Linux Fundamentals

A Practical Guide to Learning Linux

Matthew West

Linux Fundamentals: A Practical Guide to Learning Linux

by Matthew West

Published 2005-01-25 19:56:07

Table of Contents

List of Figures

This page intentionally left blank

List of Tables

This page intentionally left blank

Chapter 1. Details of requirements for the courses

Machine Requirements

You do not need a very poweful Computer to run Linux. If you did a console only installation (without a GUI windows environment) a Pentium 100 with 32 MB of memory and 500 MB of hard drive space would be more than enough . Of course the more recent versions of Linux like Red Hat™ Enterprise 3 and SUSE Linux™ 9 would need a computer with more capabilities than a Pentium 1

Fundamentals

It is possible to run this course on Linux or in Windows - however when working in Windows you will have to install a Linux emulator. Matthew West created a Debian GNU/Linux image you can use with Bochs, an Open Source Emulator. We suggest you use this image and Bochs, since it also includes the sample files that Matthew West uses as examples and Exercises in the Fundamentlas course.

What this does is to create a virtual enviroment, that runs inside your current Operating System, which runs Debian GNU/Linux. Download and installation instructions can be found the section called “Setup of Linux Emulator for Fundamentals Course” [44] here.

Because this emulated (virtual) machine needs to use the resources of the installed Operating System you need a faster computer to run it successfully. We recommend a Pentium 2 500 Mhz computer with 32 MB memory. If you already have Linux installed you do not need to use the simulator.

System Administration

For the System Administration course you need only one machine. The System Administration course begins by explaining how to install Debian. Run this course after you have completed and are familiar with the material covered in the Fundamentals course.

Networking

For the Networking courses, you would need to have two machines with Linux installed on them. Both machines should have a Network card installed. If you are using a machine that is already connected to a Local Area Network then you do not

need a second machine.

Shell Scripting and Internals courses

You need to have a Linux distribution installed to take part in these courses

Course Pre-requisites And Outcomes

A theory that works

In the technical world of computers today it is no longer really enough to just know or specialise in one area of technology. With the emphasis being on networking, you will need to know something of everything to really get by. (e.g. operating systems, routers and other networking equipment, system and network administration, system and network design, latest technology trends etcetera.)

You will also need knowledge of the business structure in your company. Become a technical person able or capable of enhancing the business of the company or companies that you work for - almost operating as a business yourself (self-sustaining and self-enhancing).

Some examples would be:

- Regular operating system maintenance will ensure good solid and consistent performance - this could save the business a lot of money.
- Another example would be that if you know the Open Source and Free Software products available you could advise your company to go with that solution rather than a propriety solution and this could go a long way to ensuring that the business saves money.
- Think laterally and carefully when supporting and Operating System like Linux or Unix, being so powerful means that there is more to it than a simple stream of instructions.
- Become aware of what is happening around you in the computer industry and in business and become a real asset.

A note aside:- “Whilst working as a Unix technician, a new "client" phoned me one day and asked me to install another disk drive on the Unix server for them as their first hard drive was full. They had been running this server for 4 years and I asked them to wait until I arrived before purchasing another disk but I was too late they had ordered one already. When I arrived I cleaned up the primary hard disk drive from 100% full to 40% full just by doing thorough house-keeping (could have been handled with a Shell Script running automatically each week/month), tidying up the

log files, temporary directories, superfluous software versions etcetera." Simone Demblon

Once you have learnt one operating system in the way that we have structured this course, it is much easier to pick up other knowledge on hardware / operating systems, system configuration and even development.

As it would be almost impossible to learn everything about all technology available, cultivate a technical way of thinking laterally, it will be a decided advantage to you.

Pre-requisite knowledge and outcomes

The text below is a mere guideline to equivalent knowledge, as you know if you have a talent for working with computers OR if you are an extremely hard-worker who is prepared to play with the operating system until you are sure of yourself, then you are likely to not need to follow these guidelines and you will exceed the qualification levels that are suggested here.

Therefore when I say below that your knowledge would be equivalent to a System Administrator, what I am really saying is that although you will have an extensive knowledge of Linux (we have structured the courses to ensure that there are sufficient labs and Exercises), the additional knowledge - the knowledge of specific company set-ups or specific pieces of hardware - will still have to be gained by your experience.

Now let's look at each course or course range and discuss the relevant issues:

1. In order to successfully complete the Fundamentals course you will need to have knowledge of PC Computers (operating systems and hardware).

   After completion on the Fundamentals course (approximately 18 hours of study), you would have a basic grounding of the Linux Operating System.

   Please note however that although an introductory course to Linux it is not an introduction to computers or operating systems. We assume that you have some technical knowledge already.

