OPERATING SYSTEM LAB MANUAL

SYLLABUS

CSE- 308 E

			Class Work	:	25
			Exam	:	25
L	T	P	Total	:	50
-	-	2	Duration of Exam	:	3 Hrs

- Study of WINDOWS 2000 Operating System.
- Administration of WINDOWS 2000 (including DNS, LDAP, Directory Services).
- Study of LINUX Operating System (Linux Kernel, Shell, Basic commands, pipe & filter commands).
- Administration of LINUX Operating System.
- Writing of Shell Scripts (Shell Programming).
- AWK Programming.

RATIONAL BEHIND THE OPERATING SYSTEM LAB

Operating systems are an essential part of any computer system. Similarly, a practical study on operating system is an essential part of any computer-science education. Operating system is a part that ensures the correct operation of computer system. We should know what the operating system does for the user, and what it does for the computer system operator, how the services provided by operating system are useful for user. This lab is intended as an introductory practical study of operating system. It provides a clear description of basic concepts that underlie operating systems.

In this Lab we concentrate on WINDOWS 2000 operating system , LINUX operating system. We discuss fundamental concepts that are applicable to WINDOWS 2000 operating system and LINUX operating system. We have a focus on LINUX commands. Also we have a practice of shell programming.

S/W & H/W REQUIREMENTS

Software Requirements :

1. Windows 2000 Operating System.
2. Linux Operating System.
3. Windows XP.

Hardware Requirements :

Processor	:	P-IV (1.8 GHz)
RAM	:	256 MB
Hard Disk	:	40 GB

LIST OF EXPERIMENTS

Sr. No.	Experiments
1.	Study of Windows 2000 operating system
2.	Administration of Windows 2000 operating system
3.	Study of Linux operating system
4.	Linux basic commands, pipe and filter commands
5.	Administration of Linux operating system
6.	Writing of Shell scripts: 1. To print the name, address and date of birth of a student 2. To print the date and the present working directory. 3. To calculate and print the sum of the 3 numbers 4. To find largest of three numbers. 5. To check for a number whether it is even or odd.
7.	AWK Programming

PRACTICAL-1

OBJECTIVE : Study of Windows 2000 operating System.

THEORY :

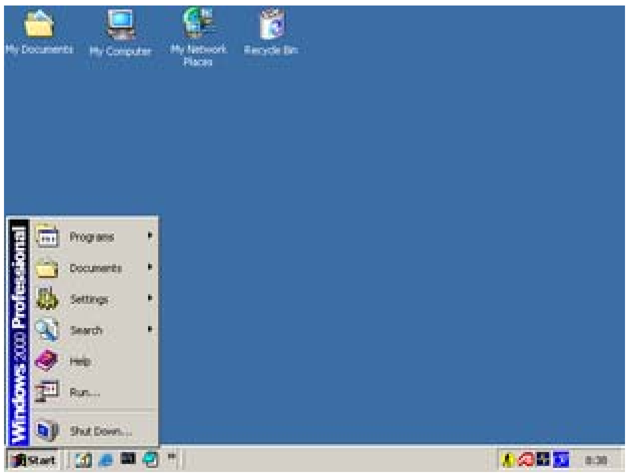
Windows 2000

Also called "Win2K" and "W2K," Windows 2000 was a major upgrade to Windows NT 4, launched in early 2000. Available in one client and three server versions, Windows 2000 added support for Plug and Play. Windows 2000 uses the same interface as Windows 95/98, but added considerably more features, dialogs and options.

Versions

Windows 2000 Version	Use	SMP Support	RAM
Professional	Client		2GB
Server	Server	4-way	2GB
Advanced Server*	Server	8-way	8GB
DataCenter Server*	Server	32-way	64GB

Windows 2000



Windows 2000 succeeded Windows NT 4.

Developer	Microsoft
OS family	Windows NT
Source model	Closed source
Latest release	Service Pack 4 / June 2003
Kernel type	Hybrid kernel
License	Microsoft EULA
Working state	Historic, Microsoft support ends July 1, 2005
Website	www.microsoft.com/windows2000

Windows 2000 (also referred to as **Win2K**, **W2K** or **Windows NT 5.0**) is a 32-bit preemptible, interruptible graphical business-oriented operating system, which has been designed to work with either uniprocessor or symmetrical multi processor (SMP) based Intel x86 computers. It is part of the Microsoft Windows NT line of operating systems and was released on February 17, 2000. Windows 2000 comes in four versions: Professional, Server, Advanced Server, and Datacenter Server. Additionally, Microsoft offers Windows 2000 Advanced Server, Limited Edition, released in 2001, which runs on Intel Itanium 64-bit processors. Windows 2000 is classed as a hybrid-kernel operating system and its architecture is divided into two modes, a user mode and a kernel mode. The user mode is highly restrictive and has very limited system resource access, while the kernel mode has no such restrictions.

All versions of Windows 2000 have common functionality, including many system utilities such as the Microsoft Management Console (MMC) and standard system management applications such as a disk defragmenter. Support for people with disabilities has also been improved by Microsoft across their Windows 2000 line, and they have included increased support for different languages and locale information. All versions of the operating system support the Microsoft filesystem, NTFS 5, the Encrypted File System (EFS) and basic and dynamic disk storage. Dynamic disk storage allows different types of volumes to be used. The Windows 2000 Server family has enhanced functionality, including the ability to provide Active Directory services (a hierarchical framework of resources), Distributed file system (a file system that supports sharing of files) and fault-redundant storage volumes.

Windows 2000 can be installed and deployed to an enterprise through either an attended or unattended installation. Unattended installations rely on the use of answer files to fill in installation information, and can be performed through a bootable CD using Microsoft System Management Server (SMS), by the System Preparation Tool (Sysprep).

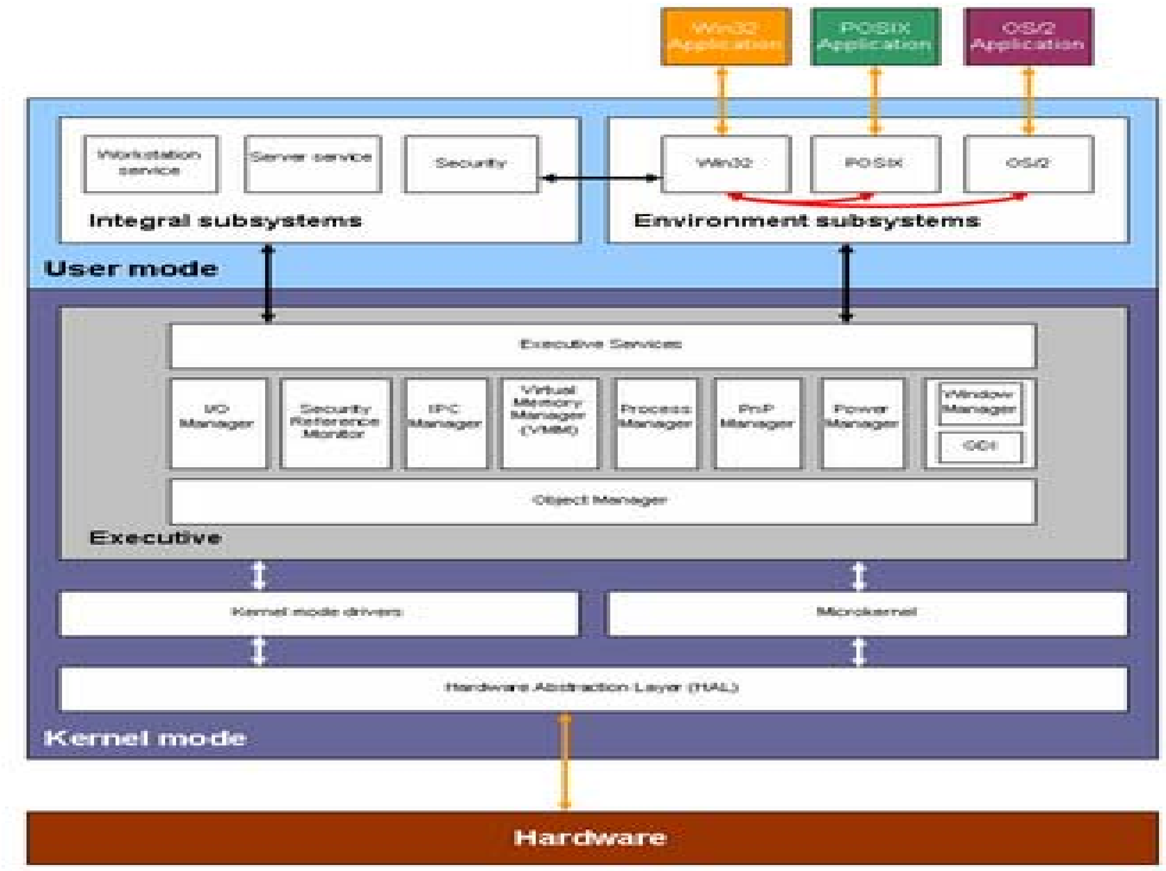
History

Windows 2000 originally descended from the Microsoft Windows NT operating system product line. Originally called Windows NT 5, Microsoft changed the name to Windows 2000 on October 27th, 1998. It was also the first Windows version that was released without a code name. The first beta for Windows 2000 was released on September 27, 1997 and several further betas were released until Beta 3 which was released on April 29, 1999. From here Microsoft issued three release candidates from between July to November 1999 and finally released the operating system to partners on December 12, 1999. The public received the full version of Windows 2000 on February 17, 2000 and the press immediately hailed it as the most stable operating system Microsoft had ever released. Novell, however, was not so impressed with Microsoft's new directory service architecture as they found it to be less scalable or reliable than their own Novell Directory Services (NDS) technology. On September 29, 2000, Microsoft released Windows 2000 Datacenter. Microsoft released Service Pack 2 on August 18, 2003, Service Pack 3 on October 31, 2003 and its last Service Pack (SP4) on November 11, 2003. Microsoft phased out all development of their Java Virtual Machine (JVM) from Windows 2000 in Service Pack 3.

Windows 2000 has since been superseded by newer Microsoft operating systems. Microsoft has replaced Windows 2000 Server products with Windows Server 2003, and Windows 2000 Professional with Windows XP Professional. Windows Neptune started development in 1999, and was supposed to be the home-user edition of Windows 2000. However, the project lagged in production time – and only one alpha release was built. Windows Me was released as a substitute, and the Neptune project was forwarded to the production of Whistler (Windows XP). The only elements of the Windows project which were included in Windows 2000 were the ability to upgrade from Windows 95 or Windows 98, and support for the FAT32 subsystem.

Several notable security flaws have been found in Windows 2000. Code Red and Code Red II were famous (and highly visible to the worldwide press) computer worms that exploited vulnerabilities of the indexing service of Windows 2000's Internet Information Services (IIS). In August 2003, two major worms named the Sobig worm and the Blaster worm began to attack millions of Microsoft Windows computers, resulting in the largest down-time and clean-up cost ever. The worms have also had political consequences as many companies in several countries started to call for government action to prevent further damages from Windows worms.

Architecture



The Windows 2000 operating system architecture consists of two layers (user mode and kernel mode), with many different modules within both of these layers.

Windows 2000 is a highly modular system that consists of two main layers: a user mode and a kernel mode. The user mode refers to the mode in which user programs are run. Such programs are limited in terms of what system resources they have access to, while the kernel mode has unrestricted access to the system memory and external devices. All user mode applications access system resources through the executive which runs in kernel mode.

User mode

User mode in Windows 2000 is made of subsystems capable of passing I/O requests to the appropriate kernel mode drivers by using the I/O manager. Two subsystems make up the user mode layer of Windows 2000: the environment subsystem and the integral subsystem.

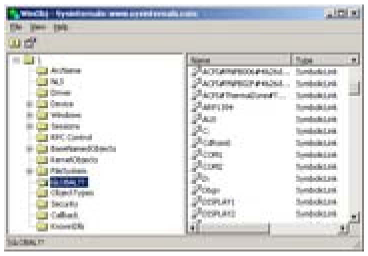
The environment subsystem was designed to run applications written for many different types of operating systems. These applications, however, run at a lower priority than kernel mode processes. There are three main environment subsystems:

1. Win32 subsystem runs 32-bit Windows applications and also supports Virtual DOS Machines (VDMs), which allows MS-DOS and 16-bit Windows 3.x (Win16) applications to run on Windows.
2. OS/2 environment subsystem supports 16-bit character-based OS/2 applications and emulates OS/2 1.3 and 1.x, but not 2.x or later OS/2 applications.
3. POSIX environment subsystem supports applications that are strictly written to either the POSIX.1 standard or the related ISO/IEC standards.

The integral subsystem looks after operating system specific functions on behalf of the environment subsystem. It consists of a *security subsystem* (grants/denies access and handles logons), *workstation service* (helps the computer gain network access) and a *server service* (lets the computer provide network services).

Kernel mode

Kernel mode in Windows 2000 has full access to the hardware and system resources of the computer. The kernel mode stops user mode services and applications from accessing critical areas of the operating system that they should not have access to.



Each object in Windows 2000 exists in its own namespace. This is a screenshot from *SysInternal's* WinObj (*http://www.sysinternals.com/ntw2k/freeware/winobj.shtml*)

The executive interfaces with all the user mode subsystems. It deals with I/O, object management, security and process management. It contains various components, including:

- **Object manager**: a special executive subsystem that all other executive subsystems must pass through to gain access to Windows 2000 resources. This essentially is a resource management infrastructure service that allows Windows 2000 to be a object oriented operating system.
- **I/O Manager:** allows devices to communicate with user-mode subsystems by translating user-mode read and write commands and passing them to device drivers.
- **Security Reference Monitor (SRM):** the primary authority for enforcing the security rules of the security integral subsystem.

- **IPC Manager:** short for Interprocess Communication Manager, manages the communication between clients (the environment subsystem) and servers (components of the executive).
- **Virtual Memory Manager:** manages virtual memory, allowing Windows 2000 to use the hard disk as a primary storage device (although strictly speaking it is secondary storage).
- **Process Manager:** handles process and thread creation and termination
- **PnP Manager:** handles Plug and Play and supports device detection and installation at boot time.
- **Power Manager:** the power manager coordinates power events and generates power IRPs.
- The display system is handled by a device driver contained in *Win32k.sys*. The **Window Manager** component of this driver is responsible for drawing windows and menus while the **GDI** (graphical device interface) component is responsible for tasks such as drawing line (mathematics)lines and curves, rendering fonts and handling palettes.

The Windows 2000 Hardware Abstraction Layer, or HAL, is a layer between the physical hardware of the computer and the rest of the operating system. It was designed to hide differences in hardware and therefore provide a consistent platform to run applications on. The HAL includes hardware specific code that controls I/O interfaces, interrupt controllers and multiple processors.

The microkernel sits between the HAL and the executive and provides multiprocessor synchronization, thread and interrupt scheduling and dispatching, trap handling and exception dispatching. The microkernel often interfaces with the process manager. The microkernel is also responsible for initializing device drivers at boot up that are necessary to get the operating system up and running.

Kernel mode drivers exist in three levels: highest level drivers, intermediate drivers and low level drivers. The Windows Driver Model (WDM) exists in the intermediate layer and was mainly designed to be binary and source compatible between Windows 98 and Windows 2000. The lowest level drivers are either legacy Windows NT device drivers that control a device directly or can be a PnP hardware bus.

PRACTICAL-2

OBJECTIVE : Administration of Windows 2000 (including DNS, LDAP, Directory Services).

THEORY :

Introduction

Make computers easier and more comfortable to use with accessibility features and utilities built right into Microsoft® Windows 2000 (Professional and Home versions).Display and ReadabilityThe job description for a systems administrator says that little creativity is required, but this is not necessarily the case. Often, the job requires a great deal of creative thinking in order to solve a problem quickly. If a systems problem crops up, there may be more than one "fix," and it is the job of the systems administrator to determine which solution will be the most effective. He may also be in charge of upgrading the system and deciding which software is the most effective for the job it does.Generally , All these works done by administrator are part of System Administration.

