit journaled filesystem.

(c) ____ is an extension of the FAT filesystem known as FAT32.

(d) The loading and execution of the boot block is usually controlled by _____ in ROM or PROM.

(e) A request to access any file requires access to the filesystem's _____.

(f) The only information not included in an inode, is the file ____ and _____.

(g) ____ returns live information so it does not use a database.

(h) LILO does not depend on a specific _____, and can boot an operating system from floppy disks and hard disks.

(i) By dividing the hard disk in _____, data can be grouped and separated.

(j) The ____ process is the first user level process started by the kernel.

1.10 LET US SUM UP

- Linux is a fast, secure, stable and open source operation system which is based on Unix.
- The owner of proprietary software exercises certain exclusive rights over the software.
- Open-source software (OSS) is computer software with its source code made available and licensed with an open-source license.
- The goals of Free Software are reached by granting the freedom to users to run, copy, distribute, study, change and improve the software.
- A Linux distribution is a collection of software on top of a Linux kernel.
- Red Hat discontinued the Red Hat Linux line in favor of Red Hat Enterprise Linux (RHEL) for enterprise environments.
- The Linux kernel consists of various modules and it interacts directly with the underlying hardware.
- The Linux shell is the software that provides an interface for the user of an operation system which needs services of a kernel.
- The Linux filesystem includes the methods and data structures that an operating system uses to keep track of files on a disk or partition.
- The bootblock is a program at some fixed location on a hard disk, floppy disk or other media, which is loaded when the computer is turned on.
- LILO is a boot loader for Linux and was the default boot loader for most Linux distributions.
- GRUB is a bootloader capable of loading a variety of free and proprietary operating systems.
- A run level is a state of init and the whole system that defines what system services are operating.

1.11 ANSWERS TO CHECK YOUR PROGRESS

1.
   (a) kernel
   (b) Proprietary
   (c) Open-source
   (d) Free
   (e) CentOS, major, minor
   (f) Red Hat Enterprise
   (g) Ubuntu
   (h) Gentoo, Portage package management.
   (i) System libraries, Kernel.
   (j) hardware, software
   (k) process access
   (l) interface

2.
   (a) Superblock
   (b) XFS
   (c) vfat
   (d) firmware
   (e) superblock
   (f) name, directory
   (g) Find
   (h) file system
   (i) partitions
   (j) init

1.12 FURTHER READINGS

- Linux Documentation Project [http://www.tldp.org/]
- Documentation for Linux enthusiasts [http://www.linuxdocs.org/]
- Linux Man pages installed on local system.

1.13 MODEL QUESTIONS

1. Write a short note on the Linux operating system.
2. Differentiate between Proprietary, Open Source and Free software.
3. Describe any one Linux distribution.
4. Explain the architecture of the Linux operating system with diagram.
5. Describe the functioning of the Linux kernel subsystem.
6. What is the Linux file system? Explain any five types of file systems available.
7. Write short notes on Bootblock, Superblock, Inode and Itable.
8. List some standard directories available in the Linux operating system.
9. What are the LILO and GRUB bootloadrs of the Linux operating system?
10. Briefly describe the methods involved in installing the Linux operating system.
11. What is Hard Disk partitioning for Linux? Explain how to create new partitions in the Linux Operating system.
12. What are Linux Init and Runlevels?

UNIT - 2: LINUX BASICS

UNIT STRUCTURE

2.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- login to a Linux system
- create user accounts and groups
- learn about Linux processes
- know the file system
- learn all the manipulation features of files
- read, search and edit files

2.2 INTRODUCTION

In the previous unit we were introduced to the Linux operating system and the introductory linux operating system concepts that provided the basic grounding for a novice Linux user. We also were introduced to some new concepts related to the different software types like proprietary, open source and free software. The various Linux distributions available were also dealt with at length. The minimum requirement necessary to get introduced to the basic

functioning and operating of a Linux system starting from installing the system were thoroughly dealt with in the previous unit.

In this unit we will deal further into more in-depth topics that will deal with some similar but advanced operations of the Linux system. These include first logging into the Linux system, creating user accounts, handling and performing numerous file oriented operations using commands etc. Also in this unit we shall discuss the concept of processes and the file system in Linux. Finally we will provide a list of the different commands available in the Linux operating system that are used to perform several tasks related to files, directories etc in Linux.

2.3 GETTING STARTED

In order to work on a Linux system directly, we will need to provide a user name and password. We always need to authenticate to the system. Most PC-based Linux systems have two basic modes for a system to run in: either quick and sober in text console mode, which looks like DOS with mouse, multitasking and multi-user features, or in graphical mode, which looks better but consumes more system resources.

2.3.1 LOGGING IN

Graphical mode –

This is the default on most desktop computers. We connect to the system using graphical mode when we are first asked for a user name, and then, in a new window, to type the password. It is generally considered a bad idea to connect (graphically) using the root user name, the system adminstrator's account, since the use of graphics includes running a lot of extra programs, in root's case with a lot of extra permissions. To keep all risks as low as possible, we should use a normal user account to connect graphically. But there are enough risks to keep this in mind as a general advice, for all use of the root account. After entering the username/password combination, it can take a little while before the graphical environment is started, depending on the CPU speed, personal settings, etc.

Another common form for a prompt is this one:

```
[user@host dir]
```

In the above example, `user` will be the login name, `host` is the name of the machine we are working on, and `dir` an indication of the current location in the file system. To disconnect from the system in graphical mode, we need to close all terminal windows and other applications, and then hit the **logout** icon or find **Log Out** in the menu. Closing everything is not really necessary, and the system can do this for us, but session management might put all currently open applications back on the screen when we reconnect again, which takes longer and is not always the desired effect. However, this behavior is configurable. The login screen

appearing again, asking to enter user name and password, indicates a successful logout.

Text mode –

We are in the text mode when the whole screen is black, showing (in most cases white) characters. A text mode login screen typically shows some information about the machine we are working on, the name of the machine and a prompt for log in:

```
RedHat Linux Release 8.0 (Psyche)
blast login:_
```

The login is different from a graphical login where we have to hit the **Enter** key after providing the user name, because there are no buttons on the screen to click. Then we should type the password, followed by another **Enter**. As we enter the password, we do not see any indication of the entered text, not even an asterisk or the cursor. But this is normal on Linux and is done for security reasons. When the system has accepted a valid user, we get to see the home screen. In text mode we should log in as root only to do setup and configuration that absolutely requires administrator privileges, such as adding users, installing software packages, and performing network and other system configuration. Once finished, we should immediately leave the special account and resume work as a non-privileged user. Alternatively, some systems, like Ubuntu, force to use `sudo`, so that we do not need direct access to the administrative account. Logging out is done by entering the logout command, followed by **Enter**.

The power button

While Linux was not meant to be shut off without application of the proper procedures for halting the system, hitting the power button is equivalent to starting those procedures on newer systems. However, powering off an old system without going through the halting process might cause severe damage. We should always use the Shutdown option when logging out from the graphical interface or, when on the login screen look around for a shutdown button.

2.3.2 CREATING ACCOUNTS AND GROUPS

Users and groups are used on Linux for access control — that is, to control access to the system's files, directories, and peripherals. Linux offers relatively simple/coarse access control mechanisms by default. On Linux systems, as on all Unix like operating systems, all files, executables, and directories belong to a user and a group. Users all have a default group, but typically also belong to multiple additional groups. Permissions are built on top of this system. While files and directories all have an owner and a group, permissions can be set granularly so that files can be read, written, or executed by their owners, groups, or the "world" (i.e. all users on the system.) The permission system allows for any combination of **read/write/execute** permission for **owner/group/world** users. While we often think of "users" as correlating directly to the human beings who use the machine, many of the "users" on a Linux

system are special users created for specific applications, so that executables – particularly ones which are accessible over the network – have a very limited ability to affect the system. For example, the Apache web server runs as `www-data` on Debian/Ubuntu systems, to limit web users from gaining unnecessary access to the system.

All Linux systems have a **superuser** or "**root**" user account, which is the first user created on a system. The root user has special access to administer the system; root has the ability to read, write, and execute any file on the system and has the ultimate authority over the administration of the system. For this reason, we recommend limiting the use of the root user account as much as possible. Furthermore, it is generally a best practice to isolate users and applications to their own user accounts to limit the potential security risk that any application or user can pose to the system as a whole.

We create normal users using the `adduser` command in the following form:

```
adduser [username]
```

With the `adduser` command we can also be more specific with regards to what the user's home directory and default shell will be. The following command creates the user "*squire*" in the groups "*morris*" and "*leader*".

```
adduser  --home /home/squire \          ## specify a conventional home
                                        directory

         --shell /bin/bash |            ## specify bash as the default
                                        shell

         --ingroup   morris   leader    ## specify to which group this
           squire                       user should belong
```

This example specifies that the user's default shell will be `bash`, and that the location of the user's home directory will be `/home/squire`. Typically, on Linux systems home directories for users are created by default in the form of `/home/[username]/`.

By default, the **primary** or default user group is the same as the user's username. Additional useful options include:

- `-no-create-home` Disables the home directory for this user. Useful for system accounts.
- `--disabled-login` Prevents user-from logging in. Useful for system accounts and administrative accounts.

If we need to modify a user account after the fact, the `usermod` command may be helpful. The syntax for this command is `usermod [options] [username]` where `[username]` is the user's specified login name. Useful options include:

- `-L` or –lock: Locks the user account and prevents it from executing programs or logging in to the system.
- `-U` or –unlock: Unlocks a previously locked group and re-enables its access to the system.

- `-g` or –gid: Changes the default group for the user in question.
- `-G` or "–groups [groupname-1] [groupname-2]": Replaces the existing supplementary groups that the user account is associated with, but does not change the default group.
- `-a` or –append: Alters the function of -G so that the existing list of supplementary groups is maintained while adding all groups specified by -G.

Creating Groups

Users may be grouped together into a "group," and users may choose to join an existing group to utilize the privileged access it grants. The command for adding groups is `groupadd` or `groupdel`.

To list all groups on the system:

```
$ cat /etc/group
```

We create new groups with the `groupadd` command. The `groupadd` command creates a new group account using the values specified on the command line plus the default values from the system. The new group will be entered into the system files as needed.

```
# groupadd [-g gid [-o]] [-r] [-f] group
```

The `groupadd` command creates a new group account using the values specified on the command line and the default values from the system. The new group will be entered into the system files as needed. The options which apply to the `groupadd` command are –

`-g` *gid*

The numerical value of the group's ID. This value must be unique, unless the `-o` option is used. The value must be non-negative. The default is to use the smallest ID value greater than 500 and greater than every other group. Values between 0 and 499 are typically reserved for *system accounts*.

`-r`

This flag instructs `groupadd` to add a *system account*. The first available *gid* lower than 499 will be automatically selected unless the `-g` option is also given on the command line.

`-f`

This is the *force* flag. This will cause `groupadd` to exit with an error when the group about to be added already exists on the system. If that is the case, the group will not be altered (or added again).
This option also modifies the way `-g` option works. When we request a *gid* that it is not unique and we do not specify the `-o` option too, the group creation will fall back to the standard behavior (adding a group as if neither `-g` or `-o` options were specified).

We add users to a group with the gpasswd command:

```
# gpasswd -a [user] [group]
```

To delete existing groups:

```
# groupdel [group]
```

To remove users from a group:

```
# gpasswd -d [user] [group]
```

If the user is currently logged in, he/she must log out and in again for the change to have effect.

2.3.3 GETTING HELP

Getting help from Linux involves using one of the different ways to get any help or assistance when operating the Linux system. Help can be found for every functionality of the Linux operating system. The manual pages of Linux are an overwhelming source of documentation. They are, however, very structured. Reading man pages is usually done in a terminal window when in graphical mode, or just in text mode if preferable. To open the man pages we type the following at the prompt, followed by **Enter**:

```
yourname@yourcomp ~> man man
```

The documentation for **man** will be displayed on the screen after pressing **Enter**:

```
man(1)
man(1)

NAME
man - format and display the on-line manual pages
manpath - determine user's search path for man
pages

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

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

```
OPTIONS
-C  config_file
lines 1-27
```

We use the space bar to browse to the next page, and the back key to go back to the previous page. When we reach the end, **man** will usually quit and we get the prompt back. Typing **q** leaves the man page before reaching the end, or if the viewer does not quit automatically at the end of the page.

Each man page usually contains a couple of standard sections, as we can see from the **man man** example:

- The first line contains the name of the command we are reading about, and the id of the section in which this man page is located. The man pages are ordered in chapters. Commands are likely to have multiple man pages, for example the man page from the user section, the man page from the system admin section, and the man page from the programmer section.
- The name of the command and a short description are given, which is used for building an index of the man pages. We can look for any given search string in this index using the `apropos` command.
- The synopsis of the command provides a technical notation of all the options and/or arguments this command can take. An option can be considered as a way of executing the command. The argument is what we execute it on. Some commands have no options or no arguments. Optional options and arguments are put in between "[" and "]" to indicate that they can be left out.
- A longer description of the command is given.
- Options with their descriptions are listed. Options can usually be combined. If not so, this section will tell us about it.
- Environment describes the shell variables that influence the behavior of this command.
- Sometimes sections specific to this command are provided.
- A reference to other man pages is given in the "SEE ALSO" section. In between parentheses is the number of the man page section in which to find this command. Experienced users often switch to the "SEE ALSO" part using the ***/*** command followed by the search string SEE and press **Enter**.
- Usually there is also information about known bugs (anomalies) and where to report new bugs.
- There might also be author and copyright information.

The Info pages

In addition to the **man** pages, we can read the **Info** pages about a command, using the `info` command. These usually contain more recent information and are somewhat easier to use. The man pages for some commands refer to the Info pages.

We start by typing `info info` in a terminal window:

```
File: info.info,    Node: Top,    Next: Getting
Started,  Up: (dir)

Info: An Introduction
*********************

   Info is a program, which you are using now,
for reading
documentation of computer programs.    The GNU
Project distributes most
of its on-line manuals in the Info format, so you
need a program called
"Info reader" to read the manuals.  One of such
programs you are using
now.

   If you are new to Info and want to learn how
to use it, type the
command `h' now.  It brings you to a programmed
instruction sequence.

   To learn advanced Info commands, type `n'
twice.  This brings you to
`Info for Experts, skipping over the `Getting
Started' chapter.

* Menu:
* Getting Started::Getting started using an Info
reader.
* Advanced Info::Advanced commands within Info.
* Creating an Info File::How to make your own
Info file.
--zz-Info: (info.info.gz) Top, 24 lines --Top----
---------------------------

Welcome to Info version 4.2. Type C-h for help, m
for menu item.
```

Using the arrow keys we browse through the text and move the cursor on a line starting with an asterisk, containing the keyword about which we want info, and then hit **Enter**. We use the **P** and **N** keys to go to the previous or next subject. The space bar will move us one page further, no matter whether this starts a new subject or an Info page for another command. We use **Q** to quit. The **info** program has more information.

The –help option

Most GNU commands support the **–help**, which gives a short explanation about how to use the command and a list of available options. Below is the output of this option with the `cat` command:

```
userprompt@host: cat --help
Usage: cat [OPTION] [FILE]...
Concatenate FILE(s), or standard input, to
standard output.
```

```
-A, --show-all  equivalent to -vET
-b,  --number-nonblank  number  nonblank  output
lines
-e                      equivalent to -vE
-E, --show-ends     display $ at end of each line
-n, --number            number all output lines
-s, --squeeze-blank   never more than one single
blank line
-t                      equivalent to -vT
-T, --show-tabs      display TAB characters as ^I
-u                      (ignored)
-v, --show-nonprinting   use ^ and M- notation,
                        except for LFD and TAB
--help     display this help and exit
--version  output version information and exit

With no FILE, or when FILE is -, read standard
input.

Report bugs to <bug-textutils@gnu.org>.
```

2.3.4 PROCESSES

Processes –

Generally, on any operating system, we say we have so many programs running. These running programs introduce the concept of processes. A process is a program in execution. It is one of the fundamental abstractions in Unix Operating Systems, the other fundamental abstraction being files. Linux is a multi-user and multi-tasking operating system. A Linux process is a program in execution on a Linux system. Therefore, whenever a program is executed, a new process is created. A process also consumes resources like the file system, memory or other CPU resources. This gives rise to the need of process management in Linux.

Identifier for Linux Processes

In Linux, every process has a unique process Identifier (ID) associated to it. A process ID (i.e. PID) is a number which is uniquely assigned as soon as the process is created. The PID's are allocated sequentially as the processes are being created. However, it generally starts from 2, as PID=1 is reserved for 'init process. As we always expect, there is a maximum limit to the PID value. Hence, whenever the sequentially allocated PID reaches the maximum value, it wraps to the lower limit (generally 300) and the next PID's allocated are the available ones starting from the lower limit. The PID of the process, as the name suggests is its identifier. Hence, most of the operations being done on a process need the PID to be mentioned.

We shall see in the following sections, how do we see display all the processes with their PID's and various operations that can be performed on a process.

Listing Processes

At any moment, the Linux user can view the list of all the processes which have been created and not terminated. The Linux command used to view list of processes is `ps` which means 'process status' (Some authors also interpret it as 'process snapshot'). To see what all this Linux command has to offer in detail, the best source is the man-page.

Try out running the command –

```
$ ps

PID TTY          TIME CMD
1779 pts/0    00:00:00 bash
2176 pts/0    00:00:00 ps
```

We see two processes although we are sure there are other processes running as well. The `ps` command without any options just lists the processes which are created by the current terminal. The first one is the 'bash' which is the running linux shell by the terminal and other is the process created by `ps` command itself.

- **PID –** The process Identifier which is '1779' for 'bash' and '2176' for 'ps'.
- **TTY –** Stands for terminal-type and is the name of the console/terminal, the process is associated to.
- **TIME –** The CPU time since the process has started. It is confusing that why the CPU time for 'bash' process is '00:00:00'? This is because, CPU time is the time for which the process is being executed by the processor. However, when bash runs commands, say `ls` command , a child process `ls` is spawned and whatever execution and cpu utilization takes place, goes under the **ls** process and not **bash**. Bash process is just the parent process.
- **CMD –** Command run to create the process.

There are many more options offered by the Linux command `ps` to explore the various processes being launched in a system.

List all processes

```
$ps -e
```

If we look at the `ps` man page, `-e` option means "Select all processes", which implies now our list of displayed processes is not limited to the ones by the current terminal. Instead, we will be able to see all the currently running processes.

Running the above command, we see a huge list of processes. Hence, to read them through reasonably, we pipe the output to `more`.

```
$ ps -e | more

PID TTY      TIME   CMD
1 ?        00:00:00 init
2 ?        00:00:00 kthreadd
```

Types of Processes

Although there is no standard classification of types of processes in Linux, the segregation could be in interactive and non-interactive processes, foreground and background processes or daemon or batch processes. It can also be classified based on the status of the processes such as zombie processes. It is good enough if we comprehend all these various terminologies in the linux system.

Interactive processes

An interactive process is one which needs user's interaction while it is active. For example, when we launch a vi-editor, it is an interactive process. Another example could be the `telnet` command. Hence, the interactive processes have to be associated to a terminal.

Under the umbrella of interactive processes, we have Foreground and Background processes.

Foreground Process

A process is a foreground process if it is in focus and can be given input from the standard input. It blocks the shell until the foreground process is complete. When we run our commands on the terminal, they generally run as foreground processes. They block the terminal until it is complete. Although most of our linux commands are quick enough for us to even realize that.

Let us create our own program which sleeps for 10 seconds and then ourselves experience what waiting for the foreground process feels like.

The C source looks like:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int time = 10;
    sleep(time);
    printf("Slept for 10 secs\n");
    return 0;
}
```

Now we compile and run the program,

```
$ gcc wait_process.c -Wall -o wait_process
$ ./wait_process
```

The terminal is blocked by the running process, and not letting the user to do anything until the program is complete.

Background Process

Background processes are ones that are running, but in the background, not taking any user input from the terminal. It does not block the terminal, and allows us to use the terminal irrespective of the background process is complete or not. The key-character to make any new process to be run in background is '&'. How we use this character, is by suffixing it with the command, as in,

&

It is time, to run our '**wait_process**' program to run as a background, so that we can avoid the terminal to get blocked while the process is sleeping.

```
$ ./wait_process &
[1] 2534
$
```

To get these background and foreground handy, linux provides certain commands to view what is running and also switch any foreground process to background and vice versa. One can use command 'jobs' to see what all is running associated with the terminal.

```
$gedit wait_process.c&
[1] 2538
$./wait_process&
[2] 2546
$jobs
[1]-  Running                 gedit
wait_process.c &
[2]+  Running                 ./wait_process &
$
```

CHECK YOUR PROGRESS

1. Fill in the blanks

(a) All Linux systems have a _____ or root user account, which is the first user created on a system.

(b) We create normal users using the _____ command.

(c) The _____ command creates a new group account.

(d) We can read the Info pages about a command, using the ____ command.

(e) A process is a _____ in execution.

(f) In Linux, every process has a unique _____ associated to it.

(g) The ___ command is used to view list of processes.

(h) When we run our commands on the terminal, they generally run as _____ processes.

2.4 FILES AND FILE SYSTEM

For most users and for most common system administration tasks, it is enough to accept that files and directories are ordered in a tree-like structure. The computer, however, does not understand a thing about trees or tree-structures. Every partition has its own file system. By imagining all those file systems together, we can form an idea of the tree-structure of the entire system, but it is not as simple as that. In a file system, a file is represented by an **inode**, a kind of serial number containing information about the actual data that makes up the file: to whom this file belongs, and where is it located on the hard disk. Every partition has its own set of **inodes**; throughout a system with multiple partitions, files with the same inode number can exist. Each **inode** describes a data structure on the hard disk, storing the properties of a file, including the physical location of the file data. When a hard disk is initialized to accept data storage, usually during the initial system installation process or when adding extra disks to an existing system, a fixed number of inodes per partition is created. This number will be the maximum amount of files, of all types (including directories, special files, links etc.) that can exist at the same time on the partition. We typically count on having 1 inode per 2 to 8 kilobytes of storage.

At the time a new file is created, it gets a free inode. In that inode is the following information:

- File type (regular, directory, ...)
- Permissions on the file.
- Number of links to this file.
- File size.
- Date and time of creation, last read and change.

The only information not included in an inode, is the file name and directory. These are stored in the special directory files. By comparing file names and inode numbers, the system can make up a tree-structure that the user understands. Users can display inode numbers using the -i option to ls. The inodes have their own separate space on the disk.

File type –

1. Regular file(-)
2. Directory files(d) Special files
3. Block file(b)
4. Character device file(c)
5. Named pipe file or just a pipe file(p)
6. Symbolic link file(l)
7. Socket file(s)

1. **Regular file type –** These are the files which are indicated with “-” in ls -l output at the starting of the line. And these files are.
    a) Readable files or
    b) A binary files or
    c) Image files or

d) Compressed files etc.

2. **Directory file type –** These type of files contains regular files/folders/special files stored on a physical device. And this type of files will be in blue in color with link greater than or equal 2.
3. **Block file type –** These files are hardware files most of which are present in /dev.
4. **Character device files –** Provides a serial stream of input or output. Our terminals are example for this type of files.
5. **Pipe files –** The other name of pipe is a “named” pipe, which is sometimes called a FIFO. FIFO stands for “First In, First Out” and refers to the property that the order of bytes going in is the same coming out. The “name” of a named pipe is actually a file name within the file system.
6. **Symbolic link files –** These are linked files to other files. They are either Directory/Regular File. The inode number for this file and its parent files are same. There are two types of links ie soft and hard link.
7. **Socket files –** A socket file is used to pass information between applications for communication purpose

Permissions on the file – The Linux security model is based on the one used on UNIX systems, and is as rigid as the UNIX security model (and sometimes even more), which is already quite robust. On a Linux system, every file is owned by a user and a group user. There is also a third category of users, those that are not the user owner and do not belong to the group owning the file. For each category of users, read, write and execute permissions can be granted or denied.

The `ls -l` command also displays file permissions for these three user categories; they are indicated by the nine characters that follow the first character, which is the file type indicator at the beginning of the file properties line. As seen in the examples below, the first three characters in this series of nine display access rights for the actual user that owns the file. The next three are for the group owner of the file, the last three for other users. The permissions are always in the same order: read, write, execute for the user, the group and the others. Some examples:

```
marise:~> ls -l To_Do
-rw-rw-r--    1 marise  users      5 Jan 15 12:39
To_Do
marise:~> ls -l /bin/ls
-rwxr-xr-x    1 root    root   45948 Aug  9 15:01
/bin/ls*
```

The first file is a regular file (first dash). Users with user name *marise* or users belonging to the group *users* can read and write (change/move/delete) the file, but they cannot execute it (second and third dash). All other users are only allowed to read this file, but they cannot write or execute it (fourth and fifth dash).

For easy use with commands, both access rights or modes and user groups have a code as shown below.

Code	Meaning
0 or -	The access right that is supposed to be on this place is not granted.
4 or r	Read access is granted to the user category defined in this place.
2 or w	Write permission is granted to the user defined in this place.
1 or x	Execute permission is granted to the user category defined in this place

Access mode codes

Code	Meaning
u	user permissions
g	group permissions
o	permissions for others

User group codes

This straight forward scheme is applied very strictly, which allows a high level of security even without network security. Among other functions, the security scheme takes care of user access to programs, it can serve files on a need-to-know basis and protect sensitive data such as home directories and system configuration files.

2.5 SEARCHING, COPYING, MOVING AND RENAMING FILES

Searching files –

The `find` command allows us to search for files for which we know the approximate filenames. The simplest form of the command searches for files in the current directory and recursively through its subdirectories that match the supplied search criteria. We can search for files by name, owner, group, type, permissions, date, and other criteria.

Typing the following command at the prompt lists all files found in the current directory.

```
find .
```

To find files that match a specific pattern, we use the "**-name**" argument. We can use filename metacharacters (such as "*"), but should either put an escape character ("\") in front of each of them or enclose them in quotes. For example, if we want to find all the files that start with "pro" in the Documents directory, we would use the "`cd` Documents/" (without the quotes) command to change to the Documents directory, and then type the following command.

```
find . -name pro\*
```

All files in the current directory starting with “pro” are listed.

The find command defaults to being case sensitive. If we want the search for a word or phrase to be case insensitive, we should use the `–iname` option with the `find` command. It is the case insensitive version of the `–name` command.

If `find` does not locate any files matching the criteria, it produces no output.

Using the Locate Command

The `locate` command is faster than the `find` command because it uses a previously built database, whereas the `find` command searches in the real system, through all the actual directories and files. The `locate` command returns a list of all path names containing the specified group of characters. The basic form of the `locate` command finds all the files on the file system, starting at the root, that contain all or any part of the search criteria.

The `-b` option can be used with the `locate` command to find all files or directories that contain exactly and only the search criteria, as follows.

```
locate -b ‘\mydata’
```

The backslash (‘\’) in the above command is a globbing character, which provides a way of expanding wildcard characters in a non-specific file name into a set of specific filenames. A wildcard is a symbol that can be replaced by one or more characters when the expression is evaluated. The most common wildcard symbols are the question mark (?) which stands for a single character and the asterisk (*) which stands for a contiguous string of characters. In the above example, the backslash disables the implicit replacement of “mydata” by “*mydata*” end up with only results containing “mydata.”

The `mlocate` command is a new implementation of locate. It indexes the entire file system, but the search results only include files to which the current user has access. When we update the `mlocate` database, it keeps timestamp information in the database. This allows `mlocate` to know if the contents of a directory changed without reading the contents again and makes updates to the database faster and less demanding on the hard drive.

Copying files –

Like so many Linux features, we have a variety of options from which to choose when manipulating files and directories. We can also use wildcards when copying, moving, or deleting files and directories.

The following command copies a file:

```
cp <source> <destination>
```

So, to copy the file *sneakers.txt* to the directory *tigger* in the login directory, we move to the login directory and type:

```
cp sneakers.txt tigger
```

We can also use relative pathnames to copy the file, and can use both relative and absolute pathnames with `cp`. Our login directory is the parent of the directory *tigger*; *tigger* is one directory down from our login directory.

The options we can use with `cp` are the following:

- `-i` — interactive. Prompts to confirm if the file is going to overwrite a file in the destination. This is a handy option because it can help prevent making mistakes.
- `-r` — recursive. Rather than just copying all the files and directories, this will copy the whole directory tree, subdirectories and all.
- `-v` — verbose. shows the progress of the files being copied.

If we use `cp` with no options, we will not see much when the command is executed. Using an option, such as `-i`, can make the process a little more useful. Copying a file to a location that already has a file with the same name, brings about the question if we really want to overwrite (or replace) the file that is already present.

We now have the file *sneakers.txt* in the tigger directory, and will use `cp -i` to copy the file again to the same location.

```
[newuser@localhost newuser]$
cp -i sneakers.txt tigger
cp: overwrite 'tigger/sneakers.txt'?
```

To overwrite the file that is already present, we press Y and then [Enter]. In case we do not want to overwrite the file, we press N and [Enter].

Moving files –

To move files, we use the `mv` command. It is similar to the `cp` command, except that with `mv` the file is physically moved from one place to another, instead of being duplicated, as with `cp`.

Common options for `mv` include the following:

- `-I` — interactive. This prompts us if the selected file will overwrite an existing file in the destination directory.
- `-f` — force. Overrides the interactive mode and moves without prompting.
- `-v` — verbose. Shows a list of the files being moved.

Renaming files –

When we copy or move files, we can also rename them. To copy the file *sneakers.txt* from the login directory to the *tigger* subdirectory, we type the following:

```
cp sneakers.txt tigger
```

To copy and rename that file from *sneakers.txt* to *piglet.txt*, we type:

```
cp sneakers.txt tigger/piglet.txt
```

To move and rename the file, we substitute `mv` for `cp` in the above example. If we `cd` to *tigger* and then type `ls`, we see the file *piglet.txt*. If we just want to rename the file and keep its location, we just `mv` in our current directory:

```
mv sneakers.txt piglet.txt
```

2.6 DELETING, LINKING AND EDITING FILES

Deleting files –

Deleting files and directories in Linux is a straightforward process with the `rm` command. Options for removing files and directories include:

- `-i` — interactive. Prompts to confirm the deletion. This option can stop from deleting a file by mistake.
- `-f` — force. Overrides interactive mode and removes the file(s) without prompting.
- `-v` — verbose. Shows a list of files as they are being removed.
- `-r` — recursive. Will delete a directory and all (if any) files and the subdirectories it contains.

To delete the file **piglet.txt** from the **tigger** directory with the `rm` command:

```
rm piglet.txt
```

The `-i` (interactive) option is helpful, because it gives a second chance to think about whether or not we really want to delete the file.

```
[newuser@localhost newuser]$
rm -i piglet.txt
rm: remove 'piglet.txt'?
```

We can also delete files using the wildcard *****,

```
rm pig*
```

The above command will remove all files in the directory which start with the letters "**pig**."

We can also remove more than one file using one command:

```
rm piglet.txt sneakers.txt
```

Options for removing files and directories include the following:

- `-i` — interactive. Prompts to confirm the deletion, thus stopping us from deleting a file by mistake.
- `-f` — force. Overrides interactive mode and removes the file(s) without prompting.
- `-v` — verbose. Shows a list of files as they are being removed.

- `-r` — recursive. Will delete a directory and all (if any) files and the subdirectories it contains.

We can use `rmdir` to remove a directory (`rmdir foo`, for example), but only if the directory is empty. To remove directories with `rm`, we must specify the -r option.

For example, to recursively remove the directory **tigger** we would type:

```
rm -r tigger
```

To combine options, such as forcing a recursive deletion, we type:

```
rm -rf tigger
```

Linking files –

A link is nothing more than a way of matching two or more file names to the same set of file data. There are two ways to achieve this:

- **Hard link:** Associate two or more file names with the same inode. Hard links share the same data blocks on the hard disk, while they continue to behave as independent files. There is an immediate disadvantage: hard links cannot span partitions, because inode numbers are only unique within a given partition.

- **Soft link:** Soft link or symbolic link (or for short: symlink): a small file that is a pointer to another file. A symbolic link contains the path to the target file instead of a physical location on the hard disk. Since inodes are not used in this system, soft links can span across partitions.

Removing the target file for a symbolic link makes the link useless. Each regular file is in principle a hardlink. Hardlinks cannot span across partitions, since they refer to inodes, and inode numbers are only unique within a given partition. It may be argued that there is a third kind of link, the user-space link, which is similar to a shortcut in MS Windows. These are files containing meta-data which can only be interpreted by the graphical file manager. To the kernel and the shell these are just normal files. They may end in a .desktop or .lnk suffix.

Editing files –

Editing files in Linux can be performed by using different commands available. These are discussed below.

Vi editor: The vi editor (short for visual editor) is a screen editor which is available on almost all Unix systems. VI is a fast and powerful editor. VI has no menus but instead uses combinations of keystrokes in order to accomplish commands.

We start using vi, at the Unix prompt by typing vi followed by a file name. To edit an existing file we type in its name, and to create a new file we type in the name we wish to give to the new file.

```
%vi filename
```

vi has two modes – the command mode and the insert mode. It is essential to know which mode we are in at any given point in time. When in the command mode, letters of the keyboard will be interpreted as commands. When in the insert mode the same letters of the keyboard will type or edit text. vi always starts out in command mode. To move between the two modes we- type **i** to enter the insert mode, and hit **ESC** to leave insert mode and return to the command mode. In case we are not sure where we are, hitting **ESC** a couple of times puts us back in command mode.