   In this course, some internal operations of the operating system are covered simply, and this is in order to ensure that you are able to think a problem through laterally. This will also assist if wishing to complete the range all the way through to the Internals course, by giving you a grounding in simple terms to build on throughout the other courses.

2. In order to successfully complete the System Administration course you would need enough knowledge to install an operating system.

After completion of the Fundamentals and System Administration courses (18 + 30 hours), you would have the equivalent knowledge of a Junior Administrator in Linux. You will have enough knowledge and experience (through intensive labs) to assist a fully qualified System Administrator in a commercial business situation. (RHCT)

At this stage all you will lack is further experience to enable you to perform the function of System Administrator.

3. After further completing the Network Administrators course (30 hours), and this would include all associated Exercises, labs and simulated problem labs, you would be able to work as a Junior Network Administrator.

4. After completing the Elective course subjects, affiliated to the Networking course, (18 hours) you would be qualified to do System and Network Administration including monitoring and maintaining your network. (RHCE)

5. Shell Scripting (20 hours) is a course that will clarify the power of Linux for you and will also excite you as pieces of the "operating system puzzle" fall into pace at an alarming rate. This is a stunning course and no matter what you intend to do with your Linux knowledge this course is a must.Ensure that you have completed the following courses or that you have equivalent knowledge prior to attempting this course: Fundamentals, System Administration and Networking Introduction.

6. Internals^-1 is a technical course written to enable a System Administrator to become a visionary systems engineer able to attend a full internals course if so inclined. A cautionary note would be that although we have kept it as generically inclined as possible you may have to check up the variances with the Linux or Unix kernel that you are working with.

   As we have said from the beginning, support of such an operating system is going to take a fair amount of lateral thinking, and as not all of us are interested in the nitty-gritty details of how an operating system was written (see reference material used if you are interested), so internals^-1 will give you the workings in a more simple technical form.

Chapter 2. Linux Distributions and Certifications

Licensing and availability details for Debian, Red Hat and SuSE

There are literally hundreds of Linux Distributions available. Size-wise they extend from versions that fit onto one 3.5 inch disk to those that are a few gigabytes big.

Although there are differences from one version of this operating system to another, they all use the Linux Kernel (albeit different versions of the kernel

We will discuss three of the Linux distributions that are most widely used, namely: SuSE, Red Hat, and Debian

SUSE Linux™

Novell announced[1] on November the 4th 2003, that they have made an agreement to acquire SUSE Linux™ [2], this purchase is subject to regulatory approval, but is expected to be allowed and finalized by the first quarter of 2004.

Home Users

This is how SUSE™ describes SUSE Linux 9.0, which is the family of products aimed at the home user:[3]

“ Migrating from Windows has never been easier: SUSE Linux 9.0 is secure and stable. In addition to a powerful operating system, SUSE Linux 9.0 delivers all the applications you need for Internet, Office, Multimedia, and Home networking. Its installation routine is now almost fully automated, so you'll be up and running with little effort. And, of course, you are assured all the advantages of using Open Source software. ”

System Requirements for SUSE Linux 9.0 ™

- Processor
    - Intel: Celeron, Pentium to Pentium 4, Xeon
    - AMD: K6/II/III, Duron™, Athlon™, Athlon™ XP/MP, Athlon 64™

[1] Novell.com [http://www.novell.com/news/press/archive/2003/11/pr03069.html]
[2] SUSE is a registered trademark of SUSE Linux™
[3] http://www.suse.com/us/private/products/suse_Linux/

    - IBM
    - 286, 386, 486 and Cyrix processors are not supported

- Main Memory
    - At least 64 MB are required for the installation with YaST2 in graphical mode; 128 MB recommended

- Hard disk
    - 400 MB to more than 3 GB (Personal Edition) or 6 GB (Professional Edition) for the installation of all packages; 2 GB or more recommended
    - LBA48 hard disks are supported

SUSE Linux™ 9.0 can be downloaded via FTP for free. You can choose to download the complete installation directory, a CD image, from which you can create a bootable CD that will download and install SUSE from the FTP server. It is not possible to create the installation CD's for SUSE™ from the directories on the FTP server. You can also download a demonstration version of SUSE™ that runs from a bootable CD[4]

Enterprise Users

SUSE™ currently has three products that businesses are suggested to use, these are SUSE Linux Standard Sever 8™, SUSE Linux Enterprise Server 8™ and SUSE Linux Openexchange Server 4.1™

SUSE Linux Standard Server 8™:

This is how SUSE describes SUSE Linux Standard Server 8™

“ With its comprehensive graphical administration, SUSE Linux Standard Server was designed for small organizations and departments that want to implement their Internet access as well as e-mail, print, and file services in a reliable, secure way. Standard Server is available for 32-bit processors (x86) from AMD and Intel and supports up to two CPUs” [5]