LDAP
(Lightweight Directory Access Protocol)

Lightweight Directory Access Protocol (LDAP) is an established Internet standard that enables cross-network operating system interoperability between directory services that support it. In Windows 2000, LDAP is the primary way the Operating System accesses the Active Directory database. By using this open standard, Microsoft is enabling 3rd party vendors and other platforms (Unix) to be able to work in a Windows 2000 environment and use AD services.

Active Directory and LDAP

LDAP fills a huge hole in NT and provides authenticated access to all network resources. By Mark Minasi , *Windows NT Magazine,* December 1997

Common LDAP RFCs

Microsoft Knowledge Base Article: 221606 - Enumerates the RFCs that define Lightweight Directory Access Protocol (LDAP).

How to Change a Windows 2000 User's Password Through LDAP

Microsoft Knowledge Base Article: 269190 - You can set a Windows 2000 user's password through the Lightweight Directory Access Protocol (LDAP) given certain restrictions. This article describes how to set or change the password attribute.

The Evolution of LDAP
A brief history of the evolution of LDAP. Source: Windows & .NET Magazine, April 1998.

Using Ldp.exe to Find Data in the Active Directory

Microsoft Knowledge Base Article: 224543 - LDP.EXE is a Windows 2000 Resource Kit utility that can be used to perform LDAP (Lightweight Directory Access Protocol) searches against the Active Directory for specific information given search criteria.

Microsoft LDAP Error Codes

Microsoft Knowledge Base Article: 218185 - Microsoft Windows 2000 Active Directory uses the Internet-standard Lightweight Directory Access Protocol (LDAP) to access information. In response to various LDAP requests, a domain controller returns responses containing field LDAP error

Multiple LDAP Binds to the Same Connection Cause Memory Leak

Microsoft Knowledge Base Article: 289644 - On a Windows 2000-based domain controller, multiple Lightweight Directory Access Protocol (LDAP) binds over the same connection to that domain controller cause a memory leak.

Stand-Alone "LDAP://" Query Does Not Work

Microsoft Knowledge Base Article: 220594 - When you attempt a stand-alone query using LDAP://, you may receive the following error message: An error occurred while performing the search. Your computer, your Internet service provider, or the specified directory service may be disconnected. Check your connections and try again.

DNS (Domain Name System):

DDNS will replace WINS as Microsoft's name resolution system. Like WINS, DDNS is dynamic, however, it will be fully compatible with Unix and other systems that are DNS based.

With Windows 2000, all Active Directory domain names are DNS domain names, but--and this is important--not every DNS domain name is an Active Directory domain name.[1] So while an organization's Active Directory namespace resembles its DNS namespace, the two don't have to be and probably won't be identical. While it's beyond the scope of this book to give an exhaustive explanation of Active Directory namespace design, we can give you some examples to clarify the connection between the naming of Active Directory domains and DNS domains

The Active Directory integrated, Internet standards-based Domain Name System (DNS) service simplifies object naming and location through Internet protocols, and improves scalability, performance and interoperability. Systems that receive addresses from a Dynamic Host Configuration Protocol (DHCP) server are automatically registered in DNS. Replication options with legacy DNS system and through Active Directory can simplify and strengthen name replication infrastructure.

Domain Local Groups

- are valid only on a single (local) domain
- used to assign permissions in the local domain.
- Can contain users and Global groups, including Global groups from other trusted domains.

Global Groups

- can access resources in other domains.
- Used to export user accounts to other domains, where they can be - imported into Local Groups on trusting domains
- With NT4, contains user accounts only.
- With Windows 2000, a Global group can nest within other global - - groups from within their own domain, but such groups don't appear in the GC.

Universal Groups

New to Windows 2000.This is available only if Windows 2000 is running under native mode. Members of Universal groups can be from any domain.

Special groups

Used by Windows Server for system access, and do not contain user or group accounts.

Group (Printer) Permissions Usage

Permissions are never assigned directly to global groups.

Organize computer users based on administrative needs (locations and job functions) into global groups such as “LA Sales".

Identify common resources (such as files and printers) into domain local groups such as “Color printer X2".

Add global groups to a Domain Local Group.

Assign permissions (on specific resources) to the Domain Local Group.

DIRECTORY SERVICES

A central component of the Windows platform, Active Directory service provides the means to manage the identities and relationships that make up network environments. Windows Server 2003 makes Active Directory simpler to manage, easing migration and deployment.

Active Directory Integration

Active Directory integration with the underlying security infrastructure provides a focal point of security management of users, computers and devices making Windows 2000 easier to manage.

Multi-master Replication

Active Directory uses multi-master replication to ensure high scalability and availability in distributed network configurations. "Multi-master" means that each directory replica in the network is a peer of all other replicas; changes can be made to any replica and will be reflected across all of them

Integrated Directory Services

Windows 2000 introduces Active Directory, a scalable, standard-compliant directory service that makes Windows 2000 easier to manage, more secure, and more interoperable with existing investments. Active Directory centrally manages Windows- based clients, and servers through a single consistent management interface, reducing redundancy and maintenance costs

Directory interoperability

Meta directory technologies enable companies to use Active Directory to manage identity information stored in heterogeneous directory services.

Directory synchronization tools

Maintain and synchronize data between Active Directory and Microsoft Exchange and Novell NDS directories.

DELEGATION

Delegation is a powerful feature of Windows 2000 that helps administrators shuffle off some of their administrative responsibility to other trusted (trustworthy) users before overwork causes them to "shuffle off this mortal coil."

DHCP

If you are going to deploy and manage IP addressing on Windows 2000 using DHCP, you might want to disable the Automatic Private IP Addressing (APIPA) feature on your machines. APIPA causes an IP address to be automatically assigned to a client machine from the reserved address range 169.254.0.1 through 169.254.255.254 when the system is configured for DHCP but is unable to contact a DHCP server when it first starts up. This can be nasty, since no warning message indicates that the system has used APIPA instead of DHCP to obtain its address, resulting in an inability to access other machines on the network because they are on a different subnet.

Disk Quotas

A good tip when implementing disk quotas is to configure global quotas only and not quotas for individual users. Not following this can make quota administration a real headache.

Disks

Microsoft has borrowed the concept of mounted volumes from Unix and implemented the ability to mount a FAT or NTFS volume in an empty folder on an NTFS volume in Windows 2000. This feature helps you get beyond the 24-letter limit for mapped drives. See *disks* in Chapters 3 and 4 for details. Note that you can cause problems for yourself with this feature: nothing prevents you from mounting a volume in a folder on a mounted volume, or even mounting a volume in a folder on itself!

MMC

The Microsoft Management Console can be used for building customized administrative tools, which can then be distributed by email or by storing them on a network share. See the first part of Chapter 5 for information on the MMC and how to customize it.

Permissions

Like the earlier Windows NT operating system, Windows 2000 provides you with two sets of permissions for security access to files and folders: NTFS permissions and shared-folder permissions. The basic approach for secure shared resources is the same as in NT, but NTFS permissions will require some relearning in Windows 2000.

Printers

One terrific feature of Windows 2000 is that you can manage printers remotely across a network (or even over the Internet) using only a web browser. See in and for more information about this feature. By the way, to print to a Windows 2000 print server over the Internet, open the printer in your web browser and click Connect. This installs the appropriate drivers on your computer and creates a network printer to let you print to the remote print device. Let Windows 2000 detect Plug and Play printers and install drivers for them automatically. If you install the driver manually and reboot your machine, you may end up with two printers for the same print device!

PRACTICAL-3

OBJECTIVE : Study of LINUX operating system (Linux Kernel, Shell).

THEORY :

Introduction

Linux is a true 32-bit operating system that runs on a variety of different platforms, including Intel, Sparc, Alpha, and Power-PC (on some of these platforms, such as Alpha, Linux is actually 64-bit). Linux can and should be considered a full-blown implementation of unix. However, it can not be called "Unix"; not because of incompatibilities or lack of functionality, but because the word "Unix" is a registered trademark owned by AT&T, and the use of the word is only allowable by license agreement. Linux runs on 386/486/Pentium machines with ISA, EISA, PCI and VLB busses. MCA (IBM's proprietary bus) is not well-supported in 2.0.x and earlier versions, but support has been added to the current development tree, 2.1.x. There is a port to multiple Motorola 680x0 platforms (currently running on some Amigas, Ataris, and VME machines), which now works quite well. It requires a 68020 with an MMU, a 68030, 68040, or a 68060, and also requires an FPU. Networking and X now work. Linux is being actively ported to the PowerPC architecture, including PowerMac (Nubus and PCI), Motorola, IBM, and Be machines. Ports to other machines, including MIPS and ARM, are under way and showing various amounts of progress. The Linux operating system is currently the operating system of choice for computational physicists in academia. Linux is a freely redistributable clone of the UNIX operating system, which runs on most personal computer (PC) platforms (Intel x86, Mac, *etc*.) Computational physicists in academia generally prefer Linux for a number of reasons. Firstly, Linux is free! This is a powerful incentive for academics, who generally have to work on very tight budgets. Secondly, the hardware platforms, which support Linux, namely PCs, are relatively inexpensive, since the market for PCs is vast and highly competitive. By contrast, scientific workstations which run proprietary brands of UNIX (*e.g.*, Sun Sparc, IBM RS/6000, *etc.*) tend to be relatively expensive, since the market for such workstations is minuscule compared to that for PCs. Thirdly, Linux is a very powerful operating system. It has full UNIX features, and is, therefore, a true multi-user, multi-tasking, networked, operating system. Fourthly, Linux is extremely stable and almost never crashes. Finally, because Linux is free and runs on PCs, academics (not to mention graduate and undergraduate students!) can easily afford to run the same operating system at home as at work.

The primary author of Linux is Linus Towards. Since his original versions, it has been improved by countless numbers of people.

Linux is a freely distributed, multitasking, multi-user operating system that behaves like UNIX. The term "Linux" is actually somewhat vague. "Linux" is used in two ways: specifically to refer to the kernel itself –the heart of any version of Linux –and more generally to refer to any collection of applications that run on the kernel, usually referred to a distribution. The kernel's job is to provide the basic environment in which applications can run, including the basic interfaces with hardware.

Although there are a large number of Linux implementations, you will find a lot of similarities in the different distributions, if only because every Linux machine is a box with building blocks that you may put together following your own needs and views. Installing the system is only the beginning of a longterm relationship. Just when you think you have a nice running system, Linux will stimulate your imagination and creativeness, and the more you realize what power the system can give you, the more you will try to redefine its limits.
Linux may appear different depending on the distribution, your hardware and personal taste, but the fundamentals on which all graphical and other interfaces are built, remain the same. The Linux system is based on GNU tools (Gnu's Not UNIX), which provide a set of standard ways to handle and use the system. All GNU tools are open source, so they can be installed on any system. Most distributions offer pre-compiled packages of most common tools, such as RPM packages on RedHat and Debian packages (also called deb or dpkg) on Debian, so you needn't be a programmer to install a package on your system. However, if you are and like doing things yourself, you will enjoy Linux all the better, since most distributions come with a complete set of development tools, allowing installation of new software purely from source code. This setup also allows you to install software even if it does not exist in a pre-packaged form suitable for your system.

A list of common GNU software:

- Bash: The GNU shell
- GCC: The GNU C Compiler
- GDB: The GNU Debugger
- coreutils: a set of basic UNIX-style utilities, such as **ls**, **cat** and **chmod**
- Findutils: to search and find files
- Fontutils: to convert fonts from one format to another or make new fonts
- The Gimp: GNU Image Manipulation Program
- Gnome: the GNU desktop environment
- Emacs: a very powerful editor
- Ghostscript and Ghostview: interpreter and graphical frontend for PostScript files.
- GNU Photo: software for interaction with digital cameras
- Octave: a programming language, primarily intended to perform numerical computations and image processing.
- GNU SQL: relational database system
- Radius: a remote authentication and accounting server

Many commercial applications are available for Linux, and for more information about these packages we refer to their specific documentation. Throughout this guide we will only discuss freely available software, which comes (in most cases) with a GNU license.

To install missing or new packages, you will need some form of software management. The most common implementations include RPM, dpkg and Ximian Red Carpet. RPM is the RedHat Package Manager, which is used on a variety of Linux systems, eventhough the name does not suggest this. Dpkg is the Debian package management system, which uses an interface called **apt-get**, that can manage RPM packages as well. Ximian Red Carpet is a third party implementation of RPM with a graphical front-end. Other third party software vendors may have their own installation procedures, sometimes resembling the InstallShield and such, as known on MS Windows and other platforms. As you advance into Linux, you will likely get in touch with one or more of these programs.

LINUX

The Linux kernel (the *bones* of your system, see the section called “The kernel”) is not part of the GNU project but uses the same license as GNU software. A great majority of utilities and development tools (the *meat* of your system), which are not Linux-specific, are taken from the GNU project. Because any usable system must contain both the kernel and at least a minimal set of utilities, some people argue that such a system should be called a *GNU/Linux* system. In order to obtain the highest possible degree of independence between distributions, this is the sort of Linux that we will discuss throughout this course. If we are not talking about a Linux system, the specific distribution, version or program name will be mentioned.

Since every Linux distribution contains the basic packages and can be built to meet almost any requirement (because they all use the Linux kernel), you only need to consider if the distribution will run on your hardware. LinuxPPC for example has been made to run on MacIntosh and other PowerPCs and does not run on an ordinary x86 based PC. LinuxPPC does run on the new Macs, but you can't use it for some of the older ones with ancient bus technology. Another tricky case is Sun hardware, which could be an old SPARC CPU or a newer UltraSparc, both requiring different versions of Linux.

Some Linux distributions are optimized for certain processors, such as Athlon CPUs, while they will at the same time run decent enough on the standard 486, 586 and 686 Intel processors. Sometimes distributions for special CPUs are not as reliable, since they are tested by fewer people.

Most Linux distributions offer a set of programs for generic PCs with special packages containing optimized kernels for the x86 Intel based CPUs. These distributions are well-tested and maintained on a regular basis, focusing on reliant server implementation and easy installation and update procedures. Examples are Debian, Ubuntu, Fedora, SuSE and Mandriva, which are by far the most popular Linux systems and generally considered easy to handle for the beginning user, while not blocking professionals from getting the most out of their Linux machines. Linux also runs decently on laptops and middle-range servers. Drivers for new hardware are included only after extensive testing, which adds to the stability of a system.

While the standard desktop might be Gnome on one system, another might offer KDE by default. Generally, both Gnome and KDE are available for all major Linux distributions. Other window and desktop managers are available for more advanced users.

The standard installation process allows users to choose between different basic setups, such as a workstation, where all packages needed for everyday use and development are installed, or a server installation, where different network services can be selected. Expert users can install every combination of packages they want during the initial installation process.

The goal of this guide is to apply to all Linux distributions. For your own convenience, however, it is strongly advised that beginners stick to a mainstream distribution, supporting all common hardware and applications by default. The following are very good choices for novices:

- Fedora Core
- Debian
- SuSE Linux
- Mandriva (former MandrakeSoft)
- Knoppix: an operating system that runs from your CD-ROM, you don't need to install anything.

Downloadable ISO-images can be obtained from LinuxISO.org. The main distributions can be purchased in any decent computer shop.

Linux Properties

Linux Pros

A lot of the advantages of Linux are a consequence of Linux' origins, deeply rooted in UNIX, except for the first advantage, of course:

Linux is free:

As in free beer, they say. If you want to spend absolutely nothing, you don't even have to pay the price of a CD. Linux can be downloaded in its entirety from the Internet completely for free. No registration fees, no costs per user, free updates, and freely available source code in case you want to change the behavior of your system.