Pico: pico (**Pi**ne **co**mposer) is a text editor for Linux. PICO is a very simple and easy-to-use text editor offering paragraph justification, cut/paste, and a spelling checker. Pico does not support working with several files simultaneously and cannot perform a find and replace across multiple files. It also cannot copy text from one file to another (though it is possible to read text into the editor from a file in its working directory). Pico does support search and replace operations. Pico's interface is in many ways very similar to that found in Windows editors, such as Notepad.

Ed: ed is a line editor for the Unix operating system. It was one of the first end-user programs hosted on the system and has been standard in Unix-based systems ever since.

Syntax: `ed [-C] [-p string] [-s] [-] [-x] filename`

-C	Encryption option; the same as the -x option, except that ed simulates a C command. The C command is like the X command, except that all text read in is assumed to have been encrypted.
-p string	Allows the user to specify a prompt string. By default, there is no prompt string.
-s or -	Suppresses the writing of character counts by e, r, and w commands, of diagnostics from e and q commands, and of the ! prompt after a !shell command.
-x	Encryption option; when used, ed simulates an X command and prompts the user for a key. The X command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the -x option.
filename	The name of the file to edit.

Assuming the file **myfile.txt** has the following lines in it:

Hello world
this is a test
of the ed editor

`ed myfile.txt` - Would open the file myfile.txt in the ed editor.

$	Reads the last line in this case "of the ed editor"
-	Moves back one line. For example, if $ was typed, if - was entered would move up to "this is a test". A number can be added to move more than one line at a time. For example, if at the last line typing -2 would move back to the first line in this example.
+	The + moves one line forward instead of back like the -. Like the - the + can have a number added after it, for example, +2 to move two lines instead of one line.
/.	Reads the first line in this case "Hello world" additional dots can be added to read other lines.
/text/	Searches for the text typed in-between the forward slashes and displays the next line that has this text. For example, if we were at the first line "Hello world" and typed /test/ the line displayed would be "this is a test"
I	Inserts text above the current line. Once finished we press CTRL + C to exit out of the insert option.
J	Joins lines of text.
T	Copies the line.
C	Used to change text. For example, if the current line was "this is a test". Typing c would allow us to enter a new line. Typing "this is a tester" and then pressing CTRL + C would replace the previous line with this new line.
R	Removes the specific line we are currently on.
X	Allows to skip to a line. For example, typing X and when asked to enter a key we enter 3 would display the line "of the ed editor"

2.7 LINUX COMMANDS

Linux has a variety of commands that can be used to create, manipulate, edit and perform several other functions related to the individual files and directories present in the system. Below are some of those commands that are used for performing different tasks related to files or directories in the linux system.

ls -

When invoked without any arguments, `ls` lists the files in the current working directory. A directory that is not the current working directory can be specified and `ls` will list the files there. The user also may specify any list of files and directories. In this case, all files and all contents of specified directories will be listed.

Files whose names start with "." are not listed, unless the `-a` flag is specified, the `-a` flag is specified, or the files are specified explicitly.

Without options, `ls` displays files in a bare format. This bare format however makes it difficult to establish the type, permissions, and size of the files. The most common options to reveal this information or change the list of files are:

- `-l` long format, displaying Unix file types, permissions, number of hard links, owner, group, size, last-modified date and filename
- `-f` do not sort. Useful for directories containing large numbers of files.
- `-F` appends a character revealing the nature of a file, for example, * for an executable, or / for a directory. Regular files have no suffix.
- `-a` lists all files in the given directory, including those whose names start with "." (which are hidden files in Unix). By default, these files are excluded from the list.
- `-R` recursively lists subdirectories. The command ls -R / would therefore list all files.
- `-d` shows information about a symbolic link or directory, rather than about the link's target or listing the contents of a directory.
- `-t` sort the list of files by modification time.
- `-h` print sizes in human readable format. (e.g., 1K, 234M, 2G, etc.)

The following example demonstrates the output of the `ls` command given two different arguments (pwd is a command that shows the present working directory, or in other words, the folder we are currently in):

```
$ pwd
/home/fred
$ ls -l
drwxr--r--  1 fred  editors 4096 drafts
-rw-r--r--  1 fred editors 30405 edition-32
-r-xr-xr-x  1 fred  fred   8460  edit
$ ls -F
drafts/
edition-32
edit*
```

In this example, the user **fred** has a directory named **drafts**, a regular file called **edition-32**, and an executable named **edit** in his home directory.

rm –

`rm` (short for remove) is a basic UNIX command used to remove objects such as files, directories, device nodes, symbolic links, and so on from the filesystem. To be more precise, `rm` removes references to objects from the filesystem, where those objects might have had multiple references (for example, a file with two different names), and the objects themselves are discarded only when all references have been removed and no programs still have open handles to the objects. This allows for scenarios where a program can open a file, immediately remove it from the filesystem, and then use it for temporary space, knowing that the file's space will be reclaimed after the program exits, even if it exits by crashing. `rm` generally does not destroy file data, since its purpose is really merely to unlink references, and the filesystem space freed may still contain leftover data from the removed file.

To remove a file named "foo" from a directory we type:

```
% rm foo
```

Normally, no output is produced by `rm`, since it typically only generates messages in the event of an error. The -v option can be used to get `rm` to detail successful removal actions.

rmdir –

`rmdir` (or rd) is a command which will remove an empty directory on a Unix, DOS, OS/2 or Microsoft Windows operating system. In Unix, Linux, and OS X, it is case sensitive, whereas DOS, OS/2 and Windows (95, 98, ME), we can type the characters in any combination of upper case and lower case letters, and rd/rmdir will recognize and remove that directory. Normal usage is straightforward where one types:

```
rmdir name_of_directory
```

where `name_of_directory` corresponds with the name of the directory one wishes to delete. There are options to this command such as `-p` in Unix which removes parent directories if they are also empty.

For example:

```
rmdir -p foo/bar/baz
```

will first remove `baz/`, then `bar/` and finally `foo/` thus removing the entire directory tree specified in the command argument.

`rmdir` will not remove a directory if it is not empty in UNIX. The correct way to remove a directory and all its contents recursively is with the `rm` command. For example:

```
rm -r foo/bar/baz
rm -rf foo/bar/baz
```

pwd –

The `pwd` command reports the full path to the *current directory*. The current directory is the directory in which a user is currently operating while using a command line interface. A command line interface is an all-text display mode and it is provided via a *console* (i.e., a display mode in which the entire screen is text only) or via a *terminal window* (i.e., a text-only window in a GUI). The full path, also called an *absolute path*, to a directory or file is the complete hierarchy of directories from the *root directory* to and including that directory or file. The root directory, which is designated by a forward slash (/), is the base directory on the *filesystem* (i.e., hierarchy of directories), and it contains all other directories, subdirectories and files on the system. Thus, the full path for any directory or file always begins with a forward slash.

`pwd` is one of the most basic commands in Linux and other Unix-like operating systems, along with `ls`, which is used to list the contents of the current directory, and `cd`, which is used to change the current directory.

Syntax:

```
pwd [option]
```

Unlike most commands, `pwd` is almost always used just by itself, i.e.,

```
pwd
```

That is, it is rarely used with its options and never used with *arguments* (i.e., file names or other information provided as inputs). Anything that is typed on the same line after `pwd`, with the exception of an option, is ignored, and no error messages are returned.

As an example, if a user with the username *janis* is in its home directory, then the above command would typically return */home/janis/* (because, by default, all home directories are located in the directory */home*). Likewise, if a user were currently working in directory */usr/share/config* (which contains a number of program configuration files), then the same command would return */usr/share/config*. `pwd` is useful for confirming that the current directory has actually been changed to what the user intended after using cd. For example, after issuing the cd command to change the current directory from /home/janis to /usr/share/config,

pwd could be used for confirmation; that is, the following sequence of commands would be issued:

```
cd /usr/share/config/
pwd
```

The standard version of pwd has a mere two options, both of which are employed only infrequently. The `-help` option is used as follows:

```
pwd -help
```

This option displays information about pwd, of which there is very little because it is such a simple command (i.e., it only has two options and accepts no arguments).

The other option is `– version`, which displays the version number, i.e.,

```
pwd -version
```

Although it is often thought of as standing for *present working directory*, pwd is actually an acronym for *print working directory*. The word *print* is traditional UNIX terminology for *write* or *display*, and it originated when computer output was typically printed on paper by default because CRT (cathode ray tube) display monitors were not yet widely available. Despite its extreme simplicity, pwd remains one of the most useful and popular of the commands for Unix-like operating systems. Actually, this simplicity is completely consistent with the Unix philosophy, which emphasizes small, specialized and modular programs rather than the large and complex programs that are favored by some other operating systems.

more –

`more` is a command to view (but not modify) the contents of a text file one screen at a time. The command-syntax is:

```
more [options] [file_name]
```

If no file name is provided, `more` looks for input from `stdin`.

Once `more` has obtained input, it displays as much as can fit on the current screen and waits for user input to advance, with the exception that a form feed (^L) will also cause more to wait at that line, regardless of the amount of text on the screen. In the lower-left corner of the screen is displayed the text "-More-" and a percentage, representing the percent of the file that more has paged through. (This percentage includes the text displayed on the current screen.) When more reaches the end of a file (100%) it exits. The most common methods of navigating through a file are Enter, which advances the output by one line, and Space, which advances the output by one screen.

Options: Options are typically entered before the file name, but can also be entered in the environment variable $MORE. Options entered in the actual command line will override those entered in the $MORE environment variable. Available options may vary between Unix systems, but a typical set of options is as follows:

- `-num`: This option specifies an integer which is the screen size (in lines).
- `-d`: more will prompt the user with the message "[Press space to continue, 'q' to quit.]" and will display "[Press 'h' for instructions.]" instead of ringing the bell when an illegal key is pressed.
- `-l`: more usually treats ^L (form feed) as a special character, and will pause after any line that contains a form feed. The -l option will prevent this behavior.
- `-f`: Causes more to count logical, rather than screen lines (i.e., long lines are not folded).
- `-p`: Do not scroll. Instead, clear the whole screen and then display the text.
- `-c`: Do not scroll. Instead, paint each screen from the top, clearing the remainder of each line as it is displayed.
- `-s`: Squeeze multiple blank lines into one.
- `-u`: Backspaces and carriage returns to be treated as printable characters;
- `+/`: The **+/** option specifies a string that will be searched for before each file is displayed. (Ex.: more +/Preamble gpl.txt)
- `+num`: Start at line number num.

less –

`less` is a program similar to `more`, but which allows backward movement in the file as well as forward movement. Also, `less` does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like vi. `Less` uses **termcap** (or terminfo on some systems), so it can run on a variety of terminals. There is even limited support for hardcopy terminals. (On a hardcopy terminal, lines which should be printed at the top of the screen are prefixed with a caret.) `less` can be invoked with options to change its behaviour, for example, the number of lines to display on the screen. A few options vary depending on the operating system. While `less` is displaying the file, various commands can be used to navigate through the file. These commands are based on those used by both more and vi. It is also possible to search for character patterns in the file. By default, `less` displays the contents of the file to the standard output (one screen at a time). If the file name argument is omitted, it displays the contents from standard input (usually the output of another command through a pipe). If the output is redirected to anything other than a terminal, for example a pipe to another command, `less` behaves like `cat`.

Syntax:

```
less [options] [file_name]
```

Options:

- `- g`: Highlights just the current match of any searched string.
- `- I`: Case-insensitive searches.
- `- M`: Shows more detailed prompt, including file position.
- `- N`: Shows line numbers (useful for source code viewing).

- - S: Disables line wrap ("chop long lines"). Long lines can be seen by side scrolling.
- - ?: Shows help.

Example –

```
less -M readme.txt   # Read "readme.txt."
```

grep –

`grep` is a command-line utility for searching plain-text data sets for lines matching a regular expression. `Grep` was originally developed for the Unix operating system, but is available today for all Unix-like systems. Its name comes from the `ed` command *g/re/p* (***g****lobally search a* ***r****egular* ***e****xpression* and ***p****rint*), which has the same effect: doing a global search with the regular expression and printing all matching lines. `Grep` searches files specified as arguments, or, if missing, the program's standard input. By default, it reports matching lines on standard output, but specific modes of operation may be chosen with command line options. A simple example of a common usage of grep is the following, which searches the file *fruitlist.txt* for lines containing the text string *apple*:
`$ grep apple fruitlist.txt`

sort –

The `sort` command is used to sort the lines in a text file.

Syntax:

```
sort [options]... [file]
```

Options:

`-b`	Ignores spaces at beginning of the line.
`-c`	Check whether input is sorted, does not sort
`-d`	Uses dictionary sort order and ignores the punctuation.
`-f`	Ignores caps
`-g`	Compare according to general numerical value
`-i`	Ignores nonprinting control characters.
`-k`	Start a key at POS1, end it at POS2 (origin 1)
`-m`	Merges two or more input files into one sorted output.
`-M`	Treats the first three letters in the line as a month (such as may.)
`-n`	Sorts by the beginning of the number at the beginning of the line.
`-o`	Write result to FILE instead of standard output
`-r`	Sorts in reverse order
`-s`	Stabilize sort by disabling last-resort comparison
`-t`	Use SEP instead of non-blank to blank transition
`-T`	Uuse DIR for temporaries, not $TMPDIR or /tmp; multiple options specify multiple directories
`-u`	If line is duplicated only display once

`-z`	End lines with 0 byte, not newline
`+fields`	Sorts by fields , usually by tabs
`filename`	The name of the file that needs to be sorted.
`-o outputfile`	Sends the sorted output to a file.

cat –

The `cat` program is a standard Unix utility that concatenates and lists files. The name is an abbreviation of catenate, a synonym of concatenate. When the "cat" program is given files in a sequence as arguments, it will output their contents to the standard output in the same sequence. It mandates the support of one option flag, u (unbuffered), by which each byte is written to standard output without buffering as it is read. Many operating systems do this by default and ignore the flag. If one of the input filenames is specified as a single hyphen (-), then cat reads from standard input at that point in the sequence. If no files are specified, cat reads from standard input only.

Syntax:

```
cat [options] [file_names]
```

`cat` will concatenate (put together) the input files in the order given, and if no other commands are given, will print them on the screen as standard output. It can also be used to print the files into a new file as follows:

```
cat [options] [file_names] > newfile.txt
```

head –

The `head` command reads the first few lines of any text given to it as an input and writes them to standard output (which, by default, is the display screen).

Syntax:

```
head [options] [file(s)]
```

The square brackets indicate that the enclosed items are optional. By default, `head` returns the first ten lines of each file name that is provided to it. For example, the following will display the first ten lines of the file named *aardvark* in the current directory (i.e., the directory in which the user is currently working):

```
head aardvark
```

If more than one input file is provided, `head` will return the first ten lines from each file, precede each set of lines by the name of the file and separate each set of lines by one vertical space. The following is an example of using head with two input files:

```
head aardvark armadillo
```

If it is desired to obtain some number of lines other than the default ten, the `-n` option can be used followed by an integer indicating the number of lines desired. For example, the above example could be modified to display the first 15 lines from each file:

```
head -n15 aardvark armadillo
```

-n is a very tolerant option. For example, it is not necessary for the integer to directly follow it without a space in between. Thus, the following command would produce the same result:

```
head -n   15 aardvark armadillo
```

In fact, the letter *n* does not even need to be used at all. Just the hyphen and the integer (with no intervening space) are sufficient to tell head how many lines to return. Thus, the following would produce the same result as the above commands:

```
head -15 aardvark armadillo
```

head can also return any desired number of bytes (i.e., a sequence of eight bits and usually long enough to represent a single character) from the start of each file rather than a desired number of lines. This is accomplished using the *-c* option followed by the number of bytes desired. For example, the following would display the first five bytes of each of the two files provided:

```
head -c 5 aardvark anteater
```

When head counts by bytes, it also includes the *newline character*, which is a *non-printing* (i.e, invisible) character that is designated by a backslash and the letter *n* (i.e., *n*). Thus, for example, if there are three new, blank lines at the start of a file, they will be counted as three characters, along with the *printing characters* (i.e., characters that are visible on the monitor screen or on paper). The number of bytes or lines can be followed by a multiplier suffix. That is, adding the letter *b* directly after the number of bytes multiplies it by 512, *k* multiplies it by 1024 and *m* multiplies it by 1048576. Thus, the following command would display the first five kilobytes of the file aardvark:

```
head -c5k aardvark
```

The -c option is less tolerant than the -n option. That is, there is no default number of bytes, and thus some integer must be supplied. Also, the letter *c* cannot be omitted as can the letter *n*, because in such case head would interpret the hyphen and integer combination as the -n option. Thus, for example, the following would produce an error message something like *head: aardvark: invalid number of bytes*:

```
head -c aardvark
```

If head is used without any options or *arguments* (i.e., file names), it will await input from the keyboard and will successively repeat (i.e., each line will appear twice) on the monitor screen each of the first ten lines typed on the keyboard. If it were desired to repeat some number of lines other than the default ten, then the -n option would be used followed by the integer representing that number of lines (although, again, it is not necessary to include the letter *n*), e.g.,

```
head -n3
```

As is the case with other command line (i.e., all-text mode) programs in Linux and other Unix-like operating systems, the output from head can *redirected* from the display monitor to a file or printer using the *output redirection operator* (which is

represented by a rightward-pointing angular bracket). For example, the following would copy the first 12 lines of the file *Yuriko* to the file *December*:

```
head -n 12 Yuriko > December
```

If the file named December did not yet exist, the redirection operator would create it; if it already existed, the redirection operator would overwrite it. To avoid erasing data on an existing file, the *append operator* (which is represented by two consecutive rightward pointing angle brackets) could be used to add the output from head to the end of a file with that name if it already existed (or otherwise create a new file with that name), i.e.,

```
head -n 12 Yuriko >> December
```

The output from other commands can be sent via a *pipe* (represented by the vertical bar character) to head to use as its input. For example, the following sends the output from the *ls* command (which by default lists the names of the files and directories in the current directory) to head, which, in turn, displays the first ten lines of the output that it receives from ls:

```
ls | head
```

This output could easily be redirected, for example to the end of a file named *file1* as follows:

```
ls | head >> file1
```

It could also be piped to one or more filters for additional processing. For example, the *sort* filter could be used with its *-r* option to sort the output in reverse alphabetic order prior to appending file1:

```
ls | head | sort -r >> file1
```

The *-q* (i.e., *quiet*) option causes head to not show the file name before each set of lines in its output and to eliminate the vertical space between each set of lines when there are multiple input sources. Its opposite, the *-v* (i.e., *verbose*) option, causes head to provide the file name even if there is just a single input file.

tail –

The `tail` command reads the final few lines of any text given to it as an input and writes them to *standard output* (which, by default, is the monitor screen).

Syntax:

```
tail [options] [filenames]
```

The square brackets indicate that the enclosed items are optional. By default, tail returns the final ten lines of each file name that is provided to it. For example, the following command will *print* (traditional Unix terminology for *write*) the last ten lines of the file named *aardvark* in the *current directory* (i.e., the director in which the user is currently working) to the display screen:

```
tail aardvark
```

If more than one input file is provided, `tail` will print the last ten lines from each file to the monitor screen. Each set of lines will be preceded by the name of the file and separated by one vertical space from other sets of lines. The following is an example of using tail with multiple input files:

```
tail aardvark anteater armadillo
```

If it is desired to print some number of lines other than the default ten, the *-n* option can be used followed by an integer indicating the number of lines desired. For example, to print the final 15 lines from each file in the above example, the command would be modified as follows:

```
tail -n15 aardvark anteater armadillo
```

-n is a very tolerant option. For example, it is not necessary for the integer to directly follow it without a space in between. Thus, the following command would produce the same result:

```
tail -n   15 aardvark anteater armadillo
```

In fact, the letter *n* does not even need to be used at all. Just the hyphen and the integer (with no intervening space) are sufficient to tell tail how many lines to print. Thus, the following would produce the same result as the above commands:

```
tail -15 aardvark anteater armadillo
```

tail can also print any desired number of *bytes* (i.e., a sequence of eight bits and usually long enough to represent a single character) from the end of each file rather than a desired number of lines. This is accomplished using the *-c* option followed by the number of bytes desired. For example, to view the final five bytes of each of the two files aardvark and anteater, the following command would be used:

```
tail -c 5 aardvark anteater
```

When `tail` counts by bytes, it also includes the *newline character*, which is a *non-printing* (i.e, invisible) character that is designated by a backward slash and the letter *n* (i.e., *n*). Thus, for example, if there are three new, blank lines at the end of a file, they will be counted as three characters, along with the *printing* characters (i.e., characters that are visible on the monitor screen or paper). The number of bytes or lines can be followed by a multiplier suffix. That is, adding the letter *b* directly after the number of bytes multiplies it by 512, *k* multiplies it by 1024 and *m* multiplies it by 1048576. Thus, the following command would print the last five kilobytes of the file aardvark:

```
tail -c5k aardvark
```

The `-c` option is less tolerant than the `-n` option. That is, there is no default number of bytes, and thus some integer must be supplied. Also, the letter *c* cannot be omitted as can the letter *n*, because in such case tail would interpret the hyphen and integer combination as the -n option. Thus, for example, the following would produce an error message something like *tail: aardvark: invalid number of bytes*:

```
tail -c aardvark
```

If `tail` is used without any options or *arguments* (i.e., inputs), it will await input from the keyboard and will successively repeat (i.e., each line will appear twice) on the monitor screen each of the final ten lines typed on the keyboard. If it were desired to repeat some number of lines other than the default ten, then the -n option would be used followed by the integer representing that number of lines (although, again, it is not necessary to include the letter *n*), e.g.,

```
tail -n3
```

As is the case with other *command line* (i.e., all-text mode) programs in Unix-like operating systems, the output of tail can be *redirected* from the monitor to a file or printer using the *redirection operator* (which is represented by a rightward pointing angular bracket). For example, the following would write the final 12 lines of the file *Yuriko* to the file *December*:

```
tail -n 12 Yuriko > December
```

If the file named *December* did not yet exist, the redirection operator would create it; if it already existed, the redirection operator would overwrite it. To avoid erasing data on an existing file, the *append operator* (which is represented by two rightward pointing angular brackets) could be used to add the output from tail to the end of a file with that name if it already existed (or otherwise create a new file with that name), i.e.,

```
tail -n 12 Yuriko >> December
```

The output from other commands can be *piped* (i.e., sent) to tail to use as its input. For example, the following sends the output from the *ls* command (which by default lists the names of the files and directories in the current directory) to tail, which, in turn, prints the final ten lines of the output that it receives from ls to the monitor screen:

```
ls | tail
```

This output could easily be redirected, for example to a file named *last_filenames* as follows:

```
ls | tail >> last_filenames
```

It could also be piped to one or more filters for additional processing. For example, the *sort* filter could be used with its *-r* option to sort the output in reverse alphabetic order prior to writing to a file:

```
ls | tail | sort -r >> last_filenames
```

The *-q* (i.e., *quiet*) option causes tail to not print the file name before each set of lines and to eliminate the vertical space between each set of lines when there are multiple input sources. The *-v* (i.e., *verbose*) option causes tail to print the file name even if there is just a single input file. Tail could be viewed as a counterpart of the *head* command, which always starts reading from the beginning of files and which can continue until any specified distance from the beginning. However, there are a few differences. Perhaps the most useful of these is that tail is somewhat more flexible in that, in addition to being able to start reading any specified distance from the end of a file, it can also

start at any specified distance from the beginning of a file. Tail can be instructed to begin printing from some number of lines or bytes from the start of a file by preceding the number with a plus sign instead of a minus sign. For example, the following would print each of the designated files to the display monitor beginning with the seventh line and until the end:

```
tail +7 aardvark anteater armadillo
```

The c option could be used to tell tail to print each of the designated files beginning with the seventh byte instead of the seventh line:

```
tail +7c aardvark anteater armadillo
```

A particularly common application for tail is examining the most recent entries in log files. This is because the newest entries are appended to the ends of such files, which tail excels in showing. As log files can be a rather long, this can eliminate a lot of scrolling that would be necessary if some other command were used to read them. For example, the most recent entries to the log */var/log/messages* can easily be viewed by using the following:

```
tail /var/log/messages
```

As is the case with other programs on Unix-like operating systems, additional information, including variations on specific versions, can be obtained about the tail and head commands by consulting the built-in manual and information pages using commands such as:

```
man tail
```

or

```
info head
```

wc –

The program reads either standard input or a list of files and generates one or more of the following statistics: newline count, word count, and byte count. If a list of files is provided, both individual file and total statistics follow.

Example:

```
$ wc foo bar
     40     149     947 foo
   2294   16638   97724 bar
   2334   16787   98671 total
```

The first column is the count of newlines, meaning that the text file **foo** has 40 newlines while bar has 2294 newlines- resulting in a total of 2334 newlines. The second column indicates the number of words in each text file showing that there are 149 words in **foo** and 16638 words in bar- giving a total of 16787 words. The last column indicates the number of characters in each text file, meaning that the file **foo** has 947 characters while bar has 97724 characters- 98671 characters all in all.

Usage:

- `wc -l <filename>` print the line count
- `wc -c <filename>` print the byte count
- `wc -m <filename>` print the character count
- `wc -L <filename>` print the length of longest line
- `wc -w <filename>` print the word count

tee –

`tee` is normally used to *split* the output of a program so that it can be displayed and saved in a file. The command can be used to capture intermediate output before the data is altered by another command or program. The `tee` command reads standard input, then writes its content to standard output. It simultaneously copies the result into the specified file(s) or variables. The syntax differs depending on the command's implementation:

Syntax:

```
tee [-a][-i] [File...]
```

Arguments:

- **File** One or more files that will receive the "tee-d" output.

Flags:

- **- a:** Appends the output to the end of File instead of writing over it.
- **- i:** Ignores interrupts.

The command returns the following exit values (exit status):

- **0:** The standard input was successfully copied to all output files.
- **>0:** An error occurred.

ps –

The `ps` command (short for "process status") displays the currently-running processes.

For example,

```
# ps
  PID TTY          TIME CMD
 7431 pts/0    00:00:00 su
 7434 pts/0    00:00:00 bash
18585 pts/0    00:00:00 ps
```

top –

The `top` command is useful for monitoring systems continuously for processes that take more system resources like the CPU time and the memory. `top` periodically updates the display showing the high resource consuming processes at the top. top is an excellent aid in checking a system. If the system is giving a slow response time, we just run `top` and look for statistics like - load average, CPU utilization, memory and swap usage and the top CPU and

memory intensive processes, which would give us a fair idea of what is happening in the system.

Command Line Options

Option	Description
`-h, -v`	print program version, usage prompt and quit
`-b`	work in "batch" mode. No inputs are accepted and top quits after -n number iterations
`-n`	work for the given number of iterations and quit.
`-d`	delay time interval between iterations in in the format ss[.tt] seconds
`-H`	Show threads. By default, processes are displayed. LWP ids are displayed under PID.
`-i`	do not display "idle" processes
`-u`	Report only processes with the given effective user id or user name
`-U`	Report only processes with the given real, effective, saved or filesystem user id or user name
`-p`	Monitor the processes identified by the given list of process ids.
`-s`	work in secure mode
`-S`	Display cumulative CPU time for each process and its children which have died and have been waited for by it

tar –

The `tar` (i.e., tape archive) command is used to convert a group of files into an archive. An archive is a single file that contains any number of individual files plus information to allow them to be restored to their original form by one or more extraction programs. Archives are convenient for storing files as well as for transmitting data and distributing programs. Moreover, they are very easy to work with, often much more so than dealing with large numbers of individual files. Although tar was originally designed for backups on magnetic tape, it can now be used to create archive files anywhere on a filesystem. Archives that have been created with `tar` are

commonly referred to as **tarballs**. Unlike some other archiving programs, and consistent with the Unix philosophy that each individual program should be designed to do only one thing but do it well, `tar` does not perform compression. However, it is very easy to compress archives created with tar by using specialized compression utilities.

Syntax:

```
tar option(s) archive_name file_name(s)
```

`tar` has numerous options, many of which are not frequently used. Unlike many commands, `tar` requires the use of at least one option, and usually two or more are necessary. `tar` files are created by using both the -c and -f options. The former instructs `tar` to create an archive and the latter indicates that the next argument (i.e., piece of input data in a command) will be the name of the new archive file. Thus, for example, the following would create an archive file called file.tar from the three files named file1, file2 and file3 that are located in the current directory (i.e., the directory in which the user is currently working):

```
tar -cf file.tar file1 file2 file3
```

It is not absolutely necessary that the new file have the .tar extension; however, the use of this extension can be very convenient because it allows the type of file to be visually identified. It is necessary, however, that the -f option be the final option in a sequence of contiguous, single-letter options; otherwise, the system will become confused as to the desired name for the new file and will use the next option in the sequence as the name. The -v (i.e., verbose) option is commonly used together with the -c and -f options in order to display a list of the files that are included in the archive. In such case, the above example would become

```
tar -cvf file.tar file1 file2 file3
```

`tar` can also be used to make archives from the contents of one or more directories. The result is recursive; that is, it includes all objects (e.g., directories and files) within each level of directories. For example, the contents of two directories named dir1 and dir2 could be archived into a file named dir.tar with the following:

```
tar -cvf dir.tar dir1 dir2
```

It is often convenient to use `tar` with a wildcard (i.e., a character which can represent some specific class of characters or sequence of characters). The following example uses the **star** wildcard (i.e., an asterisk), which represents any character or sequence of characters, to create an archive of every object in the current directory:

```
tar -cf *
```

By default, `tar` creates an archive of copies of the original files and/or directories, and the originals are retained. However, they can be removed when using `tar` by adding the **-remove-files option**. As it has no compression and decompression capabilities of its own, `tar` is commonly used in combination with an external

compression utility. A very handy feature of the GNU version (which is standard on Linux) is the availability of options that will cause standard compression programs to compress a new archive file as soon as it has been created. They are -j (for bzip2), -z (for gzip) and -Z (for compress). Thus, for example, the following would create an archive named files.tar.bz2 of the files file4, file5 and file6 that is compressed using bzip2:

```
tar -cvjf files.tar.bz2 file4 file5 file6
```

`tar` can also be used for unpacking tar files. However, before doing this, there are several steps that should be taken. One is to confirm that sufficient space is available on the hard disk drive (HDD). Another is to move to an empty directory (which usually involves creating one with an appropriate name) to prevent the reconstituted files from cluttering up the current directory and overwriting any files or directories with same names that are in it. In addition, if the archive has been compressed, it must first be decompressed using the appropriate decompression program (which can usually be determined by the filename extension). In order to unpack a tar file, the -x (for extract) and -f options are required. It is also common to add the -v option to provide a running listing of the files being unpacked. Thus, for example, to unpack the archive file.tar created in a previous example the following would be used:

```
tar -xvf file.tar
```

Just as options are available to allow three compression programs to automatically compress newly created tar files, the same options can be used to have the compression programs automatically decompress tar files prior to extraction. Thus, for instance, the following would decompress and extract the contents of the compressed archive files.tar.bz2 that was created in an above example:

```
tar -xjvf files.tar.bz2
```

Files can be added to an existing archive using the **-r** option. As is always the case with **tar**, it is also necessary to use the **-f** option to indicate that the following string (i.e., sequence of characters) is the name of the archive. For example, the following would append a file named file7 to file.tar:

```
tar -rf file.tar file7
```

The **–delete** option allows specified files to be completely removed from a **tar** file (except when the tar file is on magnetic tape). However, this is different from an extraction, as copies of the removed files are not made and placed in the current directory. Thus, for example, the files file1 and file2 can be removed from file.tar with the following:

```
tar -f file.tar -delete file1 file2
```

The **-t** option tells tar to list the contents of an uncompressed archive without performing an extraction. Thus, the following would list the contents of file.tar:

```
tar -tf file.tar
```

One of the very few options that can be used alone with tar is –**help**, which provides a relatively compact listing of the numerous options that are available. Another is **–version**, which shows the version number for the installed tar program as well as its copyright information.

unzip –

`unzip` will list, test, or extract files from a ZIP archive, commonly found on MS-DOS systems. The default behavior (with no options) is to extract into the current directory (and subdirectories below it) all files from the specified ZIP archive. A companion program, zip(1L), creates ZIP archives; both programs are compatible with archives created by PKWARE's PKZIP and PKUNZIP for MS-DOS, but in many cases the program options or default behaviors differ.

nice –

`nice` is used to invoke a utility or shell script with a particular priority, thus giving the process more or less CPU time than other processes. A niceness of −20 is the highest priority and 19 or 20 is the lowest priority. The default niceness for processes is inherited from its parent process, usually 0. **nice** becomes useful when several processes are demanding more resources than the CPU can provide. In this state, a higher priority process will get a larger chunk of the CPU time than a lower priority process. If the CPU can deliver more resources than the processes are requesting, then even the lowest priority process can get up to 99% of the CPU. Only the superuser (root) may set the niceness to a smaller (higher priority) value. On Linux it is possible to change /etc/security/limits.conf to allow other users or groups to set low nice values. If a user wanted to compress a large file, but not slow down other processes, they might run the following:

```
$ nice -n 19 tar cvzf archive.tgz largefile
```

netstat –

The `netstat` command in Linux is a very useful tool when dealing with networking issues. This command is capable of producing information related to network connections, routing tables, interface statistics etc. This utility also helps the network administrators to keep an eye on the invalid or suspicious network connections.