[4]A list of FTP mirror sites can be found here. [http://www.suse.com/us/private/download/ftp/int_mirrors.html] From here you would be able to download the files you need to install SUSE Linux 9.0™

System Requirements for SuSE Linux Standard Server 8™:

- Reccomended CPU: 700 Mhz
- Minmum RAM: 256 MB
- Hard Disk Space Required for Installation: 1 GB

Features of SuSE Linux Standard Server 8™:

- File and print services for Linux and Windows™
- Primary Domain Controller (PDC) for Windows™
- Central user administration with directory service (LDAP)
- E-mail server (IMAP) for all e-mail clients including:
    - Definition of the mailbox quota
    - SPAM filter
    - Dial on demand
    - Fetching mail from other providers
- Internet gateway server including web cache, web content filter, and firewall
- Automatic assignment of IP addresses via DHCP server
- Administration of host names with Dynamic Name Service (DNS)
- Secure access for clients, i.e. for external staff via Virtual Private Network (VPN)
- Application server

SuSE Linux Enterprise Server 8™

This is how SUSE describes SUSE Linux Enterprise Server 8™

[5]From: http://www.suse.com/us/business/products/server/

“ SUSE Linux Enterprise Server 8 is a leading server operating system for professional deployment in heterogeneous IT environment of all sizes and sectors. It is available for all relevant hardware platforms, ranging from AMD/Intel 32-bit and 64-bit processors to the entire IBM eServer series including mainframes - one single server operating system with a uniform code basis!” [6]

System Requirements for SUSE Linux Enterprise Server 8™:

- Recommended CPU: 800 MHz
- Minimum RAM: 256 MB
- Hard Disk Space required for installation: 1.2 GB

For a full list of SuSE Linux Enterprise Server 8™ Features, visit this [http://www.SuSE.com/en/business/products/server/sles/features.html] page

A comparison of SUSE Linux Standard Server 8™ and SUSE Linux Enterprise Server 8™ can be found here.
[http://www.suse.com/en/business/products/server/which_version/index.html]

SUSE Linux Openexchange Server 4.1™

This is how SUSE describes SUSE Linux Openexchange Server 4.1™

“ SUSE Linux Openexchange Server 4.1 is the trend-setting groupware and communication solution that helps your company to progress - with superior technical features, far-reaching hardware independence, smooth migration, and a wide range of supported clients including Outlook clients from Outlook 98 and various web browsers.For All Requirements”

“On the basis of standardized protocols and Open Source components, SUSE Linux Openexchange Server offers everything modern enterprises and organizations need for communication: e-mail server, web server, groupware, collaboration, and messaging.” [7]

System Requirements for SUSE Linux Openexchange Server 4.1™

- CPU(s): AMD Athlon™ /Duron™, Intel Pentium III/4 or compatible AMD K6

[6] From: http://www.suse.com/us/business/products/server/
[7] From:http://www.suse.com/us/business/products/openexchange/

is not supported!

- RAM: 256 MB
- Hard disk space: 9 GB

Red Hat

Red Hat Linux is probably the most well known Linux distribution. The first version of Red hat was released in October 1994.

Its success can be attributed to the commitment to support and the development of the RHCE certification. For this reason, many of the corporate companies choosing to use Open Source Software have selected Red Hat Products. They knew that with Red Hat they would be able to have reliable updates to the products and that there was a pool of trained Red Hat Support Engineers.

Red Hat for Home Users: Fedora

Late in 2003 Red Hat announced that they would stop supporting Red hat Linux 9 and instead release two new product lines, aimed at very different markets.

Up until that time Red Hat Linux 9 and the earlier versions, were used in the corporate environment and by home users.

The version now meant for home users is called "the Fedora Project"

This is how Red Hat describes the Fedora Project: "The Fedora Project is a Red-Hat-sponsored and community-supported open source project. It is also a proving ground for new technology that may eventually make its way into Red Hat products. It is not a supported product of Red Hat, Inc." [8]

A more open development process is used for Fedora, than is for Red Hat Enterprise Linux. The Red Hat Developers are still taking part in the development of Fedora but more community driven development is encouraged.

Fedora License

In my understanding, a user may make and distribute unmodified copies of Fedora's source code and binary code. If the product is modified you may only redistribute these files if all images that contain the Fedora trademark is changed.

Fedora's binary and source files may be downloaded, for free, via FTP.

Please see this [http://fedora.redhat.com/download/] page, for exact instructions on

[8] From:http://fedora.redhat.com/

how to download Fedora's installation files.