Most of all, Linux is free as in free speech:

The license commonly used is the GNU Public License (GPL). The license says that anybody who may want to do so, has the right to change Linux and eventually to redistribute a changed version, on the one condition that the code is still available after redistribution. In practice, you are free to grab a kernel image, for instance to add support for teletransportation machines or time travel and sell your new code, as long as your customers can still have a copy of that code.

Linux is portable to any hardware platform:

A vendor who wants to sell a new type of computer and who doesn't know what kind of OS his new machine will run (say the CPU in your car or washing machine), can take a

Linux kernel and make it work on his hardware, because documentation related to this activity is freely available.

Linux was made to keep on running:

As with UNIX, a Linux system expects to run without rebooting all the time. That is why a lot of tasks are being executed at night or scheduled automatically for other calm moments, resulting in higher availability during busier periods and a more balanced use of the hardware. This property allows for Linux to be applicable also in environments where people don't have the time or the possibility to control their systems night and day.

Linux is secure and versatile:

The security model used in Linux is based on the UNIX idea of security, which is known to be robust and of proven quality. But Linux is not only fit for use as a fort against enemy attacks from the Internet: it will adapt equally to other situations, utilizing the same high standards for security. Your development machine or control station will be as secure as your firewall.

Linux is scalable:

From a Palmtop with 2 MB of memory to a petabyte storage cluster with hundreds of nodes: add or remove the appropriate packages and Linux fits all. You don't need a supercomputer anymore, because you can use Linux to do big things using the building blocks provided with the system. If you want to do little things, such as making an operating system for an embedded processor or just recycling your old 486, Linux will do that as well.

The Linux OS and quite some Linux applications have very short debu times:

Because Linux has been developed and tested by thousands of people, both errors and people to fix them are usually found rather quickly. It sometimes happens that there are only a couple of hours between discovery and fixing of a bug.

Linux Cons

There are far too many different distributions:

"Quot capites, tot rationes", as the Romans already said: the more people, the more opinions. At first glance, the amount of Linux distributions can be frightening, or ridiculous, depending on your point of view. But it also means that everyone will find what he or she needs. You don't need to be an expert to find a suitable release.

When asked, generally every Linux user will say that the best distribution is the specific version he is using. So which one should you choose? Don't worry too much about that: all releases contain more or less the same set of basic packages. On top of the basics, special third party software is added making, for example, TurboLinux more suitable for the small and medium enterprise, RedHat for servers and SuSE for workstations. However, the differences are likely to be very superficial. The best strategy is to test a couple of distributions; unfortunately not everybody has the time for this. Luckily, there is plenty of advice on the subject of choosing your Linux. A quick search on Google, using the keywords “choosing your distribution” brings up tens of links to good advise. The Installation HOWTO also discusses choosing your distribution.

Who uses Linux

Linux is freely available, and no one is required to register their copies with any central authority, so it is difficult to know how many people use Linux. Several businesses now survive solely on selling and supporting Linux, and the Linux newsgroups are some of the most heavily read on the internet, so the number is likely in the millions, but firm numbers are hard to come by. The best market research currently indicates between 7.5 and 11 million users.

The Future

After Linux 1.0 was released, work was done on several enhancements. Linux 1.2 included disk access speedups, TTY improvements, virtual memory enhancements, multiple platform support, quotas, and more. Linux 2.0, the current stable version, has even more enhancements, including many performance improvements, several new networking protocols, one of the fastest TCP/IP implementations in the world, and far, far more. Even higher performance, more networking protocols, and more device drivers will be available in Linux 2.2.

Even with over 3/4 million lines of code in the kernel, there is plenty of code left to write, and even more documentation. Please join the linux-doc@vger.rutgers.edu mailing list if you would like to contribute to the documentation. Send mail to majordomo@vger.rutgers.edu with a single line containing the word ``help" in the body (**NOT** the subject) of the message.

PRACTICAL-4

OBJECTIVE : Administration of Linux operating system.

THEORY :
Systems Administration is one of the most complex, fulfilling and misunderstood professions within the computing arena. Everybody who uses the computer depends on the Systems Administrator doing his or her job correctly and efficiently. However the only time users tend to give the Systems Administrator a second thought is when the computer system is not working Very few people, including other computing professionals, understand the complexity and the time-consuming nature of Systems Administration. Even fewer people realize the satisfaction and challenge that Systems Administration presents to the practitioner. It is one of the rare computing professions in which the individual can combine every facet of the computing field into one career. The aim of this chapter is to provide you with some background to Systems Administration so that you have some idea of why you are reading this and what you may learn via this text.

What Systems Administrators do?

Systems Administration is an old responsibility gaining newfound importance and acceptance as a profession. It has come into existence because of the increasing complexity of modern computer systems and networks and because of the economy's increasing reliance on computers. Any decent size business now requires at least one person to keep the computers running happily. If the computers don't work the business suffers.

It can be said that Systems Administrators have two basic reasons for being

- Ensuring that the computing system runs correctly and as efficiently as possible, and
- Ensuring that all users can and do use the computing system to carry out their required work in the easiest and most efficient manner.

These two reasons often conflict with one another. Management will wish to restrict the amount of money spent on computer systems. The users on the other hand will always want more disk space and faster CPUs. The System Administrator must attempt to balance these two conflicting aims.

The real work required to fulfill these aims depends on the characteristics of the particular computing system and the company it belongs to. Factors that affect what a Systems Administrator needs to do come from a number of categories including: users, hardware and support

Users

Users, your colleagues and workmates that use computers and networks to perform their tasks contribute directly to the difficulty (or ease) of your task as a Systems Administrator. Some of the characteristics of people that can contribute to your job include:

- How many users are there?
  Two hundred users are more difficult to help than two users and also require completely different practices. With two, or even ten/twenty, users it is possible to become well known to them and really get to know their requirements. With two hundred, or in some cases two thousand users, this is simply not possible.
- The level of the user's expertise.
  This is a combination of the user's actual expertise and their perceived expertise. A user who thinks they know a lot (but doesn't really) can often be more trouble than a user who knows nothing and admits it.

Users who know what they know.

Picture it. You are a Systems Administrator at a United States Air Force base. The people using your machines include people who fly million dollar weapons of destruction that have the ability to reduce buildings if not towns to dust. Your users are supremely confident in their ability. What do you do when an arrogant, abusive Colonel contacts you saying he cannot use his computer? What do you say when you solve the problem by telling him he did not have it plugged in? What do you do when you have to do this more than once? It has happened.

- What are the users trying to do?
  If the users are scientists doing research on ground breaking network technology you will be performing completely different tasks than if your users are all doing word processing and spreadsheets.
- Are they responsible or irresponsible?
  Do the users follow the rules or do they make their own? Do the users like to play with the machines? Being the Systems Administrator in a computing department at a University, where the users are computing students who want to play and see how far they can go is completely different from working in a government department, where the users hate computing and only use them when necessary.
- Who do the users know?
  A user, who has a 15-year-old, computer nerd son can often be the cause of problems since the son will tell the parent all sorts of things about computers and what can be done. Very few people have an appreciation of the constraints placed
- On a Systems Administrator and the computers under their control. Looking after a home PC is completely different to managing a collection of computers at a place of work.

Hardware/Software

The computers, software, networks, printers and other peripherals that are at a site also contribute to the type and amount of work a Systems Administrator must perform. Some considerations include:

- How many, how big and how complex?
  Once again greater numbers imply more work. Also it may be more work looking after a network of Windows NT machines than a collection of Windows 3.1 computers. Some sites will have supercomputers, which require specialized knowledge.
- Is there a network?
  The existence of a network connecting the machines together raises additional problems and further increases the workload of the Systems Administrator.
- Are the computers heterogeneous or homogenous?
  Is the hardware and software on every machine the same, or is it different. A great variety in hardware and software will make it much more difficult to manage, especially when there are large numbers. The ability to specify a standard for all computers, in both hardware and software, makes the support job orders of magnitude easier.

Support

One other area, which makes a difference to the difficulty of a job as a Systems Administrator, is the level of support in the form of other people, time and resources. The support you do (or don't?) receive can take many forms including:

- Are you alone?
  At some sites there is one administrator who does everything from installing peripherals, fixing computers, doing backups, helping users find the enter key and a range of other tasks. At other sites these tasks are split amongst a range of administrators, operators and technicians.
- Are you a full time administrator?
  In some cases the administrator looks after the machines in addition to performing their "real job".
- What are the feelings of staff and management towards the Systems Administrators?
  In many companies the management and staff see Systems Administrators or other computer support people as overhead. This impression of Systems Administrators as an unnecessary expense influences how the users will act. Similar feelings can occur if previous Systems Administrators have been unprofessional or unable to perform their jobs.

What Systems Administrators need to know?

The answer is that to be a really good Systems Administrator you need to know everything about the entire computer system including the operating system, hardware, software, users, management, network and anything else you can think of that might affect the system in any way.

Failing that lofty aim the System Administrator must have the ability to gain this all-encompassing knowledge. The discovery process may include research, trial and error, or begging. The abilities to learn and problem solve may well be the two most important for a Systems Administrator.

At some time during their career a Systems Administrator will make use of knowledge from the following (far from exhaustive) list of fields, both computing and non-computing:

- Programming,
  Systems Administrators have to be able to program. They might have to write scripts that automate regular tasks or a Visual Basic program to help users perform certain tasks.
- Hardware maintenance and installation,
  this may range from installing new hardware to cleaning old hardware so that it continues to work.
- Documentation and testing,
- Human Computer Interface,
- Networks and computer communication,
- User education,
- Diplomacy,
- Legal issues and contracts,
- Detective work,
- Management and policy setting, and
- Public relations.

SYSTEM SERVICES

If you happen to open 'System' - 'Services' in the Mandrake Control Center, you see a lot of columns, starting with a more or less cryptic name for the service. The second column reads either 'stopped' or 'running', the third is a button which reveals some basic information. Then there's a check box labeled 'On boot' followed by two buttons: 'Start' and 'Stop'.

This layout describes in essence what you can do with a service: you can start or stop it and you can configure it to be 'started' automatically at boot time

But what ***is*** a service? In contrast to a program, *services* do not require user input (they 'run in the background') apart from starting or stopping them, and even this can be

automated.
There are two kinds of services:

→ Services which are started and keep running for the duration of a session (i.e. until the system gets shut down). In Unix slang these are also called *daemons* ('helpful spirit'). These are usually servers of some kind which are started and then wait for incoming requests, like a web server, a mail server, the printer service or a font server.

→ Services which are started, run and terminated when finished. These are usually scripts for system maintenance or for enabling certain features, like the 'num lock' script whose sole purpose is to turn on the num lock feature - i.e. being able to use the right-hand number pad on most keyboards for number input - during boot.

Apart from the Mandrake Control Center, there's a slew of other graphical configuration utilities you can use to configure services. Webmin and Linux conf both come with modules to do that. KDE and GNOME each offer there own brand of service configuration utility. Furthermore there's 'tksysv' and its console based parent 'ntsysv'. But the MCC module does what needs to be done, but maybe you like one of the other applications better. You don't need to be afraid of causing inconsistencies when using different utilities since they all use the same (command line) commands, 'service' and 'chkconfig'.

'service' and 'chkconfig'

The 'service' command, a simple shell script in '/sbin', is used to display the status of a service, to start, stop or restart it. This command takes two arguments, the name of the service (i.e. the name of the file in '/etc/init.d') and what should be done in regard to this service:

- # service *service_name* start
- # service *service_name* stop
- # service *service_name* restart
- # service *service_name* status

'restart' and 'status' are not supported by all service scripts.

'chkconfig' lists, adds, removes and configures services permanently. To have a service started automatically at boot time, you would use:

# chkconfig *service_name* on

To have it ***not*** started automatically:

# chkconfig *service_name* off

To list all available services and their current configuration:

```
# chkconfig --list
```

The output of this last command will become clearer to you when you've read the next section. More on this command can be found in man chkconfig.

Like their graphical counterparts, these commands require you to be 'root'. Nothing forces you to use them instead of MCC or the other utilities, I prefer them because I'm faster at typing a command than at clicking through a graphical interface ;-).

Advantages Of The Services Mechanism

Being able to control services has several advantages:

- **Reducing system load:**
  Although daemons are 'sleeping' most of time they nevertheless use up a certain amount of system memory. The 'service' interface allows you to start services on demand, for example you can start the printer daemon right before printing and stop it when finished.
- **Increasing system security:**
  Daemons are listening on certain ports for events. More daemons running mean more open ports which in turn provide more possible points of attack. On the other hand there are services which actively increase system security, like the 'bastille-firewall' service.
- **Avoiding reboots:**
  If you change the configuration of a daemon, the daemon usually has to be started to let the change take effect. If you install a package which contains a service, the service usually won't start right away but will be configured to be started automatically at boot time.
  By controlling services you can fulfill these tasks during runtime.

Shortening boot time:

A good chunk of the time your Linux system needs to boot is taken up by starting or running daemons and other services. If you configure your system to start only those services on boot you need immediately or all the time, you can reduce the boot up time considerably.

How Services Work

This section is intended for people who not only want to know what to do but also why things are done this way. You can live on Linux without this, but in my opinion it's more fun when you get a grasp of the concept behind the scenes.

Service Scripts

If you are curious, you might want to know now how the system knows which services are available. The service scripts are located in '/etc/init.d' ('/etc/rc.d/init.d' on older releases).
Graphical utilities like the Mandrake Control Center just assume that every script in this directory controls a service, so if you put a script there, it will appear on the Services module of the Mandrake Control Center and in similar utilities, too, and can also be handled directly via the commands 'service' and 'chkconfig'.

A service script contains the commands to at least start or stop a service. Have a look at a basic template for a service script:

```
#!/bin/sh

# chkconfig: runlevels order_number_start_link order_number_stop_link

# description: short description of service

. /etc/rc.d/init.d/functions

case "$1" in
  start)
        echo -n "Starting service: "
        command(s) to start service
        echo
        ;;
  stop)
        echo -n "Shutting down service: "
        command(s) to stop service
        echo
        ;;
  status)
        status service_name
        ;;
  *)
        echo "*** Usage: service_name {start|stop|status}"
exit 1
esac
exit 0
```

If you've already seen a shell script, it's pretty simple. 'chkconfig' and 'description' are explained in the next subsection. The 'functions' line is only needed here to have the 'status' command available. Then there's a 'case' fork which tells the script which commands to execute if the last argument to the 'service' command is either 'stop', 'start' or 'status'. The 'echo' lines provide some feedback, '*)' matches all cases in which the last argument isn't one of 'start', 'stop' or 'status' and thus invalid
Of course you have to make sure that *service_name* really *is* the name of the script and that the script has the executable bit set.

Annotated List of System Services

This list tries to cover all the scripts in '/etc/init.d'. It depends on your scale of installation how many of these services are available on your system.