Syntax:

```
netstat [options]...
```

Disk related commands

It is very common to have to examine disk use to determine where there is free space, where the disk may be nearing full and where we may need to add disk or move files. One of the most essential commands used to examine disks is the `df` command. The `df` command is used to display information about mounted file systems. By default the `df` command will typically return disk information in kilobytes. Since there can be variation on this

default behavior it is often nice to use the `-k` option which will force `df` to displays disk space usage in kilobytes as seen in this example:

```
$ df –k
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/hdf1	18727836	2595832	15180656	15%	**/**
/dev/hda1	101086	5945	89922	7%	/boot
None	128560	**0**	128560	0%	/dev/shm

These results show that two file systems are mounted. The Filesystem column of the output shows the path to the disk device which is currently mounted at the Mounted on location. The 1K-blocks column displays the size of the entire partition while the Used and Available columns indicate the number of 1K blocks on that device used and available. The Use% column will show what percentage of the disk is currently used and is the quickest way to identify disks which are getting full.

To get the display in a friendlier format, the –h option can be used:

```
$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hdf1	18G	2.5G	15G	15%	/
/dev/hda1	99M	5.9M	88M	7%	/boot
None	126M	0	126M	0%	/dev/shm

The `-h` option will show output in the familiar gigabyte, megabyte or kilobyte (G, M or K respectively) scales. This makes things more human readable, hence the h. It is easy to see that the first file system is 18GB in size, with 2.5GB used and 15GB of available free space. It is mounted on the root (/) mount point. The second file system is 99MB in size, with 5.9MB used and 88MB of available free space. It is mounted on the /boot mount point. This output also shows a shared memory space of 126MB currently available (/dev/shm).

Creating a File System

Some file systems are created automatically during the Linux installation process. There are many different types of file systems. Microsoft Windows administrators are familiar with filesystems like FAT16, FAT32, and NTFS. The comparable options on Linux are ext2, ext3, and Linux-swap. The differences between these filesystem types are beyond the scope of this book. During the lifetime of a Linux system it is not uncommon to want to add additional disk space to a system by adding disks or replace a current drive with a larger capacity. Here are some of the most useful commands for setting up disks:

Note: One needs root privileges to perform most of these tasks.

Command	Function
`fdisk`	Partition a hard disk
`fsck`	Check and optionally repair one or more Linux file systems
`mkdir`	Make a new file directory
`mkfs`	Make a file system
`mkswap`	Make a swap area on a device or in a file
`mount`	Mount a file system (umount to unmount)
`parted`	Disk partitioning and partition resizing program. It allows the user to create, destroy, resize, move and copy ext2, ext3, Linux-swap, FAT and FAT32 partitions.
`sfdisk`	List the size of a partition, the partitions on a device, check the partitions on a device, and repartition a device.

Commands for file system creation

CHECK YOUR PROGRESS

2. Fill in the blanks

(a) Each inode describes a _____ on the hard disk.

(b) The only information not included in an inode, is the _____ and _____.

(c) A _____ file is used to pass information between applications for communication purpose.

(d) On a Linux system, every file is owned by a ____ and a _____ user.

(e) The ____ command allows us to search for files for which we know the approximate _____.

(f) The _____ command is faster than the ____ command because it uses a previously built database.

(g) We use the ____ command to move files.

(h) _____ cannot span partitions, because inode numbers are only unique within a given _____.

(i) ____ does not support working with several files simultaneously and cannot perform a _____ across multiple files.

(j) __ is used to remove objects such as files, directories, device nodes, symbolic links, and so on from the _____.

2.8 LET US SUM UP

- ***Users*** and ***groups*** are used on Linux for access control.
- The Linux ***access-control*** system allows for any combination of read/write/execute permission for owner/group/world users.
- A Linux ***process*** is a program in execution on a Linux system.
- An ***interactive process*** is one which needs user's interaction while it is active.
- A process is a ***foreground process*** if it is in focus and can be given input from the standard input.
- ***Background processes*** are ones that are running in the background, not taking any user input from the terminal.
- In a Linux ***file system***, a file is represented by an inode.
- ***Hard links*** share the same data blocks on the hard disk, while they continue to behave as independent files.
- A ***Soft link*** contains the path to the target file instead of a physical location on the hard disk..

2.9 ANSWERS TO CHECK YOUR PROGRESS

1.
(a) superuser.
(b) adduser.
(c) groupadd.
(d) info.
(e) program.
(f) process Identifier.
(g) ps.
(h) foreground.

2.
(a) data structure.
(b) file name, directory.
(c) socket.
(d) user, group.
(e) find, approximate filenames.
(f) locate, find.
(g) mv.
(h) hard links, partition.
(i) pico, find and replace.
(j) rm, filesystem.

2.10 FURTHER READINGS

- Linux Documentation Project [http://www.tldp.org/]
- Documentation for Linux enthusiasts [http://www.linuxdocs.org/]
- Linux Man pages installed on local system.

2.11 MODEL QUESTIONS

1. What are accounts and groups in Linux? How do you use them?
2. What are Linux processes? Describe their types.
3. Write a note on Linux file system.

4. What are the different options for searching files in Linux?
5. How do you copy, move and rename files in Linux?
6. What is linking in Linux? Describe its use.
7. Differentiate between the **more** and **less** commands in Linux.
8. Illustrate the use of the **head** and the **tail** commands.

UNIT- 3 LINUX SHELL SCRIPT

UNIT STRUCTURE

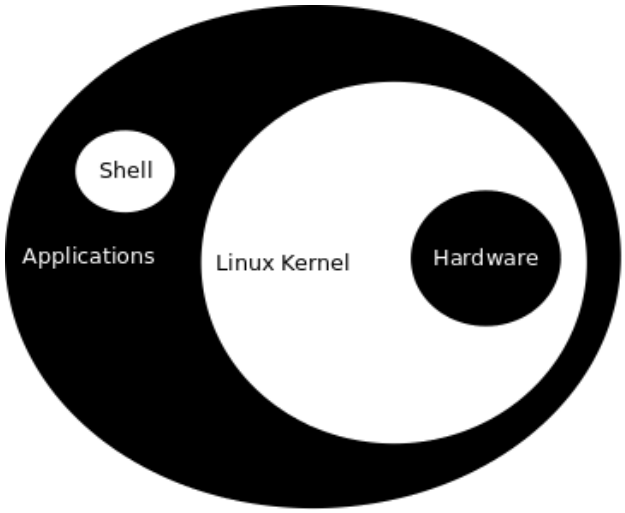
3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn what a Linux Shell is
- learn about the various shells available in Linux
- compare the various shells available in Linux
- learn the basics of shell programming
- write shell programs

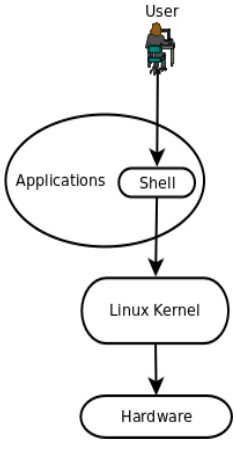
3.2 INTRODUCTION

To interact with the Linux Operating System there is a special program called the "Shell". The Linux Shell accepts instructions and passes it to the Linux Kernel. The Linux Shell is a user mode program and is more commonly seen as a command language interpreter that executes commands which is read either from the standard input device (keyboard) or from a file. The Linux Shell is not part of the system kernel, but uses the system kernel to execute programs, create files etc. The shell is the traditional user interface for the Linux Operating System.



Several shells are available with Linux depending on your distribution. You can view the currently installed shells on your system by typing "**cat /etc/shells**" in the terminal. To view the current shell you are using type "**echo $SHELL**" in the terminal.

Linux shells are interactive and you can either type in single commands or sequences of commands. It is possible to also store sequences of commands to a text file and call the Linux Shell to execute this text file instead of entering the commands individually. This text file containing the sequence of commands is known as a shell script.



In this unit, we will be specifically looking at the Linux Shell Scrips. This unit assumes that you have already installed a Linux distribution on a computer and ready to try out all the examples stated in the unit as we progress. This unit also assumes that you are aware of the basics of using Linux. If you need to brush up on that topic, you are highly encouraged to refer to the numerous ebooks and tutorials available online on the Internet.

3.3 VARIOUS TYPES OF SHELL AVAILABLE IN LINUX

Bourne Shell (/bin/sh):

The Bourne Shell or just **sh** was named after its developer Steve Bourne at Bell Labs in 1977. It introduced the basic features common to all the shells, including piping, command substitution, variables, control structures for condition-testing and looping, filename wildcarding.

C Shell (/bin/csh):

The C Shell or just **csh**, was written by Bill Joy, while a graduate student at the University of California, Berkeley in 1978. Designed to allow users to write shell script programs using a syntax very similar to that of the C programming language. The C Shell also introduced a large number of features for interactive work, including the history and editing mechanisms, aliases, directory stacks, tilde notation, cd path, job control and path hashing.

TC Shell (/bin/tcsh):

The TC shell or just **tcsh** was developed by Ken Greer (Carnegie Mellon University), Mike Ellis (Fairchild A.I. Labs), Paul Placeway (Ohio State University), Christos Zoulas, et al between 1975 to 1983. It is an expansion upon the C Shell (csh). This shell adds the ability to use keystrokes from the Emacs word processor program to edit text on the command line.

Korn Shell (/bin/ksh):

The Korn Shell or just **ksh** was developed by David Korn (Bell Labs) in 1983. This shell merges the features of the C Shell (csh) and the TC Shell (tcsh) together with a shell programming language similar to that of the original Bourne shell.

Bourne Again Shell (/bin/bash):

The Bourne Again Shell or just **bash** was written by Brian Fox for the GNU Project as a free software replacement for the Bourne Shell (sh) in 1989. Ultimately it is intended to be a full implementation of the IEEE Posix Shell and Tools specification. This shell is the most widely used in the Linux community and the default is most Linux distributions. The bash shell provides all the interactive features of the C Shell (csh) and the Korn Shell (ksh). Its programming language is compatible with the Bourne Shell (sh).

There are numerous shells available other than the ones mentioned above. However, for our example purposes we will be using the bash shell (bash) as this is the shell you will have by default in most of the Linux distributions.

3.4 COMPARISONS BETWEEN VARIOUS SHELLS

In this section we will briefly compare the Linux Shells discussed in the previous section based on certain features that are commonly expected these days.

command-line completion:

Command-line completion allows a user to type the first few characters of a command, program, or filename, and press the completion key (normally the **Tab**) to fill in the rest of the item.

The **csh**, **tcsh**, **ksh**, **bash** shells support command-line completion.

command history:

Command history is a feature in a operating system shell that allows a user to recall, edit and rerun previous commands without having to retype the entire thing. The general invocation could be the following:

Typing "**history**" at the shell prompt would list all the commands typed in previously.

Typing “**!10**” at the shell prompt would mean the command listed at no. 10 in history.
Typing “**!!**” at the shell prompt would mean the entire previous command.
Typing “**!$**” at the shell prompt would mean just the last word of the previous command.
Typing “**!ls**” at the shell prompt would mean the command that started with “**ls**”.

The **csh**, **tcsh**, **ksh**, **bash** shells support command history.

value prompt :
A shell script can prompt the interactive user for a value.

The **sh**, **csh**, **tcsh**, **ksh**, **bash** shells support value prompt.

menu/options selector:
A shell script can present the interactive user with a list of choices.

The **ksh**, **bash** shells support menu/options selector.

The comparisons between the various Linux Shells briefly discussed here does in no means cover the full features of any of these shells. These shells have their own purpose and if you want to know more about them you are encouraged to refer to the numerous resources available online on the Internet.

3.5 SHELL PROGRAMMING IN BASH

Let us briefly go through some basics of bash shell scripting before diving into the outlined topics. As we have already covered in the introduction that a shell script is a plain-text file that contains commands. It can be executed by typing its name into a shell, or by placing its name in another shell script. However, for a shell script file to be executable, the file must meet a couple of basic conditions as stated below.

1. A bash shell script file must start with a special first line "**#!/bin/bash**" also called the “hashbang”. What this line does is names the command processor, which in

our case is "/bin/bash" and executes the file using “bash”, the Bourne Again Shell.

2. A bash shell script file must be made executable by changing its permission bits using "**chmod +x**". So, if “myscriptfile.sh” is the bash script file, you need to run “chmod +x myscriptfile.sh” before being able to run the shell script.

Additionally, for easy identification of shell script files you may suffix your shell scripts with the "**.sh**" extension though the command processor would check only for the executable bit and the “#!/bin/bash” line to decide how to handle the shell script file.

For running your shell scripts use the "**./myscriptfile.sh**" method.

To demonstrate what we have learn till now let us write and run a simple bash script file. We will use the bash terminal and the “vim” editor to write our bash script file.

At the bash prompt (**$**) type the following, assuming that you have the vim editor already installed.

```
vim myscriptfile.sh
```

Press the “**i**” key to go into -- *INSERT* -- mode of the vim editor, and add the following lines to the “myscriptfile.sh” file.

```
#!/bin/bash
echo "Hello KKHSOU"
```

After you are done entering the above text, press the “**Esc**” key and then the sequence of keys “**:wq**” to write and quit the vim editor. On successful write and exit from the vim editor you should be back at the bash prompt (**$**). Next, we will need to make our script “myscriptfile.sh” executable by setting its executable bit as shown below.

```
chmod +x myscriptfile.sh
```

Finally, we will run our script file “myscriptfile.sh” by typing the following at the bash prompt (**$**).

```
./myscriptfile.sh
```

```
Hello KKHSOU
```

If you get an output “Hello KKHSOU” on your terminal as well, you have successfully written and executed your bash shell script.

3.5.1 READ COMMAND

The read command is one of the shell built-in commands. The syntax of the read command is as follows.

General Syntax:
read *[options] NAME1 NAME2 ... NAMEn*

Full options:

read [-ers] [-a aname] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt] [-t timeout] [-u fd] [name ...]

read reads one line from the standard input, or from the file descriptor (**fd**) supplied as an argument to the **-u** option, and the first word is assigned to the first **name**, the second word to the second **name**, and so on, with leftover words and their intervening separators assigned to the last **name**. If there are fewer words read from the input stream than **names**, the remaining **names** are assigned empty values. The characters in internal field separator (**IFS**) are used to split the line into words. The backspace character (****) may be used to remove any special meaning for the next character read and for line continuation.

Options, if supplied, have the following meanings.
-e : If the standard input is coming from a terminal, **readline** is used to obtain the line. **readline** uses the current (or default, if line editing was not previously active) editing settings.

-r : Backslash does not act as an escape character. The backslash is considered to be part of the line. In particular, a backslash-newline pair may not be used as a line continuation.

-s : Silent mode. If input is coming from a terminal, characters are not echoed.

-a aname : The words are assigned to sequential indices of the array variable **aname**, starting at 0. **aname** is unset before any new values are assigned. Other name arguments are ignored.

-d delim : The first character of **delim** is used to terminate the input line, rather than newline.

-i text : If **readline** is being used to read the line, **text** is placed into the editing buffer before editing begins.

-n nchars : **read** returns after reading **nchars** characters rather than waiting for a complete line of input, but honor a delimiter if fewer than **nchars** characters are read before the delimiter.

-N nchars : **read** returns after reading exactly **nchars** characters rather than waiting for a complete line of input, unless end-of-file (**EOF**) is encountered or **read** times out. Delimiter characters encountered in the input are not treated specially and do not cause **read** to return until **nchars** characters are read.

-p prompt : Display **prompt** on standard error, without a trailing newline, before attempting to read any input. The **prompt** is displayed only if input is coming from a terminal.

-t timeout : Cause **read** to time out and return failure if a complete line of input is not read within **timeout** seconds. **timeout** may be a decimal number with a fractional portion following the decimal point. This option is only effective if **read** is reading input from a terminal, pipe, or other special file; it has no effect when reading from regular files. If **timeout** is 0, **read** returns success if input is available on the specified file

descriptor **fd**, failure otherwise. The exit status is greater than 128 if the timeout is exceeded.

-u fd : Read input from file descriptor **fd**. If no names are supplied, the line read is assigned to the variable **REPLY**. The return code is zero, unless end-of-file (**EOF**) is encountered, **read** times out (in which case the return code is greater than 128), or an invalid file descriptor is supplied as the argument to **-u**.

The above **read** syntax and options are found in the BASH version 4.2.45(1)-release, which was used while writing this unit. You can check the BASH version installed on your system by typing "**echo $BASH_VERSION**" at the bash prompt.

3.5.2 CONDITIONAL AND LOOPING STATEMENTS

An important aspect of programming is being able to perform some specific actions depending upon the outcome of a test condition being true or false. This can be especially useful when used with the exit status of a previous command in bash. Bash Shell scripting supplies us with several ways for testing conditions and then performing specific actions based upon the result of the test condition. Some examples of conditions or conditional statements are the **if condition** and the **case statement**. Conditions are also tested in the **while loop** and the **for loop** where specific actions are repeated.

Conditional Statements

if

The syntax of the if command is:

```
if [ test-condition ]
        then
                consequent-actions
elif [ more-test-condition ]
        then
                more-consequent-actions
else
        alternate-consequent-actions
```

```
fi
```

The **test-condition** list is executed, and if its return status is zero (true), the **consequent-actions** list is executed. If the **test-condition** returns a non-zero status (false), each **elif** list is executed in turn, and if its exit status is zero (false), the corresponding **more-consequent-actions** is executed and the **if** command completes. If the **else alternate-consequent-actions** is present, and the final command in the final **if** or **elif** clause has a non-zero (false) exit status, then the **alternate-consequent-actions** is executed. The return status is the exit status of the last command executed, or zero if no condition tested true.

Example:

```
#!/bin/bash
echo "* * * This Bash Script will compare two INTEGERS X and Y given as input * * *"
read -p "Type the value for X and press [ENTER]: " x
read -p "Type the value for Y and press [ENTER]: " y
if [ $x -eq $y ]
        then
                echo "X is equal to Y"
elif [ $x -ge $y  ]
        then
                echo "X is greater than Y"
else
        echo "X is less than Y"
fi
```

case

The syntax of the case command is:

```
case word in
        pattern1a | pattern1b | pattern1c ) command-list ;;
        pattern2a | pattern2b ) command-list ;;
esac
```

The **case** will selectively execute the **command-list** corresponding to the first **pattern** that matches **word**. If the shell option nocasematch is enabled, the match is performed without regard to the case of alphabetic characters. The "**|**" operator is used

to separate multiple patterns, and the "**)**" operator terminates a pattern list. A list of patterns and an associated command-list is known as a clause. Each clause must be terminated with "**;;**", "**;&**", or "**;;&**". The **word** undergoes *tilde expansion*, *parameter expansion*, *command substitution*, *arithmetic expansion*, and *quote removal* before matching is attempted. Each **pattern** undergoes *tilde expansion*, *parameter expansion*, *command substitution*, and *arithmetic expansion*. There may be an arbitrary number of case clauses, each terminated by a "**;;**", "**;&**", or "**;;&**". The first pattern that matches determines the **command-list** that is executed. If the "**;;**" operator is used, no subsequent matches are attempted after the first **pattern** match. Using "**;&**" in place of "**;;**" causes execution to continue with the **command-list** associated with the next clause, if any. Using "**;;&**" in place of "**;;**" causes the shell to test the **patterns** in the next clause, if any, and execute any associated **command-list** on a successful match. The return status is zero if no **pattern** is matched. Otherwise, the return status is the exit status of the **command-list** executed.

Example:

```
#!/bin/bash
read -p "Type in a vegetable name and press [ENTER]: " vegetable
case $vegetable in
        "onion" ) echo "For $vegetable Rs. 100.00 per Kilogram." ;;
        "potato" ) echo "For $vegetable Rs. 20.00 per Kilogram." ;;
        "tomato" ) echo "For $vegetable Rs. 40.00 per Kilogram." ;;
        "cabbage" ) echo "For $vegetable Rs. 30.00 per Kilogram." ;;
        * ) echo "No $vegetable in stock." ;;

esac
```

Looping Statements

while

The syntax of the while command is:

```
while test-commands
        do consequent-commands
done
```

Execute **consequent-commands** as long as the **test-commands** has an exit status of zero (true). The return status is the exit status of the last command executed in **consequent-commands**, or zero if none was executed.

Example:

```
#!/bin/bash
read -p "Which multiplication table you want to view [enter number]: " number
i=1
while [ $i -le 10 ]
	do
		echo "$number x $i = `expr $i \* $number`"
		i=`expr $i + 1`
done
```

for

The syntax of the for command is:

```
for name [ in words ]
		do commands
done
```

Expand **words**, and execute **commands** once for each member in the resultant list, with **name** bound to the current member. If "**in words**" is not present, the **for** command executes the **commands** once for each positional parameter that is set, as if "**in "$@"**" had been specified. The return status is the exit status of the last command that executes. If there are no items in the expansion of **words**, no **commands** are executed, and the return status is zero.

Example:

```
#!/bin/bash
 for i in {0..10}
	do echo "$i"
done
```

The **break** and **continue** builtins may be also used to control loop execution, as shown in the examples below.

```
#!/bin/bash
for i in {1..5}
do
  if [ $i -gt 3 ]
  then
        break
  fi
  echo "$i"
done
```

The example shown above will print the numbers from 1 to 3 as the condition for break is for $i > 3. **break** is used to exit the loop. In the example shown below we use the **continue** builtin to print all the even numbers between 1 and 10.

```
#!/bin/bash
for i in {1..10}
do
  if [ $(($i % 2)) -eq 0 ]
        then
              echo "$i"
  else
        continue
  fi
done
```

3.5.3 PARAMETER PASSING AND ARGUMENTS

The variables within a Bash Script that represent the values that will be passed to it for processing are called **parameters** and the values you pass to the script are called **arguments**. Both parameters and arguments can be used interchangeably however for a clear understanding we will use the simple example shown below.

```
#!/bin/bash
number=$1
echo "[ Multiplication Table of $number ]"
i=1
while [ $i -le 10 ]
      do
            echo "$number x $i = `expr $i \* $number`"
            i=`expr $i + 1`
done
```

The variable **number** within the bash script shown above, is a **parameter**.
$1 is the first **argument** passed to the bash script.

We run this script with an **argument** 2, as shown below.

```
./myscriptfile.sh 2
[ Multiplication Table of 2 ]

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

2 x 5 = 10

2 x 6 = 12
```

```
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Therefore, like any other command line program, we can pass arguments to a Bash script. Bash has special variables set aside for this, the arguments are stored in variables with a number in the order of the argument starting at 0 as shown below.

$0 - the bash script's name.

$1 - first argument

$2 - second argument

$3 - third argument

$# - total number of argument

$@** or **$* - all the arguments

3.5.4 BOURNE SHELL VARIABLES

Bash uses certain shell variables in the same way as the Bourne shell. In some cases, Bash assigns a default value to the variable.

- **CDPATH** : A colon-separated list of directories used as a search path for the cd builtin command.

- **HOME** : The current user's home directory; the default for the cd builtin command. The value of this variable is also used by tilde expansion (see Tilde Expansion).

- **IFS** : A list of characters that separate fields; used when the shell splits words as part of expansion.

- **MAIL** : If this parameter is set to a filename or directory name and the MAILPATH variable is not set, Bash informs the user of the arrival of mail in the specified file or Maildir-format directory.

- **MAILPATH** : A colon-separated list of filenames which the shell periodically checks for new mail. Each list entry can specify the message that is printed when new mail arrives in the mail file by separating the file name from the message with a '?'. When used in the text of the message, $_ expands to the name of the current mail file.

- **OPTARG** : The value of the last option argument processed by the getopts builtin.
- **OPTIND** : The index of the last option argument processed by the getopts builtin.
- **PATH** : A colon-separated list of directories in which the shell looks for commands. A zero-length (null) directory name in the value of PATH indicates the current directory. A null directory name may appear as two adjacent colons, or as an initial or trailing colon.
- **PS1** : The primary prompt string. The default value is '\s-\v\$ '. See Printing a Prompt, for the complete list of escape sequences that are expanded before PS1 is displayed.
- **PS2** : The secondary prompt string. The default value is '> '.

You can view the current values of these variables by typing "echo $*variable-name*" at the terminal. Replace the *variable-name* with any of the above mentioned bash shell variables.

3.5.5 BASH VARIABLES

These variables are set or used by Bash, but other shells do not normally treat them specially. This is not an exhaustive list and we will list out only a few of them.

Please read the “man bash” man pages for more information.

- **BASH** : The full pathname used to execute the current instance of Bash.
- **BASHOPTS** : A colon-separated list of enabled shell options. If this variable is in the environment when Bash starts up, each shell option in the list will be enabled before reading any startup files. This variable is read only.
- **BASHPID** : Expands to the process ID of the current Bash process. This differs from $$ under certain circumstances, such as subshells that do not require Bash to be re-initialized.
- **BASH_COMMAND** : The command currently being executed or about to be executed, unless the shell is executing a command as the result of a trap, in which case it is the command executing at the time of the trap.
- **BASH_VERSINFO** : A readonly array variable whose members hold version information for this instance of Bash.
- **BASH_VERSION** : The version number of the current instance of Bash.
- **EUID** : The numeric effective user id of the current user. This variable is readonly.
- **GROUPS** : An array variable containing the list of groups of which the current user is a member. Assignments to GROUPS have no effect and return an error status. If GROUPS is unset, it loses its special properties, even if it is subsequently reset.
- **HOSTNAME** : The name of the current host.
- **HOSTTYPE** : A string describing the machine Bash is running on.
- **LANG** : Used to determine the locale category for any category not specifically selected with a variable starting with LC_.
- **LINENO** : The line number in the script or shell function currently executing.
- **MACHTYPE** : A string that fully describes the system type on which Bash is executing, in the standard GNU cpu-company-system format.
- **OLDPWD** : The previous working directory as set by the cd builtin.
- **OSTYPE** : A string describing the operating system Bash is running on.

You can also view the current values of these Bash variables by typing “echo $*variable-name*” at the terminal. Replace the *variable-name* with any of the above mentioned bash variables.

3.5.6 SHELL RESERVED WORDS

In Bash, **shell reserved words** are words that have a special meaning to the shell and therefore should not be used for anything else. Below is a list of a few of these reserved words. Please read the “man bash” man pages for more information.

! : A pipeline is a sequence of simple commands separated by one of the control operators ‘|’ or ‘|&’.

[or **]** : Conditional constructs.

{ or **}** : Command Grouping.

case : Conditional constructs.

esac : Conditional constructs.

if : Conditional constructs.

elif : Conditional constructs.

else : Conditional constructs.

fi : Conditional constructs.

in : Conditional constructs.

select : Conditional Constructs

do : Looping Constructs.

done : Looping Constructs.

for : Looping Constructs.

then : Conditional Constructs.

time : Pipelines.

until : Looping Constructs.

while : Looping Constructs.

CHECK YOUR PROGRESS

Q1. Write the name of few commonly used Shells.

Q2. How do you terminate a shell script if statement?

Q3. Within a UNIX shell scripting loop construct, what is the difference between the break and continue?

3.6 LET US SUM UP

In this unit we have tried to get acquainted briefly with bash scripting in Linux. Though we have covered only a few of the topics, these topics are intended to inspire and point you in the direction to further explore. This unit was not intended to provide you with an exhaustive in-depth on Bash Scripting but merely to introduce you to get started on bash scripting.

What we have learned in this unit.

- What a Linux Shell is.
- The various shells available in Linux.
- Comparison between the various shells available in Linux.
- The basics of shell programming.
- The **read** command and its use.
- The conditional statements **if** and **case** and their use.
- The looping statements **while** and **for** and their use.
- Using the **break** and **continue** builtins within the looping statements.
- Using **parameters** and **arguments** in bash scripts.
- About the Bourne Shell Variables, Bash Variables and Shell Reserved Words in Bash.

3.7 ANSWERS TO CHECK YOUR PROGRESS

1. List of commonly used UNIX shells:
   - · The Bourne Shell (sh)
   - · The C Shell (csh or tsch)
   - · The Bourne Again Shell (bash)
   - · The Korn Shell (ksh)

2. With fi, which is "if" spelled backwards.

3. Using break within a shell scripting loop construct will cause the entire loop to terminate. A continue will cause the current iteration to terminate, but the loop will continue on the next iteration.

3.8 FURTHER READINGS

- Bourne-Again SHell manual at “http://www.gnu.org/software/bash/manual/”
- The “man bash” manual page.

3.9 MODEL QUESTIONS

Q1.Which bash command displays the current shell?

Q2.Which Linux shell is the default and common in most distributions of Linux today?

Q3.Which Linux shells support command history?

Q4.Which **hashbang** is used in bash scripting?

Q5.What is the general convention for naming bash script file extension?

Q6.Which option can you use with **read** to display a prompt for input?

Q7.What does "-**eq**" mean in condition-testing?

Q8.What does "**$0**" contain when working with bash arguments?

Q9.What do you get when you type "**echo $PS1**" at the bash prompt?

Q10. Which command at the bash prompt will display your current bash version?

*****

UNIT- 4 SYSTEM ADMINISTRATION

UNIT STRUCTURE

4.1 LEARNING OBJECTIVE

After going through this unit, you will be able to:

- define System Administration
- describe the common System Administrative tasks
- describe the various roles of a System Administrator
- manage users and groups on a Linux System
- change file ownerships and permissions on Linux
- create and mount file systems on Linux
- check and monitor system performance on Linux
- use the “su” command to become a super user

- use commands to get system information
- backup and restore file in Linux

- install and remove software in Linux

- discuss the graphical user interfaces in Linux

4.2 INTRODUCTION

System administration is concerned mainly with the design, construction, and maintenance of computer systems and networks. The duties of a system administrator are diverse and are not limited to installing, supporting and maintaining servers or other computer systems, or planning for and responding to service outages and other infrastructure problems. A system administrator may also perform scripting or programming, project management for systems-related projects, supervising or training computer operators, and may also be the consultant for computer problems beyond the knowledge of technical support staff. To perform his or her job well, a system administrator must demonstrate a blend of both technical skills and responsibility. That said, a system administrator is expected to know about both the hardware and software he or she is dealing with.

In this unit, we will be specifically looking at System Administration on Linux Operating System. This unit assumes that you have already installed a Linux distribution on a computer and ready to try out all the examples stated in the unit as we progress. This unit also assumes that you are aware of the hardware configuration of your computer, details about what happens when a computer boots; starting from turning on the Computer by pressing the power button till the Operating System completes booting to a login prompt. If you need to brush up on that topic, you are highly encouraged to refer to the numerous ebooks and tutorials available online on the Internet.

4.3 COMMON ADMINISTRATIVE TASKS

Some of the tasks a System Administrator performs on a day-to-day basis, but not limited to, may include performing routine system audits; performing backups of critical data; ensuring that the network infrastructure is up and running; taking responsibility for security; system performance tuning, answering technical queries and assisting users; troubleshooting any reported problems; introducing and integrating new technologies into existing infrastructure; analyzing system logs and identifying potential

issues; applying operating system updates, patches, and configuration changes; adding, removing, or updating user account information, resetting passwords, etc.; installing and configuring new hardware and software; responsibility for documenting the configuration of the system.

4.3.1 IDENTIFYING ADMINISTRATIVE FILES

On a Linux system, everything is a file; if something is not a file, it is a process. The Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Unix/Unix-like operating systems. The FHS is maintained by the Linux Foundation. The current version is 2.3, announced on 29 January 2004. [Note: you are encouraged to read about development of the FHS 3.0 standard online at http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs]

.

As per the current FHS specifications, following directories, or symbolic links to directories, are required in "/" or the root directory.

Directory	Description
/bin	Essential command binaries. Contains commands that may be used by both the system administrator and by users, but which are required when no other filesystems are mounted (e.g. in single user mode). It may also contain commands which are used indirectly by scripts. There must be no subdirectories in /bin.
/boot	Static files of the boot loader. This directory contains everything required for the boot process except configuration files not needed at boot time and the map installer. Thus /boot stores data that is used before the kernel begins executing user-mode programs. This may include saved master boot sectors and sector map files.
/dev	The /dev directory is the location of special or device files.
/etc	Host-specific system configuration. The /etc hierarchy contains configuration files. A "configuration file" is a local file used to control the operation of a program; it must be static and cannot be an executable binary.
/home	User home directories.
/lib	Essential shared libraries and kernel modules. The /lib directory contains those shared library images needed to boot the system and run the commands in the root filesystem, ie. by binaries in /bin and /sbin.
/media	Mount point for removeable media. This directory contains subdirectories which are used as mount points for removeable media such as floppy disks, cdroms and zip disks.
/mnt	Mount point for mounting a filesystem temporarily. This directory is provided so that the system administrator may temporarily mount a filesystem as needed. The content of this directory is a local issue and should not affect the manner in which any program is run.
/opt	Add-on application software packages. /opt is reserved for the installation of add-on application software packages.
/root	Home directory for the root user.
/sbin	Essential system binaries. Utilities used for system administration (and other root-only commands) are stored in /sbin, /usr/sbin, and /usr/local/sbin. /sbin contains binaries essential for booting, restoring, recovering, and/or repairing the system in addition to the binaries in /bin. Programs executed after /usr is known to be mounted (when there are no problems) are generally placed into /usr/sbin. Locally-installed system administration programs should be placed into /usr/local/sbin.
/srv	Data for services provided by this system.
/tmp	Temporary files
/usr	Secondary hierarchy. /usr is the second major section of the filesystem. /usr is shareable, read-only data. That means that /usr should be shareable between various FHS-compliant hosts and must not be written to. Any information that is host-specific or varies with time is stored elsewhere.
/var	Variable data. /var contains variable data files. This includes spool directories and files, administrative and logging data, and transient and temporary files.

Aside: To view files on a Linux system, we use the "ls" command which is used to simply list the current directory contents onto the screen [Note: you are encouraged to use the "man ls" or the "info ls" command to know more about this command or how to use it on your Linux system].

Example#1:

When we use the command "ls -l" inside a directory, the "-l" option with the "ls" displays the file type, using the first character of each input line. Don't worry if you did not understand what you just read in the previous sentence. We will try to understand it in this example.

INPUT

Line1: [nanu.kachari@beefy-miracle KKHSOU]$ **ls -l**

The input **Line1** contains the bash command prompt with some values between "[" and "]$". The value "nanu.kachari" is the username of the logged in user followed by the "@beefy-miracle" which indicates the hostname of the host. "KKHSOU" is the current directory the user is in. "ls -l" is the command entered by the user. Every command has to be followed by a carriage return or the enter key in order to be executed.

OUTPUT

Line1: total 596
Line2: -rw-rw-r--. 1 nanu.kachari nanu.kachari 18242 Jul 1 15:45 MCA Linux Syllabus.docx
Line3: drwxrwxr-x. 2 nanu.kachari nanu.kachari 4096 Jul 9 15:00 Unit4
Line4: -rw-------. 1 nanu.kachari nanu.kachari 577536 Jul 1 16:19 Unit 4 - System Administration.doc
Line5: -rw-rw-r--. 1 nanu.kachari nanu.kachari 4857 Jul 9 14:56 unit4.txt

The output **LINE1** "total 596" is the Total no. of blocks in the directory. The **LINE2** to **LINE5** first column (example: -rw-rw-r--.) first character (example: -) starting from the left denotes the type of file. The rest nine characters deals with permissions and will be explained later in this unit.

A "**-**" indicates a regular file
A "**d**" indicates a directory (files that are lists of other files)
A "**l**" indicates a link (link to a file or directory)
A "**c**" indicates a special file (the mechanism used for input and output. Most special files are located in "/dev")
A "**s**" indicates a socket (a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control)
A "**p**" indicates a named pipe (act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics)
A "**b**" indicates a block device

The output **LINE2** to **LINE5** second column (example: 1) which is numeric, is the no. of links pointing to the file. This will always be "1" for a regular file and "2" for directories. **LINE2** to **LINE5** third column (example: nanu.kachari) and fourth column (example: nanu.kachari) deals with ownerships and will be explained later in this unit. **LINE2** to **LINE5** fifth column (example: 18242) is the file size in bytes. **LINE2** to **LINE5** sixth, seven and eight columns are the files' last modification or creation data and time

(example: Jul 1 15:45). **LINE2** to **LINE5** ninth column (example: MCA Linux Syllabus.docx) is the name of the file.

In order to perform any administrative task on a Linux System you need to have root or super user privileges. On Linux we use the "su" or the "sudo" commands to switch user privileges to achieve this. [Note: you are encouraged to use the "man su", "info su" or "man sudo", "man sudo" commands to known more about this command or how to use it on your Linux system]

Example#2:

```
[nanu.kachari@beefy-miracle ~]$ whoami
nanu.kachari
[nanu.kachari@beefy-miracle ~]$ su -
Password:
[root@beefy-miracle ~]#
[root@beefy-miracle ~]# whoami
root
```

4.3.2 CONFIGURATION AND LOG FILES

Configuration files can be global config files or local config files. Global config files apply to all users and are usually located in the "/etc" folder. Local config files applies to a specific user and are stored in the users' home directory, for example as "~/.example" or "~/.config/example" files with the "." suffixed generally.

The location of config files may differ with different Linux Distributions. Therefore, the files listed below may not match the Linux installation you may be using, however they are generally similar.

Some common configuration files:

File	Information/service
/etc/aliases	Mail aliases file for use with the Sendmail and Postfix mail server. Running a mail server on each and every system has long been common use in the UNIX world, and almost every Linux distribution still comes with a Sendmail package. In this file local user names are matched with real names as they occur in E-mail addresses, or with other local addresses.
/etc/apache	Config files for the Apache web server.
/etc/bashrc	The system-wide configuration file for the Bourne Again SHell. Defines functions and aliases for all users. Other shells may have their own system-wide config files, like cshrc.
/etc/crontab and /etc/cron.*	Configuration of tasks that need to be executed periodically - backups, updates of the system databases, cleaning of the system, rotating logs etc.
/etc/default	Default options for certain commands, such as useradd.
/etc/filesystems	Known file systems: ext3, vfat, iso9660 etc.
/etc/fstab	Lists partitions and their mount points.
/etc/group	Configuration file for user groups. Use the shadow utilities groupadd, groupmod and groupdel to edit this file. Edit manually only if you really know what you are doing.
/etc/hosts	A list of machines that can be contacted using the network, but without the need for a domain name service. This has nothing to do with the system's network configuration, which is done in /etc/sysconfig.
/etc/inittab	Information for booting: mode, number of text consoles etc.
/etc/issue	Information about the distribution (release version and/or kernel info).
/etc/ld.so.conf	Locations of library files.
/etc/logrotate.*	Rotation of the logs, a system preventing the collection of huge amounts of log files.
/etc/mail	Directory containing instructions for the behavior of the mail server.
/etc/motd	Message Of The Day: Shown to everyone who connects to the system (in text mode), may be used by the system admin to announce system services/maintenance etc.
/etc/mtab	Currently mounted file systems. It is advised to never edit this file.
/etc/nsswitch.conf	Order in which to contact the name resolvers when a process demands resolving of a host name.
/etc/pam.d	Configuration of authentication modules.
/etc/passwd	Lists local users. Use the shadow utilities useradd, usermod and userdel to edit this file. Edit manually only when you really know what you are doing.
/etc/printcap	Outdated but still frequently used printer configuration file. Don't edit this manually unless you really know what you are doing.
/etc/profile	System wide configuration of the shell environment: variables, default properties of new files, limitation of resources etc.
/etc/rc*	Directories defining active services for each run level.
/etc/resolv.conf	Order in which to contact DNS servers (Domain Name Servers only).
/etc/sendmail.cf	Main config file for the Sendmail server.
/etc/services	Connections accepted by this machine (open ports).
/etc/sound or sndconfig	Configuration of the sound card and sound events.
/etc/ssh	Directory containing the config files for secure shell client and server.
/etc/sysconfig	Directory containing the system configuration files: mouse, keyboard, network, desktop, system clock, power management etc. (specific to RedHat)
/etc/X11	Settings for the graphical server, X. RedHat uses XFree, which is reflected in the name of the main configuration file, XFree86Config. Also contains the general directions for the window managers available on the system, for example gdm, fvwm, twm, etc.
/etc/xinetd.* or inetd.conf	Configuration files for Internet services that are run from the system's (extended) Internet services daemon (servers that don't run an independent daemon).

Some common log files:

/var/log/messages	System messages
/var/log/secure	Logging by PAM of network access attempts
/var/log/dmesg	Log of system boot. Also see command dmesg
/var/log/boot.log	Log of system init process
/var/log/lastlog	Requires the use of the lastlog command to examine contents
/var/log/maillog	log from the sendmail daemon

You can use the "tail" command to view the log files. [Note: you are encouraged to use the "man tail" or "info tail" commands to know more about the "tail" command on your Linux installation].

Example#3:

```
[root@beefy-miracle ~]# tail -f  /var/log/messages
Jul 11 11:46:04 beefy-miracle kernel: [60730.950120] device p5p1 left promiscuous mode
Jul 11 11:46:08 beefy-miracle kernel: [60735.260284] vboxnetflt: dropped 0 out of 89521 packets
Jul 11 11:56:15 beefy-miracle dbus-daemon[703]: dbus[703]: [system] Activating service name='net.reactivated.Fprint' (using servicehelper)
Jul 11 11:56:15 beefy-miracle dbus[703]: [system] Activating service name='net.reactivated.Fprint' (using servicehelper)
Jul 11 11:56:15 beefy-miracle dbus-daemon[703]: dbus[703]: [system] Successfully activated service 'net.reactivated.Fprint'
Jul 11 11:56:15 beefy-miracle dbus[703]: [system] Successfully activated service 'net.reactivated.Fprint'
Jul 11 11:56:15 beefy-miracle dbus-daemon[703]: Launching FprintObject
Jul 11 11:56:15 beefy-miracle dbus-daemon[703]: ** Message: D-Bus service launched with name: net.reactivated.Fprint
Jul 11 11:56:15 beefy-miracle dbus-daemon[703]: ** Message: entering main loop
Jul 11 11:56:45 beefy-miracle dbus-daemon[703]: ** Message: No devices in use, exit
```

4.3.3 MANAGING USER ACCOUNTS

For performing the task of managing user accounts in Linux you are required to have super user privileges. Depending on the distribution of Linux you have installed you should be able to achieve this by either using the command "su -" or "sudo -i" and entering the root user password. The command prompt should change from "$" to "#" on a successful super user login.

Example#4:

```
[nanu.kachari@beefy-miracle ~]$ su -
```

```
Password:
[root@beefy-miracle ~]#
```

Adding users:

When it comes to adding users in Linux there are several methods you could use. We will be discussing only the command line interface methods. The fundamental low level tool for adding users in Linux is "useradd". [Note: you are encouraged to use the "man useradd" or "info useradd" on your Linux installation for further reference]

View the system default configuration for adding user using "useradd".

Example#5:

```
[root@beefy-miracle ~]# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

GROUP: This is the only option which will not be taken as default. Because if you don't specify -n option a group with same name as the user will be created and the user will be added to that group. To avoid that and to make the user as the member of the default group you need to give the option -n.

HOME: This is the default path prefix for the home directory. Now the home directory will be created as /home/USERNAME.

INACTIVE: -1 by default disables the feature of disabling the account once the user password has expired. To change this behavior you need to give a positive number which means if the password gets expired after the given number of days the user account will be disabled.

EXPIRE: The date on which the user account will be disabled.

SHELL: Users login shell.

SKEL: Contents of the skel directory will be copied to the users home directory.

CREATE_MAIL_SPOOL: According to the value creates or does not create the mail spool.

Create users with the default configurations using "useradd".

Example#6: You will notice in this example that the default "useradd" command creates "rahuli" as both a username and a group.

```
[root@beefy-miracle root]# useradd rahul
[root@beefy-miracle root]# passwd rahul
Changing password for user rahul.
New password:
BAD PASSWORD: The password fails the dictionary check - it is too simplistic/systematic
Retype new password:
passwd: all authentication tokens updated successfully.
```

```
[root@beefy-miracle root]# grep rahul /etc/passwd
rahul:x:1005:1005::/home/rahul:/bin/bash
[root@beefy-miracle root]# grep rahul /etc/group
rahul:x:1005:
```

Instead of using the default "useradd" values (group, shell etc. located at /etc/default/useradd) as shown in the above method, you could specify custom values in the command line as parameters to the useradd command to add new users.

Syntax:
useradd -s SHELL -m -d HOME -c "COMMENT" -g GROUP USERNAME

-s SHELL The login shell for the user.

-m	Create user’s home directory if it does not exist.
-d HOME	Home directory of the user.
-c COMMENT	Comment.
-g GROUP	Group name or number of the user.
USERNAME	Login id of the user.

Example#7:

```
[root@beefy-miracle root]# useradd -s /bin/bash -m -d /home/rahul -c "Rahul Sarma" -g rahul rahul
```

Using the above method for creating multiple users can be very tedious and time consuming. Thankfully, the "newusers" command available in Linux provides us the option to create users in bulk.

Syntax: newusers FILENAME

The FILENAME is in the same format as the "/etc/password" file.

loginname:password:uid:gid:comment:homedir:loginshell

Example#8:

Let us suppose that we have a "newusers.txt" text file containing the following lines. We will use this file to add new users in bulk.

The contents of the a "newusers.txt" text file.

```
binod.deka:abcd1234:1006:1006:Binod Deka:/home/binod.deka:/bin/bash
smiritisikha:a1b2c3d4:1007:1007:Smiritisikha Choudhury:/home/smiritisikha:/bin/bash
```

Lets use the "newusers.txt" text file to add users in bulk.

```
[root@beefy-miracle root]# newusers newusers.txt
```

Now, let us check to see the users we added using the **cat** command.

```
[root@beefy-miracle root]# cat /etc/passwd
```

```
.
..
...
rahul:x:1005:1005::/home/rahul:/bin/bash
binod.deka:x:1006:1006:Binod Deka:/home/binod.deka:/bin/bash
smiritisikha:x:1007:1007:Smiritisikha Choudhury:/home/smiritisikha:/bin/bash
```

Also, let us check to see if the respective groups were added as well.

```
[root@beefy-miracle root]# cat /etc/group
.
..
...
rahul:x:1005:
binod.deka:x:1006:
smiritisikha:x:1007:
```

Deleting users:

We use the "userdel" command to delete a user account and related files in Linux. The userdel command must be run as root user. [Note: you are encouraged to use the "man userdel" or "info userdel" on your Linux installation for further reference. Please use caution while using this command.]

Syntax: userdel USERNAME

Example#9:

In this example we will use the "userdel" command to remove the user but not the user home directory.

Let us view the contents of the "/home" folder before deleting the user "rahul".

```
[root@beefy-miracle root]# ls -l /home/
drwx------.  4 binod.deka  binod.deka  4096 Jul 10 16:20 binod.deka
drwx------.  2 root        root       16384 Aug  2  2012 lost+found
```

```
drwx------.  2 smiritisikha     smiritisikha     4096 Jul 10 17:12 smiritisikha
drwx------.  4 rahul     rahul  4096 Jul 10 15:53 rahul
```

Now we use the "userdel" command to delete the user "rahul".

```
[root@beefy-miracle root]# userdel rahul
```

Let us have a look at the contents of the "/home" folder after deleting the user "rahul".

```
[root@beefy-miracle root]# ls -l /home/
total 40
drwx------.  4 binod.deka     binod.deka     4096 Jul 10 16:20 binod.deka
drwx------.  2 root     root                16384 Aug  2  2012 lost+found
drwx------.  2 smiritisikha     smiritisikha     4096 Jul 10 17:12 smiritisikha
drwx------.  4 1005     1005                4096 Jul 10 15:53 rahul
```

You can see that the folder "rahul" still exists even though the user "rahul" has been deleted.

Example#10:

In this example we will use the "userdel" command to remove all user files along with the user home directory itself and the user's mail spool. Please note that files located in other file systems will have to be searched for and deleted manually.

We list the contents of the "/home" folder before deleting the user "binod.deka".

```
[root@beefy-miracle root]# ls -l /home
total 40
drwx------.  4 binod.deka     binod.deka     4096 Jul 10 16:20 binod.deka
drwx------.  2 root     root                16384 Aug  2  2012 lost+found
drwx------.  2 smiritisikha     smiritisikha     4096 Jul 10 17:12 smiritisikha
drwx------.  4 1005     1005                4096 Jul 10 15:53 rahul
```

We now use the "userdel" with the "-r" option to delete the user "binod.deka".

```
[root@beefy-miracle root]# userdel -r binod.deka
```

List the "/home" folder after deleting the user "binod.deka".

```
[root@beefy-miracle root]# ls -l /home
total 36
drwx------. 2 root        root          16384 Aug  2  2012 lost+found
drwx------. 2 smiritisikha smiritisikha  4096 Jul 10 17:12 smiritisikha
drwx------. 4 1005        1005           4096 Jul 10 15:53 rahul
```

You can see that the folder "binod.deka" no longer exists.

4.3.4 CHANGING PERMISSIONS AND OWNERSHIPS

The file permissions available on a Linux system are read (r), write (w) and execute (x). Read file permissions will allow the user to view the contents of the file. Write file permissions will allow the user to change the contents of the file or delete the file. Execute file permissions will allow the user to run the file as a program.

The directory permissions available on a Linux system are same as the file permissions however they do mean different things. Read directory permissions will allow the user to list the contents of the directory. Write directory permissions will allow the user to add or delete files in the directory. Execute directory permissions will allow the user to list information about the files in the directory.

Directory or file permissions are assigned to three entities, the user, group and others.

On a Linux system, every file and directory, is owned by a specific user and group. File permissions are defined separately for users, groups, and others. Please refer to **Example#1** where the "ls" command is used with the "-l" option to list the contents of a directory as show below.

```
LINE1: [nanu.kachari@beefy-miracle KKHSOU]$ ls -l
LINE2: total 596
LINE3: -rw-rw-r--. 1 nanu.kachari nanu.kachari 18242 Jul 1 15:45 MCA Linux Syllabus.docx
```

LINE4: drwxrwxr-x. 2 nanu.kachari nanu.kachari 4096 Jul 9 15:00 Unit4
LINE5: -rw-------. 1 nanu.kachari nanu.kachari 577536 Jul 1 16:19 Unit 4 - System Administration.doc
LINE6: -rw-rw-r--. 1 nanu.kachari nanu.kachari 4857 Jul 9 14:56 unit4.txt

LINE3 to **LINE6** first column (example: -rw-rw-r--.), second to fourth character (example: rw-) represents the owner permissions. "r" is read, "w" is write and "x" is execute permission. "-" indicates no permission. The order is "rwx".

LINE3 to **LINE6** first column (example: drwxrwxr-x.), fifth to seventh character (example: rwx) represents the group permissions. "r" is read, "w" is write and "x" is execute permission. "-" indicates no permission. The order is "rwx".

LINE3 to **LINE6** first column (example: -rw-------.), eight to tenth character (example: ---) represents the other (or everyone) permissions. "r" is read, "w" is write and "x" is execute permission. "-" indicates no permission. The order is "rwx".

LINE3 to **LINE6** The third and fourth column (example: nanu.kachari nanu.kachari) of the above output shows the owner and group information respectively.

We use the "chmod" command in Linux to change file mode bits. [Note: you are encouraged to use the "man chmod" or "info chmod" commands to known more about the "chmod" command and the available options].

Example#11:
Syntax (Alphabetical Mode): chmod [ugoa][+-=][rwx] file

The "owner" or "u" is the username of the person who owns the file. In Linux by default, the user who creates a file or directory becomes the owner of that file.

The "group" or "g" is the group that owns the file. All users who belong to a group that owns a file will have the same access permissions to that file.

The "other" or "o" is a user who is not the owner of the file and also does not belong to the usergroup of the file. In other words, if you set a permission for "other", it will affect "everyone" by default.

Let us see an example, where we list the contents of a folder "KKHSOU" as given below. The "ll" command shown in this example is an alias of the "ls -l", we use it as a shorthand.

```
[nanu.kachari@beefy-miracle KKHSOU]$ ll
total 596
-rw-rw-r--. 1 nanu.kachari nanu.kachari 18242 Jul 1 15:45 MCA Linux Syllabus.docx
drwxrwxr-x. 2 nanu.kachari nanu.kachari 4096 Jul 9 15:00 Unit4
-rw-------. 1 nanu.kachari nanu.kachari 577536 Jul 1 16:19 Unit 4 - System Administration.doc
-rw-rw-r--. 1 nanu.kachari nanu.kachari 4857 Jul 9 14:56 unit4.txt
```

Now, let us change the "owner" or owners' permissions for the "MCA Linux Syllabus.docx" file. The "u" option is used to indicate that we are changing the users' or "owner" permissions. The "+" option indicates that we are adding these permissions (you could use "-" to indicate removing permissions and "=" to indicate exact permissions). The "rwx" indicates the "read", "write' and "execute" permissions. If the filename consists of "spaces" we will need to escape these with "\". Otherwise, the command line interface will interpret whatever follows after the spaces as parameters to the command.

You should know here, that we are allowed to use the "chmod" command on this file because we happen to be the owner of this file. Otherwise, only a root user will be allowed to "chmod" file permissions other than the owner.

```
[nanu.kachari@beefy-miracle KKHSOU]$ chmod u+rwx MCA\ Linux\ Syllabus.docx
```

If we use the "ll" command now, we can see that the permissions have been added to the "MCA Linux Syllabus.docx" file.

```
[nanu.kachari@beefy-miracle KKHSOU]$ ll
total 596
-rwxrw-r--. 1 nanu.kachari nanu.kachari 18242 Jul 1 15:45 MCA Linux Syllabus.docx
```

```
drwxrwxr-x. 2 nanu.kachari nanu.kachari   4096 Jul  9 15:00 Unit4
-rw-------. 1 nanu.kachari nanu.kachari 577536 Jul  1 16:19 Unit 4 - System Administration.doc
-rw-rw-r--. 1 nanu.kachari nanu.kachari   4857 Jul  9 14:56 unit4.txt
```

You will notice that the "MCA Linux Syllabus.docx" file permissions of the "owner" was "rw-" prior to chmod and we set it to "rwx".

Similarly, we can change the file permissions of the "group" and "others" using the "g" and "o" respectively in place of the "u" option. You can also use the "a" option with chmod to indicate "all" meaning "user", "group" and "others". You are encouraged to experiment with these options.

Here is an example how to reset the file permission for all to "r".

```
[nanu.kachari@beefy-miracle KKHSOU]$ chmod a=r MCA\ Linux\ Syllabus.docx
[nanu.kachari@beefy-miracle KKHSOU]$ ll
total 596
-r--r--r--. 1 nanu.kachari nanu.kachari  18242 Jul  1 15:45 MCA Linux Syllabus.docx
drwxrwxr-x. 2 nanu.kachari nanu.kachari   4096 Jul  9 15:00 Unit4
-rw-------. 1 nanu.kachari nanu.kachari 577536 Jul  1 16:19 Unit 4 - System Administration.doc
-rw-rw-r--. 1 nanu.kachari nanu.kachari   4857 Jul  9 14:56 unit4.txt
```

You could also use the octal or numeric mode to set file and directory permissions with the "chmod" command.

Numeric	Symbolic
0	---
1	--x
2	-w-
3	-wx
4	r--

5	r-x
6	rw-
7	rwx

Read (r)	4
Write (w)	2
Execute (x)	1
No Permission (-)	0

Example#12:

Syntax (Octal or Numeric Mode):

chmod [owner permissions: 0-7][group permissions: 0-7][others permissions: 0-7] file

Let us set file permissions using this mode on the file "MCA Linux Syllabus.docx" located within the "KKHSOU" folder. We will set the owner permission to "rwx", the group permission to "r-x" and the others permission to "r-x". Hence, in the octal or numeric mode these permissions will translate to "755".

```
[nanu.kachari@beefy-miracle KKHSOU]$ chmod 755 MCA\ Linux\ Syllabus.docx
[nanu.kachari@beefy-miracle KKHSOU]$ ll
total 596
-rwxr-xr-x. 1 nanu.kachari nanu.kachari  18242 Jul  1 15:45 MCA Linux Syllabus.docx
drwxrwxr-x. 2 nanu.kachari nanu.kachari   4096 Jul  9 15:00 Unit4
-rw-------. 1 nanu.kachari nanu.kachari 577536 Jul  1 16:19 Unit 4 - System Administration.doc
-rw-rw-r--. 1 nanu.kachari nanu.kachari   4857 Jul  9 14:56 unit4.txt
```

Therefore, you can see that we could use either of the two modes to set file or directory permissions.

There is a special permission called the sticky bit that can be added to world writable directories to prevent users from deleting other users' files. The sticky bit is added to the front of the permissions.

Example#13:

Let us suppose that we have a folder "tmp" which we want to assign permissions for everyone to be able to read, write or execute. Using the "ll" command you can view the current permissions assigned to the "tmp" folder.

```
nanu@nanu-Vostro1510:~/Desktop/KKHSOU/temp$ ll
total 12
drwxrwxr-x 3 nanu nanu 4096 Jul 12 20:48 ./
drwx------ 3 nanu nanu 4096 Jul 12 20:47 ../
drwxrwxr-x 2 nanu nanu 4096 Jul 12 20:48 tmp/
```

Now, we assign the sticky bit using the "1777" assignment to the "tmp" folder.

```
nanu@nanu-Vostro1510:~/Desktop/KKHSOU/temp$ chmod 1777 tmp
```

You can now see using the "ll" command that our permissions have been set. The letter "t" at the end of the permissions column indicates that the sticky bit set.

```
nanu@nanu-Vostro1510:~/Desktop/KKHSOU/temp$ ll
total 12
drwxrwxr-x 3 nanu nanu 4096 Jul 12 20:48 ./
drwx------ 3 nanu nanu 4096 Jul 12 20:47 ../
drwxrwxrwt 2 nanu nanu 4096 Jul 12 20:48 tmp/
```

4.3.5 TEMPORARILY DISABLE USER ACCOUNTS

In certain situations a System Administrator may need to temporarily disable system accounts. This situation could arise because of administrator policy, security threat or system maintenance.

The command "passwd" available on Linux systems has the option "-l" to lock a user account and the option "-u" to unlock. We could use this to temporarily disable a user account which we will see in our next example.

Example#14:

Let us disable a user that we had created previously in **Example#8**. And to understand what is happening in the background of this all we will also observe where exactly the change is happenning. We know that the files "/etc/passwd" and "/etc/shadow" contains the user account related information. Therefore, we will observe these files.

The "/etc/passwd" file contains an entry for the user "smiritisikha" as shown below.

```
smiritisikha:x:1007:1007:Smiritisikha Choudhury:/home/smiritisikha:/bin/bash
```

Also, the "/etc/shadow" file contains the following entry for the user "smiritisikha" as shown below.

```
smiritisikha:$6$76m6owD....45hBre.:15896:0:99999:7:::
```

As root or superuser we will lock the user account "smiritisikha" using the "passwd" command.

```
[root@beefy-miracle ~]# passwd smiritisikha -l
Locking password for user smiritisikha.
passwd: Success
[root@beefy-miracle ~]#
```

Now, we check and see "/etc/passwd" file and find that there is no change.

```
smiritisikha:x:1007:1007:Smiritisikha Choudhury:/home/smiritisikha:/bin/bash
```

However, in the "/etc/shadow" file you can see that the encrypted password field has been appended with a "!" character.

```
smiritisikha:!!$6$76m6owD...45hBre.:15896:0:99999:7:::
```

We could simply undo the operation performed above by using the "-u" option with the "passwd" command.

```
[root@beefy-miracle ~]# passwd smiritisikha -u
Unlocking password for user smiritisikha.
passwd: Success
[root@beefy-miracle ~]#
```

And see that the encrypted password field has been restored in the "/etc/shadow" file for the user.

```
smiritisikha:$6$76m6owD...45hBre.:15896:0:99999:7:::
```

Though editing the "/etc/shadow" file manually as root or superuser will result in the same effect. However, it is not recommended. [Note: you are encouraged to use the "man passwd" or the "info passwd" command to known more about this command or how to use it on your Linux system].

There are other methods to lock and unlock a user account, the "usermod" command with the "-L" option for locking and the "-U" option for unlocking could also be used for this purpose. [Note: you are encouraged to use the "man usermod" or the "info usermod" command to known more about this command or how to use it on your Linux system].

4.3.6 CREATING AND MOUNTING FILE SYSTEMS

A filesystem is comprised of the methods and data structures that an operating system uses to keep track of files on a disk or partition. Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called making a filesystem.

Filesystems are created or initialized with the “mkfs” command. “mkfs” is a front end that runs the appropriate program depending on the desired filesystem type. The type is selected with the “-t fstype” option. [Note: you are encouraged to use the "man mkfs" or the "info mkfs" command to known more about this command or how to use it on your Linux system].

Example#15:

Since, we do not want to temper with the filesystem of the existing system we are trying out our examples on. We will use a USB pendrive to try out this example. We will also

monitor the system messages in a separate terminal windows by using the “tail -f /var/log/messages” command as root or superuser.

Now, as we plug in this USB device we should get something similar to the “/var/log/messages” as show below.

```
Aug  8 17:21:09 beefy-miracle kernel: [200363.028636] usb 2-1.2: new high-speed USB device number 8 using ehci-pci
Aug  8 17:21:10 beefy-miracle kernel: [200363.116485] usb 2-1.2: New USB device found, idVendor=0951, idProduct=1623
Aug  8 17:21:10 beefy-miracle kernel: [200363.116491] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
Aug  8 17:21:10 beefy-miracle kernel: [200363.116494] usb 2-1.2: Product: DataTraveler 120
Aug  8 17:21:10 beefy-miracle kernel: [200363.116496] usb 2-1.2: Manufacturer: Kingston
Aug  8 17:21:10 beefy-miracle kernel: [200363.116499] usb 2-1.2: SerialNumber: XXXXXXXXXXXXXXXXXXXXXXXX
Aug  8 17:21:10 beefy-miracle kernel: [200363.117181] scsi13 : usb-storage 2-1.2:1.0
Aug  8 17:21:10 beefy-miracle mtp-probe: checking bus 2, device 8: "/sys/devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.2"
Aug  8 17:21:10 beefy-miracle mtp-probe: bus: 2, device: 8 was not an MTP device
Aug  8 17:21:11 beefy-miracle kernel: [200364.120024] scsi 13:0:0:0: Direct-Access     Kingston DataTraveler 120 1.00 PQ: 0 ANSI: 2
Aug  8 17:21:11 beefy-miracle kernel: [200364.120475] sd 13:0:0:0: Attached scsi generic sg3 type 0
Aug  8 17:21:11 beefy-miracle kernel: [200364.122229] sd 13:0:0:0: [sdc] 7827392 512-byte logical blocks: (4.00 GB/3.73 GiB)
Aug  8 17:21:11 beefy-miracle kernel: [200364.124850] sd 13:0:0:0: [sdc] Write Protect is off
Aug  8 17:21:11 beefy-miracle kernel: [200364.125483] sd 13:0:0:0: [sdc] Incomplete mode parameter data
Aug  8 17:21:11 beefy-miracle kernel: [200364.125487] sd 13:0:0:0: [sdc] Assuming drive cache: write through
Aug  8 17:21:11 beefy-miracle kernel: [200364.128715] sd 13:0:0:0: [sdc] Incomplete mode parameter data
Aug  8 17:21:11 beefy-miracle kernel: [200364.128718] sd 13:0:0:0: [sdc] Assuming drive cache: write through
Aug  8 17:21:11 beefy-miracle kernel: [200364.129370]  sdc:
Aug  8 17:21:11 beefy-miracle kernel: [200364.131838] sd 13:0:0:0: [sdc] Incomplete mode parameter data
Aug  8 17:21:11 beefy-miracle kernel: [200364.131843] sd 13:0:0:0: [sdc] Assuming drive cache: write through
Aug  8 17:21:11 beefy-miracle kernel: [200364.131847] sd 13:0:0:0: [sdc] Attached SCSI removable disk
```

We observe that the USB pen drive has been detected and assigned the “/dev/sdc” device string. We could find this out similarly using the “dmesg” command which is also show below.

```
[200363.028636] usb 2-1.2: new high-speed USB device number 8 using ehci-pci
[200363.116485] usb 2-1.2: New USB device found, idVendor=0951, idProduct=1623
[200363.116491] usb 2-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[200363.116494] usb 2-1.2: Product: DataTraveler 120
[200363.116496] usb 2-1.2: Manufacturer: Kingston
[200363.116499] usb 2-1.2: SerialNumber: XXXXXXXXXXXXXXXXXXXXXXXX
[200363.117181] scsi13 : usb-storage 2-1.2:1.0
[200364.120024] scsi 13:0:0:0: Direct-Access     Kingston DataTraveler 120 1.00 PQ: 0 ANSI: 2
[200364.120475] sd 13:0:0:0: Attached scsi generic sg3 type 0
[200364.122229] sd 13:0:0:0: [sdc] 7827392 512-byte logical blocks: (4.00 GB/3.73 GiB)
[200364.124850] sd 13:0:0:0: [sdc] Write Protect is off
[200364.124856] sd 13:0:0:0: [sdc] Mode Sense: 16 23 09 51
[200364.125483] sd 13:0:0:0: [sdc] Incomplete mode parameter data
[200364.125487] sd 13:0:0:0: [sdc] Assuming drive cache: write through
[200364.128715] sd 13:0:0:0: [sdc] Incomplete mode parameter data
[200364.128718] sd 13:0:0:0: [sdc] Assuming drive cache: write through
[200364.129370]  sdc:
[200364.131838] sd 13:0:0:0: [sdc] Incomplete mode parameter data
```

```
[200364.131843] sd 13:0:0:0: [sdc] Assuming drive cache: write through
[200364.131847] sd 13:0:0:0: [sdc] Attached SCSI removable disk
[200364.353949] SELinux: initialized (dev sdc, type vfat), uses genfs_contexts
```

Now, we will use the “fdisk” command with the “-l” option on “/dev/sdc” to list the partitions on this disk.

```
[root@beefy-miracle ~]# fdisk -l /dev/sdc
Disk /dev/sdc: 4007 MB, 4007624704 bytes, 7827392 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
  Device Boot      Start         End      Blocks   Id  System
[root@beefy-miracle ~]#
```

We can see that this disk does not contain any partitions. Let us go ahead and create a single primary partition on this disk for our example sake. We will do that by using the “fdisk” command on “/dev/sdc”.

```
[root@beefy-miracle ~]# fdisk /dev/sdc
Welcome to fdisk (util-linux 2.22.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
Command (m for help):
```

Now, press “p” to view the partition information of “/dev/sdc”.

```
Command (m for help): p
Disk /dev/sdc: 4007 MB, 4007624704 bytes, 7827392 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
  Device Boot      Start         End      Blocks   Id  System
Command (m for help):
```

Press “n” to create a new partition. Then press “p” to select primary partition and thereafter just enter the default values for the partition number, first sector, last sector.

```
Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-7827391, default 2048): 2048
Last sector, +sectors or +size{K,M,G} (2048-7827391, default 7827391): 7827391
Partition 1 of type Linux and of size 3.7 GiB is set
Command (m for help):
```

Now, we can view the primary partition we just created by pressing “p”.

```
Command (m for help): p
Disk /dev/sdc: 4007 MB, 4007624704 bytes, 7827392 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
  Device Boot      Start         End      Blocks   Id  System
/dev/sdc1            2048     7827391     3912672   83  Linux
```

Do note that here that we need to save the changes and we do that by pressing “w” to write out the changes to “/dev/sda”.

```
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
[root@beefy-miracle ~]#
```

You can type “fdisk -l /dev/sdc” to check what you we just did.

```
[root@beefy-miracle ~]# fdisk -l /dev/sdc
Disk /dev/sdc: 4007 MB, 4007624704 bytes, 7827392 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

```
  Device Boot      Start         End      Blocks   Id  System
/dev/sdc1            2048     7827391     3912672   83  Linux
```

So, now we have a successfully created a primary partition on “/dev/sdc” which is identified as “/dev/sdc1”. However, we are not done yet, as we need to create a filesystem on the partition to be able to mount and use it. We will use the “mkfs” command with the option “-t ext4” (-t is used to specify the type of filesystem) to create an “ext4” filesystem on “/dev/sdc1”.