Fedora System Requirements

- CPU: Note: The following CPU specifications are stated in terms of Intel processors. Other processors (notably, offerings from AMD, Cyrix, and VIA) that are compatible with and equivalent to the following Intel processors may also be used with Fedora Core.
    - Pentium-class Note: Fedora Core 1 is optimised for Pentium PRO (and later) CPUs, but also supports Pentium-class CPUs. This approach has been taken because Pentium-class optimisations actually result in reduced performance for non-Pentium-class processors.
    - Recommended for text-mode: 200 MHz Pentium-class or better
    - Recommended for graphical: 400 MHz Pentium II or better
- Hard Disk Space (NOTE: Additional space will be required for user data):
    - Custom Installation (Minimal): 520MB
    - Server: 870MB
    - Personal Desktop: 1.9GB
    - Workstation: 2.4GB
    - Custom Installation (Everything): 5.3GB
- Memory:
    - Minimum for text-mode: 64MB
    - Minimum for graphical: 192MB
    - Recommended for graphical: 256MB

Red Hat Enterprise Linux

Red Hat has four products that are aimed for use in the corporate environment:

Red Hat Enterprise Linux ws

Designed for desktop/client systems, therefore it does not include the server applications found in the Red Hat Enterprise Linux ES or AS. (see below).

Red Hat Enterprise Linux ws capabilities and System Requirements

A Test Paragraph to see if it fixes the issue

- Capabilities:
    - Mail
    - Document Processing
    - Browsing
    - Instant Messaging
- Supported hardware:
    - Intel X86
    - Intel Itanium
    - AMD
    - AMD64

Red Hat Enterprise Linux ES

Designed for small/midrange servers. Has the same capabilities as Red Hat Enterprise Linux AS, but its hardware support is less extensive. It supports x-86 based systems, with up to 2 CPU's and 8 GB of memory.

It is ideally suited for network, file, print, mail, Web, and custom or packaged business applications.

- Capabilities:

    - Mail
    - File (SMB/NFS)
    - Print
    - Accelerated Web (tux)
    - Advanced Firewall (arptables)
    - Extended Remote Shell Access/Mgmt
    - DHCP
    - DNS Nameserver
    - Network Authentication (Kerberos)
    - News
    - Backup
    - Dump Server (Netdump)
    - Directory Server (LDAP)
    - Remote Boot/Image Server
    - SSL

- Supported Hardware:

    - x86 architectures, up to 2 CPU's

Red Hat Enterprise Linux AS

Designed for high-end and mission-critical systems. This is Red Hat's top of the range distribution and is certified on systems provided by Dell, HP, IBM, and Sun. [9]

It supports the largest commodity-architecture servers with up to 16 CPUs and 64GB of main memory and is available with the highest levels of support.

[9] See this [http://www.redhat.com/software/rhel/as/] page for full specifications for Red Hat enterprise Linux AS

Red Hat Enterprise Linux AS Capabilities and Supported Hardware

- Capabilities:
  - Mail
  - File (SMB/NFS)
  - Print
  - Accelerated Web (tux)
  - Advanced Firewall (arptables)
  - Extended Remote Shell Access/Mgmt
  - DHCP
  - DNS Nameserver
  - Network Authentication (Kerberos)
  - News
  - Backup
  - Dump Server (Netdump)
  - Directory Server (LDAP)
  - Remote Boot/Image Server
  - SSL
- Supported Hardware:
  - Intel X86
  - Intel Itanium
  - AMD AMD64
  - IBM zSeries
  - IBM iSeries

- IBM pSeries
- IBM S/390

Red Hat Professional Workstation

Enterprise Linux for use in the home. Supports up x86 hardware, with up to 2 CPU's. Meaning that other than Fedora, this distribution does have official support from Red Hat, this includes technical support as well as security updates.

Red Hat Professional Workstation capabilities and Supported Hardware

- Capabilities:
  - Bluecurve, Ximian Evolution, OpenOffice.org, Mozilla
  - Samba, NFS, CUPS
  - GCC3.2
- Supported Hardware:
  - x86 architectures

Red Hat Enterprise Family Licensing

As with the Fedora Project, you are allowed to make and distribute unmodified copies of the binary and source code. If you want to distribute modified copies of the software or source code, you need to modify those files which contain the trademark images of Red Hat.

The licenses may vary depending on the country and product, to view the specific license for the product you are interested in, visit this [http://www.redhat.com/licenses/] page.

Debian GNU/Linux

Debian GNU/Linux (to use the correct term, Debian used for short) is a completely

free operating system, by this I mean that it contains no software that is released under a proprietary license. Some other Linux distributions contain code that is not free software. It currently uses the Linux kernel, but there are plans to use the GNU kernel in future releases. The latest stable release is 3.0r.1

Debian can be downloaded via FTP or HTTP from this [http://www.debian.org/distrib/cd] page

Or purchased from vendors, see this [http://www.debian.org/CD/vendors/] page

There are three versions of Debian that are available: stable, testing and unstable. This describes the amount of development that has been done on the particular version, and what environment it is suited to. You would not want to run your company's servers with the unstable version of Debian!