In this list services are either 'optional', 'common' or 'essential'. 'Optional' means you can turn this service safely off without loosing vital functionality, 'common' means that this service isn't vital but usually enabled, and 'essential' means you should *not* turn it off, unless you know exactly what you are doing and why you are doing it.

acon
Needed for arabic languages to be displayed correctly.
Pertinence: Optional. Package: acon. Doc: Files in '/usr/share/doc/acon-[...]'

acpid
ACPI (Advanced Configuration Power Interface) is the successor to APM (Advanced Power Management). 'acpid' is maintained by the ACPI4Linux project. Since essential functions like 'suspend' and 'resume' haven't been implemented yet, 'apmd' is still used as the default power management service in Mandrake.
Pertinence: Optional. Package: acpid. Doc: Linux ACPI HOWTO

adsl
Control script for ADSL (Asyncronous Digital Subscriber Line) connections via PPPoE (Point-to-Point Protocol over Ethernet).
Pertinence: Optional. Package: rp-pppoe. Doc: `man pppoe`, `man pppoe.conf`

alsa
This starts and stops the ALSA (Advanced Linux Sound Architecture) sound driver. If you don't want sound (or your card uses an OSS driver), turn it off.
Pertinence: Optional. Package: initscripts.

amd
The Automounter Daemon. Useful for automatically mounting (hey!) network file systems or removable media. Since removable media are handled by 'supermount' in Mandrake and 'amd' does have its handling quirks, you will possibly only need it for mounting network shares (NFS and the like). Do not run this if you don't need it as it poses a potential security hole.
Pertinence: Optional. Package: am-utils. Doc: MU on 'automount', man pages.

anacron
The cousin of the 'cron'-daemon. 'cron' runs tasks like system maintenance at certain times, but skips them if the system isn't running at that time. That's where 'anacron' comes in: it checks delayed 'cron'-tasks at boot-time and executes them. If your machine doesn't run all the time, you should leave it enabled.
Pertinence: Optional. Package: anacron. Doc: `man anacron`, MUO on using anacron

apcupsd
apcupsd manages UPS (Uninterruptable Power Supply) hardware manufactured by APC (American Power Conversion).
Pertinence: Optional. Package: apcupsd. Doc: The APCUPSD Users Manual

apmd
The Advanced Power Management BIOS Daemon. Usually you will only need it if your computer runs on battery, i.e. a laptop. Some laptop BIOSes do not take kindly to apmd, causing massive installation problems.
Pertinence: Optional. Package: apmd. Doc: `man apmd`

arpwatch
Keeps track of Ethernet/IP address pairings (no, I don't know what's that good for either).
Pertinence: Optional. Package: arpwatch. Doc: `man arpsnmp`.

atd
The At Daemon. Manages scheduled ('at a certain time') jobs. Related to 'crond'.
Pertinence: Optional. Package: at. Doc: `man atd`, `man at`, MUO article on 'at'.
auth2.init, codasrv.init, update.init, venus.init

autofs
Controls the automount daemon (amd). Usually not enabled. You might need it if you want to mount network-shares automatically.
Pertinence: Optional. Doc: `man autofs`, `man automount`.

bayonne
Bayonne provides a telephony application server.
Pertinence: Optional. Package: bayonne. Doc: Bayonne User Manual.

cfengine The Configuration Engine provides software agents and a language for central configuration and administration of large scale networks.
Pertinence: Optional. Package: cfengine.

chronyd
chronyd can keep your system's time in step with the true time or keep a network of computers in time sync with each other.
Pertinence: Optional. Package: chrony. Doc: FAQ, `info chrony`

crond
The Cron Daemon. Manages repeated tasks ('chronological'). Related to 'atd'.
Pertinence: Essential. Package: vixie-cron. Doc: `man crond`, `man cron`, MUO article on 'cron'

cups

CUPS is the standard printing service on Mandrake Linux.
Pertinence: Optional. Package: cups. Doc: /usr/share/doc/cups/documentation.html, MUO on using CUPS

dhcp-relay
You will need if your DHCP (Dynamic Host Configuration Protocol) server is located in another subnet than its clients.
Pertinence: Optional. Package: dhcp-relay. Doc: `man dhcrelay`
dhcp-server

fcron
Fcron is a replacement for 'cron' as well as 'anacron'. Notice that ML 8.1 comes with an outdated version which has security issues. Get 2.0 from the Fcron website.
Pertinence: Optional. Package: fcron. Doc: `man fcron`, `man fcrontab`

fetchmail
Daemon for the Fetchmail mail retriever.
Pertinence: Optional. Package: fetchmail-daemon. Doc: MUO on configuring fetchmail, `man fetchmail`

functions
Contains code blocks to be used by other service scripts.
Pertinence: Essential. Package: initscripts. Doc: Read the script ;-)

gated
GateD is a network routing daemon.
Pertinence: Optional. Package: gated. Doc: `man gated`

gdips
GnuDIP can be used by an Internet provider to assign static DNS names to its clients even if those clients have their IPs dynamically assigned.
Pertinence: Optional. Package: gnudip-server. Doc: Files in /usr/share/doc/gnudip-server-[...]

halt
The script executed when the system gets halted or rebooted. This script is not meant to be executed directly from the commandline.
Pertinence: Essential. Package: initscripts. Doc: Read script.

hpoj
Script for the CUPS HP OfficeJet printer / scanner driver.
Pertinence: optional. Package: hpoj. Doc: hpoj documentation index

httpd
The daemon necessary to run the Apache web-server. In ML, it runs as a standalone service and not via '(x)inetd'. If you do not intend to run a web-server, turn it off: it uses a considerable amount of system resources (more than 15 MB of system memory) and

makes your box vulnerable to outside attacks if not configured properly (via '/etc/httpd/conf/httpd.conf').
Pertinence: Optional. Package: apache-conf. Doc: `man httpd`.

ibod IBOD is the 'ISDN Bandwidth On Demand Daemon'. It supports opening or closing a second B-channel automatically upon a certain amount of traffic.
Pertinence: Optional. Package: ibod. Doc: `man ibod`.

innd
Control script for the InterNetNews Usenet server.
Pertinence: Optional. Package: inn. Doc: lots ;-) Some 30 man pages and an FAQ in '/usr/share/doc/inn-[...]/faq'

ipchains
'ipchains' is the standardfirewalling method in Linux kernel 2.2 based systems, Linux kernel 2.4 (Mandrake Linux 8.x) uses 'iptables' instead. You'll need this if you want to keep on using your old (2.2) firewall rules and software.
Pertinence: Optional. Package: ipchains. Doc: `man ipchains`, IPCHAINS-HOWTO

ippl
ippl (IP Protocols Logger) logs incoming network traffic.
Pertinence: Optional. Package: ippl. Doc: `man ippl`

ipsec
Part of Linux FreeS/WAN, an implementation of the IPSEC (Internet Protocol SECurity) protocol. IPSEC allows you to connect trusted networks via untrusted ones using a technique called 'tunneling' (in short: all the traffic between the trusted networks gets encrypted and decrypted automatically).
Pertinence: Optional. Package: freeswan. Doc: lots

isdn4linux
For users of ISDN-*cards* (terminal-adapters are handled like modems). Notice that you might still have to configure this service, read '/usr/doc/isdn4net-[...]/doc/INSTALL.quick' for more. In ML 7.2 and later, use the 'draknet' utility.
Pertinence: Optional. Package: isdn4net. Doc: Files in '/usr/share/doc/isdn4net-[...]/', MUO on configuring ISDN

jabber, jabber-icq
Control scripts for the Open Source Instant Messaging server Jabber. You'll need jabber-icq if you want to allow clients to use the ICQ service.
Pertinence: Optional. Package: jabber, jabber-icq. Doc: docs.jabber.org
kadmin, kprop, krb524, krb5kdc, krb5server
Control scripts for a Kerberos 5 server. Kerberos is a network authentication protocol. In order for clients to access a Kerberos server, they need special client software (like 'ftp-client-krb5').
Pertinence: Optional. Package: krb5-server. Doc: `info krb5-admin`

keytable Not a service in the strict sense of the word. The 'keytable' script loads the selected console keyboard map as set in '/etc/sysconfig/keyboard' (variable KEYTABLE).
Pertinence: Common. Package: console-tools. Doc: `man loadkeys`

killall
Not a service, but a mere short helper script to stop renitent services.
Pertinence: Common. Package: initscripts. Doc: None

kudzu
Detects and configures new or changed hardware during boot. You can turn it off and your box will boot faster. You can also start kudzu during normal system operation to configure new hardware.
Pertinence: Common. Package: kudzu. Doc: `man kudzu`

ldap
LDAP is short for Lightweight Directory Access Protocol, a central network service for information stored in databases ('directories'). This script is part of the packages openldap1 and openldap-servers, which are maintained by the OpenLDAP project

linuxconf
Startscript for the Linuxconf central administration system.
Pertinence: Common. Package: linuxconf. Doc: '/usr/lib/linuxconf/help.[language-code]', online help system.

mandrake_firsttime
Not a service. Determines which commands should be run the first time the system is booted.
Pertinence: Common. Package: initscripts. Doc: Read comments in the script

mon
mon is a resource monitor (from networking to room temperature).
Pertinence: Optional. Package: mon. Doc: Files in '/usr/share/doc/mon-[...]'

mosix
MOSIX extends the Linux kernel by support for scalable cluster computing.
Pertinence: Optional. Package: mosix-utils. Doc: `man mosix`

mysql
MySQL (SQL = Structured Query Language) provides a database.
Pertinence: Optional. Package: MySQL. Doc: 'manual.html' in '/usr/share/doc/MySQL-[...]'

named

Part of BIND (Berkeley Internet Name Domain), the standard Domain Name Server (DNS) software on Linux. A name server maps IP addresses to machine names. Pertinence: Optional. Package: bind. Doc: Files in '/usr/share/doc/bind-[...], `man named.conf` and others

network
Not a service. Activates all network interfaces at boot time (or whenever invoked by the 'service' command) by calling the scripts in '/etc/sysconfig/network-scripts'. Pertinence: Common. Package: initscripts. Doc: `man ifconfig` and the appropriate documentation for the initiated interfaces

numlock
No service. 'Locks' the NumLock key at boot, thus making it possible to use the number block on most keyboards to type in numbers. This can get pretty funny when enabled on laptops ...
Pertinence: Common. Package: numlock. Doc: `man enable_X11_numlock`tice that there's an administrator account with a default password you should change. Pertinence: Optional. Package: opennap. Doc: 'manual.html' in '/usr/share/doc/opennap-[...]'

pcmcia
Part of the Card Services for Linux Project software which supports PCMCIA (PC-Memory Card International Association) cards frequently used in laptops. Pertinence: Optional. Package: kernel-pcmcia-cs. Doc: `man pcmcia`, Linux PCMCIA HOWTO

portmap
Security tool needed for Remote Procedure Calls, especially in NIS and NFS connections.Pertinence: Optional. Package: portmap. Doc: `man portmap`

postfix
The standard mail server software in Mandrake Linux, a replacement for sendmail. You will need it when your mail reader can't get or send its mail on its own or if you want to provide mail services for a network.
Pertinence: Optional. Package: postfix. Doc: `man postfix`, '/usr/doc/postfix-[version]/html/' MUO on configuring PostFix

postgresql
PostgreSQL provides a database server similar to MySQL. Pertinence: Optional. Package: postgresql-server. Doc: In package postgresql-docs
prelude
Prelude is a sophisticated Intrusion Detection System (IDS). Pertinence: Optional. Package: prelude. Doc: In package 'prelude-doc'

proftpd Control script for the standard Mandrake Linux FTP server, ProFTPd. Pertinence: Optional. Package: proftpd. Doc: on project home page

psacct
Allows monitoring process activities.
Pertinence: Optional. Package: psacct. Doc: `info accounting`

rawdevices
No service. "This scripts assigns raw devices to block devices (such as hard drive partitions)." Useful for databases (or so I'm told).
Pertinence: Common. Package: initscripts. Doc: Example in '/etc/sysconfig/rawdevices'

routed
'routed' is the Network Routing Daemon. You will need this if your machine acts as a router (obviously) for other machines in your network (e.g. as a gateway into another network like the Internet).
Pertinence: Optional. Package: routed. Doc: `man routed`

rstatd, rusersd, rwalld, rwhod
Provide access to information (status, users) or services (send messages) useful when maintaining a multiuser network.
Pertinence: Optional. Package: rusers, rwall, rwho. Doc: respective man pages

sendmail
The standard Internet mail server. In Mandrake Linux now replaced by PostFix (but still part of the distribution). If your mail client is capable of sending and receiving mail itself (e.g. Netscape Mail, kmail, Pine), you don't need it.
Pertinence: Optional. Package: sendmail. Doc: `man sendmail`

sensors
This skript starts or stops 'sensord', a hardware monitoring daemon and part of the lm_sensors hardware monitoring software.
Pertinence: Optional. Package: lm_utils. Doc: `man sensors`

shorewall
Shorewall is a firewall / masquerader to be used on single systems (i.e. a desktop firewall).
Pertinence: Optional. Package: shorewall. Doc: Files in '/usr/share/doc/shorewall-[...]/documentation'

sound
No service. Starts / stops sound device and loads / saves mixer settings.
Pertinence: Common. Package: initscripts. Doc: None

syslog

System Message Logger, i.e. it writes system or service messages to so called log files ('/var/log/*').
Pertinence: Essential. Package: sysklogd. Doc: `man sysklogd`

snmpd
Control script for the project formerly known as 'ucd-snmp', now as NET-SNMP. It is an implementation of the Simple Network Management Protocol (nomen est omen). If you know more, you know more than me.
Pertinence: Optional. Package: ucd-snmp. Doc: `man snmpd` and lost more. Also check out the project homepage

ups
Part of the former 'smartupstools', now called NUT (Network UPS Tools). This package allows the control of Uninterruptable Power Supplies over a network (you might have guessed that).
Pertinence: Optional. Package: smartupstools. Doc: Files in '/usr/share/doc/smartupstools-[...]'

usb
No service. When called (usually during boot) this script starts or stops USB (Universal Serial Bus) devices by loading or unloading the appropriate drivers (kernel modules). You are advised *not* to run this script during system operation.
Pertinence: Common. Package: initscripts. Doc: Comments in script, also have a look at '/etc/sysconfig/usb'

vncserver
Part of the Virtual Network Computing software. VNC allows you to remotely display / view / manage the desktop of another machine. It is platform-independent.
Pertinence: Optional. Package: vnc-server. Doc: in package 'vnc-doc'

vrrpd
Control script for the Virtual Router Redundancy Protocol Daemon, used to elect a master server on a local network and provide fallback in case the master fails.
Pertinence: Optional. Package: vrrpd. Doc: `man vrrpd` watchdog 'watchdog' monitors a machine and reboots it if a system freeze is detected (sounds funny, but works).
Pertinence: Optional. Package: watchdog. Doc: `man watchdog`

webmin
Webmin is a web based system administration suite (an alternative to Linuxconf).
Pertinence: Optional. Package: webmin. Doc: on project home page and inline

wine
WINE tries to build a Windows compatibility layer software (i.e. it intends to enable you to run Windows programs on Linux). This script enables users to run Windows applications by just clicking on them (well, in theory at least ...)
Pertinence: Optional. Package: wine. Doc: Files in '/usr/share/doc/wine-[...]/wine-doc'

Xfs
X Font Server. In the standard X configuration of Mandrake Linux, X won't run without a working font server.
Pertinence: Common. Package: XFree86-xfs. Doc: `man xfs`

xinetd
xinetd controls, logs or redirects accesses to other servers running on the same machine. It replaces inetd.
Pertinence: Optional. Package: xinetd. Doc: MUO article on xinetd, `man xinetd`

ypbind
NIS Binder. Only needed if your computer is part of a NIS (Network Information Service) domain (yp for its old name 'yellow pages').
Pertinence: Optional. Package: ypbind. Doc: `man ypbind`

yppasswdd, ypserv
scripts to control an NIS server.
Pertinence: Optional. Package: ypserv. Doc: `man ypserv`

PRACTICAL-5

OBJECTIVE : Linux Basic commands, pipe & filter commands.