```
[root@beefy-miracle ~]# mkfs -t ext4 /dev/sdc1
mke2fs 1.42.5 (29-Jul-2012)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
244800 inodes, 978168 blocks
48908 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1002438656
30 block groups
32768 blocks per group, 32768 fragments per group
8160 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736
Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
[root@beefy-miracle ~]#
```

Finally, we now have to mount this filesystem to a local mount point to be able to read or write to this newly created filesystem. Though most of the modern Linux distributions would probably auto mount for you, we will go ahead and discuss the manual mounting process.

Generally, the “/mnt” folder is used for mounting purposes on Linux and we will use this folder to mount our newly created filesystem. But, first just to keep things organized let us create a folder named “usb-disk” under “/mnt” which will be our mount point.

```
[root@beefy-miracle mnt]# mkdir /mnt/usb-disk
[root@beefy-miracle mnt]# ls -l /mnt
total 4
drwxr-xr-x. 2 root root 4096 Aug  8 18:21 usb-disk
[root@beefy-miracle mnt]#
```

Now, we use the “mount” command to mount our newly created filesystem on “/mnt/usb-disk”.

```
[root@beefy-miracle mnt]# mount -t ext4 /dev/sdb1 /mnt/usb-disk/
[root@beefy-miracle mnt]# ll /mnt/usb-disk/
total 24
drwxr-xr-x 3 root root  4096 Aug  8 18:08 ./
drwxr-xr-x 3 root root  4096 Aug  8 20:55 ../
drwx------ 2 root root 16384 Aug  8 18:08 lost+found/
[root@beefy-miracle mnt]#
```

To unmount we use the "umount" command with the specific mount point as a parameter.

```
[root@beefy-miracle mnt]# umount /mnt/usb-disk
[root@beefy-miracle mnt]# ls -l /mnt/usb-disk/
total 0
[root@beefy-miracle mnt]#
```

Do note here that the above example was performed using a USB pendrive which was only to demonstrate the creation and mounting or unmounting of a filesystem. In actual practice you would follow the same process however in all probability it would be on either IDE, SCSI, or SATA disks.

4.3.7 CHECKING AND MONITORING A LINUX SYSTEM

Probably the first duty of a System Administrator is to be aware of the hardware or software being used on a System. Linux has plenty of tools for checking and monitoring a Linux system. In this section, we will look at some of the tools used by System Administrators worldwide.

uname or Unix Name prints the system information about the current machine and the operating system running on it. [Note: you are encouraged to use the "man uname" or the "info uname" command to known more about this command or how to use it on your Linux system. Additionally, you can find the current Linux distribution you are running by typing "cat /etc/redhat-release" or "cat /etc/lsb-release" in the terminal]

```
[nanu.kachari@beefy-miracle ~]$ uname -a
Linux beefy-miracle.iitg.ernet.in 3.9.11-200.fc18.x86_64 #1 SMP Mon Jul 22 21:04:50 UTC 2013
x86_64 x86_64 x86_64 GNU/Linux
[nanu.kachari@beefy-miracle ~]$
```

dmidecode is a tool for dumping a computer's DMI (Desktop Management Interface) table which contains a description of the system's hardware components, as well as other useful information such as serial numbers and BIOS revision etc. You will need root or superuser privileges run this command. [Note: you are encouraged to use the "man dmidecode" or the "info dmidecode" command to known more about this command or how to use it on your Linux system.]

```
[root@beefy-miracle ~]# dmidecode --type bios
# dmidecode 2.12
SMBIOS 2.6 present.
Handle 0x0000, DMI type 0, 24 bytes
BIOS Information
        Vendor: American Megatrends Inc.
        Version: 0211G
        Release Date: 11/21/2011
        Address: 0xF0000
        Runtime Size: 64 kB
```

```
        ROM Size: 1024 kB
        Characteristics:
                ISA is supported
                PCI is supported
                PNP is supported
                BIOS is upgradeable
                BIOS shadowing is allowed
                ESCD support is available
                Boot from CD is supported
                Selectable boot is supported
                BIOS ROM is socketed
                EDD is supported
                5.25"/1.2 MB floppy services are supported (int 13h)
                3.5"/720 kB floppy services are supported (int 13h)
                3.5"/2.88 MB floppy services are supported (int 13h)
                Print screen service is supported (int 5h)
                8042 keyboard services are supported (int 9h)
                Serial services are supported (int 14h)
                Printer services are supported (int 17h)
                CGA/mono video services are supported (int 10h)
                ACPI is supported
                USB legacy is supported
                AGP is supported
                BIOS boot specification is supported
                Targeted content distribution is supported
        BIOS Revision: 2.11
Handle 0x003D, DMI type 13, 22 bytes
BIOS Language Information
        Language Description Format: Abbreviated
        Installable Languages: 1
                eng
        Currently Installed Language: eng
[root@beefy-miracle ~]#
```

Valid dmidecode type keywords are bios, system , baseboard, chassis, processor, memory , cache, connector, slot . Run the dmidecode with all these options on your system.

lscpu displays information about the CPU architecture. Similarly, there are also the **lspci**, **lsusb**, **lshw**, etc. for displaying information about hardware. Some of these tools maynot be installed by default on your Linux system. Therefore, you may have to install them. We will cover all about installing packages on Linux in another section later in this unit. [Note: you are encouraged to use the man pages or the info pages of these commands to known more about them or how to use them on your Linux system.]

```
[nanu.kachari@beefy-miracle ~]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 42
Stepping:              7
CPU MHz:               1600.000
BogoMIPS:              6185.94
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              6144K
NUMA node0 CPU(s):     0-3
[nanu.kachari@beefy-miracle ~]$
```

The **top** program provides a dynamic real-time view of a running system. It displays the system summary information as well as a list of processes or threads currently being managed by the Linux kernel. Though **top** is available by default on most distributions of Linux, you may use the **htop** interactive process viewer. It is similar to top, but allows you to scroll vertically and horizontally, so you can see all the processes running on the system, along with their full command lines. Tasks related to processes (killing, renicing) can be done without entering their PIDs. [Note: you are encouraged to use the man pages or the info pages of these commands to known more about them or how to use them on your Linux system.]

free displays the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel.

```
[root@beefy-miracle ~]# free -h
                 total       used       free     shared    buffers     cached
Mem:              3.7G       2.7G       989M       0B       178M       1.8G
-/+ buffers/cache:          741M       3.0G
Swap:             4.0G       218M       3.8G
[root@beefy-miracle ~]#
```

df displays the amount of disk space available on the file system containing each file name argument. If no file name is given, the space available on all currently mounted file systems is shown.

```
[root@beefy-miracle ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        1.9G     0  1.9G   0% /dev
tmpfs           1.9G  216K  1.9G   1% /dev/shm
tmpfs           1.9G  5.2M  1.9G   1% /run
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/sda3        61G   23G   38G  38% /
tmpfs           1.9G   56K  1.9G   1% /tmp
/dev/sda1       477M   45M  407M  10% /boot
/dev/sda2       394G  294G   81G  79% /home
[root@beefy-miracle ~]#
```

du estimates the file space usage. For example lets say, we want to know how much space in total a folder name KKHSOU is using we could use the "du" command with the "-sh" options (s indicates summary, h indicates human readable) as shown below.

```
[nanu.kachari@beefy-miracle Desktop]$ du -sh KKHSOU
2.0M    KKHSOU/
[nanu.kachari@beefy-miracle Desktop]$
```

netstat prints information about the Linux networking subsystem. For example, say you want to know what tcp connections are active currently on your system. You can use the "-t" option along with "netstat" as shown below.

```
[root@beefy-miracle ~]# netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0    208 beefy-miracle:ssh       172.xxx.xxx.xxx:10194 ESTABLISHED
[root@beefy-miracle ~]#
```

4.3.8 BACKUP AND RESTORE FILES

One of the top priorities of a System Administrator is a regular backup of critical data which can be restored in an advent of system crash or failure. That being said you should backup all critical data on separate media such as tape, writeable CD/DVD, removable USB disks, Network Storage etc., and then store your backup sets in a location separate from your Linux system. There are a variety of methods of performing backups with Linux. Example: **dump** which is a ext2/3/4 filesystem backup tool; **dd** is a tool to convert and copy a file; **cpio** is a tool for copying files to and from archives; **tar** saves many files together into a single tape or disk archive, and can restore individual files from the archive; **rsync** is a fast, versatile, remote (and local) file-copying tool. We will discuss only **tar** here. However, you are highly encouraged to check out the other tools mentioned earlier.

tar (originally for tape archive) is useful for archiving files. You can see more information by reading the man page (type "man tar" in your Linux terminal).

The tar program takes one operation mode argument. The most common ones are listed below.

c - Create a tar file from files

t - List all files in a tar file verbosely

x - Extract all files from a tar file

In addition to a function command line arguments the following arguments mentioned below are also used.

f - use archive file

z - filter the archive through gzip

v - verbosely list files processed

Now, let us try out an example where we create a tar archive with the options we just went through. I have a folder with the following files.

```
[nanu.kachari@beefy-miracle filestotar]$ ll -h
total 436K
-rw-rw-r--. 1 nanu.kachari nanu.kachari  77K Sep  5 23:36 a.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 166K Sep  5 23:37 b.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 169K Sep  5 23:37 c.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari  16K Sep  5 23:35 README.txt
[nanu.kachari@beefy-miracle filestotar]$
```

Creating a tar archive file:

First we will create a tar archive without using any compression.

```
[nanu.kachari@beefy-miracle filestotar]$ tar -cvf archive.tar a.txt b.txt c.txt README.txt
a.txt
b.txt
c.txt
README.txt
[nanu.kachari@beefy-miracle filestotar]$ ll -h
total 868K
-rw-rw-r--. 1 nanu.kachari nanu.kachari 430K Sep  5 23:49 archive.tar
-rw-rw-r--. 1 nanu.kachari nanu.kachari  77K Sep  5 23:36 a.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 166K Sep  5 23:37 b.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 169K Sep  5 23:37 c.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari  16K Sep  5 23:35 README.txt
[nanu.kachari@beefy-miracle filestotar]$
```

Now, let us create a tar archive with compression and compare the sizes of both these archives.

```
[nanu.kachari@beefy-miracle filestotar]$ tar -zcvf archive.tar.gz a.txt b.txt c.txt README.txt
a.txt
b.txt
```

```
c.txt
README.txt
[nanu.kachari@beefy-miracle filestotar]$ ll -h
total 996K
-rw-rw-r--. 1 nanu.kachari nanu.kachari 430K Sep  5 23:49 archive.tar
-rw-rw-r--. 1 nanu.kachari nanu.kachari 127K Sep  5 23:50 archive.tar.gz
-rw-rw-r--. 1 nanu.kachari nanu.kachari  77K Sep  5 23:36 a.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 166K Sep  5 23:37 b.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 169K Sep  5 23:37 c.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari  16K Sep  5 23:35 README.txt
[nanu.kachari@beefy-miracle filestotar]$
```

You can see that the compressed file obviously takes up less disk space. Alternatively, we can create a tar archive of the folder containing these files.

```
[nanu.kachari@beefy-miracle KKHSOU]$ tar -zcvf archive-folder.tar filestotar
filestotar/
filestotar/c.txt
filestotar/archive.tar
filestotar/b.txt
filestotar/a.txt
filestotar/archive.tar.gz
filestotar/README.txt
[nanu.kachari@beefy-miracle KKHSOU]$ ll -h
total 388K
-rw-rw-r--. 1 nanu.kachari nanu.kachari 383K Sep  5 23:54 archive-folder.tar
drwxrwxr-x. 2 nanu.kachari nanu.kachari 4.0K Sep  5 23:50 filestotar
[nanu.kachari@beefy-miracle KKHSOU]$
```

Testing a tar archive file:

```
[nanu.kachari@beefy-miracle filestotar]$ tar -tvf archive.tar
-rw-rw-r-- nanu.kachari/nanu.kachari 78561 2013-09-05 23:36 a.txt
-rw-rw-r-- nanu.kachari/nanu.kachari 169306 2013-09-05 23:37 b.txt
-rw-rw-r-- nanu.kachari/nanu.kachari 172462 2013-09-05 23:37 c.txt
-rw-rw-r-- nanu.kachari/nanu.kachari  16232 2013-09-05 23:35 README.txt
```

```
[nanu.kachari@beefy-miracle filestotar]$ tar -tvf archive.tar.gz
```

```
-rw-rw-r-- nanu.kachari/nanu.kachari 78561 2013-09-05 23:36 a.txt
-rw-rw-r-- nanu.kachari/nanu.kachari 169306 2013-09-05 23:37 b.txt
-rw-rw-r-- nanu.kachari/nanu.kachari 172462 2013-09-05 23:37 c.txt
-rw-rw-r-- nanu.kachari/nanu.kachari  16232 2013-09-05 23:35 README.txt
```

```
[nanu.kachari@beefy-miracle KKHSOU]$ tar -tvf archive-folder.tar
drwxrwxr-x nanu.kachari/nanu.kachari 0 2013-09-05 23:50 filestotar/
-rw-rw-r-- nanu.kachari/nanu.kachari 172462 2013-09-05 23:37 filestotar/c.txt
-rw-rw-r-- nanu.kachari/nanu.kachari 440320 2013-09-05 23:49 filestotar/archive.tar
-rw-rw-r-- nanu.kachari/nanu.kachari 169306 2013-09-05 23:37 filestotar/b.txt
-rw-rw-r-- nanu.kachari/nanu.kachari  78561 2013-09-05 23:36 filestotar/a.txt
-rw-rw-r-- nanu.kachari/nanu.kachari 129877 2013-09-05 23:50 filestotar/archive.tar.gz
-rw-rw-r-- nanu.kachari/nanu.kachari  16232 2013-09-05 23:35 filestotar/README.txt
```

Extracting a tar archive file:

Let us now create a folder named "temp" in our current working directory. We will use this folder to extract our tar archives into.

```
[nanu.kachari@beefy-miracle filestotar]$ mkdir temp
[nanu.kachari@beefy-miracle filestotar]$ cd temp/
[nanu.kachari@beefy-miracle temp]$ ll
total 0
```

Now, copy the tar archive file into this folder as shown.

```
[nanu.kachari@beefy-miracle temp]$ cp ../archive.tar .
[nanu.kachari@beefy-miracle temp]$ ll
total 432
-rw-rw-r--. 1 nanu.kachari nanu.kachari 440320 Sep  6 00:15 archive.tar
[nanu.kachari@beefy-miracle temp]$
```

Extract the tar archive file.

```
[nanu.kachari@beefy-miracle temp]$ tar -xvf archive.tar
a.txt
b.txt
c.txt
README.txt
[nanu.kachari@beefy-miracle temp]$ ll
total 436
```

```
-rw-rw-r--. 1 nanu.kachari nanu.kachari  78561 Sep  5 23:36 a.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 169306 Sep  5 23:37 b.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari 172462 Sep  5 23:37 c.txt
-rw-rw-r--. 1 nanu.kachari nanu.kachari  16232 Sep  5 23:35 README.txt
[nanu.kachari@beefy-miracle temp]$
```

Similarly, copy the other tar archives created in the previous steps to the "temp" folder and try extracting them one by one. Needless to say however, that you may delete the contents of the "temp" folder before trying out the other files.

```
[nanu.kachari@beefy-miracle temp]$ rm *
[nanu.kachari@beefy-miracle temp]$ ll
total 0
[nanu.kachari@beefy-miracle temp]$
```

4.3.9 INSTALLING AND REMOVING PACKAGES IN LINUX

One of the most common system administration task is maintaining software on a system. Maintaining software includes installaling and removing softwares and performing regular updates of already installed software. On a Linux system the package management utility used depends on the distribution of Linux. For RPM (Redhat Package Manager) based systems like the RedHat Enterprise Linux (RHEL), CentOS, Fedora, etc. the package management is usually done using the YUM (Yellowdog Updater, Modified) package manager. For Debian GNU/Linux distribution and its variants like Debian, Ubuntu, etc. the package management is usually done using the APT (Advanced Packaging Tool) package manager. Both the **yum** and **apt** package management utilities have their individual GUIs (graphical user interfaces) in addition to command line interfaces. We will be discussing the command line interface in this section only.

Generally, the repositories for both **yum** and **apt** are located accross the Internet though you can configure local repositories. However, in most cases your repositories will be located accross the Internet. We will use the repositories that are located accross the Internet in our examples. Therefore, you will need to ensure that you have a working

Internet connection. Also, you could be either directly connected to the Internet or connected via a proxy server. We will discuss both these configurations in our examples.

Using YUM:

To view the repositories currently configured on your Linux system you can use the command "yum repolist". We are using Fedora 18 Linux in our examples below.

```
[root@beefy-miracle ~]# yum repolist
Loaded plugins: langpacks, presto, refresh-packagekit
repo id                          repo name                          status
fedora/18/x86_64                 Fedora 18 – x86_64                 33,868
updates/18/x86_64                Fedora 18 - x86_64 – Updates       17,769
```

The yum repository configuration files are located in the "/etc/yum.repos.d/" folder. As we can see in the above example we have two repositories configured the *fedora* repository and the *fedora updates* repository in our Linux system. Do note that you can install and/or enable as many repositories that is needed. You are encouraged to lookup the "man yum.conf" command for more information.

Let us look at the *fedora* repository configuration which is specified by the "fedora.repo" file located in the "/etc/yum.repos.d/" folder. The yum repository configuration files have the ".repo" extension. Similarly, the *fedora updates* repository configuration is specified in the "fedora-updates.repo" file.

```
[root@beefy-miracle ~]# vim /etc/yum.repos.d/fedora.repo
[fedora]
name=Fedora $releasever - $basearch
failovermethod=priority
#baseurl=http://download.fedoraproject.org/pub/fedora/linux/releases/$releasever/Everything/$basearch/os/
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=fedora-$releasever&arch=$basearch
enabled=1
#metadata_expire=7d
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$basearch

[fedora-debuginfo]
name=Fedora $releasever - $basearch - Debug
failovermethod=priority
#baseurl=http://download.fedoraproject.org/pub/fedora/linux/releases/$releasever/Everything/$basearch/debug/
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=fedora-debug-$releasever&arch=$basearch
enabled=0
metadata_expire=7d
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$basearch
```

```
[fedora-source]
name=Fedora $releasever - Source
failovermethod=priority
#baseurl=http://download.fedoraproject.org/pub/fedora/linux/releases/$releasever/Everything/source/SRPMS/
mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=fedora-source-$releasever&arch=$basearch
enabled=0
metadata_expire=7d
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-$basearch
```

You can see that we have enabled the *fedora* repository and disabled the *fedora-debuginfo* and *fedora-source* repositories in our example.

Let us now run "yum update" to update all the currently installed packages on our system.

```
[root@beefy-miracle ~]# yum update
Loaded plugins: langpacks, presto, refresh-packagekit
fedora/18/x86_64/metalink                                | 9.4 kB  00:00:00
fedora                                                   | 4.2 kB  00:00:00
updates/18/x86_64/metalink                               | 7.4 kB  00:00:00
updates                                                  | 4.7 kB  00:00:00
Trying other mirror.
(1/2): updates/primary_db                                |  11 MB  00:00:12
(2/2): fedora/primary_db                                 |  17 MB  00:00:12
Resolving Dependencies
--> Running transaction check
---> Package graphviz.x86_64 0:2.28.0-26.fc18 will be updated
---> Package graphviz.x86_64 0:2.28.0-27.fc18 will be an update
---> Package graphviz-gd.x86_64 0:2.28.0-26.fc18 will be updated
---> Package graphviz-gd.x86_64 0:2.28.0-27.fc18 will be an update
---> Package libfm.x86_64 0:1.1.2.2-1.fc18 will be updated
---> Package libfm.x86_64 0:1.1.2.2-2.fc18 will be an update
---> Package libfm-devel.x86_64 0:1.1.2.2-1.fc18 will be updated
---> Package libfm-devel.x86_64 0:1.1.2.2-2.fc18 will be an update
--> Finished Dependency Resolution
Dependencies Resolved
================================================================================
 Package          Arch          Version             Repository          Size
================================================================================
Updating:
 graphviz         x86_64        2.28.0-27.fc18      updates             1.2 M
 graphviz-gd      x86_64        2.28.0-27.fc18      updates             30 k
 libfm            x86_64        1.1.2.2-2.fc18      updates             255 k
 libfm-devel      x86_64        1.1.2.2-2.fc18      updates             33 k
Transaction Summary
================================================================================
Upgrade  4 Packages
```

```
Total download size: 1.5 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
(1/4): graphviz-2.28.0-27.fc18.x86_64.rpm                  | 1.2 MB  00:00:05
(2/4): graphviz-gd-2.28.0-27.fc18.x86_64.rpm               |  30 kB  00:00:00
(3/4): libfm-1.1.2.2-2.fc18.x86_64.rpm                     | 255 kB  00:00:01
(4/4): libfm-devel-1.1.2.2-2.fc18.x86_64.rpm               |  33 kB  00:00:00
--------------------------------------------------------------------------------
Total                                           194 kB/s  | 1.5 MB     00:07
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Updating   : libfm-1.1.2.2-2.fc18.x86_64                   1/8
  Updating   : graphviz-2.28.0-27.fc18.x86_64                2/8
  Updating   : graphviz-gd-2.28.0-27.fc18.x86_64             3/8
  Updating   : libfm-devel-1.1.2.2-2.fc18.x86_64             4/8
  Cleanup    : graphviz-gd-2.28.0-26.fc18.x86_64             5/8
  Cleanup    : libfm-devel-1.1.2.2-1.fc18.x86_64             6/8
  Cleanup    : graphviz-2.28.0-26.fc18.x86_64                7/8
  Cleanup    : libfm-1.1.2.2-1.fc18.x86_64                   8/8
  Verifying  : graphviz-2.28.0-27.fc18.x86_64                1/8
  Verifying  : libfm-1.1.2.2-2.fc18.x86_64                   2/8
  Verifying  : libfm-devel-1.1.2.2-2.fc18.x86_64             3/8
  Verifying  : graphviz-gd-2.28.0-27.fc18.x86_64             4/8
  Verifying  : libfm-1.1.2.2-1.fc18.x86_64                   5/8
  Verifying  : graphviz-gd-2.28.0-26.fc18.x86_64             6/8
  Verifying  : graphviz-2.28.0-26.fc18.x86_64                7/8
  Verifying  : libfm-devel-1.1.2.2-1.fc18.x86_64             8/8
Updated:
        graphviz.x86_64 0:2.28.0-27.fc18        graphviz-gd.x86_64 0:2.28.0-27.fc18      libfm.x86_64   0:1.1.2.2-2.fc18
        libfm-devel.x86_64 0:1.1.2.2-2.fc18
Complete!
[root@beefy-miracle ~]#
```

Alternatively, we could have also used the "yum update -y" command to answer yes to all the updates found by yum. Do remember that the output on your system will differ from this example mentioned above as the update state of your system will differ from the system where this example was performed.

In case you connect to the Internet using a http proxy server you will need the following lines added to the "~/.bashrc" file. Generally, we use the root user to perform updates using **yum** and therfore we need to add these lines to the "/root/.bashrc" file.

```
[root@beefy-miracle ~]# vim /root/.bashrc
```

```
export http_proxy="http://username:password@proxyserver:port"
export https_proxy="http://username:password@proxyserver:port"
export ftp_proxy="http://username:password@proxyserver:port"
```

You have to replace username with your proxy username, password with your proxy password, proxyserver and port with your proxy server IP address or hostname and port. Once this is in place we use the "source ~/.bashrc" command to refresh the environment changes we just made.

To install packages we use the command "yum install *package-name*". For example let us try installing **vim**.

```
[root@beefy-miracle ~]# yum install vim
Loaded plugins: langpacks, presto, refresh-packagekit
Package 2:vim-enhanced-7.4.016-1.fc18.x86_64 already installed and latest version
Nothing to do
[root@beefy-miracle ~]#
```

You can see that vim was already installed therefore yum reports that indeed we already have vim installed on our system. This gives us an opportunity to remove this package as well as install this package to complete our example. Let us remove the package **vim** using "yum erase vim".

```
[root@beefy-miracle ~]# yum erase vim
Loaded plugins: langpacks, presto, refresh-packagekit
Resolving Dependencies
--> Running transaction check
---> Package vim-enhanced.x86_64 2:7.4.016-1.fc18 will be erased
--> Finished Dependency Resolution
Dependencies Resolved
================================================================================
 Package          Arch      Version                      Repository        Size
================================================================================
Removing:
 vim-enhanced  x86_64  2:7.4.016-1.fc18        @updates        2.1 M
Transaction Summary
================================================================================
Remove  1 Package
Installed size: 2.1 M
Is this ok [y/N]: y
```

```
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Erasing    : 2:vim-enhanced-7.4.016-1.fc18.x86_64            1/1
  Verifying  : 2:vim-enhanced-7.4.016-1.fc18.x86_64            1/1
Removed:
          vim-enhanced.x86_64              2:7.4.016-1.fc18
Complete!
[root@beefy-miracle ~]#
```

Now, we install the package **vim** using the command "yum install vim".

```
[root@beefy-miracle ~]# yum install vim
Loaded plugins: langpacks, presto, refresh-packagekit
Resolving Dependencies
--> Running transaction check
---> Package vim-enhanced.x86_64 2:7.4.016-1.fc18 will be installed
--> Finished Dependency Resolution
Dependencies Resolved
================================================================
 Package          Arch     Version                     Repository        Size
================================================================================
Installing:
 vim-enhanced     x86_64 2:7.4.016-1.fc18      updates               1.0 M
Transaction Summary
================================================================================
Install  1 Package
Total download size: 1.0 M
Installed size: 2.1 M
Is this ok [y/N]: y
Downloading Packages:
vim-enhanced-7.4.016-1.fc18.x86_64.rpm                    | 1.0 MB  00:00:04
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
```

```
  Installing : 2:vim-enhanced-7.4.016-1.fc18.x86_64                1/1
  Verifying  : 2:vim-enhanced-7.4.016-1.fc18.x86_64                1/1
Installed:
  vim-enhanced.x86_64              2:7.4.016-1.fc18
Complete!
[root@beefy-miracle ~]#
```

Though we have seen only a few examples of using **yum**, you should be able to install and remove packages now. However, you are encouraged to look up the "man yum" on your system and read about the various options to use. Alternatively, you can visit the official page located at "http://yum.baseurl.org/wiki/YumCommands" and read more about it.

Using APT:

Apt stores a list of repositories or software channels in the file "/etc/apt/sources.list" and will look similar to the one shown. We will be using Ubuntu 12.04 LTS for our examples below.

```
root@lano:~#
vim /etc/apt/sources.list
#

# deb cdrom:[Ubuntu-Server 11.04 _Natty Narwhal_ - Release amd64 (20110426)]/ natty main restricted

# deb cdrom:[Ubuntu-Server 11.04 _Natty Narwhal_ - Release amd64 (20110426)]/ natty main restricted



# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to

# newer versions of the distribution.

deb http://in.archive.ubuntu.com/ubuntu/ precise main restricted

deb-src http://in.archive.ubuntu.com/ubuntu/ precise main restricted



## Major bug fix updates produced after the final release of the

## distribution.

deb http://in.archive.ubuntu.com/ubuntu/ precise-updates main restricted
```

```
deb-src http://in.archive.ubuntu.com/ubuntu/ precise-updates main restricted

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team. Also, please note that software in universe WILL NOT receive any
## review or updates from the Ubuntu security team.
deb http://in.archive.ubuntu.com/ubuntu/ precise universe
deb-src http://in.archive.ubuntu.com/ubuntu/ precise universe
deb http://in.archive.ubuntu.com/ubuntu/ precise-updates universe
deb-src http://in.archive.ubuntu.com/ubuntu/ precise-updates universe

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## multiverse WILL NOT receive any review or updates from the Ubuntu
## security team.
deb http://in.archive.ubuntu.com/ubuntu/ precise multiverse
deb-src http://in.archive.ubuntu.com/ubuntu/ precise multiverse
deb http://in.archive.ubuntu.com/ubuntu/ precise-updates multiverse
deb-src http://in.archive.ubuntu.com/ubuntu/ precise-updates multiverse

## Uncomment the following two lines to add software from the 'backports'
## repository.
## N.B. software from this repository may not have been tested as
```

```
## extensively as that contained in the main release, although it includes

## newer versions of some applications which may provide useful features.

## Also, please note that software in backports WILL NOT receive any review

## or updates from the Ubuntu security team.

# deb http://in.archive.ubuntu.com/ubuntu/ natty-backports main restricted universe multiverse

# deb-src http://in.archive.ubuntu.com/ubuntu/ natty-backports main restricted universe multiverse
```

The default repositories that come alongwith the Ubuntu distribution used in the example above suffices our current needs. However, you could enable/disable the existing repositories by uncommenting/commenting the corresponding apt line (i.e. delete the '#' at the beginning of the line) or even add new repositories as well. You are encouraged to lookup "man sources.list" as well as "man apt-get" for more information. You need to run the **apt-get** command with supervisor previlages therefore you can use either "sudo apt-get install" or first "sudo -i" then enter your password, to get to a root prompt and then use "apt-get install". We will use the later of the two options.

Let us run the "apt-get update" command to update or resynchronize the package index files from their sources.

```
nanu@lano:~$ sudo -i

[sudo] password for nanu:

root@lano:~# apt-get update

Hit http://dl.google.com stable Release.gpg

Hit http://dl.google.com stable Release

Hit http://dl.google.com stable/main i386 Packages

Ign http://dl.google.com stable/main TranslationIndex

Hit http://archive.ubuntu.com precise Release.gpg

Hit http://archive.canonical.com precise Release.gpg
```

```
Hit http://extras.ubuntu.com precise Release.gpg

Get:1 http://archive.ubuntu.com precise-updates Release.gpg [198 B]

Hit http://archive.canonical.com precise Release

Hit http://extras.ubuntu.com precise Release

Get:2 http://archive.ubuntu.com precise-security Release.gpg [198 B]

Hit http://archive.canonical.com precise/partner i386 Packages

Hit http://extras.ubuntu.com precise/main Sources

Hit http://archive.ubuntu.com precise Release

Ign http://dl.google.com stable/main Translation-en_IN

Ign http://archive.canonical.com precise/partner TranslationIndex

Get:3 http://archive.ubuntu.com precise-updates Release [49.6 kB]

Hit http://extras.ubuntu.com precise/main i386 Packages

Ign http://dl.google.com stable/main Translation-en

Ign http://extras.ubuntu.com precise/main TranslationIndex

Get:4 http://archive.ubuntu.com precise-security Release [49.6 kB]

Hit http://archive.ubuntu.com precise/main Sources

Hit http://archive.ubuntu.com precise/restricted Sources

Hit http://archive.ubuntu.com precise/universe Sources
```

```
Hit http://archive.ubuntu.com precise/multiverse Sources

Hit http://archive.ubuntu.com precise/main i386 Packages

Hit http://archive.ubuntu.com precise/restricted i386 Packages

Ign http://archive.canonical.com precise/partner Translation-en_IN

Hit http://archive.ubuntu.com precise/universe i386 Packages

Ign http://archive.canonical.com precise/partner Translation-en

Hit http://archive.ubuntu.com precise/multiverse i386 Packages

Ign http://extras.ubuntu.com precise/main Translation-en_IN

Hit http://archive.ubuntu.com precise/main TranslationIndex

Ign http://extras.ubuntu.com precise/main Translation-en

Hit http://archive.ubuntu.com precise/multiverse TranslationIndex

Hit http://archive.ubuntu.com precise/restricted TranslationIndex

Hit http://archive.ubuntu.com precise/universe TranslationIndex

Get:5 http://archive.ubuntu.com precise-updates/main Sources [416 kB]

Get:6 http://archive.ubuntu.com precise-updates/restricted Sources [7,031 B]

Get:7 http://archive.ubuntu.com precise-updates/universe Sources [95.7 kB]

Get:8 http://archive.ubuntu.com precise-updates/multiverse Sources [8,343 B]

Get:9 http://archive.ubuntu.com precise-updates/main i386 Packages [707 kB]
```

Get:10 http://archive.ubuntu.com precise-updates/restricted i386 Packages [11.4 kB]

Get:11 http://archive.ubuntu.com precise-updates/universe i386 Packages [219 kB]

Get:12 http://archive.ubuntu.com precise-updates/multiverse i386 Packages [14.0 kB]

Hit http://archive.ubuntu.com precise-updates/main TranslationIndex

Hit http://archive.ubuntu.com precise-updates/multiverse TranslationIndex

Hit http://archive.ubuntu.com precise-updates/restricted TranslationIndex

Hit http://archive.ubuntu.com precise-updates/universe TranslationIndex

Get:13 http://archive.ubuntu.com precise-security/main Sources [85.2 kB]

Get:14 http://archive.ubuntu.com precise-security/restricted Sources [2,494 B]

Get:15 http://archive.ubuntu.com precise-security/universe Sources [28.0 kB]

Get:16 http://archive.ubuntu.com precise-security/multiverse Sources [1,804 B]

Get:17 http://archive.ubuntu.com precise-security/main i386 Packages [331 kB]

Get:18 http://archive.ubuntu.com precise-security/restricted i386 Packages [4,620 B]

Get:19 http://archive.ubuntu.com precise-security/universe i386 Packages [84.5 kB]

Get:20 http://archive.ubuntu.com precise-security/multiverse i386 Packages [2,640 B]

Hit http://archive.ubuntu.com precise-security/main TranslationIndex

Hit http://archive.ubuntu.com precise-security/multiverse TranslationIndex

Hit http://archive.ubuntu.com precise-security/restricted TranslationIndex

```
Hit http://archive.ubuntu.com precise-security/universe TranslationIndex

Hit http://archive.ubuntu.com precise/main Translation-en

Hit http://archive.ubuntu.com precise/multiverse Translation-en

Hit http://archive.ubuntu.com precise/restricted Translation-en

Hit http://archive.ubuntu.com precise/universe Translation-en

Hit http://archive.ubuntu.com precise-updates/main Translation-en

Hit http://archive.ubuntu.com precise-updates/multiverse Translation-en

Hit http://archive.ubuntu.com precise-updates/restricted Translation-en

Hit http://archive.ubuntu.com precise-updates/universe Translation-en

Hit http://archive.ubuntu.com precise-security/main Translation-en

Hit http://archive.ubuntu.com precise-security/multiverse Translation-en

Hit http://archive.ubuntu.com precise-security/restricted Translation-en

Hit http://archive.ubuntu.com precise-security/universe Translation-en

Fetched 2,119 kB in 43s (48.7 kB/s)

Reading package lists... Done

root@lano:~#
```

Now, we will use "apt-get upgrade" to install the newest versions of all packages currently installed on the system from the sources enumerated in /etc/apt/sources.list.