- Stable: current version is 3.0r.2, codenamed 'Woody'.
    - Debian suggests that end users use this version, Debian's security team supports it. Released in July 2002. Latest update to this version done on November 21st, 2003
- Testing: Current version is codenamed 'Sarge'.
    - Contains packages (applications) that have not yet been released in the Stable version, but which are planned to be released in the Stable version in the future. It has more recent versions of the software than Woody, but has no support from the Debian security team.
- Unstable: Codenamed 'Sid'.
    - This version of Debian is the one where new packages are actively being developed, generally this is only used by developers working on the packages for Sid.
- Hardware Supported by Debian:
    - Alpha ARM

- HP PA-RISC
- Intel x86
- Intel IA-64
- MIPS
- Motorola 680x0
- MIPS (DEC)
- PowerPC
- IBM S/390
- SPARC

Debian comes with more than 8700 packages, most of which are released under GPL licenses (or licenses that can be compared to the GPL). To view the list of packages that are available, visit this [http://www.debian.org/distrib/packages] page.

The installation manuals for the different distributions and hardware architectures can be found on this [http://www.debian.org/releases/stable/installmanual] page. The respective system requirements are found in the installation manuals.

When doing a minimal Debian installation, you need very little system resources. For example on the Intel x86 architecture you need 12 MB of memory and 250 MB hard drive space to install a console-based system. To install a system that includes the X Windows System, you need at least 400MB hard drive space.

On Line Classmates Information and Registration

Apart from the normal HTML version of the manual, we also use Moodle, a Open Source Package . By using the moodle version of the manual, the student not only has access to the material but also would have easy access to fellow students who are also taking part in the courses. Some of the features of Moodle is its forums which are dedicated to specific parts of the course as well as the chat facilities. This allows the student to communicate more efficiently with fellow students as well as the maintainers and lecturers of the courses.

Moodle was developed by Martin Dougiamas, visit www.moodle.org [http://www.moodle.org] for more information.

The Moodle implementation of this manual can be found at http://learnlinux.tsf.org.za/moodle/

Certification

Linux Professional Institute Certification

The Linux Professional Institute currently has two levels of certification, Junior Level Administration (LPIC1) and Intermediate Level Administration (LPIC2). At the time of writing the Senior Level Administration (LPIC3) course was still being developed.

With the permission of the LPI, we have created a detailed list of the skills you need to have to gain the different levels of certification. We have included this in the first appendix of this course Appendix A [113]

SAIR Linux and GNU Certification

We intended to cover the SAIR Linux GNU certification as well, but it seems as if it no longer exists. From their home page http://www.Linuxcertification.org/ it seems that this certification has been taken over by Thomson Learning corporate and professional training operations. We have made repeated attempts at contacting the new administrators of this certification, with no response. SAIR-GNU offered three levels of certification: Administrator (LCA), Engineer (LCE) and Master Engineer (MLCE).

Red Hat Certification

One of the reasons Red Hat has been so widely accepted in the Enterprise environment, I feel, is becasue it has created (and updated) its own certification for its products. People respect the RHCE certification, because it is not easy to attain. Unlike other certifications, the RHCE and RHCT exams are done on live test systems, that simulate tasks that System Administrators should be capable of doing.

There are two levels of certification offered by Red Hat; RHCT (Red Hat Certified Technitian) and RHCE (Red Hat Certified Engineer). The RHCE is the more advanced certification. The RHCE and RHCT exams are performance based practical labs. In other words you are given a exam based on situations that you will find in the real world. As of the end of 2003 these exams are based on the Red Hat Enterprise family of products, not Red Hat 9 as it was previoulsy.

RHCT Certification

According to Red Hat [10]

> RHCT tests a technician-specific subset of the skills tested in RHCE: * RHCTs will typically not be making the decisions about

[10] http://www.redhat.com/training/

how to set up production network services and network security. Thus, RHCT does not test the networking services and network security skills required to earn RHCE.

The RHCT consists of the following exams:

1. Troubleshooting and System Maintenance (1 hour)
2. Installation and Configuration (2 hours)

To earn the RHCT certification one must successfully complete all the requirements in Troubleshooting and System Maintenance and must attain at least 70% for Installation and configuration.