An A-Z Index of the Linux command

alias	Create an alias
apropos	Search Help manual pages (man -k)
awk	Find and Replace text, database sort/validate/index
break	Exit from a loop
builtin	Run a shell builtin
bzip2	Compress or decompress named file(s)

cal	Display a calendar
case	Conditionally perform a command
cat	Display the contents of a file
cd	Change Directory
cfdisk	Partition table manipulator for Linux
chgrp	Change group ownership
chmod	Change access permissions
chown	Change file owner and group
chroot	Run a command with a different root directory
cksum	Print CRC checksum and byte counts
clear	Clear terminal screen
cmp	Compare two files
comm	Compare two sorted files line by line
command	Run a command - ignoring shell functions
continue	Resume the next iteration of a loop
cp	Copy one or more files to another location
cron	Daemon to execute scheduled commands
crontab	Schedule a command to run at a later time
csplit	Split a file into context-determined pieces
cut	Divide a file into several parts

date	Display or change the date & time
dc	Desk Calculator
dd	Data Dump - Convert and copy a file
declare	Declare variables and give them attributes
df	Display free disk space
diff	Display the differences between two files
diff3	Show differences among three files
dir	Briefly list directory contents
dircolors	Colour setup for `ls'
dirname	Convert a full pathname to just a path
dirs	Display list of remembered directories
du	Estimate file space usage

echo	Display message on screen
egrep	Search file(s) for lines that match an extended expression
eject	Eject removable media
enable	Enable and disable builtin shell commands
env	Environment variables
ethtool	Ethernet card settings
eval	Evaluate several commands/arguments
exec	Execute a command
exit	Exit the shell
expand	Convert tabs to spaces
export	Set an environment variable
expr	Evaluate expressions

false	Do nothing, unsuccessfully
fdformat	Low-level format a floppy disk
fdisk	Partition table manipulator for Linux

fgrep	Search file(s) for lines that match a fixed string
file	Determine file type
find	Search for files that meet a desired criteria
fmt	Reformat paragraph text
fold	Wrap text to fit a specified width.
for	Expand *words*, and execute *commands*
format	Format disks or tapes
free	Display memory usage
fsck	File system consistency check and repair
ftp	File Transfer Protocol
function	Define Function Macros
gawk	Find and Replace text within file(s)
getopts	Parse positional parameters
grep	Search file(s) for lines that match a given pattern
groups	Print group names a user is in
gzip	Compress or decompress named file(s)
hash	Remember the full pathname of a name argument
head	Output the first part of file(s)
history	Command History
hostname	Print or set system name
id	Print user and group id's
if	Conditionally perform a command
import	Capture an X server screen and save the image to file
install	Copy files and set attributes
join	Join lines on a common field
kill	Stop a process from running
less	Display output one screen at a time
let	Perform arithmetic on shell variables
ln	Make links between files
local	Create variables
locate	Find files
logname	Print current login name
logout	Exit a login shell
look	Display lines beginning with a given string
lpc	Line printer control program
lpr	Off line print
lprint	Print a file
lprintd	Abort a print job
lprintq	List the print queue
lprm	Remove jobs from the print queue
ls	List information about file(s)
lsof	List open files
make	Recompile a group of programs
man	Help manual
mkdir	Create new folder(s)
mkfifo	Make FIFOs (named pipes)
mkisofs	Create an hybrid ISO9660/JOLIET/HFS filesystem
mknod	Make block or character special files
more	Display output one screen at a time
mount	Mount a file system
mtools	Manipulate MS-DOS files

mv	Move or rename files or directories
netstat	Networking information
nice	Set the priority of a command or job
nl	Number lines and write files
nohup	Run a command immune to hangups
passwd	Modify a user password
paste	Merge lines of files
pathchk	Check file name portability
ping	Test a network connection
popd	Restore the previous value of the current directory
pr	Prepare files for printing
printcap	Printer capability database
printenv	Print environment variables
printf	Format and print data
ps	Process status
pushd	Save and then change the current directory
pwd	Print Working Directory
quota	Display disk usage and limits
quotacheck	Scan a file system for disk usage
quotactl	Set disk quotas
ram	ram disk device
rcp	Copy files between two machines.
read	read a line from standard input
readonly	Mark variables/functions as readonly
remsync	Synchronize remote files via email
return	Exit a shell function
rm	Remove files
rmdir	Remove folder(s)
rsync	Remote file copy (Synchronize file trees)
screen	Terminal window manager
scp	Secure copy (remote file copy)
sdiff	Merge two files interactively
sed	Stream Editor
select	Accept keyboard input
seq	Print numeric sequences
set	Manipulate shell variables and functions
sftp	Secure File Transfer Program
shift	Shift positional parameters
shopt	Shell Options
shutdown	Shutdown or restart linux
sleep	Delay for a specified time
sort	Sort text files
source	Run commands from a file \`.'
split	Split a file into fixed-size pieces
ssh	Secure Shell client (remote login program)
strace	Trace system calls and signals
su	Substitute user identity
sum	Print a checksum for a file
symlink	Make a new name for a file
sync	Synchronize data on disk with memory
tail	Output the last part of files
tar	Tape ARchiver

tee	Redirect output to multiple files
test	Evaluate a conditional expression
time	Measure Program running time
times	User and system times
touch	Change file timestamps
top	List processes running on the system
traceroute	Trace Route to Host
trap	Run a command when a signal is set(bourne)
tr	Translate, squeeze, and/or delete characters
true	Do nothing, successfully
tsort	Topological sort
tty	Print filename of terminal on stdin
type	Describe a command

ulimit	Limit user resources
umask	Users file creation mask
umount	Unmount a device
unalias	Remove an alias
uname	Print system information
unexpand	Convert spaces to tabs
uniq	Uniquify files
units	Convert units from one scale to another
unset	Remove variable or function names
unshar	Unpack shell archive scripts
until	Execute commands (until error)
useradd	Create new user account
usermod	Modify user account
users	List users currently logged in
uuencode	Encode a binary file
uudecode	Decode a file created by uuencode

v	Verbosely list directory contents (`ls -l -b')
vdir	Verbosely list directory contents (`ls -l -b')
vi	Text Editor

watch	Execute/display a program periodically
wc	Print byte, word, and line counts
whereis	Report all known instances of a command
which	Locate a program file in the user's path.
while	Execute commands
who	Print all usernames currently logged in
whoami	Print the current user id and name (`id -un')
Wget	Retrieve web pages or files via HTTP, HTTPS or FTP

xargs	Execute utility, passing constructed argument list(s)
yes	Print a string until interrupted

.period	Run commands from a file
###	Comment / Remark

Details of some useful Linux Commands

break

Exit from a for, while, until, or select loop

SYNTAX

    break [*n*]

If *n* is supplied, the *n*th enclosing loop is exited. *n* must be greater than or equal to 1.

The return status is zero unless *n* is not greater than or equal to 1.

cal

Display a calendar

SYNTAX

    cal [-mjy] [[*month*] *year*]

options:

-m   Display monday as the first day of the week.

case

Conditionally perform a command, case will selectively execute the *command-list* corresponding to the first *pattern* that matches *word*.

SYNTAX

    case *word* in [[(] *pattern* [| *pattern*]...) *command-list* ;;]... esac

The `|' 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 `;;'.

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 `;;'. The first pattern that matches determines the command-list that is executed.

Here is an example using case in a script that could be used to describe one interesting feature of an animal:

```
echo -n "Enter the name of an animal: "
read ANIMAL
echo -n "The $ANIMAL has "
case $ANIMAL in
  horse | dog | cat) echo -n "four";;
  man | kangaroo ) echo -n "two";;
  *) echo -n "an unknown number of";;
esac
echo " legs."
```

The return status is zero if no *pattern* is matched. Otherwise, the return status is the exit status of the *command-list* executed.

-j Display julian dates (days one-based, numbered from January 1).

-y Display a calendar for the current year.

A single parameter specifies the 4 digit year (1 - 9999) to be displayed.

Two parameters denote the Month (1 - 12) and Year (1 - 9999).

If arguments are not specified, the current month is displayed.

A year starts on 01 Jan.

<u>cat</u>

Display the contents of a file (concatenate)

SYNTAX

```
    cat [Options] [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.

Examples:

Display a file
```
$ cat myfile
```

Concatenate two files:
```
$ cat file1 file2 >> file3.dat
```

Put the contents of a file into a variable
```
$my_variable=`cat $myfile.txt`
```

<u>cd</u>

Change Directory - change the current working directory to a specific Folder.

SYNTAX
    cd [-LP] [*directory*]

OPTIONS
   -P : Do not follow symbolic links
   -L : Follow symbolic links (default)
If *directory* is not given, the value of the HOME shell variable is used.

If the shell variable CDPATH exists, it is used as a search path.
If *directory* begins with a slash, CDPATH is not used.

If *directory* is `-', this will change to the previous directory location (equivalent to $OLDPWD).

The return status is zero if the directory is successfully changed, non-zero otherwise.
Examples

move to the sybase folder
```
$ cd  /usr/local/sybase
$ pwd
/usr/local/sybase
```

Change to another folder
```
$ cd /var/log
$ pwd
/var/log
```

Quickly get back

```
$ cd –
$ pwd
/usr/local/sybase
```

move up one folder

```
$cd ..
$ pwd
/usr/local/
```

```
$ cd     (Back to your home folder)
```

cmp

Compare two files, and if they differ, tells the first byte and line number where they differ.

You can use the `cmp' command to show the offsets and line numbers where two files differ. `cmp' can also show all the characters that differ between the two files, side by side.

SYNTAX
    cmp *options... FromFile* [*ToFile*]

OPTIONS
    Multiple single letter options (unless they take an argument)
    can be combined into a single command line word:
    so `-cl' is equivalent to `-c -l'.

`-c'
   Print the differing characters. Display control characters as a
   `^' followed by a letter of the alphabet and precede characters
   that have the high bit set with `M-' (which stands for "meta").

`--ignore-initial=BYTES'
   Ignore any differences in the the first BYTES bytes of the input
   files. Treat files with fewer than BYTES bytes as if they are
   empty.

`-l'
   Print the (decimal) offsets and (octal) values of all differing
   bytes.

`--print-chars'
   Print the differing characters. Display control characters as a
   `^' followed by a letter of the alphabet and precede characters

that have the high bit set with `M-' (which stands for "meta").

`--quiet'
`-s'
`--silent'
Do not print anything; only return an exit status indicating whether the files differ.

`--verbose'
Print the (decimal) offsets and (octal) values of all differing bytes.

`-v'
`--version'
Output the version number of `cmp'.

The file name `-' is always the standard input. `cmp' also uses the standard input if one file name is omitted.

An exit status of 0 means no differences were found, 1 means some differences were found, and 2 means trouble.

Example

```
$ cmp tnsnames.ora tnsnames.old
```

cp

Copy one or more files to another location

Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

Syntax

cp [*options*]... *Source Dest*
cp [*options*]... *Source... Directory*

Key

-a, --archive	same as -dpR
-b, --backup	make backup before removal
-d, --no-dereference	preserve links
-f, --force	remove existing destinations, never prompt

```
  -i, --interactive           prompt before overwrite

  -l, --link                  link files instead of copying

  -p, --preserve              preserve file attributes if possible

  -P, --parents               append source path to DIRECTORY

  -r                          copy recursively, non-directories as files

      --sparse=WHEN           control creation of sparse files

  -R, --recursive             copy directories recursively

  -s, --symbolic-link         make symbolic links instead of copying

  -S, --suffix=SUFFIX         override the usual backup suffix

  -u, --update                copy only when the SOURCE file is newer
                              than the destination file or when the
                              destination file is missing

  -v, --verbose               explain what is being done

  -V, --version-control=WORD  override the usual version control

  -x, --one-file-system       stay on this file system
      --help                  display this help and exit
      --version               output version information and exit.
```

Example - copy home directory to floppy

```
$ cp -f /mnt/floppy/* /home/simon
```

<u>date</u>

Display or change the date

```
SYNTAX
      date [OPTION]... [+FORMAT]
   or:
      date [OPTION] [MMDDhhmm[[CC]YY][.ss]]
```

`date' with no arguments prints the current time and date, in the format of the `%c' directive (described below).

```
If given an argument that starts with a `+', `date' prints the
current time and date (or the time and date specified by the `--date'
option, see below) in the format defined by that argument, which is the
same as in the `strftime' function.  Except for directives, which start
with `%', characters in the format string are printed unchanged.  The
directives are described below.

OPTIONS
  -d, --date=STRING         display time described by STRING, not `now'
                          DATESTR can be in almost any common format.
                          It can contain month names, timezones, `am' and `pm',
                          `yesterday', `ago', `next', etc.

  -f, --file=DATEFILE       like --date once for each line of DATEFILE
                          If DATEFILE is `-', use standard input.  This is
                          useful when you have many dates to process,
                          because the system overhead of starting up the
                          `date' executable many times can be considerable.

  -I, --iso-8601[=TIMESPEC] output an ISO-8601 compliant date/time string., `%Y-%m-
%d'.
                          TIMESPEC=`date' (or missing) for date only,
                          `hours', `minutes', or `seconds' for date and
                          time to the indicated precision.
                          If showing any time terms, then include the time zone
                          using the format `%z'.  If `--utc' is also specified,
                          use `%Z' in place of `%z'.

  -r, --reference=FILE      display the last modification time of FILE

  -R, --rfc-822             output RFC-822 compliant date string

  -s, --set=STRING          set time described by STRING (see -d above)

  -u, --utc, --universal    print or set Coordinated Universal Time

      --help                display this help and exit
      --version             output version information and exit

FORMAT controls the output.  The only valid option for the second form
specifies Coordinated Universal Time.  Interpreted sequences are:

  %%   a literal %
  %a   locale's abbreviated weekday name (Sun..Sat)
  %A   locale's full weekday name, variable length (Sunday..Saturday)
  %b   locale's abbreviated month name (Jan..Dec)
  %B   locale's full month name, variable length (January..December)
  %c   locale's date and time (Sat Nov 04 12:02:33 EST 1989)
  %d   day of month (01..31)
  %D   date (mm/dd/yy)
  %e   day of month, blank padded ( 1..31)
  %h   same as %b, locale's abbreviated month name (Jan..Dec)
  %H   hour :24 hour(00..23)
  %I   hour :12 hour(01..12)
  %j   day of year (001..366)
  %k   hour :24 hour(00..23)
  %l   hour :12 hour(01..12)
  %m   month (01..12)
  %M   minute (00..59)
  %n   a newline
  %p   locale's AM or PM
  %r   time, 12-hour (hh:mm:ss [AP]M)
  %s   seconds since 00:00:00, Jan 1, 1970 (a GNU extension)
       Note that this value is defined by the localtime system
       call.  It isn't changed by the `--date' option.
  %S   second (00..60)
  %t   a horizontal tab
  %T   time, 24-hour (hh:mm:ss)
  %U   week number of year with Sunday as first day of week (00..53)
  %V   week number of year with Monday as first day of week (01..53)
       If the week containing January 1 has four or
       more days in the new year, then it is considered week 1;
       otherwise, it is week 53 of the previous year, and the next week
       is week 1. (See the ISO 8601: 1988 standard.)

  %w   day of week (0..6);  0 represents Sunday
  %W   week number of year with Monday as first day of week (00..53)
  %x   locale's date representation (mm/dd/yy)
  %X   locale's time representation (%H:%M:%S)
  %y   last two digits of year (00..99)
  %Y   year (1970...)
  %z   RFC-822 style numeric timezone (-0500) (a nonstandard extension)
       This value reflects the _current_ time zone.
       It isn't changed by the `--date' option.
  %Z   time zone (e.g., EDT), or nothing if no time zone is determinable
       This value reflects the _current_ time zone.
       It isn't changed by the `--date' option.

By default, date pads numeric fields with zeroes.  GNU date recognizes
the following modifiers between `%' and a numeric directive.