```
root@lano:~# apt-get upgrade

Reading package lists... Done
```

Building dependency tree

Reading state information... Done

The following packages will be upgraded:

dhcp3-client dhcp3-common firefox firefox-branding firefox-globalmenu

isc-dhcp-client isc-dhcp-common

7 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

Need to get 28.8 MB of archives.

After this operation, 1,630 kB of additional disk space will be used.

Do you want to continue [Y/n]? **Y**

Get:1 http://archive.ubuntu.com/ubuntu/ precise-updates/main isc-dhcp-client i386 4.1.ESV-R4-0ubuntu5.9 [288 kB]

Get:2 http://archive.ubuntu.com/ubuntu/ precise-updates/main isc-dhcp-common i386 4.1.ESV-R4-0ubuntu5.9 [346 kB]

Get:3 http://archive.ubuntu.com/ubuntu/ precise-updates/universe dhcp3-client all 4.1.ESV-R4-0ubuntu5.9 [2,196 B]

Get:4 http://archive.ubuntu.com/ubuntu/ precise-updates/universe dhcp3-common all 4.1.ESV-R4-0ubuntu5.9 [1,750 B]

Get:5 http://archive.ubuntu.com/ubuntu/ precise-updates/main firefox i386 24.0+build1-0ubuntu0.12.04.1 [28.1 MB]

Get:6 http://archive.ubuntu.com/ubuntu/ precise-updates/main firefox-branding i386 24.0+build1-0ubuntu0.12.04.1 [8,956 B]

Get:7 http://archive.ubuntu.com/ubuntu/ precise-updates/main firefox-globalmenu i386 24.0+build1-0ubuntu0.12.04.1 [8,958 B]

Fetched 28.8 MB in 1min 2s (459 kB/s)

(Reading database ... 940092 files and directories currently installed.)

Preparing to replace isc-dhcp-client 4.1.ESV-R4-0ubuntu5.8 (using .../isc-dhcp-client_4.1.ESV-R4-0ubuntu5.9_i386.deb) ...

Unpacking replacement isc-dhcp-client ...

Preparing to replace isc-dhcp-common 4.1.ESV-R4-0ubuntu5.8 (using .../isc-dhcp-common_4.1.ESV-R4-0ubuntu5.9_i386.deb) ...

Unpacking replacement isc-dhcp-common ...

Preparing to replace dhcp3-client 4.1.ESV-R4-0ubuntu5.8 (using .../dhcp3-client_4.1.ESV-R4-0ubuntu5.9_all.deb) ...

Unpacking replacement dhcp3-client ...

Preparing to replace dhcp3-common 4.1.ESV-R4-0ubuntu5.8 (using .../dhcp3-common_4.1.ESV-R4-0ubuntu5.9_all.deb) ...

Unpacking replacement dhcp3-common ...

Preparing to replace firefox 23.0+build2-0ubuntu0.12.04.1 (using .../firefox_24.0+build1-0ubuntu0.12.04.1_i386.deb) ...

Unpacking replacement firefox ...

Preparing to replace firefox-branding 23.0+build2-0ubuntu0.12.04.1 (using .../firefox-branding_24.0+build1-0ubuntu0.12.04.1_i386.deb) ...

Unpacking replacement firefox-branding ...

```
Preparing to replace firefox-globalmenu 23.0+build2-0ubuntu0.12.04.1 (using .../firefox-globalmenu_24.0+build1-0ubuntu0.12.04.1_i386.deb) ...

Unpacking replacement firefox-globalmenu ...

Processing triggers for man-db ...

Processing triggers for bamfdaemon ...

Rebuilding /usr/share/applications/bamf.index...

Processing triggers for desktop-file-utils ...

Processing triggers for gnome-menus ...

Setting up isc-dhcp-common (4.1.ESV-R4-0ubuntu5.9) ...

Setting up isc-dhcp-client (4.1.ESV-R4-0ubuntu5.9) ...

Setting up dhcp3-client (4.1.ESV-R4-0ubuntu5.9) ...

Setting up dhcp3-common (4.1.ESV-R4-0ubuntu5.9) ...

Setting up firefox (24.0+build1-0ubuntu0.12.04.1) ...

Please restart all running instances of firefox, or you will experience problems.

Setting up firefox-branding (24.0+build1-0ubuntu0.12.04.1) ...

Setting up firefox-globalmenu (24.0+build1-0ubuntu0.12.04.1) ...

root@lano:~#
```

In case you connect to the Internet using a http proxy server you will need the following lines added to the "/etc/apt/apt.conf" file.

```
root@lano:~# vim /etc/apt/apt.conf
```

```
Acquire::http::proxy "http://username:password@proxyserver:port";

Acquire::https::proxy "http://username:password@proxyserver:port";

Acquire::ftp::proxy "http://username:password@proxyserver:port";
```

You have to replace username with your proxy username, password with your proxy password, proxyserver and port with your proxy server IP address or hostname and port.

To install packages we use the command "apt-get install *package-name*". For example let us try installing **vim**.

```
root@lano:~# apt-get install vim

Reading package lists... Done

Building dependency tree

Reading state information... Done

vim is already the newest version.

0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

root@lano:~#
```

You can see that apt-get reports that we already have **vim** installed on our system. This gives us an opportunity to remove this package as well as install this package to complete our example. Let us remove the package **vim** using "apt-get remove vim".

```
root@lano:~# apt-get remove vim

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following packages will be REMOVED:
```

```
  vim

0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.

After this operation, 1,950 kB disk space will be freed.

Do you want to continue [Y/n]? Y

(Reading database ... 940091 files and directories currently installed.)

Removing vim ...

update-alternatives: using /usr/bin/vim.tiny to provide /usr/bin/vi (vi) in auto mode.

update-alternatives: using /usr/bin/vim.tiny to provide /usr/bin/view (view) in auto mode.

update-alternatives: using /usr/bin/vim.tiny to provide /usr/bin/ex (ex) in auto mode.

update-alternatives: using /usr/bin/vim.tiny to provide /usr/bin/rview (rview) in auto mode.

root@lano:~#
```

Now, we install the package **vim** using the command "apt-get install vim".

```
root@lano:~# apt-get install vim

Reading package lists... Done

Building dependency tree

Reading state information... Done

Suggested packages:

  ctags vim-doc vim-scripts

The following NEW packages will be installed:
```

```
  vim

0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.

Need to get 979 kB of archives.

After this operation, 1,950 kB of additional disk space will be used.

Get:1 http://archive.ubuntu.com/ubuntu/ precise-updates/main vim i386 2:7.3.429-2ubuntu2.1 [979 kB]

Fetched 979 kB in 4s (214 kB/s)

Selecting previously unselected package vim.

(Reading database ... 940086 files and directories currently installed.)

Unpacking vim (from .../vim_2%3a7.3.429-2ubuntu2.1_i386.deb) ...

Setting up vim (2:7.3.429-2ubuntu2.1) ...

update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vim (vim) in auto mode.

update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vimdiff (vimdiff) in auto mode.

update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rvim (rvim) in auto mode.

update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rview (rview) in auto mode.

update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vi (vi) in auto mode.

update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/view (view) in auto mode.

update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/ex (ex) in auto mode.

root@lano:~#
```

Though we have seen only a few examples of using **apt-get**, you should now be able to install and remove packages. However, you are encouraged to look up the "man apt-get" on your system and read about the various options to use.

4.3.10 GRAPHICAL INTERFACES IN LINUX

Throuhgout this unit in all the examples discussed we were using the command line interface or CLI to perform all our tasks which required commands to be typed on the keyboard. Though CLI is always the preferred way on Linux you should also be aware of the various Graphical User Interfaces or GUIs that are available today. If you are currently using any of the recent distribution of Linux be it Fedora, Ubuntu, etc. you are most likely to have either GNOME or KDE as the default GUI. These GUIs are the most widely used ones. You are encouraged to read more about these GUIs at the website links provided below.

"http://www.gnome.org/gnome-3/"

"http://www.kde.org/workspaces/"

CHECK YOUR PROGRESS

1. Write the Command to do the following:

   a. Create a new directory "KKHSOU"
   b. Create a user "abc"
   c. Change the password of the existing user say "kkhsou")
   d. Display IP address
   c. Extract "kkhsou.tar"

2. Fill in the Blanks
   i) To clear the screen we type _____.
   ii) We can install the "vlc" player in UBUNTU OS using the command _____.
   iii) To get the date in the command line we use _____ .

4.4 LET US SUM UP

In this unit we have tried to get acquainted with System Administration using the “with examples” approach on Linux. Though we have covered only a few of the topics so far, these topics are intended to inspire are point you in the right direction to further explore the realm of Linux System Administration.

What we learned in this unit.

- We used the "ls" command to list files/directories.
- We used "su -" (on Fedora Linux) and "sudo -i" (on Ubuntu Linux) to switch to the root user.
- We saw that the system configuration files are generally located within the "/etc/" folder.
- We saw that the system log files are generally located wihtin the "/var/log/" folder.
- We added users using the "useradd" and "newusers" command.
- We deleted users using the "userdel" command.
- We changed permissions of files/directories using the "chmod" command and also applied the sticky bit permission to world writable directories to prevent users from deleting other users' files.
- We changed ownerships of files/firectories using the "chown" command.
- We used the command "passwd" with the option "-l" to lock a user account and the option "-u" to unlock.
- We used the "fdisk" utility to list and create partition on a disk.

- We used the "mkfs" utility to create filesystems on a disk.

- We used the "mount" utility to mount a filesystem.

- We saw that there are some useful utilities like uname, dmidecode, lscpu, free, df, du, netstat that are bundled with Linux and can be used to monitor/check the system.
- We performed backup and restore operation using the “tar” utility.

- We performed software update, installation, removal using the “yum” and “apt-get” utilities.

- We very briefly discussed about the two popular GUIs on Linux namely GNOME and KDE. Though we did not discuss in detail you are encouraged to explore further.

4.5 ANSWERS TO CHECK YOUR PROGRESS

1. a) mkdir KKHSOU
   b) useradd abc
   c) passwd kkhsou
   d) ifconfig
   c) tar -xvf kkhsou.tar

2. i) clear
   ii) apt-get install vlc
   iii) date

4.6 FURTHER READINGS

- Linux Documentation Project [http://www.tldp.org/]
- Documentation for Linux enthusiasts [http://www.linuxdocs.org/]
- Linux Man pages installed on local system.

4.7 MODEL QUESTIONS

Q1.List some of the common system administrative tasks.

Q2.Which command will display the file/directory ownerships.

Q3.Where are the default system configuration files located in Linux.

Q4.Where are the default system log files located in Linux.

Q5.Which command is used to add multiple users.

Q6.Which file contains the Group ID of users.

Q7.Which file contains user's Home Directory location.

Q8.Which command can be used to temporary disable a user login.

Q9.Which command is used to delete a user including the users' Home Directory.

Q10. Which command is used to make a file executable by all.

Q11. Which command can be used to create a Directory.

Q12. Which command can be used to view the current partitions on a disk.

Q13. Which command can be used to view the current RAM usage.

Q14. Which command can be used to view the currently active TCP connections.

Q15. What does "f" stand for in the command "tar -cvf archive.tar a.txt b.txt c.txt README.txt".

UNIT- 5 LINUX NETWORKING

UNIT STRUCTURE

5.1 LEARNING OBJECTIVE

After going through this unit, you will be able to:

- install and configure a simple LAN
- install and configure a Proxy Server using Squid
- install and configure a DNS Server using BIND
- install and configure a Web Server using Apache
- install and configure a File Server using Samba
- install and configure a simple Mail Server using Postfix
- install and configure a DHCP Server
- install and configure a SSH Server and Client
- install and configure a FTP Server and Client

5.2 INTRODUCTION

Most of the computers today are networked in some way to each other either on the Internet or privately in universities, offices, campus wide, at home, etc. Therefore, though any standalone Linux System can act as a fully functional server. Its true power and usefulness can only be realized if the Linux System is networked and interconnected with other systems.

In this unit, we will be specifically looking at networking using the Linux Operating System. This unit assumes that you have already installed a Linux distribution on a computer or computers and ready to try out all the examples stated in the unit as we progress. This unit also assumes that you are aware of the basics of TCP/IP networking. If you need to brush up on that topic, you are highly encouraged to refer to the numerous ebooks and tutorials available online on the Internet.

5.3 INSTALLATION AND CONFIGURATION OF A LAN

A Local Area Network or LAN is generally a small setup of interconnected computers connected via Ethernet Switches. A LAN would consist of both passive and active components. The passive components would include CAT6 cables, Patch Cords, RJ45 ports, Patch Panels, etc. The active components would include the Ethernet Switches, etc.

5.3.1 INSTALLATION

In order to setup a simple LAN of two computers and walk through our examples in this unit, we require the following items.

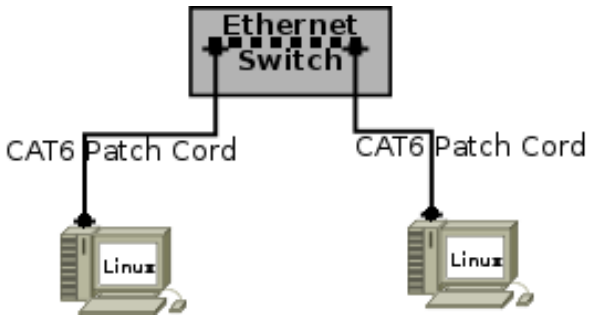
Passive Components : 2 nos. of CAT 6 patch cords.

Active Components : 1 nos. Ethernet Switch.

Computers : 2 nos. of computers running Linux. Both the computers should have at least one Network Interface each on them.

Operating System : Fedora 18 (64 bit) and Ubuntu 12.04.2 LTS (64 bit).

Once we have all the components we connect them as shown in the figure below.



We will need at least two IP Addresses to configure the network interfaces on these two computers. Let us suppose that we will use the following configurations.

For Computer A (Fedora 18):

IP Address: 192.168.1.1 Subnet Mask: 255.255.255.0 Gateway: 192.168.1.1

For Computer B (Ubuntu 12.04.2 LTS):

IP Address: 192.168.1.2 Subnet Mask: 255.255.255.0 Gateway: 192.168.1.2

5.3.2 CONFIGURATION

Configuring the IP Address:

In the examples below the **bold-face** font indicates the commands or text that needs to be typed in.

On Computer A (Fedora 18)

```
[root@computer-a ~]# vim /etc/sysconfig/network-scripts/ifcfg-p5p1
UUID="48eed959-2611-46c0-b0b4-b90d38c05bd2"
NM_CONTROLLED="yes"
BOOTPROTO=none
```

```
DEVICE="p5p1"
ONBOOT="yes"
TYPE=Ethernet
IPV4_FAILURE_FATAL=yes
IPV6INIT=no
IPADDR0=192.168.1.1
PREFIX0=24
HWADDR=AA:BB:CC:DD:EE:FF
```

After making these changes to the file we need to type "**:wq**" and press the enter key on the keyboard to write the changes and quit. Now, in order to apply these changes we need to restart the network service.

```
[root@computer-a ~]# systemctl restart network.service
```

On Computer B (Ubuntu 12.04.2 LTS)

```
root@computer-b:~# vim /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth1 inet static
    address 192.168.1.2
    netmask 255.255.255.0
```

After making these changes to the file we need to type "**:wq**" and press the enter key on the keyboard to write the changes and quit. In order to apply these changes we need to stop and start networking.

```
root@computer-b:~# /etc/init.d/networking stop
root@computer-b:~# /etc/init.d/networking start
```

Once both the computers are setup with the IP Addresses successfully, we test the connectivity with the **ping** utility.

On Computer A (Fedora 18)

```
[root@computer-a ~]# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.142 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.141 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.129 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.139 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.143 ms
^C
--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.129/0.138/0.143/0.015 ms
```

On Computer B (Ubuntu 12.04.2 LTS)

```
root@computer-b:~# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=0.447 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=0.257 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=0.182 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=0.238 ms
64 bytes from 192.168.1.1: icmp_req=5 ttl=64 time=0.416 ms
^C
--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.182/0.308/0.447/0.104 ms
```

You can observe from the ping replies that we now have a basic working TCP/IP network.

5.4 INSTALLATION AND CONFIGURATION OF A PROXY SERVER – SQUID

The Squid proxy server is a cache based proxy. It will fetch all web requests to populate its cache and then allow access to its cache based on its configuration. Squid is a widely used proxy server as it permits you to save Internet Bandwidth and is feature rich. It runs on most available operating systems, including Windows and is licensed under the GNU GPL.

In this section, we will install and configure a basic squid proxy server on both the Fedora 18 and Ubuntu 12.04.2 operating systems. You need to ensure that an active Internet connection is available on the system you are performing the installation steps mentioned below, as the software repositories being used are located on the Internet.

5.4.1 INSTALLATION

On Computer A (Fedora 18)

```
[root@computer-a ~]# yum install squid
Loaded plugins: langpacks, presto, refresh-packagekit
Resolving Dependencies
--> Running transaction check
---> Package squid.x86_64 7:3.2.13-1.fc18 will be installed
--> Finished Dependency Resolution
Dependencies Resolved
================================================================================
 Package         Arch     Version                   Repository       Size
================================================================================
Installing:
 squid           x86_64 7:3.2.13-1.fc18             updates          2.5 M
Transaction Summary
================================================================================
Install  1 Package
```

```
Total download size: 2.5 M
Installed size: 8.5 M
Is this ok [y/N]: y
Downloading Packages:
squid-3.2.13-1.fc18.x86_64.rpm                    | 2.5 MB  00:00:46
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : 7:squid-3.2.13-1.fc18.x86_64                 1/1
  Verifying  : 7:squid-3.2.13-1.fc18.x86_64                 1/1

Installed:
  squid.x86_64  7:3.2.13-1.fc18
Complete!
[root@computer-a ~]#
```

After the successful installation we need to perform steps to enable the squid service and start the squid service. We will perform these steps as examples below.

Let us first check the status of the squid service on *computer-a*.

```
[root@computer-a ~]# systemctl status squid.service
squid.service - Squid caching proxy
   Loaded: loaded (/usr/lib/systemd/system/squid.service; disabled)
   Active: inactive (dead)

Sep 19 12:31:56 computer-a systemd[1]: Stopped Squid caching proxy.
[root@computer-a ~]#
```

You can see that the squid service is disabled and inactive. Therefore, we need to enable the squid service prior to starting it.

```
[root@computer-a ~]# systemctl enable squid.service
ln -s '/usr/lib/systemd/system/squid.service' '/etc/systemd/system/multi-user.target.wants/squid.service'
[root@computer-a ~]#
```

Now, when we check the status of the squid service we see that its status has changed to enabled. However, its still inactive and we need to start it.

```
[root@computer-a ~]# systemctl status squid.service
squid.service - Squid caching proxy
   Loaded: loaded (/usr/lib/systemd/system/squid.service; enabled)
   Active: inactive (dead)
Sep 19 12:31:56 computer-a systemd[1]: Stopped Squid caching proxy.
[root@computer-a ~]#
```

Let us start the squid service.

```
[root@computer-a ~]# systemctl start squid.service
[root@computer-a ~]#
```

Now, when we check the status of the squid service we can see that it is enabled and active.

```
[root@computer-a ~]# systemctl status squid.service
squid.service - Squid caching proxy
   Loaded: loaded (/usr/lib/systemd/system/squid.service; enabled)
   Active: active (running) since Thu 2013-09-19 14:47:07 IST; 3s ago
  Process: 5568 ExecStart=/usr/sbin/squid $SQUID_OPTS -f $SQUID_CONF (code=exited, status=0/SUCCESS)
  Process: 5562 ExecStartPre=/usr/libexec/squid/cache_swap.sh (code=exited, status=0/SUCCESS)
 Main PID: 5571 (squid)
   CGroup: name=systemd:/system/squid.service
           ├─5571 /usr/sbin/squid -f /etc/squid/squid.conf
           ├─5573 (squid-1) -f /etc/squid/squid.conf
           └─5574 (logfile-daemon) /var/log/squid/access.log

Sep 19 14:47:07 computer-a systemd[1]: Starting Squid caching proxy...
Sep 19 14:47:07 computer-a squid[5571]: Squid Parent: will start 1 kids
Sep 19 14:47:07 computer-a squid[5571]: Squid Parent: (squid-1) process 5573 started
Sep 19 14:47:07 computer-a systemd[1]: Started Squid caching proxy.
[root@computer-a ~]#
```

We can also check the system log for any issues if you want to be double sure :-).

```
[root@computer-a ~]# tail -f /var/log/messages
Sep 19 14:47:07 computer-a systemd[1]: Starting Squid caching proxy...
Sep 19 14:47:07 computer-a squid[5571]: Squid Parent: will start 1 kids
Sep 19 14:47:07 computer-a squid[5571]: Squid Parent: (squid-1) process 5573 started
```

```
Sep 19 14:47:07 computer-a systemd[1]: Started Squid caching proxy.
```

On Computer B (Ubuntu 12.04.2 LTS)

```
root@computer-b:~# apt-get install squid
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  squid
0 upgraded, 1 newly installed, 0 to remove and 230 not upgraded.
Need to get 6,254 B of archives.
After this operation, 128 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu/ precise-updates/universe squid amd64 3.1.19-1ubuntu3.12.04.2 [6,254 B]
Fetched 6,254 B in 1s (5,543 B/s)
Selecting previously unselected package squid.
(Reading database ... 201082 files and directories currently installed.)
Unpacking squid (from .../squid_3.1.19-1ubuntu3.12.04.2_amd64.deb) ...
Setting up squid (3.1.19-1ubuntu3.12.04.2) ...
root@computer-b:~#
```

After the successful installation of the squid proxy we need to check if its running.

```
root@computer-b:~# service squid3 status
squid3 start/running, process 1261
root@computer-b:~#
```

You can see that on Ubuntu 12.04.2 LTS the installation process took care of installing, enabling and starting to squid proxy for us.

We now have Squid Version 3.2.13 installed on Fedora 18 and Squid Version 3.1.19 installed on Ubuntu 12.04.2.

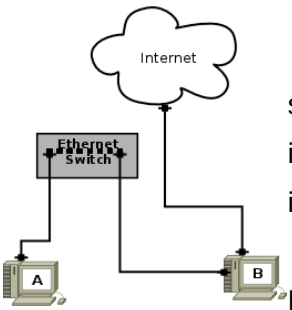
5.4.2 CONFIGURATION

For testing our Squid proxy server we do require a working Internet connection on the computer where our Squid proxy is installed. Now, there could be several

situations but for our example, since all we need is a simple proxy server, we will consider only two situations 1) Squid proxy server is directly connected to the Internet by whatever means and 2) Squid proxy server connects to the Internet via another proxy server.

The Squid configuration is located in the file "/etc/squid/squid.conf" on Fedora 18 and in the file "/etc/squid3/squid.conf" on Ubuntu 12.04.2 LTS.

Case 1: When the Squid proxy server is directly connected to the Internet.



It should pretty much work out of the box after a successful squid installation if the system running the Squid proxy is directly connected to the Internet. Nevertheless, we will look into the squid.conf file and see some of the configuration options.

Let us open the "squid.conf" file using the a text editor and look for the line with the entry "**acl localnet src 192.168.0.0/16**" and uncomment this line. The "#" character is used to comment lines within the "squid.conf" file. Therefore, by just deleting the "#" character in the beginning of a line you can uncomment a line. Fedora 18 has this line by default uncommented while on Ubuntu 12.04.2 LTS you need to uncomment this line manually. This line is an access control list or **acl** defining **localnet** as the source or **src** with IP Addresses in the range **192.168.0.0/16**. This will allow computers on the 192.168.0.0/16 network (our example LAN) to be able to use this proxy server.

Next, look for the line with the entry "**http_access allow localnet**" and uncomment it if commented. This line basically defines who should be allowed to use the proxy service. In our case this configuration would allow **http_access** to hosts on the **localnet**.

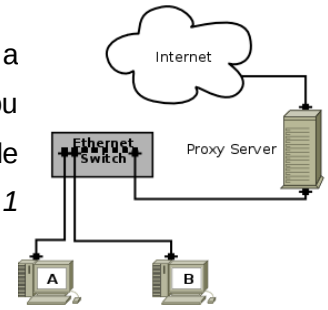
The line with the entry "**http_port 3128**" defines the port on which the http requests will be handled. You can change this port however we will go with the defaults in our example.

Restart the squid service using the command "**systemctl restart squid.service**" on Fedora 18 or "**service squid3 status**" on Ubuntu 12.04.2 LTS to apply all the changes you made to the "squid.conf" file.

Thats it, we should now be able to use our Squid proxy server with our direct Internet connection.

Case 2: When the Squid proxy server connects to the Internet via another proxy server.

In cases when your Squid proxy server does not have a direct Internet connection but uses another proxy server you really have to just add a couple of lines to the "squid.conf" file as shown below, in addition to the ones mentioned in *Case 1* above.



[If the proxy allows anonymous access]
cache_peer *parent-proxy-ip* **parent** *parent-proxy-port* **0 no-query default**
never_direct allow all

For example:
cache_peer *192.168.1.1* **parent** *3128* **0 no-query default**
never_direct allow all

[If the proxy requires a username and a password for access]
cache_peer *parent-proxy-ip* **parent** *parent-proxy-port* **0 no-query login=***username***:***password*
never_direct allow all

For example:
cache_peer *192.168.1.1* **parent** *3128* **0 no-query login=***binod.deka***:***abcd1234*
never_direct allow all

We should now be able to use our Squid proxy server via another proxy. That is via another Squid proxy server in our example. Of course we need to restart the squid service using the command "**systemctl restart squid.service**" on Fedora 18 or

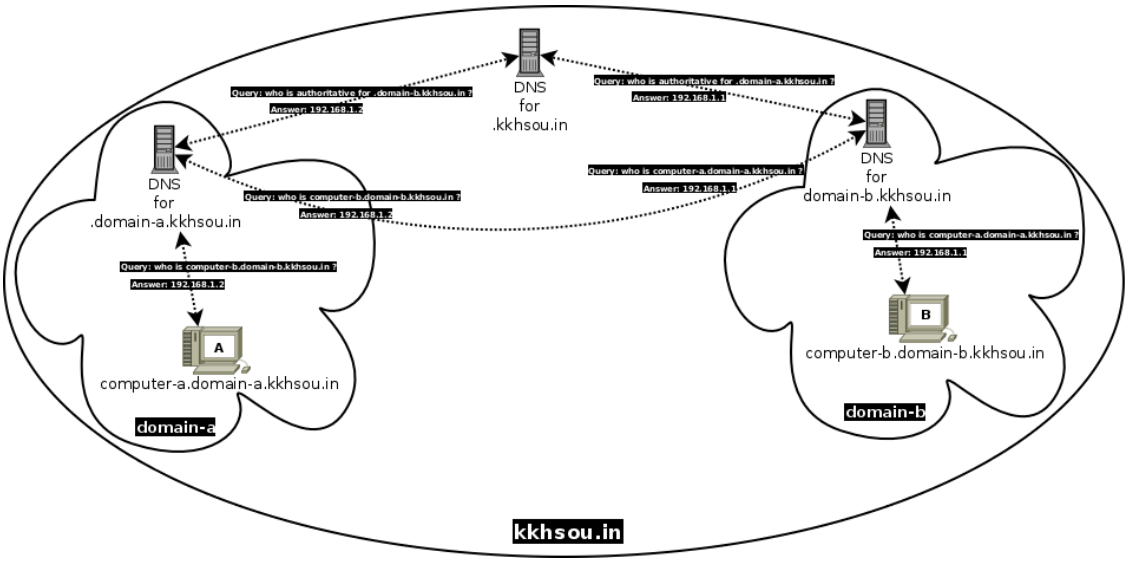
"**service squid3 status**" on Ubuntu 12.04.2 LTS to apply all the changes you made to the "squid.conf" file.

To use this proxy server we just configured, specify the proxy server as either "**192.168.1.1**" with port "**3128**" or "**192.168.1.2**" with port "**3128**" as we have installed on both *computer-a* as well as on *computer-b*. Another thing to note if you have a firewall running on your system is to allow the port "3128" or else other computers on the network would not be able to connect to the Squid Proxy on that port.

Please note that the Squid proxy server is a very powerful proxy server and there are a lot more configuration options that we have not discussed as its out of the scope of this unit. However, our goal was to configure a simple proxy server using Squid and we have done so.

5.5 INSTALLATION AND CONFIGURATION OF A DNS SERVER – BIND

Domain Name System or DNS is a service which provides resolution of fully qualified domain names (FQDN) into IP addresses and vice-versa. What this means is that domain names like say "www.kkhsou.in" is easier to remember than some IP address like "182.50.130.66". Moreover, IP addresses may change. Just think how difficult it would be to keep remembering numerical addresses. BIND (Berkeley Internet Name Domain) is the most widely used DNS software on the Internet and we will install and configure it in this section. DNS servers are also known as a nameservers as they provide a network service that associates hostnames with their respective IP addresses. DNS is usually implemented using one or more centralized servers that are authoritative for certain domains. When a client host requests information from a nameserver, it usually connects to port 53. The nameserver then attempts to resolve the name requested. If it does not have an authoritative answer, or does not already have the answer cached from an earlier query, it queries other nameservers, called root nameservers, to determine which nameservers are authoritative for the name in question, and then queries them to get the requested name.



We will setup the DNS servers more-or-less as depicted in the figure above. However, we will not be actually doing this over the Internet and neither will we be making any modifications on the DNS server for “kkhsou.in”. The figure above is only for an understanding of how DNS might be setup and work on the Internet. Since, in our current setup we only have two computers, *computer-a* and *computer-b* will both act as the DNS Server and the DNS Client for their respective domains.

5.5.1 INSTALLATION

On Computer A (Fedora 18)

```
[root@computer-a ~]# yum install bind
Loaded plugins: langpacks, presto, refresh-packagekit
Resolving Dependencies
--> Running transaction check
---> Package bind.x86_64 32:9.9.3-4.P2.fc18 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package         Arch     Version                    Repository        Size
================================================================================
```

```
Installing:
 bind               x86_64 32:9.9.3-4.P2.fc18         updates            2.1 M

Transaction Summary
================================================================================
Install  1 Package

Total download size: 2.1 M
Installed size: 6.2 M
Is this ok [y/N]: y
Downloading Packages:
bind-9.9.3-4.P2.fc18.x86_64.rpm                                | 2.1 MB  00:00:04
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : 32:bind-9.9.3-4.P2.fc18.x86_64        1/1
  Verifying  : 32:bind-9.9.3-4.P2.fc18.x86_64        1/1

Installed:
  bind.x86_64    32:9.9.3-4.P2.fc18
Complete!
[root@beefy-miracle ~]#
```

BIND is installed as the **named** service. Check the status of the named service.

```
[root@computer-a ~]# systemctl status named
named.service - Berkeley Internet Name Domain (DNS)
   Loaded: loaded (/usr/lib/systemd/system/named.service; disabled)
   Active: inactive (dead)
[root@computer-a ~]#
```

Enable the named service.

```
[root@computer-a ~]# systemctl enable named
ln -s '/usr/lib/systemd/system/named.service' '/etc/systemd/system/multi-user.target.wants/named.service'
[root@computer-a ~]#
```

Start the named service.

```
[root@computer-a ~]# systemctl start named
```

Stop the named service.

```
[root@computer-a ~]# systemctl stop named
```

On Computer B (Ubuntu 12.04.2 LTS)

```
root@computer-b:~# apt-get install bind9
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bind9utils
Suggested packages:
  bind9-doc
The following NEW packages will be installed:
  bind9 bind9utils
0 upgraded, 2 newly installed, 0 to remove and 6 not upgraded.
Need to get 455 kB of archives.
After this operation, 1,269 kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
WARNING: The following packages cannot be authenticated!
  bind9utils bind9
Install these packages without verification [y/N]? y
Get:1 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main bind9utils amd64
1:9.8.1.dfsg.P1-4ubuntu0.7 [108 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main bind9 amd64 1:9.8.1.dfsg.P1-
4ubuntu0.7 [347 kB]
Fetched 455 kB in 7s (57.6 kB/s)
Preconfiguring packages ...
Selecting previously unselected package bind9utils.
(Reading database ... 50965 files and directories currently installed.)
Unpacking bind9utils (from .../bind9utils_1%3a9.8.1.dfsg.P1-4ubuntu0.7_amd64.deb) ...
Selecting previously unselected package bind9.
Unpacking bind9 (from .../bind9_1%3a9.8.1.dfsg.P1-4ubuntu0.7_amd64.deb) ...
Processing triggers for man-db ...
Processing triggers for ufw ...
Processing triggers for ureadahead ...
```

```
Setting up bind9utils (1:9.8.1.dfsg.P1-4ubuntu0.7) ...
Setting up bind9 (1:9.8.1.dfsg.P1-4ubuntu0.7) ...
Adding group `bind' (GID 116) ...
Done.
Adding system user `bind' (UID 108) ...
Adding new user `bind' (UID 108) with group `bind' ...
Not creating home directory `/var/cache/bind'.
wrote key file "/etc/bind/rndc.key"
#
 * Starting domain name service... bind9                    [ OK ]
root@computer-b:~#
```

BIND is installed as **bind9**. Start the bind9 service.