Pre_requisite skills:

- use standard command line tools (e.g., ls, cp, mv, rm, tail, cat, etc.) to create, remove, view, and investigate files and directories
- use grep, sed, and awk to process text streams and files
- use a terminal-based text editor, such as vi/vim, to modify text files
- use input/output redirection
- understand basic principles of TCP/IP networking, including IP addresses, netmasks, and gateways
- use su to switch user accounts
- use passwd to set passwords
- use tar, gzip, and bzip2v
- configure an email client on Red Hat Enterprise Linux
- use mozilla and/or links to access HTTP/HTTPS URLs
- use lftp to access FTP URLs

Skills needed for Troubleshooting and System Maintainenance exam:

- boot systems into different run levels for troubleshooting and system maintenance
- diagnose and correct misconfigured networking
- diagnose and correct hostname resolution problems
- configure the X Window System and a desktop environment
- add new partitions, filesystems, and swap to existing systems
- use standard command-line tools to analyze problems and configure system

Skills needed for Installation and Configuration Exam:

- perform network OS installation
- implement a custom partitioning scheme
- configure printing
- configure the scheduling of tasks using cron and at
- attach system to a network directory service, such as NIS or LDAP
- configure autofs
- add and manage users, groups, and quotas
- configure filesystem permissions for collaboration
- install and update RPMs
- properly update the kernel RPM
- modify the system bootloader
- implement software RAID at install-time and run-time
- use /proc/sys and sysctl to modify and set kernel run-time parameters

The RHCE certification:

The RHCE consists of the following exams:

1. Troubleshooting (2.5 hours)
2. Multiple Choice (1 hour)
3. Installation and Configuration (2.5 hours)

To earn the RHCE one must successfully complete all the troubleshooting tests, score a minimum of 50% for the multiple choice exam, score at least 70% for both the RHCE and RHCT components of the exams and attain a minimum of at least 80% for the all the exams as a whole.

Pre_requisite skills:

- use standard command line tools (e.g., ls, cp, mv, rm, tail, cat, etc.) to create, remove, view, and investigate files and directories
- use grep, sed, and awk to process text streams and files
- use a terminal-based text editor, such as vi/vim, to modify text files
- use input/output redirection
- understand basic principles of TCP/IP networking, including IP addresses, netmasks, and gateways
- use su to switch user accounts
- use passwd to set passwords
- use tar, gzip, and bzip2v
- configure an email client on Red Hat Enterprise Linux
- use mozilla and/or links to access HTTP/HTTPS URLs
- use lftp to access FTP URLs

Skills needed for Troubleshooting and System Maintainenance exam:

- boot systems into different run levels for troubleshooting and system maintenance

- diagnose and correct misconfigured networking
- diagnose and correct hostname resolution problems
- configure the X Window System and a desktop environment
- add new partitions, filesystems, and swap to existing systems
- use standard command-line tools to analyze problems and configure system
- use the rescue environment provided by first installation CD
- diagnose and correct bootloader failures arising from bootloader, module, and filesystem errors
- diagnose and correct problems with network services (see Installation and Configuration below for a list of these services)
- add, remove, and resize logical volumes

Skills needed for Installation and Configuration Exam:

- perform network OS installation
- implement a custom partitioning scheme
- configure printing
- configure the scheduling of tasks using cron and at
- attach system to a network directory service, such as NIS or LDAP
- configure autofs
- add and manage users, groups, and quotas
- configure filesystem permissions for collaboration
- install and update RPMs
- properly update the kernel RPM
- modify the system bootloader
- implement software RAID at install-time and run-time
- use /proc/sys and sysctl to modify and set kernel run-time parameters

People wanting to attain the RHCE certification must be also capable of configuring the following network services:

- HTTP/HTTPS
- SMB
- NFS
- FTP
- Web proxy
- SMTP
- IMAP, IMAPS, and POP3
- SSH
- DNS

For each of these services, RHCEs must be able to:

- install the packages needed to provide the service
- configure the service to start when the system is booted
- configure the service for basic operation
- Configure host-based and user-based security for the service

RHCEs must also be able to

- configure hands-free installation using Kickstart
- implement logical volumes at install-time
- use PAM to implement user-level restrictions

Chapter 3. History and Politics A Business Oriented Background

Introduction

The goal of this section is to present a version of the history and politics of software that is released under a license allowing the user the liberty to copy, modify and redistribute the code.

Many companies using propriety software favour the FUD tactics [11] to stop people from using Open Source software, therefore it is important to know the history of Open Source software and become familiar with the role players in the Open Source community, and in this way be able to make informed decisions regarding which Operating System and software to use.

Open Source and Free Software Licenses

When you start to get familiar with the Open Source environment, you will notice that two terms are used: Open Source and Free Software.

Free Software is a term used to describe software that is freely available, which allows users and developers to copy, modify and redistribute the source code.

Open Source software allows the user the exact same privileges.