  `-' (hyphen) do not pad the field;  useful if the output is intended for
```

human consumption.

`_' (underscore) pad the field with spaces; useful if you need a fixed
number of characters in the output, but zeroes are too distracting.

The - and _ are GNU extensions.

Here is an example illustrating the differences:

```
date +%d/%m -d "Feb 1"
=> 01/02
date +%-d/%-m -d "Feb 1"
=> 1/2
date +%_d/%_m -d "Feb 1"
=>  1/ 2
```

Setting the time

If given an argument that does not start with `+', `date' sets the system clock to the time and date specified by that argument (as described below). You must have appropriate privileges to set the system clock. The `--date' and `--set' options may not be used with such an argument. The `--universal' option may be used with such an argument to indicate that the specified time and date are relative to Coordinated Universal Time rather than to the local time zone.

The argument must consist entirely of digits, which have the following meaning:

MM	month
DD	day within month
HH	hour
MM	minute
CC	first two digits of year (optional)
YY	last two digits of year (optional)
SS	second (optional)

The `--set' option also sets the system clock; see the next section.

Examples of `date'

* To print the date of the day before yesterday:
  `date --date='2 days ago'`

* To print the date of the day three months and one day hence:
  `date --date='3 months 1 day'`

* To print the day of year of Christmas in the current year:
```
date --date='25 Dec' +%j
```

* To print the current full month name and the day of the month:
```
date '+%B %d'
```

  But this may not be what you want because for the first nine days of the month, the `%d' expands to a zero-padded two-digit field, for example `date -d 1may '+%B %d'' will print `May 01'.

* Print a date without the leading zero for one-digit days of the month, you can use the (GNU extension) `-' modifier to suppress the padding altogether.
```
date -d=1may '+%B %-d'
```

* Print the current date and time in the format required by many non-GNU versions of `date' when setting the system clock:
```
date +%m%d%H%M%Y.%S
```

* Set the system date and time
```
date --set="2002-6-29 11:59 AM"
```

* Set the system clock forward by two minutes:
```
date --set='+2 minutes'
```

* Print the date in the format specified by RFC-822, use `date --rfc'. I just did and saw this:
```
Mon, 25 Mar 1996 23:34:17 -0600
```

* To convert a date string to the number of seconds since the epoch (which is 1970-01-01 00:00:00 UTC), use the `--date' option with the `%s' format. That can be useful in sorting and/or graphing and/or comparing data by date. The following command outputs the number of the seconds since the epoch for the time one second later than the epoch, but in time zone five hours later (Cambridge, Massachusetts), thus a total of five hours and one second after the epoch:
```
date --date='1970-01-01 00:00:01 UTC +5 hours' +%s
18001
```

  Suppose you had _not_ specified time zone information in the example above. Then, date would have used your computer's idea of the time zone when interpreting the string. Here's what you would get if you were in Greenwich, England:

```
# local time zone used
date --date='1970-01-01 00:00:01' +%s
1
```

* If you're sorting or graphing dated data, your raw date values may be represented as seconds since the epoch. But few people can look at the date `946684800' and casually note "Oh, that's the first second of the year 2000."

```
date --date='2000-01-01 UTC' +%s
946684800
```

To convert such an unwieldy number of seconds back to a more readable form, use a command like this:

```
date -d '1970-01-01 946684800 sec' +"%Y-%m-%d %T %z"
2000-01-01 00:00:00 +0000
```

echo

Display message on screen, writes each given STRING to standard output, with a space between each and a newline after the last one.

SYNTAX
echo [*options*]... [*string*]...

OPTIONS

`-n'
Do not output the trailing newline.

`-E'
Disable the interpretation of the following backslash-escaped characters

`-e'
Enable interpretation of the following backslash-escaped characters in each STRING:

`\a'	alert (bell)
`\b'	backspace
`\c'	suppress trailing newline

`\e'	escape
`\f'	form feed
`\n'	new line
`\r'	carriage return
`\t'	horizontal tab
`\v'	vertical tab
`\\'	backslash

`\NNN'
  the character whose ASCII code is NNN (octal); if NNN is not
  a valid octal number, it is printed literally.

`\xnnn'
  the character whose ASCII code is the hexadecimal value
  nnn (one to three digits)

echo is a BASH built-in command

grep

Search file(s) for specific text.

SYNTAX
  grep <*options*> "*Search String*" [*filename*]

  grep <*options*> [-e *pattern*] [*file*...]

  grep <*options*> [-f *file*] [*file*...]

A simple example:
$grep “Needle in a Haystack” /etc/*

OPTIONS

-A *NUM*
--after-context=*NUM*
  (GNU Extension)
  Print *NUM* lines of trailing context after matching lines.

-a

--text
    (GNU Extension)
    Do not suppress output lines that contain binary data. Normally,
    if the first few bytes of a file indicate that the file contains
    binary data, grep outputs only a message saying that the file
    matches the pattern. This option causes grep to act as if the
    file is a text file, even if it would otherwise be treated as
    binary. _Warning:_ the result might be binary garbage printed to
    the terminal, which can have nasty side-effects if the terminal
    driver interprets some of it as commands.

-B *NUM*
--before-context=*NUM*
    (GNU Extension)
    Print NUM lines of leading context before matching lines.

-b
--byte-offset
    (GNU Extension)
    Print the byte offset within the input file before each line of
    output. When `grep' runs on MS-DOS or MS-Windows, the printed
    byte offsets depend on whether the `-u' (`--unix-byte-offsets')
    option is used; see below.

-C *NUM*
--context=[*NUM*]
    (GNU Extension)
    Print NUM lines (default 2) of output context.

`-c'
`--count'
    Suppress normal output; instead print a count of matching lines
    for each input file. With the `-v', `--invert-match' option,
    count non-matching lines.

-d *ACTION*
--directories=*ACTION*
    (GNU Extension)
    If an input file is a directory, use ACTION to process it. By
    default, ACTION is `read', which means that directories are read
    just as if they were ordinary files (some operating systems and
    filesystems disallow this, and will cause `grep' to print error
    messages for every directory). If ACTION is `skip', directories
    are silently skipped. If ACTION is `recurse', `grep' reads all
    files under each directory, recursively; this is equivalent to the
    `-r' option.

`-e *PATTERN'*
`--regexp=*PATTERN*'
    Use *PATTERN* as the pattern; useful to protect patterns beginning
    with a `-'.

`-f *FILE*'
`--file=*FILE*'
    Obtain patterns from *FILE*, one per line. The empty file contains
    zero patterns, and therefore matches nothing.

-H
--with-filename
    (GNU Extension)
    Print the filename for each match.

-h
--no-filename
    (GNU Extension)
    Suppress the prefixing of filenames on output when multiple files
    are searched.

--help
    (GNU Extension)
    Print a usage message briefly summarizing these command-line
    options and the bug-reporting address, then exit.

`-i'
`--ignore-case'
    Ignore case distinctions in both the pattern and the input files.

-L
--files-without-match
    (GNU Extension)
    Suppress normal output; instead print the name of each input file
    from which no output would normally have been printed. The
    scanning of every file will stop on the first match.

`-l'
`--files-with-matches'
    Suppress normal output; instead print the name of each input file
    from which output would normally have been printed. The scanning
    of every file will stop on the first match.

--mmap
    (GNU Extension)

If possible, ue the `mmap' system call to read input, instead of
the default `read' system call.  In some situations, `--mmap'
yields better performance. However, `--mmap' can cause undefined
behavior (including core dumps) if an input file shrinks while
`grep' is operating, or if an I/O error occurs.

`-n'
`--line-number'
Prefix each line of output with the line number within its input
file.

-*NUM*
(GNU Extension)
Same as `--context=*NUM*' lines of leading and trailing context.
However, grep will never print any given line more than once.

`-q'
`--quiet'
`--silent'
Quiet; suppress normal output. The scanning of every file will
stop on the first match. Also see the `-s' or `--no-messages'
option.

-r
--recursive
(GNU Extension)
For each directory mentioned in the command line, read and process
all files in that directory, recursively. This is the same as the
`-d recurse' option.

`-s'
`--no-messages'
Suppress error messages about nonexistent or unreadable files.
Portability note: unlike GNU `grep', traditional `grep' did not
conform to POSIX.2, because traditional `grep' lacked a `-q'
option and its `-s' option behaved like GNU `grep''s `-q' option.
Shell scripts intended to be portable to traditional `grep' should
avoid both `-q' and `-s' and should redirect output to `/dev/null'
instead.

-U
--binary
(GNU Extension)
Treat the file(s) as binary. By default, under MS-DOS and
MS-Windows, `grep' guesses the file type by looking at the
contents of the first 32kB read from the file. If `grep' decides

the file is a text file, it strips the `CR' characters from the
original file contents (to make regular expressions with `^' and
`$' work correctly).  Specifying `-U' overrules this guesswork,
causing all files to be read and passed to the matching mechanism
verbatim; if the file is a text file with `CR/LF' pairs at the end
of each line, this will cause some regular expressions to fail.
This option has no effect on platforms other than MS-DOS and
MS-Windows.

-u
--unix-byte-offsets
(GNU Extension)
Report Unix-style byte offsets. This switch causes `grep' to
report byte offsets as if the file were Unix style text file,
i.e., the byte offsets ignore the `CR' characters which were
stripped. This will produce results identical to running `grep' on
a Unix machine. This option has no effect unless `-b' option is
also used; it has no effect on platforms other than MS-DOS and
MS-Windows.

`-v'
`--invert-match'
Invert the sense of matching, to select non-matching lines.

-V
--version
(GNU Extension)
Print the version number of `grep' to the standard output stream.
This version number should be included in all bug reports.

-w
--word-regexp
(GNU Extension)
Select only those lines containing matches that form whole words.
The test is that the matching substring must either be at the
beginning of the line, or preceded by a non-word constituent
character. Similarly, it must be either at the end of the line or
followed by a non-word constituent character. Word-constituent
characters are letters, digits, and the underscore.

`-x'
`--line-regexp'
Select only those matches that exactly match the whole line.

-Z
--null

    (GNU Extension)
    Output a zero byte (the ASCII `NUL' character) instead of the
    character that normally follows a file name. For example, `grep
    -lZ' outputs a zero byte after each file name instead of the usual
    newline. This option makes the output unambiguous, even in the
    presence of file names containing unusual characters like
    newlines. This option can be used with commands like `find
    -print0', `perl -0', `sort -z', and `xargs -0' to process
    arbitrary file names, even those that contain newline characters.

-z
--null-data
    (GNU Extension)
    Treat the input as a set of lines, each terminated by a zero byte
    (the ASCII `NUL' character) instead of a newline.  Like the `-Z'
    or `--null' option, this option can be used with commands like
    `sort -z' to process arbitrary file names.

kill

Stop a process from running, either via a signal or forced termination.

SYNTAX
      kill [-s *sigspec*] [-n *signum*] [-*sigspec*] *jobspec* or *pid*
      kill -l [*exit_status*]

key
   -l : List the signal names
   -s : Send a specific signal
   -n : Send a specific signal number

Send a signal specified by *sigspec* or *signum* to the process named by job specification *jobspec* or process ID *pid*.

sigspec is either a signal name such as SIGINT (with or without the SIG prefix) or a signal number; *signum* is a signal number.

If *sigspec* and *signum* are not present, SIGTERM is used (Terminate).

If any arguments are supplied when `-l' is given, the names of the signals corresponding to the arguments are listed, and the return status is zero. *exit_status* is a number specifying a signal number or the exit status of a process terminated by a signal.

The return status is zero if at least one signal was successfully sent, or non-zero if an error occurs or an invalid option is encountered.

List the running process

```
$ ps
PID TTY TIME CMD
1293 pts/5 00:00:00 MyProgram
```

Then Kill it

```
$ kill 1293
[2]+ Terminated MyProgram
```

To run a command and then kill it after 5 seconds:

```
   my_command & sleep 5
   kill -0 $! && kill $!
```

ls

List information about FILEs, by default the current directory.

SYNTAX

      ls [*Options*]... [*File*]...

KEY

      Sort entries alphabetically if none of -cftuSUX nor --sort.

-a, --all	Do not hide entries starting with .
-A, --almost-all	Do not list implied . and ..
-b, --escape	Print octal escapes for nongraphic characters
--block-size=*SIZE*	Use *SIZE*-byte blocks
-B, --ignore-backups	Do not list implied entries ending with ~
-c	Sort by change time; with -l: show ctime
-C	List entries by columns
--color[=*WHEN*]	Control whether color is used to distinguish file types. WHEN may be `never', `always', or `auto'
-d, --directory	List directory entries instead of contents
-D, --dired	Generate output designed for Emacs' dired mode

-f	Do not sort, enable -aU, disable -lst
-F, --classify	Append indicator (one of */=@|) to entries
--format=*WORD*	Across -x, commas -m, horizontal -x, long -l, single-column -1, verbose -l, vertical -C
--full-time	List both full date and full time
-g	(ignored)
-G, --no-group	Inhibit display of group information
-h, --human-readable	Print sizes in human readable format (e.g., 1K 234M 2G)
-H, --si	Likewise, but use powers of 1000 not 1024
--indicator-style=*WORD*	Append indicator with style *WORD* to entry names: none (default), classify (-F), file-type (-p)
-i, --inode	Print index number of each file
-I, --ignore=*PATTERN*	Do not list implied entries matching shell *PATTERN*
-k, --kilobytes	Like --block-size=1024
-l	Use a long listing format
-L, --dereference	List entries pointed to by symbolic links
-m	Fill width with a comma separated list of entries
-n, --numeric-uid-gid	List numeric UIDs and GIDs instead of names
-N, --literal	Print raw entry names (don't treat e.g. control characters specially)
-o	Use long listing format without group info
-p, --file-type	Append indicator (one of /=@|) to entries
-q, --hide-control-chars	Print ? instead of non graphic characters
--show-control-chars	Show non graphic characters as-is (default)
-Q, --quote-name	Enclose entry names in double quotes
--quoting-style=*WORD*	Use quoting style *WORD* for entry names:

	literal, shell, shell-always, c, escape
-r, --reverse	Reverse order while sorting
-R, --recursive	List subdirectories recursively
-s, --size	Print size of each file, in blocks
-S	Sort by file size
--sort=*WORD*	time -t, version -v, status -c size -S, extension -X, none -U atime -u, access -u, use -u
--time=*WORD*	Show time as *WORD* instead of modification time: atime, access, use, ctime or status; also use this as a sort key if --sort=time
-t	sort by modification time
-T, --tabsize=*COLS*	assume tab stops at each *COLS* instead of 8
-u	sort by last access time; with -l: show atime
-U	do not sort; list entries in directory order
-v	sort by version
-w, --width=*COLS*	assume screen width instead of current value
-x	list entries by lines instead of by columns
-X	sort alphabetically by entry extension
-1	list one file per line
--help	display help and exit
--version	output version information and exit

The most common options are -a (all files) and -l (long or details)

When output to file the files are listed one per line.

By default, colour is not used to distinguish types of files. That is equivalent to using --color=none.

Using the --color option without the optional WHEN argument is equivalent to using --color=always.
With --color=auto, color codes are output only if standard output is connected to a terminal (tty).

mkdir

Create new folder(s), if they do not already exist.

SYNTAX

```
      mkdir [Options] folder...

      mkdir "Name with spaces"
```

OPTIONS

```
  -m, --mode=MODE   set permission mode (as in chmod), not rwxrwxrwx - umask
  -p, --parents     no error if existing, make parent directories as needed
      --verbose     print a message for each created directory
```

mkdir creates the standard entries . (dot) for the current folder
and .. (dot dot) for its parent

man / info / help

Display helpful information about commands.

Syntax

```
      man [-k] [command]

      man intro

      man bash

      info [command]

      help [-s] [command]
```

Options

```
      -s   Short usage synopsis, restricts the information displayed.