```
[root@computer-b ~]# service bind9 start
```

Stop the bind9 service.

```
[root@computer-b ~]# service bind9 stop
```

Additionally, you should also install "bind-utils" on Fedora 18 and "dnsutils" on Ubuntu 12.04.2 LTS which includes the client tools like nslookup, dig and host, etc. We will use these tools to test our DNS server configurations.

5.5.2 CONFIGURATION

In a DNS server such as BIND, all information is stored in basic data elements called *resource records* (RR). The resource record is usually a fully qualified domain name (FQDN) of a host. The following are examples of resource records.

computer-a.domain-a.kkhsou.in

computer-b.domain-b.kkhsou.in

Each level of the hierarchy is divided by a period (that is, .). In the examples above, "**in**" defines the top-level domain, "**kkhsou**" its subdomain, and "**domain-a**" & "**domain-b**" the subdomain of "kkhsou". In our example, "**computer-a**" & "**computer-b**" identifies a *resource record* that is part of the "domain-a.kkhsou.in" & "domain-b.kkhsou.in" domains respectively. With the exception of the part furthest to the left (that is, computer-a and computer-b), each of these sections is called a **zone** and defines a specific *namespace*.

Zones are defined on authoritative nameservers through the use of "*zone files*", which contain definitions of the resource records in each zone. Zone files are stored on primary nameservers (also called master nameservers), where changes are made to the files, and secondary nameservers (also called slave nameservers), which receive zone definitions from the primary nameservers. Both primary and secondary nameservers are authoritative for the zone and look the same to clients. Depending on the configuration, any nameserver can also serve as a primary or secondary server for multiple zones at the same time. In our examples, we will be setting up a single primary nameserver (authoritative) for each individual domain.

The main configuration file for BIND is located in "/etc/named.conf" for Fedora 18 and “/etc/bind/named.conf” for Ubuntu 12.04.2 LTS. In this section, we will configure simple DNS servers for the example domain “domain-a.kkhsou.in” on computer-a and example domain “domain-b.kkhsou.in” on computer-b.

On computer-a (Fedora 18):

Our first step would be to add our example domain by adding a zone entry in the **named.conf** file.

```
zone "domain-a.kkhsou.in" IN {
  type master;
  file "domain-a.kkhsou.in.zone";
};
```

Specify the IPv4 network interface on which to listen for / allow queries under the options statement.

```
options {
```

```
...
listen-on port 53 { 192.168.1.1; };
allow-query { 192.168.1.0/24; };
...
};
```

The "/var/named/" by default will contain all the zone files stated in the "named.conf" file. In our example, the zone file will be "domain-a.kkhsou.in.zone" and will contain the zone data.

A zone file consists of directives and resource records. Directives tell the nameserver to perform tasks or apply special settings to the zone, resource records define the parameters of the zone and assign identities to individual hosts. While the directives are optional, the resource records are required in order to provide name service to a zone. All directives and resource records should be entered on individual lines. Directives begin with the dollar sign character (that is, $) followed by the name of the directive, and usually appear at the top of the file.

Our example zone file for the "domain-a.kkhsou.in" domain will be say "/var/named/**domain-a.kkhsou.in.zone**".

In a zone file comments are identified by ";" so anything that comes after ";" will be ignored by named.
Let us go ahead and create the zone file "/var/named/domain-a.kkhsou.in.zone" as root and make the following entries.

```
$ORIGIN domain-a.kkhsou.in.
$TTL 86400 ; how long the zone record is valid in seconds. Each resource record can contain its own TTL value, which overrides this directive.
@       IN  SOA  computer-a.domain-a.kkhsou.in.  root.domain-a.kkhsou.in. (
            20130925        ; serial. You must increment this serial number each time you
                            ; make changes to the zone file before restarting the
                            ; named service.
            21600           ; refresh after 6 hours
            3600            ; retry after 1 hour
```

```
            604800          ; expire after 1 week
            86400 )         ; minimum TTL of 1 day
;
;
          IN  NS    computer-a.domain-a.kkhsou.in.
computer-a      IN  A     192.168.1.1
;
;
```

After successfully creating the zone file for domain-a.kkhsou.in domain, we start the "named" service by typing "**systemctl start named.service**" as root. Next, we will need to configure *computer-a* to use this DNS server. To do this we add the following line to the "**/etc/resolve.conf**" file.

```
nameserver 192.168.1.1
```

To check if our DNS setup is working, we can use the **nslookup** utility to query our DNS server.

```
[root@computer-a ~]# nslookup computer-a.domain-a.kkhsou.in
Server:         192.168.1.1
Address:        192.168.1.1#53

Name:  computer-a.domain-a.kkhsou.in
Address: 192.168.1.1

[root@computer-a ~]#
```

We can also use the **dig** command to query our DNS server as well.

```
[root@computer-a ~]# dig computer-a.domain-a.kkhsou.in

; <<>> DiG 9.9.3-rl.13207.22-P2-RedHat-9.9.3-4.P2.fc18 <<>> computer-a.domain-a.kkhsou.in
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10223
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;computer-a.domain-a.kkhsou.in.          IN      A

;; ANSWER SECTION:
computer-a.domain-a.kkhsou.in. 86400 IN      A       192.168.1.1

;; AUTHORITY SECTION:
domain-a.kkhsou.in.     86400   IN      NS      computer-a.domain-a.kkhsou.in.

;; Query time: 0 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Sep 25 11:31:47 IST 2013
;; MSG SIZE  rcvd: 117

[root@computer-a ~]#
```

Let us also add the following entries into our "/var/named/domain-a.kkhsou.in.zone" file. We will be using these entries in our later sections.

```
@       IN  MX    10  mail.domain-a.kkhsou.in.
mail    IN  A     192.168.1.1
;
;
ftp     IN CNAME  computer-a.domain-a.kkhsou.in.
www     IN CNAME  computer-a.domain-a.kkhsou.in.
;
;
```

Though, we find that we have a working DNS server setup, we are not done yet. We have to also setup a **Reverse Name Resolution Zone File** that will be used to resolve IP Addresses to Fully Qualified Domain Names (FQDN). It looks very similar to a standard zone file, except that the *PTR resource records* are used to link the IP addresses to a fully qualified domain name.

Firstly, we need to add the following zone statement to our “/etc/named.conf” file.

```
zone "1.168.192.in-addr.arpa" IN {
  type master;
  file "domain-a.kkhsou.in.rr.zone";
};
```

Note that a reverse name resolution zone requires the first three blocks of the IP address reversed "1.168.192" followed by ".in-addr.arpa". This allows the single block of IP numbers used in the reverse name resolution zone file to be associated with the zone.

Next, we create the reverse resolution zone file "/var/named/**domain-a.kkhsou.in.rr.zone**" and make the following entries.

```
$ORIGIN 1.168.192.in-addr.arpa.
$TTL 86400
@  IN  SOA  computer-a.domain-a.kkhsou.in.  root.domain-a.kkhsou.in. (
      2001062501          ; serial
      21600        ; refresh after 6 hours
      3600                ; retry after 1 hour
      604800       ; expire after 1 week
      86400 )      ; minimum TTL of 1 day
;
@  IN  NS   computer-a.domain-a.kkhsou.in.
;
1  IN  PTR  computer-a.domain-a.kkhsou.in.
;
```

After performing the required changed restart the named service.

```
[root@computer-a ~]# systemctl restart named.service
```

We can check to see if the reverse resolution is working using **nslookup** or **dig**.

```
[root@computer-a ~]# nslookup 192.168.1.1
Server:         192.168.1.1
Address:        192.168.1.1#53

1.1.168.192.in-addr.arpa        name = computer-a.domain-a.kkhsou.in.

[root@computer-a ~]#
```

```
[root@computer-a ~]# dig -x 192.168.1.1

; <<>> DiG 9.9.3-rl.13207.22-P2-RedHat-9.9.3-4.P2.fc18 <<>> -x 192.168.1.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48303
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;1.1.168.192.in-addr.arpa.          IN      PTR

;; ANSWER SECTION:
1.1.168.192.in-addr.arpa. 86400 IN      PTR     computer-a.domain-a.kkhsou.in.

;; AUTHORITY SECTION:
1.168.192.in-addr.arpa. 86400   IN      NS      computer-a.domain-a.kkhsou.in.

;; ADDITIONAL SECTION:
computer-a.domain-a.kkhsou.in. 86400 IN         A       192.168.1.1

;; Query time: 0 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Wed Sep 25 12:53:50 IST 2013
;; MSG SIZE  rcvd: 126

[root@computer-a ~]#
```

On computer-b (Ubuntu 12.04.2 LTS):

Similarly, on Ubuntu 12.04.2 LTS we configure the DNS server. Once thing to keep in mind though, is the location of the files. They differ in the location from that of Fedora 18.

The DNS server options need to be put in the file "**/etc/bind/named.conf.options**" .

```
options {
```

```
...
listen-on { 192.168.1.2; };
allow-query { 192.168.1.0/24; };
...
};
```

The zone statements have to be entered in "**/etc/bind/named.conf.default-zones**" .

```
zone "domain-b.kkhsou.in" IN {
  type master;
  file "/etc/bind/db.domain-b.kkhsou.in";
};

zone "1.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.1.168.192";
};
```

Enter the resource records to the file "**/etc/bind/db.domain-b.kkhsou.in**" .

```
$ORIGIN domain-b.kkhsou.in.
$TTL 86400 ; how long the zone record is valid in seconds. Each resource record can contain its own TTL value, which overrides this directive.
@       IN  SOA  computer-b.domain-a.kkhsou.in.  root.domain-b.kkhsou.in. (
             20130925        ; serial. You must increment this serial number each time you
                             ; make changes to the zone file before restarting the
                             ; named service.
             21600           ; refresh after 6 hours
             3600            ; retry after 1 hour
             604800          ; expire after 1 week
             86400 )         ; minimum TTL of 1 day
;
;
          IN  NS     computer-b.domain-b.kkhsou.in.
computer-b     IN  A      192.168.1.2
;
;
@       IN  MX     10  mail.domain-b.kkhsou.in.
```

```
mail     IN  A     192.168.1.2
;
;
ftp      IN CNAME  computer-b.domain-b.kkhsou.in.
www      IN CNAME  computer-b.domain-b.kkhsou.in.
;
;
```

Add the reverse resource records to the file "**/etc/bind/db.1.168.192**" .

```
$ORIGIN 1.168.192.in-addr.arpa.
$TTL 86400
@  IN  SOA  computer-b.domain-b.kkhsou.in.  root.domain-b.kkhsou.in. (
      2001062501          ; serial
      21600               ; refresh after 6 hours
      3600                ; retry after 1 hour
      604800              ; expire after 1 week
      86400 )             ; minimum TTL of 1 day
;
@  IN  NS   computer-b.domain-b.kkhsou.in.
;
1  IN  PTR  computer-b.domain-b.kkhsou.in.
;
```

After performing the required changed restart the bind9 service.

```
[root@computer-b ~]# service bind9 restart
```

We should now have fully functional DNS servers for both the “domain-a.kkhsou.in” and “domain-b.kkhsou.in” domains. Do note here that we have configured a simple DNS server for our example purposes only. There are other advanced configurations that we did not touch upon in this section due to the scope of this unit. You are therefore encouraged to read about these advanced options before attempting to configure any DNS server either for the Internet or for your live networks.

5.6 INSTALLATION AND CONFIGURATION OF A WEB SERVER – APACHE

The Apache HTTP Server, is a robust, full-featured open source web server developed by the Apache Software Foundation and is one of the most widely used web server software that is currently used on the Internet. In this section we will install and configure a simple Web server on both the Fedora 18 and Ubuntu 12.04.2 LTS operating systems using Apache 2 in our examples.

5.6.1 INSTALLATION

On computer-a (Fedora 18):

To install the Apache Web server (httpd), as root type in the command.

```
yum install httpd
```

To enable to httpd service type the command as root.

```
systemctl enable httpd.service
```

To disable to httpd service type the command as root.

```
systemctl disable httpd.service
```

To start the httpd service type the command as root.

```
systemctl start httpd.service
```

To stop the httpd service type the command as root.

```
systemctl stop httpd.service
```

To restart the httpd service type the command as root.

```
systemctl restart httpd.service
```

On computer-b (Ubuntu 12.04.2 LTS):

To install the Apache Web server (apache2), as root type the command.

```
apt-get install apache2
```

To start the apache2 service, as root type the command.

```
service apache2 start
```

To stop the apache2 service, as root type the command.

```
service apache2 stop
```

To restart the apache2 service, as root type the command.

```
service apache2 restart
```

5.6.2 CONFIGURATION

On computer-a (Fedora 18):

The main configuration file is located at "**/etc/httpd/conf/httpd.conf**".

The line below sets the default port where httpd would listen for connections.

```
Listen 80
```

The line below sets the admin email.

```
ServerAdmin webmaster@domain-a.kkhsou.in
```

The line below sets the web server name.

```
ServerName www.domain-a.kkhsou.in:80
```

The line below sets the location of the document root of the web server.

```
DocumentRoot "/var/www/html"
```

The configuration section below sets the file that Apache will serve if a directory is requested.

```
<IfModule dir_module>
    DirectoryIndex index.php index.html
</IfModule>
```

To check the configuration for possible errors, type the following as root.

```
service httpd configtest
```

Once you are satisfied with your configuration you need to either restart the httpd service or alternatively reload the configuration for the httpd service to use.

To restart the httpd service completely, type the following as root.

```
systemctl restart httpd.service
```

To only reload the configuration, type the following as root.

```
systemctl reload httpd.service
```

If there are no errors and the httpd service started properly you should be able to open your web browser to the default webpage page as shown in the screenshot below for **localhost**.

You should now also be able to open similarly the default webpage when using www.domain-a.kkhsou.in instead of localhost from your web browser.

On computer-b (Ubuntu 12.04.2 LTS):

Ubuntu 12.04.2 LTS uses a slightly different approach for configuring the apache2 service.

The main configuration file is located at "**/etc/apache2/apache2.conf**". However, we do not require to edit this file, in our example.

The line below in the file "**/etc/apache2/ports.conf**" sets the default port where apache2 would listen for connections.

```
Listen 80
```

To setup our example site for "www.domain-b.kkhsou.in" we will need to edit the file located in "**/etc/apache2/sites-enabled/000-default** " and add the following entry to set the admin email. And we will leave rest of the configurations intact for our example purposes.

```
<VirtualHost *:80>
        ServerAdmin webmaster@domain-b.kkhsou.in
</VirtualHost>
```

The configuration section in the file "**/etc/apache2/mods-enabled/dir.conf**" shown below, sets the file that Apache will serve if a directory is requested.

```
<IfModule mod_dir.c>
        DirectoryIndex index.html index.cgi index.pl index.php index.xhtml index.htm
</IfModule>
```

Once you are satisfied with your configuration you need to either restart the apache2 service before being able to use with the new configuration.

To restart the apache2 service, type the following as root.

```
service apache2 restart
```

To stop the apache2 service, type the following as root.

```
service apache2 stop
```

To start the apache2 service, type the following as root.

```
service apache2 start
```

If there are no errors and the apache2 service started properly you should be able to open your web browser to the default webpage page as shown in the screenshot below for **localhost**. You should also be able to open similarly the default webpage when using www.domain-b.kkhsou.in instead of localhost from your web browser.

Though we have a simple working Apache Web server, which was our goal in this section. There are an exhaustive number of options available with the Apache Web server that can be used with any custom configuration requirements. You should read all about these configuration options first and ensure that you understand them, before attempting to configure a web server on the Internet or on any live network.

5.7 INSTALLATION AND CONFIGURATION OF A FILE SERVER – SAMBA

Samba is an open source implementation of the Server Message Block (SMB) protocol. It allows the networking of Microsoft Windows, Linux, UNIX, and other operating systems together, enabling access to Windows-based file and printer shares. Samba's use of SMB allows it to appear as a Windows server to Windows clients.

Samba is comprised of three daemons (smbd, nmbd, and winbindd). The **smbd** server daemon provides file sharing and printing services to Windows clients and the default ports on which the server listens for SMB traffic are **TCP ports 139 and 445**. The **nmbd**

server daemon understands and replies to NetBIOS name service requests and the default port that the server listens to for NMB traffic is **UDP port 137**. The winbind service resolves user and group information on a server running Windows NT, 2000, 2003 or Windows Server 2008 and makes Windows user / group information understandable by UNIX platforms. However, winbind is beyond the scope of this section and therefore we will not be discussing it in this section.

In this section we will install and configure a simple file server using **samba** on both the Fedora 18 and Ubuntu 12.04.2 LTS operating systems in our examples.

5.7.1 INSTALLATION

On computer-a (Fedora 18):

To install samba server, type the following as root.

```
[root@computer-a ~]# yum install samba
Loaded plugins: langpacks, presto, refresh-packagekit
Resolving Dependencies
--> Running transaction check
---> Package samba.x86_64 2:4.0.9-1.fc18 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package         Arch     Version             Repository       Size
================================================================================
Installing:
 samba           x86_64 2:4.0.9-1.fc18    updates527 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 527 k
Installed size: 1.6 M
```

```
Is this ok [y/N]: y
Downloading Packages:
samba-4.0.9-1.fc18.x86_64.rpm                    | 527 kB  00:00:04
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : 2:samba-4.0.9-1.fc18.x86_64          1/1
  Verifying  : 2:samba-4.0.9-1.fc18.x86_64          1/1

Installed:
  samba.x86_642:4.0.9-1.fc18
Complete!
[root@computer-a ~]#
```

To install the samba client, type the following as root.

```
[root@computer-a ~]# yum install samba-client
Loaded plugins: langpacks, presto, refresh-packagekit
Resolving Dependencies
--> Running transaction check
---> Package samba-client.x86_64 2:4.0.9-1.fc18 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package         Arch     Version            Repository       Size
================================================================================
Installing:
 samba-client    x86_64 2:4.0.9-1.fc18    updates          455 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 455 k
Installed size: 1.2 M
```

```
Is this ok [y/N]: y
Downloading Packages:
samba-client-4.0.9-1.fc18.x86_64.rpm                | 455 kB  00:00:03
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : 2:samba-client-4.0.9-1.fc18.x86_64  1/1
  Verifying  : 2:samba-client-4.0.9-1.fc18.x86_64  1/1

Installed:
  samba-client.x86_64 2:4.0.9-1.fc18
Complete!
[root@computer-a ~]#
```

To check the status of the samba services, type the following as root.

```
[root@computer-a ~]# systemctl status smb.service
smb.service - Samba SMB Daemon
   Loaded: loaded (/usr/lib/systemd/system/smb.service; disabled)
   Active: inactive (dead)
```

```
[root@computer-a ~]# systemctl status nmb.service
nmb.service - Samba NMB Daemon
   Loaded: loaded (/usr/lib/systemd/system/nmb.service; disabled)
   Active: inactive (dead)
```

To start the Samba server, type the following command as root.

```
systemctl start smb.service
```

To stop the Samba server, type the following command as root.

```
systemctl stop smb.service
```

To restart the Samba server, type the following command as root.

```
systemctl restart smb.service
```

On computer-b (Ubuntu 12.04.2 LTS):

To install samba, type the following as root.

```
root@computer-b:~# apt-get install samba
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  libwbclient0 samba-common smbclient tdb-tools
Suggested packages:
  openbsd-inetd inet-superserver smbldap-tools ldb-tools ctdb cifs-utils
The following NEW packages will be installed:
  samba tdb-tools
The following packages will be upgraded:
  libwbclient0 samba-common smbclient
3 upgraded, 2 newly installed, 0 to remove and 237 not upgraded.
Need to get 22.5 MB of archives.
After this operation, 23.5 MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get:1 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main libwbclient0 amd64 2:3.6.3-2ubuntu2.8 [29.9 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main smbclient amd64 2:3.6.3-2ubuntu2.8 [14.1 MB]
Get:3 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main samba-common all 2:3.6.3-2ubuntu2.8 [326 kB]
Get:4 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main samba amd64 2:3.6.3-2ubuntu2.8 [8,049 kB]
Get:5 http://in.archive.ubuntu.com/ubuntu/ precise/main tdb-tools amd64 1.2.9-4 [23.2 kB]
Fetched 22.5 MB in 2min 24s (156 kB/s)
Preconfiguring packages ...
(Reading database ... 201772 files and directories currently installed.)
Preparing to replace libwbclient0 2:3.6.3-2ubuntu2.6 (using .../libwbclient0_2%3a3.6.3-2ubuntu2.8_amd64.deb) ...
Unpacking replacement libwbclient0 ...
Preparing to replace smbclient 2:3.6.3-2ubuntu2.6 (using .../smbclient_2%3a3.6.3-2ubuntu2.8_amd64.deb) ...
Unpacking replacement smbclient ...
Preparing to replace samba-common 2:3.6.3-2ubuntu2.6 (using .../samba-common_2%3a3.6.3-2ubuntu2.8_all.deb) ...
Unpacking replacement samba-common ...
Selecting previously unselected package samba.
Unpacking samba (from .../samba_2%3a3.6.3-2ubuntu2.8_amd64.deb) ...
Selecting previously unselected package tdb-tools.
Unpacking tdb-tools (from .../tdb-tools_1.2.9-4_amd64.deb) ...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
Processing triggers for ufw ...
Rules updated for profile 'Bind9'
```

```
Setting up libwbclient0 (2:3.6.3-2ubuntu2.8) ...
Setting up samba-common (2:3.6.3-2ubuntu2.8) ...
Setting up smbclient (2:3.6.3-2ubuntu2.8) ...
Setting up samba (2:3.6.3-2ubuntu2.8) ...
Generating /etc/default/samba...
Importing account for binoddeka...ok
Importing account for tapashi.kashyap...ok
Importing account for choudhurysmriti...ok
update-alternatives: using /usr/bin/smbstatus.samba3 to provide /usr/bin/smbstatus (smbstatus) in auto mode.
smbd start/running, process 30095
nmbd start/running, process 30129
Setting up tdb-tools (1.2.9-4) ...
update-alternatives: using /usr/bin/tdbbackup.tdbtools to provide /usr/bin/tdbbackup (tdbbackup) in auto mode.
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
root@computer-b:~#
```

To check the status of the samba services, type the following as root.

```
[root@computer-b ~]# service smbd status
smbd start/running, process 30095
```

```
[root@computer-b ~]# service nmbd status
nmbd start/running, process 30129
```

To start the samba server, type the following as root.

```
service smbd start
```

To stop the samba server, type the following as root.

```
service smbd stop
```

To restart the samba server, type the following as root.

```
service smbd restart
```

5.7.2 CONFIGURATION

On computer-a (Fedora 18):

The default samba configuration file located in "**/etc/samba/smb.conf**" allows users to view their home directories as a Samba share. It also shares all printers configured for the system as Samba shared printers. In other words, you can attach a printer to the system and print to it from the Windows machines on your network.

To specify the Windows workgroup and a brief description of the Samba server, we edit the following lines in our /etc/samba/smb.conf file.

```
#======================= Global Settings ==============================
[global]
# ---------------------- Network-Related Options -------------------------
workgroup = DOMAIN-A
server string = Samba Server Version %v

# --------------------------- Logging Options -----------------------------
log file = /var/log/samba/log.%m
max log size = 50

# ----------------------- Standalone Server Options ------------------------
security = user
passdb backend = tdbsam

# ---------------------------- Printing Options -----------------------------
load printers = yes
cups options = raw
```

To create a Samba share directory we add the following section to our /etc/samba/smb.conf file. In our example, we will allow the users **binoddeka**, **tapashi.kashyap** and **choudhurysmriti** to be able to access the share "**allusers**".

```
#========================= Share Definitions ========================
[allusers]
  comment = All Users
```

```
path = /home/shares/allusers
valid users = binoddeka, tapashi.kashyap, choudhurysmriti
create mask = 0660
directory mask = 0771
writable = yes
```

We can use the "testparm" program which checks the syntax of the /etc/samba/smb.conf file. The testparm program also displays a summary of your /etc/samba/smb.conf file and the server's role (stand-alone, domain, etc.) after testing. This is convenient when debugging as it excludes comments and concisely presents information for experienced administrators to read.

```
[root@computer-a ~]# testparm
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Processing section "[homes]"
Processing section "[printers]"
Processing section "[allusers]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions
[global]
        workgroup = DOMAIN-A
        server string = Samba Server Version %v on computer-a
        log file = /var/log/samba/log.%m
        max log size = 50
        idmap config * : backend = tdb
        cups options = raw
[homes]
        comment = Home Directories
        valid users = MYDOMAIN\%S
        read only = No
        browseable = No
[printers]
        comment = All Printers
        path = /var/spool/samba
        printable = Yes
        print ok = Yes
```

```
        browseable = No
[allusers]
  comment = All Users
  path = /home/shares/allusers
  valid users = binoddeka, tapashi.kashyap, choudhurysmriti
  create mask = 0660
  directory mask = 0771
  writable = yes
[root@computer-a ~]#
```

Next we will use the "smbpasswd" program that manages encrypted passwords. This program can be run by a superuser to change any user's password as well as by an ordinary user to change their own Samba password.

```
[root@computer-a ~]# smbpasswd -a binoddeka
New SMB password:
Retype new SMB password:
Added user binoddeka.
[root@computer-a ~]#
```

Similarly, using the "smbpasswd" program we will need to add the other users as well.

To connect to our share we will use the “smbclient” program, as shown below.

```
[root@computer-a ~]# smbclient //computer-a/allusers -U binoddeka
Enter binoddeka's password:
Domain=[DOMAIN-A] OS=[Unix] Server=[Samba 4.0.9]
smb: \>
```

At the “smb: \>” prompt you can type “?” to view the available commands. To view the files in the “allusers” shared folder you can either type “dir” or “ls”.