“So what is the difference? And which is the correct term to use?” These are questions only you can answer for yourself, once you have read the following chapter:

The definition of Open Source, Free Software and Open Content

Much of the internal debates that range in the Open Source/Free Software communities, range about the licenses used to distribute software and Operating Systems. There are two main factions when it comes to Linux;

- the Free Software Foundation (FSF)[12]
- and the Open Source Initiative (OSI)[13].

As you will see from the section below the main difference between these two

[11] Fear, Uncertainty, Doubt
[12] www.gnu.org The Homepage of the GNU and the Free Software Federation
[13] www.opensource.org [http://www.opensource.org/]The Home page of the Open Source Initiative

organisations is *their motivation* for developing the software that is released under a license that allows freedom to modify, copy and redistribute the code.

The GNU Project and the Free Software Foundation

GNU (GNU's Not Unix) was formed by Richard M. Stallman (RMS) in 1984, when he became disillusioned by changes in his working environment.

Stallman worked in MIT's Artificial Intelligence labs in an atmosphere where developers freely shared source code with each other, with the understanding that this is how software was improved. This changed in the early 1980's when the MIT lab started using new hardware, which used proprietary hardware. To use the hardware the developers had to sign non-disclosure agreements.

RMS: The rule made by the owners of proprietary software was, “If you share with your neighbour, you are a pirate. If you want any changes, beg us to make them.” [14]

Stallman left MIT and started the GNU project. He believed that if he wanted to continue developing software he needed to be able to do so in an environment that would allow him to share his work with others, even if that meant creating a new Operating System and new utilities for that Operating System, as well as creating a license to release these products under.

> It is important to understand the meaning of the word 'free' in the context of Free Software (or Open Source Software), the Free Software Foundation and Open Content. 'Free' does not refer to the cost of the product but rather that the user is allowed the Freedom to access the Source Code for the product (be that a software application, an Operating System or documentation for an application.) A popular way of expressing this idea is: "Free Software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech", not as in "free beer." [15]

The Free Software Foundations' definition of Free Software: [16]

1. Freedom to use an application, for any purpose.
2. The Freedom to modify the application, to suit your needs.
3. The Freedom to redistribute copies of the application, so as to "help your

[14]Richard Stallman www.gnu.org [http://www.gnu.org/gnu/thegnuproject.html]
[15]The GNU's explanation on the meaning of "Free Software"www.gnu.org [http://www.gnu.org/philosophy/free-sw.html]
[16]From the GNU's philosophy page, visit the following link for the exact wording:www.gnu.org [http://www.gnu.org/philosophy/free-sw.html]

neighbor".

4. Freedom to modify the application and then distribute this modified version to whoever wants to use it.

The Free Software Foundation has commented on several licenses that claim to be Free Software licenses. To view this list visit this [http://www.gnu.org/licenses/license-list.html] page

The Open Source Initiative

The Open Source Initiative (OSI) [17] promotes the development of Open Source software.

The OSI was formed in 1998 after Netscape released the source code for Navigator5.0. Netscape decided to do this after they had read "The Cathedral and the Bazaar". (see our summary of the "Cathedral and the Bazaar" the section called “The Cathedral and the Bazaar” [32]later in this section.)

Eric Raymond, John "maddog" Hall, Larry Augustin and Bruce Perens are some of the people involved with the OSI from the start. This was a group of people who felt that the term "Free Software" as used by the Free Software Foundation is misleading and that it was not correctly understood despite being in existence since 1984. They also planned to work closer with commercial companies than the Free Software Foundation does.

The Open Source Initiative's definition of Open Source software:

“When the application is released under Open Source Licenses it allows the user many more freedoms than just access to the Source Code. These are:”

1. Free redistribution

   The application can be redistributed, either on its own or as a part of a bundle of other applications. No royalty or other fees are required to do this.

   Explanation as I see it: The idea of this is to not lose the long-term benefits (discussed later) of Open Content Software just to earn some money in the short term.

2. Source Code

   The Source Code for the project must be easily available, if it is not downloaded with the compiled code (human readable code compiled into code the machine can read) clear instructions must be given as to where and how the

[17] www.opensource.org [http://www.opensource.org]

source code can be obtained.

One of the main ideas of Open Source and Free Software is to make the evolution of software as easy as possible.

3. Derived Works

   The license must allow applications to be modified and distributed again with the same license as the original application.

   Explanation as I see it: By allowing people to modify and redistribute work, the evolution of applications are improved.

4. Integrity of the Author's Source Code

   If the author wishes to keep their Source Code as is, they may stipulate that modified Source Code cannot be distributed, only if they then allow files (patches) to be distributed with the application that will modify it at runtime.

   The author must allow the distribution of applications built (Compiled) from modified Source Code.

   This allows the author to specify that the original Source Code may not be modified (so that the author can be better recognised for the work done on the original Source Code), but forces the author to still allow modifications to the application by way of patch files that will modify the application, but not the original Source Code.