      -k   Search by command description rather than command name.

    intro  An overview of basic commands
```

Press <Space bar> to view the next page
Press <return> to view next line
Press <ctrl-C> to exit

For simplicity, this website includes both **internal** GNU bash commands and **external** unix commands in a single list. Many more commands are available and the man command will list the full details of these.

Internal means a command built into the shell, it's the shell that performs the action.
External means the shell will fork and execute an external program as a new subprocess. External commands are available when running any shell.

For example, the cd command is built-in. The ls command, is external.

The man command lists all the internal commands for bash under man bash

<u>more</u>

Display output one screen at a time, less provides *more* emulation and extensive enhancements.

SYNTAX
 more [-dlfpcsu] [-*num*] [+/ *pattern*] [+ *linenum*] [*file* ...]

OPTIONS
 Command line options are described below. Options are also taken from the environment variable MORE (make sure to precede them with a dash (``-")) but command line options will override them.

 -*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 contin ue, '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   Suppress underlining.

+/   The +/ option specifies a string that will be searched for before each file is displayed.

+*num* Start at line number num.

COMMANDS

Interactive commands for more are based on vi(1). Some commands may be preceeded by a decimal number, called k in the descriptions below. In the following descriptions, ^X means control-X.

h or ?   Help: display a summary of these commands. If you forget all the other commands, remember this one.

SPACE   Display next k lines of text. Defaults to current screen size.

z   Display next k lines of text. Defaults to current screen size. Argument becomes new default.

RETURN   Display next k lines of text. Defaults to 1. Argument becomes new default.

d or ^D   Scroll k lines. Default is current scroll size, initially 11. Argument becomes new default.

q or Q or INTERRUPT   Exits the more command.

s   Skip forward k lines of text. Defaults to 1.

f   Skip forward k screenfuls of text. Defaults to 1.

b or ^B   Skip backwards k screenfuls of text. Defaults to 1.

'   Go to place where previous search started.

=   Display current line number.

/*pattern*   Search for kth occurrence of regular expression. Defaults to 1.

n   Search for kth occurrence of last r.e. Defaults to 1.

! or :!	Execute in a subshell
v	Start up /usr/bin/vi at current line
^L	Redraw screen
:n	Go to kth next file. Defaults to 1.
:p	Go to kth previous file. Defaults to 1.
:f	Display current file name and line number
.	Repeat previous command

ENVIRONMENT
More utilizes the following environment variables, if they exist:

MORE	This variable may be set with favored options to more.

mv

Move or rename files or directories.

SYNTAX
mv [*options*]... *Source Dest*

mv [*options*]... *Source... Directory*

If the last argument names an existing directory, `mv' moves each other given file into a file with the same name in that directory. Otherwise, if only two files are given, it renames the first as the second. It is an error if the last argument is not a directory and more than two files are given.

OPTIONS

-b
--backup
Make a backup of each file that would otherwise be overwritten or removed.

-f
--force
Remove existing destination files and never prompt the user.

-i

--interactive
    Prompt whether to overwrite each existing destination file,
    regardless of its permissions. If the response does not begin
    with `y' or `Y', the file is skipped.

-S *SUFFIX*
--suffix=*SUFFIX*
    Append *SUFFIX* to each backup file made with `-b'.
    The backup suffix is ~, unless set with SIMPLE_BACKUP_SUFFIX.

-u
--update
    Do not move a nondirectory that has an existing destination with
    the same or newer modification time.

-v
--verbose
    Print the name of each file before moving it.

-V *METHOD*
--version-control=*METHOD*'
    Change the type of backups made with `-b'. METHOD can be:

t, numbered	make numbered backups
nil, existing	numbered if numbered backups exist, simple otherwise
never, simple	always make simple backups

--help	display help and exit
--version	output version information and exit

Examples

Rename the file apple as orange.doc:
```
mv apple orange.doc
```

Move orange.doc to the Documents folder:
```
mv orange.doc ~/Documents/orange.doc
```

Rename a bunch of file extensions
e.g. change *.txt into *.htm
```
  for f in *.txt; do mv ./"$f" "${f%txt}htm"; done
```

`mv' can move only regular files across filesystems.

If a destination file exists but is normally unwritable, standard input is a terminal, and the `-f' or `--force' option is not given, `mv' prompts the user for whether to replace the file.

(You might own the file, or have write permission on its directory.) If the response does not begin with `y' or `Y', the file is skipped.

ps

Process status, information about processes running in memory. If you want a repetitive update of this status, use top.

SYNTAX
  Display Process info
    ps *option(s)*

  List all the keyword options
    ps [-L]

OPTIONS

  This version of ps accepts 3 kinds of option:
   -Unix98 options may be grouped and must be preceeded by a dash.
   BSD options may be grouped and must not be used with a dash.
   --GNU long options are preceeded by two dashes.

  Options of different types may be freely mixed. The PS_PERSONALITY environment variable provides more detailed control of ps behavior.

  The Options below are listed side-by-side (unless there are differences).

SIMPLE PROCESS SELECTION

-A a	select all processes (including those of other users)
-a	select all with a tty except session leaders
-d	select all, but omit session leaders
-e	select all processes
g	really all, even group leaders (does nothing w/o SunOS settings)
-N	negate selection
r	restrict output to running processes
T	select all processes on this terminal
x	select processes without controlling ttys
--deselect	negate selection

PROCESS SELECTION BY LIST

-C	select by command name
-G	select by RGID (supports names)
-g	select by session leader OR by group name
--Group	select by real group name or ID

--group	select by effective group name or ID
-p p --pid	select by process ID (PID)
-s --sid	select by session ID
-t --tty	select by terminal (tty)
-u U	select by effective user ID (supports names)
-U	select by RUID (supports names)
--User	select by real user name or ID
--user	select by effective user name or ID
-*123*	implied --sid
123	implied --pid

OUTPUT FORMAT CONTROL

-c	Different scheduler info for -l option
-f	Full listing
-j j	Jobs format
-l l	Long format
-O O	Add the information associated with the space or comma separated list of keywords specified, after the **process ID**, in the default information display.
-o o	Display information associated with the space or comma separated list of keywords specified.
--format	user-defined format
s	display signal format
u	display user-oriented format
v	display virtual memory format
X	old Linux i386 register format
-y	do not show flags; show rss in place of addr

OUTPUT MODIFIERS

C	use raw CPU time for %CPU instead of decaying average
c	true command name
e	show environment after the command
f	ASCII-art process hierarchy (forest)
-H	show process hierarchy (forest)
h	do not print header lines (repeat header lines in BSD personality)
-m m	show all threads
-n	set namelist file
n	numeric output for WCHAN and USER
N	specify namelist file
O	sorting order (overloaded)
S	include some dead child process data (as a sum with the parent)
-w w	wide output
--cols	set screen width
--columns	set screen width

--forest	ASCII art process tree
--html	HTML escaped output
--headers	repeat header lines
--no-headers	print no header line at all
--lines	set screen height
--nul	unjustified output with NULs
--null	unjustified output with NULs
--rows	set screen height
--sort	specify sorting order
--width	set screen width
--zero	unjustified output with NULs

INFORMATION

-V V	print version
L	list all format specifiers
--help	print help message
--info	print debugging info
--version	print version

OBSOLETE

A	increase the argument space (DecUnix)
M	use alternate core (try -n or N instead)
W	get swap info from ... not /dev/drum (try -n or N instead)
k	use /vmcore as c-dumpfile (try -n or N instead)

<u>pwd</u>

Print Working Directory

SYNTAX

```
pwd [-LP]
```

OPTIONS (shell builtin)

-P : The pathname printed will not contain symbolic links.
-L : The pathname printed may contain symbolic links

The default action is to show the current folder as an absolute path.
All components of the path will be actual folder names - none will be symbolic links

<u>read</u>

Read a line from standard input

SYNTAX

read [-ers] [-a *aname*] [-p *prompt*] [-t *timeout*]

[-n *nchars*] [-d *delim*] [*name*...]

OPTIONS

-a *aname*
The words are assigned to sequential indices of the array variable aname,
starting at 0. All elements are removed from aname before the assignment.
Other name arguments are ignored.

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

-e
If the standard input is coming from a terminal, Readline is used
to obtain the line.

-n *nchars*
read returns after reading nchars characters rather
than waiting for a complete line of input.

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

-r
If this option is given, 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.

-t *timeout*
Cause read to time out and return failure if a complete line
of input is not read within timeout seconds. This option has no
effect if read is not reading input from the terminal or a pipe.

rm

Remove files (delete/unlink)

SYNTAX
rm [*options*]... *file*...

OPTIONS

```
  -d, --directory      unlink directory, even if non-empty (super-user only)

  -f, --force          ignore nonexistent files, never prompt

  -i, --interactive    prompt before any removal

  -r, -R, --recursive  remove the contents of directories recursively

  -v, --verbose        explain what is being done

      --help           display this help and exit

      --version        output version information and exit
```

To remove a file you must have write permission on the file and the folder where it is stored.

rm -rf will recursively remove folders and their contents

The OWNER of a file does not need rw permissions in order to rm it.

Undeletable files

The rm command accepts the `--' option which will cause it to stop processing flag options from that point forward. This allows the removal of file names that begin with a dash (`-').

```
rm -- -filename
```

Alternatively use an absolute or relative path reference.

```
rm /home/user/-filename
rm ./-filename
```

To delete a file with non-printable characters in the name: `bad file name' Use the shell wildcard "?" for each character

```
rm bad?file?name
```

rmdir

Remove directory, this command will only work if the folders are empty.

Syntax

```
rmdir [options]... folder(s)...
```

Options

```
     --ignore-fail-on-non-empty
               Ignore each failure that is solely because the
               directory is non-empty.
  -p, --parents   Remove explicit parent directories if being emptied
     --verbose   Output a diagnostic for every directory processed
     --help      Display help and exit
     --version   Output version information and exit
```

Example

Before deleting with a wildcard, it's wise to echo the files first:

```
$ echo elvis?costello*.mp3

$ rm elvis?costello*.mp3
```

VI Editor Commands

Switch to Text or Insert mode:

		Open line above cursor	**O**				
Insert text at beginning of line	**I**	Insert text at cursor	**i**	Insert text after cursor	**a**	Append text at line end	**A**
		Open line below cursor	**o**				

Switch to Command mode:

Switch to command mode	**<ESC>**

Cursor Movement (command mode):

		Scroll Backward 1 screen	**<ctrl>b**		
		Scroll Up 1/2 screen	**<ctrl>u**		
Go to beginning	**0**	Go to line *n*	***n*G**	Go to end of	**$**

of line				line	
		Scroll Down 1/2 screen	**<ctrl>d**	Go to line number ##	**:##**
		Scroll Forward 1 screen	**<ctrl>f**		
		Go to last line	**G**		
Scroll by sentence f/b	**( )**				
Scroll by word f/b	**w b**	Move left, down, up, right	**h j k l**	Left 6 chars	**6h**
		Directional Movement	**Arrow Keys**	Go to line #6	**6G**

Deleting text (command mode):

Change word	**cw**	Replace one character	**r**		
Delete word	**dw**	Delete text at cursor	**x**	Delete entire line (to buffer)	**dd**
		Delete current to end of line	**D**	Delete 5 lines (to buffer)	**5dd**
				Delete lines 5-10	**:5,10d**

Editing (command mode):

Copy line	**yy**	Copy *n* lines	***n*yy**	Copy lines 1-2/paste after 3	**:1,2t 3**
Paste above current line	**P**				
Paste below current line	**p**			Move lines 4-5/paste after 6	**:4,5m 6**
				Join previous line	**J**

Search backward for *string*	***?string***	Search forward for *string*	***/string***	Find next *string* occurrence	**n**
% (entire file) **s** (search and replace) /old text with new/ **c** (confirm) **g** (global - all)	**:%s/oldstring/newstring/cg**			Ignore case during search	**:set ic**
Repeat last command	**.**	Undo previous command	**u**	Undo all changes to line	**U**

Save and Quit (command mode):

Save changes to buffer	**:w**	Save changes and quit vi	**:wq**	Save file to new file	**:w file**
		Quit without saving	**:q!**	Save lines to new file	**:10,15w file**

<u>wc</u>

Print byte, word, and line counts, count the number of bytes, whitespace-separated words, and newlines in each given FILE, or standard input if none are given or for a FILE of `-'.

SYNTAX

```
wc [options]... [file]...
```

OPTIONS

-c
--bytes
--chars
    Print only the byte counts.

-w
--words

Print only the word counts.

-l
--lines
Print only the newline counts.

-L
--max-line-length
Print only the length of the longest line per file,
and if there is more than one file it prints the
maximum (not the sum) of those lengths.

`wc' prints one line of counts for each file, and if the file was given as an argument, it prints the file name following the counts.

If more than one FILE is given, `wc' prints a final line containing the cumulative counts, with the file name `total'. The counts are printed in this order: newlines, words, bytes.

By default, each count is output right-justified in a 7-byte field with one space between fields so that the numbers and file names line up nicely in columns. However, POSIX requires that there be exactly one space separating columns. You can make `wc' use the POSIX-mandated output format by setting the `POSIXLY_CORRECT' environment variable.

By default, `wc' prints all three counts. Options can specify that only certain counts be printed. Options do not undo others previously given, so

```
wc --bytes --words
```

will print both the byte counts and the word counts.

who

Print who is currently logged in

SYNTAX
who [*options*] [*file*] [am i]

OPTIONS

`-m'
Print the current user id, name and domain
(Same as `who am i')

`-q'
`--count'

Print only the login names and the number of users logged on.
Overrides all other options.

`-s'
Ignored; for compatibility with other versions of `who'.

`-i'
`-u'
`--idle'
After the login time, print the number of hours and minutes that
the user has been idle. `.' means the user was active in last
minute. `old' means the user was idle for more than 24 hours.

`-l'
`--lookup'
Attempt to canonicalize hostnames found in utmp through a DNS
lookup. This is not the default because it can cause significant
delays on systems with automatic dial-up internet access.

`-H'
`--heading'
Print a line of column headings.

`-w'
`-T'
`--mesg'
`--message'
`--writable'
After each login name print a character indicating the user's
message status:

`+' allowing `write' messages
`-' disallowing `write' messages
`?' cannot find terminal device

If given no non-option arguments, `who' prints the following information for each user
currently logged on:

login name,
terminal line,
login time,
remote hostname or X display.

If given one non-option argument, `who' uses that instead of `/var/run/utmp' as the name
of the file containing the record of users logged on. `/var/run/wtmp' is commonly given as

an argument to `who' to look at who has previously logged on.

If given two non-option arguments, `who' prints only the entry for the user running it (determined from its standard input), preceded by the hostname. Traditionally, the two arguments given are `am i', as in `who am i'.

PRACTICAL-6

OBJECTIVE : Writing of Shell Scripts (Shell Programming).

Problem-1

To print the name, address and date of birth of a student

Algo :

```
1   START
2   ENTER THE VALUE FOR NAME AND READ IT
3   ENTER THE VALUE OF ADDRESS AND READ IT
4   ENTER THE VALUE FOR DATE OF BIRTH OF A STUDENT  AND READ
    IT.
5   PRINT ALL THE THREE BY THE $ECHO COMMAND.
6   STOP.
```

Source Code :

```
[user11@linuxserver user11]$ cat details
echo enter the name
read n
echo enter the address
read a
echo enter date of birth
read d
```

```
echo the name u entered is : $n
echo the address u entered is : $a
echo the DOB u entered is : $d
```

Output :

```
[user11@linuxserver user11]$ sh details
enter the name
mohit
enter ur address
gurgaon
enter ur date of birth
aug 6th
the name u entered is : mohit
the address u entered is : gurgaon
the DOB u entered is : aug 6th
[user11@linuxserver user11]$
```

<u>Problem-2</u>

To print the date and the present working directory.

Algo :

```
1   START
2   PRINT THE DATE BY THE COMMAND $DATE
3   PRINT THE PRESENT WORKING DIRECTORY BY THE COMMAND
    $PWD
4   STOP.
```

Source Code :

```
[user11@linuxserver user11]$ cat date
echo the date is : `date `
echo the directory is : `pwd`
```

Output :