```
smb: \> ls
  .                   D        0  Tue Oct  1 15:20:19 2013
  ..                  D        0  Thu Sep 26 17:17:41 2013
  b.txt               N     7480  Tue Oct  1 15:20:19 2013
  a.txt               N     7436  Tue Oct  1 15:20:13 2013
                50380 blocks of size 8388608. 3286 blocks available
smb: \>
```

To exit from the “smb: \>” prompt you can type “quit” and hit the enter key on your keyboard.

On computer-b (Ubuntu 12.04.2 LTS):

The main Samba configuration file is located in “**/etc/samba/smb.conf**”. All the configurations will be similar to the Samba configuration discussed for Fedora 18. However, we will make the modifications with respect to **domain-b** on **computer-b**. So, that our “testparm” program out put will be as shown below.

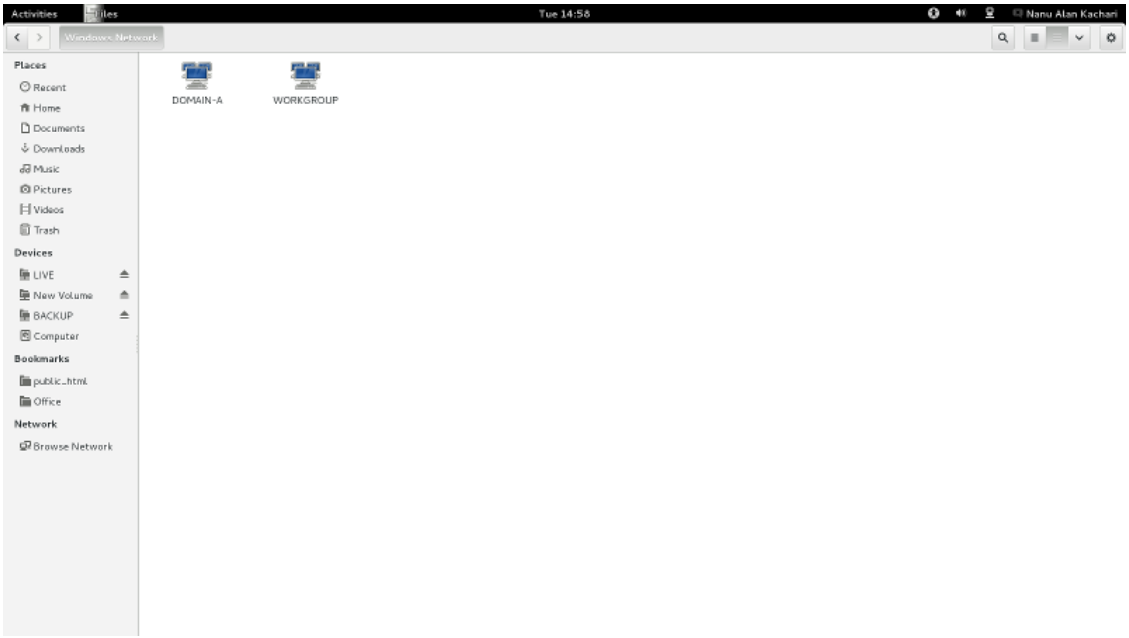
```
root@computer-b:/srv/samba/allusers# testparm
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Processing section "[printers]"
Processing section "[print$]"
Processing section "[share]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions
[global]
        workgroup = DOMAIN-B
        server string = %h server (Samba, Ubuntu)
        map to guest = Bad User
        obey pam restrictions = Yes
        pam password change = Yes
        passwd program = /usr/bin/passwd %u
        passwd chat = *Enter\snew\s*\spassword:* %n\n *Retype\snew\s*\spassword:* %n\n *password\supdated\ssuccessfully* .
        unix password sync = Yes
        syslog = 0
        log file = /var/log/samba/log.%m
        max log size = 1000
        dns proxy = No
        usershare allow guests = Yes
        panic action = /usr/share/samba/panic-action %d
        idmap config * : backend = tdb
[printers]
        comment = All Printers
```

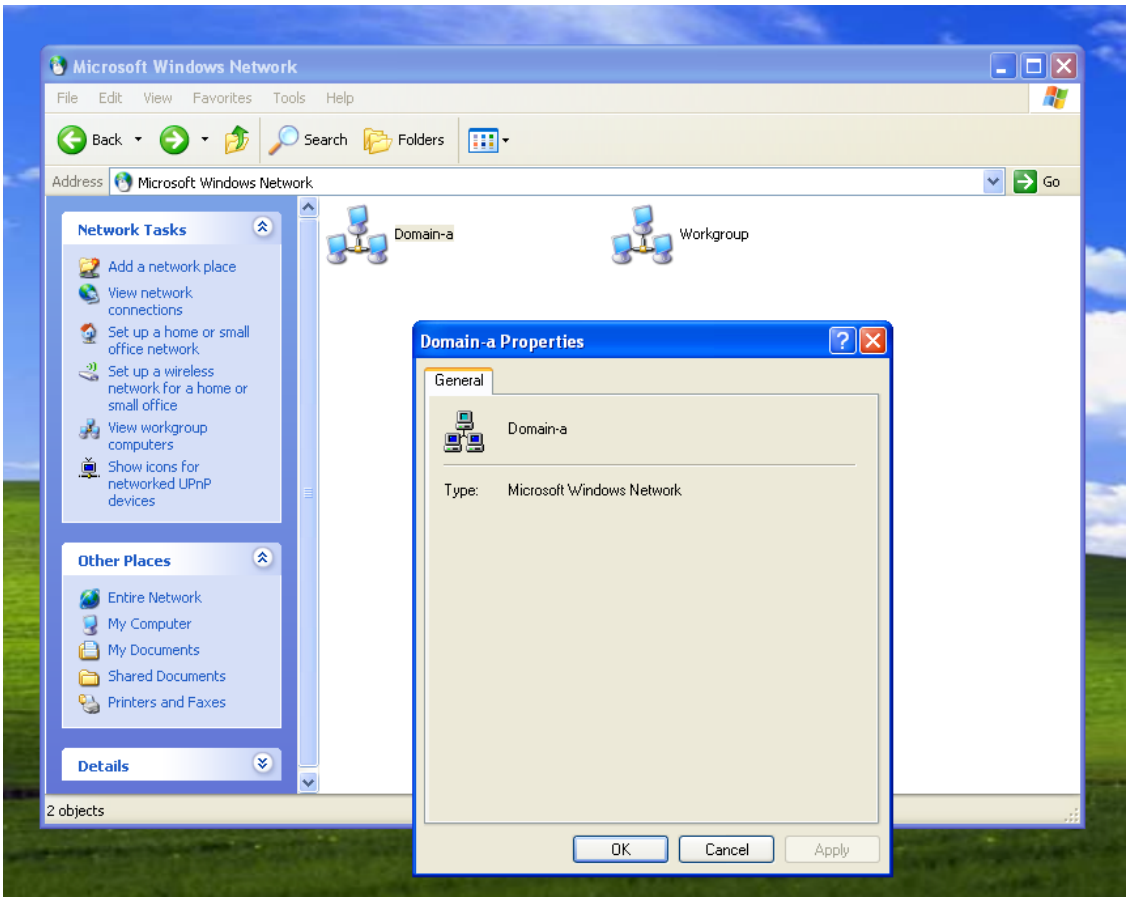
```
        path = /var/spool/samba
        create mask = 0700
        printable = Yes
        print ok = Yes
        browseable = No
[print$]
        comment = Printer Drivers
        path = /var/lib/samba/printers
[share]
        comment = Ubuntu File Server Share
        path = /srv/samba/allusers
        read only = No
        create mask = 0755
        guest ok = Yes
root@computer-b:/srv/samba/allusers#
```

Please do note that we require to restart the Samba Service on both Fedora 18 and Ubuntu 12.04.2 LTS system before we are able to use Samba. Therefore, ensure that after any configuration change to the “smb.conf” file you restart the samba services.

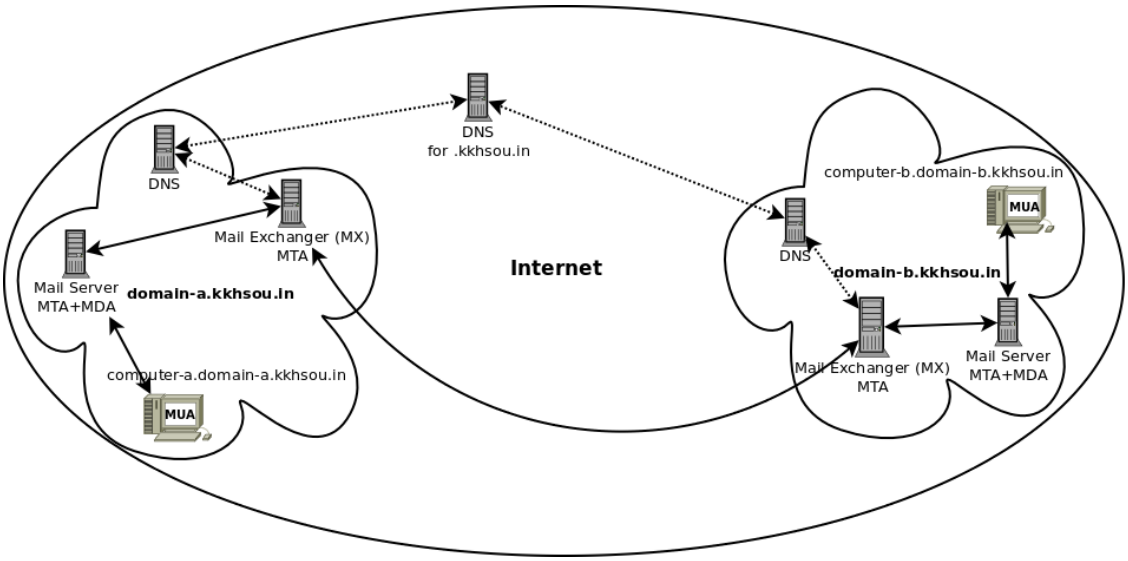
We now are done configuring a simple Samba server and you should be able to browse the shares created from both computers running Windows as well as from our Linux boxes.

Some screen-shots of our Samba configuration browse-able for Domain-A, from both Linux and Windows Clients.


Activities
Tue 14:58
Nanu Alan Kachari
Windows Network
Places
Recent
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Devices
LIVE
New Volume
BACKUP
Computer
Bookmarks
public_html
Office
Network
Browse Network
DOMAIN-A
WORKGROUP

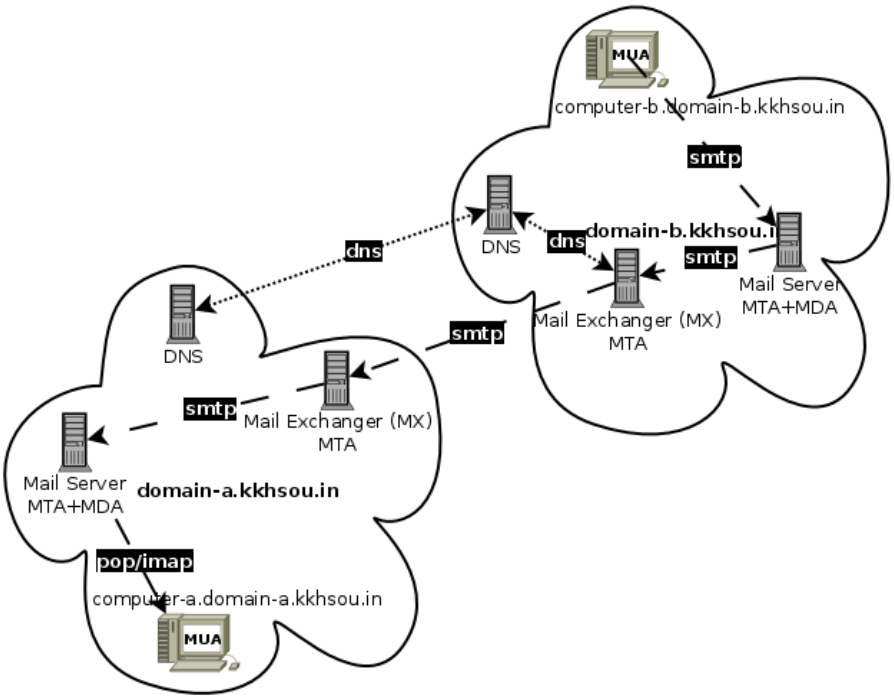

Microsoft Windows Network
File Edit View Favorites Tools Help
Back
Search
Folders
Address
Microsoft Windows Network
Go
Network Tasks
Add a network place
View network connections
Set up a home or small office network
Set up a wireless network for a home or small office
View workgroup computers
Show icons for networked UPnP devices
Other Places
Entire Network
My Computer
My Documents
Shared Documents
Printers and Faxes
Details
Domain-a
Workgroup
2 objects
Domain-a Properties
General
Domain-a
Type:
Microsoft Windows Network
OK
Cancel
Apply

5.8 INSTALLATION AND CONFIGURATION OF A MAIL SERVER – POSTFIX



The process of moving an email from one person to another over a network or the Internet involves many systems working together. Email is delivered using a client/server architecture. A typical mail system is depicted in the figure above, though we are using fictitious domain names here for our understanding.

A sender uses a Mail User Agent (**MUA**), meaning an email client like Evolution, Thunderbird, Mutt, etc., to send the message through one or more Mail Transfer Agents (**MTA**), meaning email servers like Exim, Postfix, Sendmail, etc., and ultimately will hand it off to a Mail Delivery Agent (**MDA**) like Procmail, mail, etc., for delivery to the recipient's mailbox. The recipient's email client will retrieve the email from the mailbox usually via a POP3 or IMAP server like Dovecot. The process of sending and receiving email between our example domains is depicted in the figure below.



Every email domain is required to have an MTA that would by default receive emails for it's domain. This MTA is known as the Mail Exchanger (MX) for the domain. Typically, a domain is required to also have a DNS server which is authoritative for it's domain. This DNS will maintain all the Domain Names for its domain including the MX Record which is the address of the MTA for the domain. Furthermore, the email mailboxes are generally stored on another MTA which could include an MDA as well. In our examples, we will configure all the components of a mail system on a single system.

We will install and configure postfix (as MTA), procmail (as MDA), mutt (as MUA) and dovecot (as pop server) on both Fedora 18 and Ubuntu 12.04.2 LTS operating systems. You will need to ensure that an active Internet connection is available on the system you are performing the installation steps mentioned below, as the software repositories being used for installation are located on the Internet.

5.8.1 INSTALLATION

MTA installation:

To install a Mail Transport Agent (MTA), we choose **postfix** in our examples, type the following as root.

On computer-a (Fedora 18):

```
yum install postfix
```

You can use the following to check the status, enable, start, stop or restart the postfix service.

systemctl status postfix.service
systemctl enable postfix.service
systemctl start postfix.service
systemctl stop postfix.service
systemctl restart postfix.service

On computer-b (Ubuntu 12.04.2 LTS):

```
apt-get install postfix
```

You can use the following to check the status, start, stop or restart the postfix service.

service postfix status
service postfix start
service postfix stop
service postfix restart

MDA installation:

To install the Mail Delivery Agent (MDA), we choose **procmail** in our examples, type the following as root.

On computer-a (Fedora 18):

```
yum install procmail
```

On computer-b (Ubuntu 12.04.2 LTS):

```
apt-get install procmail
```

MUA installation:

To install the Mail User Agent (MUA), we choose **mutt** (which is a text based email client) in our examples, type the following as root.

On computer-a (Fedora 18):

```
yum install mutt
```

On computer-b (Ubuntu 12.04.2 LTS):

```
apt-get install mutt
```

POP/IMAP Server installation:

This is optional and would only be required if you plan to grant access to user mailboxes via pop/imap. To install the **dovecot** (imap/pop) server, type the following as root.

On computer-a (Fedora 18):

```
yum install dovecot
```

You can use the following to check the status, enable, start, stop or restart the dovecot service.

systemctl status dovecot.service
systemctl enable dovecot.service
systemctl start dovecot.service
systemctl stop dovecot.service
systemctl restart dovecot.service

On computer-b (Ubuntu 12.04.2 LTS):

```
apt-get install dovecot-pop3d
```

You can use the following to check the status, start, stop or restart the dovecot service.

service dovecot status
service dovecot start
service dovecot stop
service dovecot restart

5.8.2 CONFIGURATION

MTA configuration:

The postfix configuration parameters are stored in "**/etc/postfix/main.cf**" file. Rather than editing the configuration file directly, you can use the "postconf" command to configure all postfix parameters. Check the man pages "man postconf" for more detailed options. We will not be using postconf for our examples since a simple configuration is sought.

On computer-a (Fedora 18):

For our simple example purpose, edit the "/etc/postfix/main.cf" file and make the following changes. Leave the rest as is default.

```
mydomain = domain-a.kkhsou.in
myorigin = $mydomain

myhostname = computer-a.domain-a.kkhsou.in
mydestination = $myhostname, localhost.$mydomain, localhost

mynetworks = 192.168.1.0/24, 127.0.0.0/8

inet_interfaces = all

home_mailbox = Maildir/
```

After adding the above to the "/etc/postfix/main.cf" file, we need to restart the postfix service for the changes to take effect. Type the following as root.

```
systemctl restart postfix.service
```

On computer-b (Ubuntu 12.04.2 LTS):

To configure postfix, run the command "dpkg-reconfigure postfix" as root or alternatively edit the "**/etc/postfix/main.cf**" and make the following changes as root. Leave the rest as default.

```
myhostname = computer-b.domain-b.kkhsou.in
myorigin = /etc/mailname

mydestination = computer-b.domain-b.kkhsou.in, localhost.domain-b.kkhsou.in, localhost

mynetworks = 127.0.0.0/8 192.168.1.0/24

inet_interfaces = all

home_mailbox = Maildir/
```

After adding the above to the "/etc/postfix/main.cf" file, we need to restart the postfix service for the changes to take effect. Type the following as root.

```
service postfix restart
```

By default Postfix will use mbox for the mailbox format. The "home_mailbox = Maildir/" setting will place new mails in a directory named "Maildir" of the respective users' home directory. We will need to configure our Mail Delivery Agent (MDA) to use the same path.

MDA configuration:

A Mail Delivery Agent (MDA) is invoked by the MTA to file incoming email in the proper user's mailbox. Any program that actually handles a message for delivery to the point where it can be read by an email client application can be considered an MDA. For this reason, some MTAs (such as Sendmail and Postfix) can fill the role of an MDA when they append new email messages to a local user's mail spool file. For our example purposes we will use the default configurations.

MUA configuration:

Since we are using mutt as our MUA, we need to add the following to the "**~/.muttrc**" file, i.e. the .muttrc file within the users' home directory. You will need to create this file if it does not already exist.

```
set mbox_type=Maildir
set folder="~/Maildir"
set mask="!^\\.[^.]"
set mbox="~/Maildir"
set record="+.Sent"
set postponed="+.Drafts"
set spoolfile="~/Maildir"
mailboxes `echo -n "+ "; find ~/Maildir -maxdepth 1 -type d -name ".*" -printf "+'%f' "`
```

POP/IMAP Server configuration:

The main configuration file for dovecot is located at "**/etc/dovecot/dovecot.conf**".

Now add the following to the dovecot configuration specifying the mailbox we are using.

```
mail_location = maildir:~/Maildir
```

Finally, we need to also add our mailserver that we have just configured as an MX Record in our DNS. If you refer to the section of this UNIT which deals with the installation and configuration of BIND as a DNS (Section 5.5), you will notice that we had added an MX for each of our example domains.

If you followed the examples in this section without running into an errors, you should have a working email server. However, prior to deployment in any live environment you should consult the documentation of each of these thoroughly and read about all the security issues related to email. Since, our aim was to setup a simple email system we did not cover most of the configurations.

5.9 INSTALLATION AND CONFIGURATION OF A DHCP SERVER

The Dynamic Host Configuration Protocol (DHCP) is a network protocol that automatically assigns TCP/IP information to client machines. In a DHCP environment each DHCP client connects to the centrally located DHCP server, which provides all

network related configuration including the IP address, gateway, and DNS servers. This reduces the administrative overhead of having to assign network related configuration manually to hosts in a network.

In this section, we will install and configure a simple DHCP server on Fedora 18 and Ubuntu 12.04.2 LTS.

5.9.1 INSTALLATION

On computer-a (Fedora 18):

To install the dhcp server type "**yum install dhcp**", as root.

```
[root@computer-a ~]# yum install dhcp
Loaded plugins: langpacks, presto, refresh-packagekit
updates/18/x86_64/metalink                              | 8.5 kB  00:00:00
updates                                                 | 4.7 kB  00:00:00
updates/primary_db                                      |  11 MB  00:00:28
Resolving Dependencies
--> Running transaction check
---> Package dhcp.x86_64 12:4.2.5-15.fc18 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package          Arch      Version                     Repository         Size
================================================================================
Installing:
 dhcp             x86_64    12:4.2.5-15.fc18   updates             506 k

Transaction Summary
================================================================================
Install  1 Package

Total download size: 506 k
Installed size: 1.4 M
Is this ok [y/N]: y
Downloading Packages:
dhcp-4.2.5-15.fc18.x86_64.rpm                           | 506 kB  00:00:03
```

```
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : 12:dhcp-4.2.5-15.fc18.x86_64     1/1
  Verifying  : 12:dhcp-4.2.5-15.fc18.x86_64     1/1

Installed:
  dhcp.x86_64 12:4.2.5-15.fc18

Complete!
[root@computer-a ~]#
```

To check the status of the dhcpd service, type the following as root.

```
[root@computer-a ~]# systemctl status dhcpd.service
```

To enable the dhcpd service, type the following as root.

```
[root@computer-a ~]# systemctl enable dhcpd.service
```

To start the dhcpd service, type the following as root.

```
[root@computer-a ~]# systemctl start dhcpd.service
```

To stop the dhcpd service, type the following as root.

```
[root@computer-a ~]# systemctl stop dhcpd.service
```

To restart the dhcpd service, type the following as root.

```
[root@computer-a ~]# systemctl restart dhcpd.service
```

On computer-b (Ubuntu 12.04.2 LTS):

To install the dhcp server type "**apt-get install isc-dhcp-server** ", as root.

```
root@computer-b:~# apt-get install isc-dhcp-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  isc-dhcp-client isc-dhcp-common
Suggested packages:
  isc-dhcp-server-ldap
The following NEW packages will be installed:
```

```
  isc-dhcp-server
The following packages will be upgraded:
  isc-dhcp-client isc-dhcp-common
2 upgraded, 1 newly installed, 0 to remove and 253 not upgraded.
Need to get 1,066 kB of archives.
After this operation, 1,010 kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get:1 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main isc-dhcp-client amd64 4.1.ESV-R4-0ubuntu5.9 [290 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main isc-dhcp-common amd64 4.1.ESV-R4-0ubuntu5.9 [347 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main isc-dhcp-server amd64 4.1.ESV-R4-0ubuntu5.9 [428 kB]
Fetched 1,066 kB in 4s (233 kB/s)
Preconfiguring packages ...
(Reading database ... 201832 files and directories currently installed.)
Preparing to replace isc-dhcp-client 4.1.ESV-R4-0ubuntu5.8 (using .../isc-dhcp-client_4.1.ESV-R4-0ubuntu5.9_amd64.deb) ...
Unpacking replacement isc-dhcp-client ...
Preparing to replace isc-dhcp-common 4.1.ESV-R4-0ubuntu5.8 (using .../isc-dhcp-common_4.1.ESV-R4-0ubuntu5.9_amd64.deb) ...
Unpacking replacement isc-dhcp-common ...
Selecting previously unselected package isc-dhcp-server.
Unpacking isc-dhcp-server (from .../isc-dhcp-server_4.1.ESV-R4-0ubuntu5.9_amd64.deb) ...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
Setting up isc-dhcp-common (4.1.ESV-R4-0ubuntu5.9) ...
Setting up isc-dhcp-client (4.1.ESV-R4-0ubuntu5.9) ...
Setting up isc-dhcp-server (4.1.ESV-R4-0ubuntu5.9) ...
Generating /etc/default/isc-dhcp-server...
isc-dhcp-server start/running, process 31631
isc-dhcp-server6 stop/waiting
root@computer-b:~#
```

To check the status of the dhcpd service, type the following as root.

```
[root@computer-b ~]# service isc-dhcp-server status
```

To start the dhcpd service, type the following as root.

```
[root@computer-b ~]# service isc-dhcp-server start
```

To stop the dhcpd service, type the following as root.

```
[root@computer-b ~]# service isc-dhcp-server stop
```

To restart the dhcpd service, type the following as root.

```
[root@computer-b ~]# service isc-dhcp-server restart
```

5.9.2 CONFIGURATION

On both Fedora and Ubuntu, the default location for the main DHCP configuration file is "**/etc/dhcp/dhcpd.conf**". A sample configuration file can be found at "**/usr/share/doc/dhcp-version/dhcpd.conf.sample**" on Fedora 18 and for Ubuntu 12.04.2 LTS at "**/usr/share/doc/isc-dhcp-server/examples/dhcpd.conf**". You can refer to this file while configuring "/etc/dhcp/dhcpd.conf".

We will configure a simple dhcp server for the example network we have setup.

```
#set the default and the maximum lease time in seconds
default-lease-time 600;
max-lease-time 7200;

#our example network configuration details
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option routers 192.168.1.254;
option domain-name-servers 192.168.1.1, 192.168.1.2;
option domain-search "kkhsou.in";
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.10 192.168.1.20;
}
```

You are required to restart the dhcp service after any changes to the configuration file.

Though we have a working dhcp server using the above configuration. In practice you may require more parameters to be included in the dhcp configuration. Therefore,

you are encouraged to read the man pages “man dhcpd.conf” before attempting to install, configure and use on a live network.

5.10 INSTALLATION AND CONFIGURATION OF A SSH SERVER AND CLIENT

The SSH (Secure Shell) is a protocol which facilitates secure communications between two systems using a client/server architecture and allows users to log into server host systems remotely. Both Fedora 18 and Ubuntu 12.04.2 LTS includes the general OpenSSH package (openssh) as well as the OpenSSH server (openssh-server) and client (openssh-clients) packages. Note that the OpenSSH packages require the OpenSSL package (openssl), which installs several important cryptographic libraries, enabling OpenSSH to provide encrypted communications. The SSH server listens on port 22 by default.

In this section, we will install and configure a simple SSH server "openssh" on Fedora 18 and Ubuntu 12.04.2 LTS.

5.10.1 INSTALLATION

During the Operating System installation, be it Fedora or Ubuntu, the openssh-client will be installed by default. Though the openssh-server may not be installed by default. We will be assuming that the openssh is not installed on our system and perform the installation on both Fedora and Ubuntu.

On computer-a (Fedora 18):

To install openssh type the following, as root.

```
[root@computer-a ~]# yum install openssh openssh-server openssh-clients openssl
```

To check the status of the openssh daemon, type the following as root.

```
[root@computer-a ~]# systemctl status sshd.service
```

To start the openssh daemon, type the following as root.

```
[root@computer-a ~]# systemctl start sshd.service
```

To stop the openssh daemon, type the following as root.

```
[root@computer-a ~]# systemctl stop sshd.service
```

To restart the openssh daemon, type the following as root.

```
[root@computer-a ~]# systemctl restart sshd.service
```

On computer-b (Ubuntu 12.04.2 LTS):

To install openssh type the following, as root.

```
root@computer-b:~# apt-get install openssh-server openssh-client openssl
```

To check the status of the openssh daemon, type the following as root.

```
root@computer-b:~# service ssh status
```

To start the openssh daemon, type the following as root.

```
root@computer-b:~# service ssh start
```

To stop the openssh daemon, type the following as root.

```
root@computer-b:~# service ssh stop
```

To restart the openssh daemon, type the following as root.

```
root@computer-b:~# service ssh restart
```

5.10.2 CONFIGURATION

In most cases, the default configuration for SSH should suffice. However, it is possible to configure the SSH server to suite your requirements.

The default configuration file for the ssh daemon is located at "**/etc/ssh/sshd_config**". Though we will not discuss it at length, you are encouraged to

use the man pages by typing “**man sshd_config**” at the terminal to read about it and have a clear understanding before attempting to configure it on a live system.

You should now be able to connect using any ssh client with the ssh server we just setup.

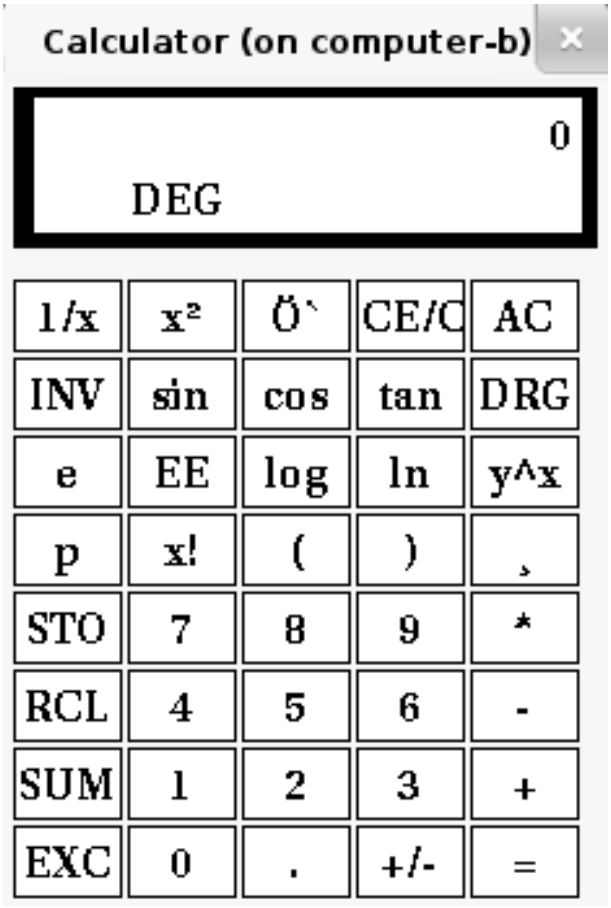
```
[binoddeka@computer-a ~]$ ssh binoddeka@computer-b
binoddeka@computer-b's password:
Last login: Fri Oct  4 21:51:18 2013 from 192.168.1.1
[binoddeka@computer-b ~]$
```

Another interesting option you should know about is the X11 forwarding, which can be used to connect to remote Linux system via ssh and work with the graphical desktop environment applications available on the remote system. To use ssh with this option simple type in the following while initiating the ssh connection.

```
[binoddeka@computer-a ~]$ ssh -Y binoddeka@computer-b
```

Now, for example if you type “xcalc” you should get a pop-up window of the graphical calculator.

```
[binoddeka@computer-b ~]$ xcalc
```



5.11 INSTALLATION AND CONFIGURATION OF A FTP SERVER AND CLIENT

The File Transfer Protocol (FTP) is one of the oldest and most commonly used protocols found on the Internet today. Its purpose is to reliably transfer files between computer hosts on a network without requiring the user to log directly into the remote host or have knowledge of how to use the remote system. It allows users to access files on remote systems using a standard set of simple commands. The ftp server listens on port 21 by default.

In this section, we will install and configure a simple FTP server "vsftpd" on Fedora 18 and Ubuntu 12.04.2 LTS. **vsftpd** (Very Secure FTP Daemon) is a fast, stable and secure FTP server.

5.11.1 INSTALLATION

On computer-a (Fedora 18):

To install **vsftpd** on Fedora 18, type the following as root.

```
[root@computer-a ~]# yum install vsftpd
Loaded plugins: langpacks, presto, refresh-packagekit
Resolving Dependencies
--> Running transaction check
---> Package vsftpd.x86_64 0:3.0.2-2.fc18 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================
 Package          Arch      Version             Repository        Size
================================================================
Installing:
 vsftpd           x86_64    3.0.2-2.fc18        updates           166 k

Transaction Summary
================================================================
Install  1 Package

Total download size: 166 k
Installed size: 355 k
Is this ok [y/N]: y
Downloading Packages:
vsftpd-3.0.2-2.fc18.x86_64.rpm                          | 166 kB  00:00:02
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : vsftpd-3.0.2-2.fc18.x86_64          1/1
  Verifying  : vsftpd-3.0.2-2.fc18.x86_64          1/1
```

```
Installed:
  vsftpd.x86_64 0:3.0.2-2.fc18
  Complete!
[root@computer-a ~]#
```

To check the status of the “vsftpd” service, type the following as root.

```
[root@computer-a ~]# systemctl status vsftpd.service
vsftpd.service - Vsftpd ftp daemon
   Loaded: loaded (/usr/lib/systemd/system/vsftpd.service; disabled)
   Active: inactive (dead)
```

To enable the “vsftpd” service, type the following as root.

```
[root@computer-a ~]# systemctl enable vsftpd.service
ln -s '/usr/lib/systemd/system/vsftpd.service' '/etc/systemd/system/multi-user.target.wants/vsftpd.service'
```

To start the “vsftpd” service, type the following as root.

```
[root@computer-a ~]# systemctl start vsftpd.service
```

To stop the “vsftpd” service, type the following as root.

```
[root@computer-a ~]# systemctl stop vsftpd.service
```

To restart the “vsftpd” service, type the following as root.

```
[root@computer-a ~]# systemctl restart vsftpd.service
```

On computer-b (Ubuntu 12.04.2 LTS):

To install **vsftpd** on Ubuntu, type the following as root.

```
root@computer-b:~# apt-get install vsftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  tdb-tools
Use 'apt-get autoremove' to remove them.
The following NEW packages will be installed:
  vsftpd
0 upgraded, 1 newly installed, 0 to remove and 256 not upgraded.
Need to get 124 kB of archives.
After this operation, 342 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu/ precise/main vsftpd amd64 2.3.5-1ubuntu2 [124 kB]
Fetched 124 kB in 2s (50.7 kB/s)
```

```
Preconfiguring packages ...
Selecting previously unselected package vsftpd.
(Reading database ... 201789 files and directories currently installed.)
Unpacking vsftpd (from .../vsftpd_2.3.5-1ubuntu2_amd64.deb) ...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
Setting up vsftpd (2.3.5-1ubuntu2) ...
vsftpd start/running, process 29005
root@computer-b:~#
```

To check the status of the “vsftpd” service, type the following as root.

```
root@computer-b:~# service vsftpd status
```

To start the “vsftpd” service, type the following as root.

```
root@computer-b:~# service vsftpd start
```

To stop the “vsftpd” service, type the following as root.

```
root@computer-b:~# service vsftpd stop
```

To restart the “vsftpd” service, type the following as root.

```
root@computer-b:~# service vsftpd restart
```

5.11.2 CONFIGURATION

In most cases, the default configuration for **vsftp** should suffice. However, it is possible to configure vsftp to suite your requirements.

On computer-a (Fedora 18):

The main configuration file for vsftpd is located at "**/etc/vsftpd/vsftpd.conf**".

The list of users not allowed to log into vsftpd is located at "**/etc/vsftpd/ftpusers**". By default, this list includes the root, bin, and daemon users, among others.

The "**/etc/vsftpd/user_list**" file can be configured to either deny or allow access to the users listed, depending on whether the userlist_deny directive is set to YES (default) or

NO in /etc/vsftpd/vsftpd.conf. If /etc/vsftpd/user_list is used to grant access to users, the usernames listed must not appear in /etc/vsftpd/ftpusers.

The folder "**/var/ftp/**" contains the files served by vsftpd. It also contains the /var/ftp/pub/ directory for anonymous users. Both directories are world-readable, but writable only by the root user.

You should now be able to connect to the ftp server we just installed. We will be loging in as an anonymous user without a password.

```
[binoddeka@computer-a ~]$ ftp
ftp> open localhost
Connected to localhost (127.0.0.1).
220 (vsFTPd 3.0.2)
Name (localhost:binoddeka): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls -l
227 Entering Passive Mode (127,0,0,1,184,157).
150 Here comes the directory listing.
drwxr-xr-x    2 0        0            4096 Sep 10 07:37 pub
226 Directory send OK.
ftp>
```

On computer-b (Ubuntu 12.04.2 LTS):

The main configuration file for vsftpd is located at "**/etc/vsftpd.conf**".

The "**/etc/ftpusers**" file contains the list of users that are disallowed for FTP access. The default list includes root, daemon, nobody, etc. To disable FTP access for additional users simply add them to the list.

During installation a user “ftp” is created with a home directory “**/srv/ftp**”. This is the default FTP directory.

You should now be able to connect to the ftp server we just installed. We will be loging in as an anonymous user without a password.

```
[binoddeka@computer-b ~]$ ftp localhost
Connected to localhost (127.0.0.1).
220 (vsFTPd 2.3.5)
Name (localhost:binoddeka): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls -la
227 Entering Passive Mode (172,16,24,1,42,218).
150 Here comes the directory listing.
drwxr-xr-x    2 0        133          4096 Oct 07 12:58 .
drwxr-xr-x    2 0        133          4096 Oct 07 12:58 ..
-rw-r--r--    1 0        0               0 Oct 07 12:58 test.txt
226 Directory send OK.
ftp>
```

Since we are setting up a simple ftp server we will go with the default configuration that is available after installation although you should be able to perform custom configurations based on your specific requirements.

CHECK YOUR PROGRESS

Q1.Give three examples of passive components.

Q2.What is Squid proxy server?

Q3.What do you mean by BIND?

Q4.What is a resource record?

Q5.What is Apache web server?

5.13 LET US SUM UP

In this unit we have tried to get acquainted with few aspects of using Linux Networking with the “with examples” approach. Though we have covered only a few of the topics, these topics are intended to inspire and point you in the direction to further explore the various tools available under Linux.

What we have learned in this unit:

- We installed and configured a simple LAN comprising of two computers, one Ethernet Switch and two CAT6 Patch cords. We also learnt how to configure IP addresses on both Fedora 18 and Ubuntu 12.04.2 LTS and to use the "ping" utility.
- We installed and configured a simple http proxy server using Squid.
- We installed and configured a simple DNS Server using BIND. We also setup DNS for two example domains, domain-a.kkhsou.in & domain-b.kkhsou.in; and learnt to configure DNS on both Fedora 18 and Ubuntu 12.04.2 LTS.
- We installed and configured a simple Web Server using Apache.
- We installed and configured a simple File Server using Samba.
- We installed and configured a simple Mail Server using Postfix. We also setup email systems for our two example domains and learnt to configure on both Fedora 18 and Ubuntu 12.04.2 LTS.
- We installed and configured a simple DHCP Server.
- We installed and configured a simple SSH Server and Client.
- We installed and configured a simple FTP Server and Client.

5.14 ANSWERS TO CHECK YOUR PROGRESS

1Ans: the names of three passive components are: CAT6 cables, Patch Cords, RJ45 ports.

2Ans: The Squid proxy server is a cache based proxy. It will fetch all web requests to populate its cache and then allow access to its cache based on its configuration. Squid

is a widely used proxy server as it permits you to save Internet Bandwidth and is feature rich. It runs on most available operating systems, including Windows and is licensed under the GNU GPL.

3Ans: BIND (Berkeley Internet Name Domain) is the most widely used DNS software on the Internet. Domain Name System or DNS is a service which provides resolution of fully qualified domain names (FQDN) into IP addresses and vice-versa. What this means is that domain names like say “www.kkhsou.in” is easier to remember than some IP address like “182.50.130.66”. Moreover, IP addresses may change. Just think how difficult it would be to keep remembering numerical addresses.

4Ans: In a DNS server such as BIND, all information is stored in basic data elements called resource records (RR). The resource record is usually a fully qualified domain name (FQDN) of a host.

5Ans: The Apache HTTP Server is a robust, full-featured open source web server developed by the Apache Software Foundation and is one of the most widely used web server software that is currently used on the Internet.

5.15 FURTHER READINGS

- The Linux man pages of the tools used.
- The Official Fedora Documentation available at “http://docs.fedoraproject.org”.
- The Official Ubuntu Documentation available at “https://help.ubuntu.com/”.

5.16 MODEL QUESTIONS

1. Which file contains the IP Address configurations in Linux?
2. What does the ping utility do?

3. What is a Proxy Server? Which command can be used to check the status of the Squid proxy server in Fedora 18?
4. What is BIND? Which file contains the main configuration for BIND in Ubuntu 12.04.2 LTS?
5. What does the "nslookup" utility do?
6. What does the "dig" utility do?
7. What is a FQDN?
8. What do you mean by a reverse name lookup in DNS?
9. What is Apache? Where is the main configuration file located for Apache?
10. What is Samba?
11. What is a MTA? Which network protocol is used by MTAs to communicate with each other?
12. Mozilla Thunderbird is a MTA, MUA or MDA?
13. What is a MX record?
14. What is DHCP?
15. Which file contains the configuration for the sshd daemon?
16. Which option can be used with the "ssh" utility for X11 forwarding?
17. What is vsftp?

****