```
[user11@linuxserver user11]$ sh date
the date is : Mon Nov 6 10:24:50 IST 2006
the directory is : /home/user11
[user11@linuxserver user11]$
```

Problem-3

To calculate and print the sum of the 3 numbers

Algo :

```
1   START
2   ENTER THE 1ST NUMBER AND READ IT
3   ENTER THE 2ND NUMBER AND READ IT
4   ENTER THE 3RD NUMBER AND READ IT
5   ADD THE 3 NUMBERS BY THE COMMAND `expr $a + $b + $c `
6   PRINT THE SUM OF THE 3 NUMBERS .
7   STOP
```

Source Code :

```
[user11@linuxserver user11]$ cat add
echo enter the 1st number
read a
echo enter the 2nd number
read b
echo enter the 3rd number
read c
echo the SUM of 3 numbers is :: `expr $a + $b + $c`
```

Output :

```
[user11@linuxserver user11]$ sh add
enter the 1st number
25
enter the 2nd number
65
enter the 3rd number
30
the SUM of 3 numbers is :: 120
[user11@linuxserver user11]$
```

<u>Problem-4</u>

To find largest of three numbers.

Algo :

```
1  START
2  ENTER THE THREE NUMBERS
3  COMPARE WHICH OF THREE IS GREATEST
4  PRINT THE VALUE OF THE GREATEST NUMBER
5  STOP
```

Source Code :

```
[user11@linuxserver user11]$ cat large
echo enter any 3 numbers
read a
read b
read c
if [ $a -gt $b -a $a -gt $b ]
then echo the greatest no. is :$a
elif [ $b -gt $a -a $b -gt $c ]
then echo the greatest no. is : $b
else
echo the greatest no is : $c
fi
```

Output :

```
[user11@linuxserver user11]$ sh 1arge
enter any 3 numbers
21
23
24
the greatest no is : 24
[user11@linuxserver user11]$
```

Problem-5

To check for a number whether it is even or odd.

Algo :

```
1   START
2   READ THE NUMBER FROM THE USER
3   IF IT GIVES A REMAINDER WHEN DEVIDED BY 2, IT IS AN ODD
    NUMBER ELSE IT'S AN EVEN NUMBER
4   PRINT THE TRUE RESULT
5   STOP
```

Source Code :

```
[user11@linuxserver user11]$ cat evodd
echo enter the no
read a
if [ $a % 2 -eq 0 ]
then echo given no. is even
else
echo given no. is odd
fi
```

Output :

```
[user11@linuxserver user11]$ sh evodd
enter the no
```

```
21
given no. is odd
[user11@linuxserver user11]$
```

PRACTICAL-7

OBJECTIVE : AWK Programming.

AWK programming

Awk is essentially a stream editor, like sed. You can pipe text to it, and it can manipulate it on a line-by-line basis. [or it can read from a file]. It is also a programming language. That basically means it can do anything sed can do, and a lot more. (But you might have to type more :-)

Unlike sed, it has the ability to remember context, do comparisons, and most things another full programming language can do. For example, it isn't just limited to single lines. It can JOIN multiple lines, if you do things right.

The simplest form of awk is a one-liner:

```
  awk '{ do-something-here }'
```

The "do-something-here" can be a single print statement, or something much more complicated. It gets somewhat 'C'-like. Simple example:

```
  awk '{print $1,$3}'
```

will print out the first and third columns, where columns are defined as "Things between whitespace". (whitespace==tab or space) Complicated example:

```
awk '{          if ( $1 = "start") {
                start=1;
                print "started";
                if ( $2 != "" )      {
                        print "Additional args:",$2,$3,$4,$5
                }
                continue;
        }
        if ( $1 = "end") {
                print "End of section";
                printf ("Summary: %d,%d,%d (first, second,
equal)\n",
                        firstcol, secondcol, tied);
                firstcol=0;
                secondcol=0;
                tied=0;
                start=0;
        }
        if ( start >0) {
                if ( $1 > $2 ) {
                        firstcol= firstcol+1
                }else
                if ( $2 > $1 ) {
                        secondcol= secondcol+1
                }else
                        tied=$tied+1
        }
}'
```
AWK's main goal in life is to manipulate its input on a line by line basis. An awk program usually goes through some version of
Process a line. Move on.
Process a line. Move on.
Process a line. ...

If what you want to do does not fit into that model, then awk may not be a good fit for what you want to do.

The general syntax used by all awk programing can be described as:

PATTERN {COMMAND(S)}

What this means is,

"For each line of input, go look and see if the PATTERN is present. If it is present, run the stuff between {}"

[If there is no pattern specified, the command gets called for EVERY line]

A specific example:

```
awk '/#/ {print "Got a comment in the line"}' /etc/hosts
```
will print out "Got a comment" for every line that contains at least one '#', **anywhere in the line**, in /etc/hosts

The '//' bit in the pattern is one way to specify matching. There are also other wasy to specify if a line matches. For example,

```
 $1 == "#" {print "got a lone, leading hash"}
```
will match lines that the first column is a single '#'. The '==' means an EXACT MATCH of the ENTIRE column1.

On the other hand, if you want a partial match of a particular column, use the '~' operator

```
  $1 ~ /#/ {print "got a hash, SOMEWHERE in column 1"}
```

NOTE THAT THE FIRST COLUMN CAN BE AFTER WHITESPACE.

Input of "# comment" will get matched
Input of " # comment" will ALSO get matched

If you specifically wanted to match "a line that begins with exactly # and a space" you should use

```
  /^# /  {do something}
```

Multiple matching

Awk will process ALL PATTERNS that match the current line. So if the following example is used,

```
  awk '
     /#/ {print "Got a comment"}
     $1 == "#" {print "got comment in first column"}
     /^# /  {print "Found comment at beginning"}
   ' /etc/hosts
```

you will get THREE printouts, for a line like

```
# This is a comment
```

TWO printouts for

```
  # This is an indented comment
```

and only one for

```
1.2.3.4 hostname # a final comment
```

So we've covered awk's general syntax . Now let's start getting into the funky stuff.

awk has "special" match strings: "BEGIN" and "END"

The BEGIN directive gets called once before any data line is read, and never again. The END directive gets called after all lines have been read. If multiple files are given, it only gets called after the very last file has been completed.

Generally, you would use the BEGIN bit to initialize things, and the END bit for summaries, or cleanup.

Example:

```
BEGIN          { maxerrors=3 ; logfile=/var/log/something ;
tmpfile=/tmp/blah}
  ...          { blah blah blah }
/^header/      { headercount += 1 }
END            { printf("total headers encountered=%d\n",
headercount);
```

This will count the number of times "header" appears in an input file,and print out the total only after processing the entire file.

AWK also has lots of other special variables, that you can use in the { } section. For example,

```
print  NF
```

will give you the total number of columns (Number of Fields) in the current line. FILENAME will be the current filename, assuming the filename was given to awk, and you're not piping to it.

You CANNOT CHANGE NF yourself.

Similarly, there is a NR variable, that tells us how many lines you have processed. ("Number of Records")

There are other special variables, and even ones we CAN change in the middle of the program.

Math tweaks

```
  +, -, /, *, sin(), cos(), tan(), atan(), sqrt(), rand(),
srand()
```

String manipulation

index() will tell you if and where a string occurs in a substring.

match() is similar, but works for regular expressions.

sprintf() gives you a way to format output, and do conversions along the way. This should be familiar to anyone who has used printf() in C. For example,

```
   newstring=sprintf("one is a number %d, two is a string %s\n",
one,  two);
   print  newstring
```

"%d" says "print the variable matching me, as a decimal number"
"%s" says "print the variable matching me, as a string"

So if you wanted to join two strings with no gaps, ONE way would be to use

```
    newstring=sprintf("%s%s", one,  two)
```

length() just gives you an easy way to count characters in a string, if you need that.

System level functions

system() lets you call potentially ANY executable on the system. The target executable can be either in your $PATH, or you can specify it by absolute path.

For example, the painful

```
    system("rm -rf $HOME");
```

or

```
    system("/bin/kill 1")
```

If you want to do more complex things, you'll probably end up doing something like

```
   sysstring=sprintf("somecommand %s %s", arg1,  arg2);
   system(sysstring)
```

close() is an important function that is often overlooked. This is probably because there isn't an explicit open() call, so people don't expect to need a close() call. And for most purposes, you dont. But you DO NEED IT, if you are dealing with more than one output file.

Awk gives you the capability to open random files on the fly. For example

```
  /^file/     { print $3 >> $2 }
```

should take the line "file output here-is-a-word", open the file 'output', and print 'here-is-a-word' to it.

AWK is "smart", in that it keeps track of what files you open, and KEEPS them open. It assumes if you opened a file once, you are likely to do it again. Unfortunately, this means if you open LOTS of files, you may run out of file descriptors. So, when you know you are done with a file, close it. So, to improve the above example, you might use something along the lines of:

```
  /^file/     { if ( $2 !=  oldfile ) { close( oldfile) };
                      print $3 >> $2 ;  oldfile = $2; }
```

Array concepts

I previously didn't have a reason to use arrays, but since someone recently emailed me an example of when you could use arrays , I feel inclined to share it with folks.

We have previously covered variables, as a name that holds a value for you. Arrays are an extension of variables. Arrays are variables that hold more than one value. How can it hold more than one value? Because it says "take a number".

If you want to hold three numbers, you could say

```
value1="one"; value2="two"; value3="three";
```

OR, you could use

```
values[1]="one"; values[2]="two"; values[3]="three";
```

You must always have a value in the brackets[] when using a variable as an array type. You can pick any name for an array variable name, but from then on, that name can ONLY be used as an array. You CANNOT meaningfully do

```
values[1]="one";
values="newvalue";
```

You CAN reassign values, just like normal variables, however. So the following IS valid:

```
values[1]="1";
print values[1];
values[1]="one";
print values[1];
```

The really interesting this is that unlike some other languages, you dont have to just use numbers. The [1],[2],[3] above are actually treated as ["1"], ["2"], ["3"]. Which means you can also use other strings as identifiers, and treat the array almost as a single column database. This formal name for this is an "associative array".

```
numbers["one"]=1;
numbers["two"]=2;
print numbers["one"];
value="two";
print numbers[value];
value=$1;
if(numbers[value] = ""){ print "no such number"; }
```

When and how to use arrays

There are different times you might choose to use arrays. As I mentioned, I personally have never needed them :-) But here are some instances that might be relevant to you.

Storing info for later

When using awk as part of a larger shellscript, I tend to just print out information to a temporary file. But if you have a reason to do so, you could save particular words in memory, and print them out all at the end, which would be faster than using a temporary file.

```
/special/{ savedwords[lnum]=$2; lnum+=1; }
END     {
                count=0;
                while(savedwords[count] != "")
                {
                        print count,savedwords[count];
                        count+=1;
                }
        }
```

Instead of just printing the words out, you could use the END section to do some additional processing that you might need, before displaying them.

Arrays, and the split() function

The other primary reason to use arrays, is if you want to do subfields. Lets say you have a line, that has some course divisions, and some fine divisions. In other words, top level fields are separated by spaces, but then you get smaller words separated by colons.

```
  This is a variable:field:type line
  There can be multiple:type:values here
```

In the above, the fourth space-separated field, has subfields separated by colons. Now let's say you wanted to know the value of the second subfield, in the fourth major field. One way to deal with this, would be to call two awks, piped together:

```
awk '{print $4}' | awk -F: '{print $2}'
```

Yet another way would be to change the field separator variable 'FS', on the fly:

```
awk '{ newline=$4; fs=FS; FS=":";  $0=newline; print $2 ; FS=fs;
}'
```

But you could also do it with arrays, using the split() function, as follows:

```
awk '{ newline=$4; split(newline,subfields,":"); print
subfields[2]} '
```

Straight output

Sometimes, you just want to use awk as a formatter, and dump the output stright to the user. The following script takes a list of users as its argument, and uses awk to dump information about them out of /etc/passwd.

Note: observe where I unquote the awk expression, so that the shell does expansion of $1, rather than awk.

```
#!/bin/sh

while [ "$1" != "" ] ; do
       awk -F: '$1 == "'$1'" { print $1,$3} ' /etc/passwd
       shift
done
```

Setting a shell variable from awk output

Sometimes you just want to use awk as a quick way to set a value for a variable. Using the passwd theme, we have a way to grab the shell for a user, and see if it is in the list of official shells.

Again, be aware of how I unquote the awk expression, so that the shell does expansion of $1, rather than awk.

```
#!/bin/sh

user="$1"
if [ "$user" ="" ] ; then echo ERROR: need a username ; exit ; fi
```

```
usershell=`awk -F: '$1 == "'$1'" { print $7} ' /etc/passwd`
grep -l $usershell /etc/shells
if [ $? -ne 0 ] ; then
       echo ERROR: shell $usershell for user $user not in
/etc/shells
fi
```

Other alternatives:

```
# See "man regex"
usershell=`awk -F: '/^'$1':/ { print $7} ' /etc/passwd`

#Only modern awks take -v. You may have to use "nawk" or "gawk"
usershell=`awk -F: -v user=$1 '$1 == user { print $7} '
/etc/passwd`
```

The explaination of the extra methods above, is left as an excercise to the reader :-)

In a pipe-line

Sometimes, you just want to put awk in as a filter for data, either in a larger program, or just a quickie one-liner from your shell prompt. Here's a quickie to look at the "Referrer" field of weblogs, and see what sites link to your top page many different types of web browsers come to look at your site.

```
#!/bin/sh

grep -h ' /index.html' $* | awk -F\" '{print $4}' | sort -u
```

AWK summary

All the features I have mentioned, make AWK a fairly decent language. Its main drawback is that it is so line-oriented. It would be kind of nice to have a proceedural language with all the power of AWK. Which is why perl was invented.

Unfortunately, Larry then decided to go waaaaay beyond the simple concept, by throwing in the kitchen sink, AND destroying the cleanness of the AWK language syntax, all in the name of reducing the number of keystrokes needed to accomplish something. The extra functions in perl are good. The syntax, however, is disgusting to programmers capable of touch-typing more than 20 words a minute. Having the shortest number of characters to implement something, should not be the judge of value for a language.

REFERENCES

- “Operating System Concepts” by Silberchatz.
- “Modern Operating System” by A. Tanenbaum.
- “Unix The Shell” by Sumita Dabas.
- “Unix Shell Programming” by Yashwant Kanetkar.

NEW IDEAS / EXPERIMENTS

- The students can go for practical study of Windows XP.
- Other operating systems like Polaris, Unix can be further added.
- Students can go for practical implementation of Scheduling Techniques like FCFS, Priority Scheduling, Round Robin Scheduling, SJF and different page replacement techniques like FIFO, LRU, Optimal page replacement using programming language like C.

FREQUENTLY ASKED QUESTIONS

1. What is the difference between windows and Linux?
2. How many processors a single operating system can support?
3. What is different between multiprogramming, multiprocessing and multithreading?
4. What is the difference between segmentation and paging ?
5. What is Windows 2000?

6. How much difference is there between Windows 2000 and Windows NT 4.0?
7. What is the difference between Windows 95/98/Me and Windows 2000?
8. What are the hardware requirements for Windows 2000?
9. What is Linux?.
10. Can I have both Windows and Linux on my computer?
11. What makes Linux different from Microsoft Windows or Mac OS?
12. Is Linux the only Open Source operating system?
13. What is the relationship of Linux and UNIX/Unix?
14. How does Linux differ technically from Microsoft Windows?
15. Is Linux compatible with Windows?
16. What does Linux look like?
17. Is Linux really more secure than Windows?
18. Which Windows users should switch to Linux? Why? When? How?
19. Write a shell script to identify the given string is palindrome or not?
20. What is the difference between writing code in shell and editor?
21. How will you list only the empty lines in a file (using grep)?
22. What does $# stand for?
23. What are the different kinds of loops available in shell script?
24. What Hardware Is Supported?
25. Who Wrote Linux?