5. No Discrimination against Persons or Groups

   The author may not discriminate against any person or group; the product must be made available to all people who want to use it.

   This is to ensure as wide a range of users as possible, to improve the evolution of the software.

6. No Discrimination against fields of Endeavour

   “The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.”

   This is to ensure that the software can be used for commercial purposes.

7. Distribution of License

   The same license applies to all the people who the application is redistributed

to.

This is to make sure that some people who the application is distributed do not have to accept an additional license, (e.g. A Non Disclosure Agreement), to use the software.

8. License must not be specific to a Product

   If an application is taken from a bundle of applications that was released as Open Content, then that application has the same license as the original bundle of applications.

   This is to ensure that applications can be freely distributed in whatever form that it may be in. (i.e. part of a bundle or on its own)

9. License must not restrict other Software

   The License the application is released under cannot specify that it can only be distributed or used in conjunction with Open Source software.

   This allows distributors of software to decide what software to use and redistribute, further enhancing the evolution of software.

10. License must be Technology Neutral

    For example: the license cannot specify that to use the software you must download it from a web page, or from a CD-ROM. The license must allow modification of the application so that it can be used in all environments.

    This is so that, if the original application only ran in a GUI environment it can be altered to so that it can run in a command line environment. Also so that the License agreement cannot be made in one specific way, i.e. "Click Wrap" to allow the user to download the file from the web.

Bruce Perens, a leading advocate and developer of Free Software developed the Open Source Definition. The latest version of the Open Source Definition can be found on this [http://www.opensource.org/docs/definition_plain.php] page

Summary of difference between Open Source and Free Software:

The Free Software Foundation believes that their cause is a social one, to allow people to freely and openly use software for the betterment of mankind.

The Open Source Initiative believes that allowing people access to source code and allowing them to modify these to improve the software is a practical need, not a social right.

Though these two organisations differ on the reasons, they are not against each other; both believe that proprietary software inhibits the development of useful software.

“Throughout the rest of this document, I will use the term "Open Source Software" when referring to either Open Source or Free Software Foundation issues or products. It was difficult to choose between the two terms. The common misconception that "free" means free-of-charge and the fact that many people relate this to a anti-business mindset made me choose to use the term, "Open Source Software". I do, however, understand what the GNU means with "free software" and admire them for what they have done for this movement. ” Riaan Bredenkamp

Open Content:

Open Content was an organization that promoted the sharing of materials, specifically those used to educate people.

To do this it developed licenses that people could use to license their works be this software or manuals.

The organisation was founded by David Wiley in 1998.

During 2003 Wiley closed the Open Content organization because he felt that the Creative Commons organisation was doing a better job at creating "licenses for content" that would be recognised in court legally.

For more information about Open Content, visit this [http://www.opencontent.org/] page.

More information about the Creative Commons Organisation can be found on this [http://creativecommons.org/] page

The History of Open Source Software

Linux typically includes many utilities that were developed by the GNU organisation.

The following section will briefly explain the history of the how the development of Free Software has led to the development of Linux, as we know it today.

Using Linux to describe an Operating System is incorrect, Linux is the kernel not the complete Operating System. The kernel is responsible for the allocation of resources in an Operating System, it allows processes to utilise the hardware of a computer.

The correct way of describing Linux as an Operating System is

GNU/Linux. Since the Operating System consists of Linux as the kernel, and many other utilities (most of which were created with the assistance of the GNU project). In the rest of this document we will use "Linux" to describe the Operating System, since that is how most people refer to it today.

The GNU Project

Richard M Stallman started the Free Software Federation in 1984, yet that is not where our story starts.

When Stallman started work in the MIT's Artificial Intelligence Laboratory in 1971, he found a community of developers who shared software that they had written with each other, other learning institutes and companies. As Stallman indicates on the FSF website, the sharing of software was not new or unique to the MIT AI Laboratory community, it is as old as computers. Just as cooking recipes are shared, so were software applications.

In the early 80's the MIT hacker community started to disintegrate, a new computer system (a PDP-10) with a proprietary Operating System that hastened the collapse of the AI Lab community. To work with the software on the new system, Stallman had to sign a Non-disclosure agreement with the company who created the PDP-10. Bear in mind that this was to use the executable files for the software, these are not human readable, one needs the Source Code of the application to truly understand what it does, how it does this, without this it is nearly impossible to modify an application to better suit your needs.

Stallman was not willing to accept an agreement that would mean that he would not be able to help his fellow developers, he saw it as actively hindering other people from being able to do their work. Stallman tells of an incident that occurred to him while he was working in the AI Lab in MIT, where the software that they used to control their printer in the lab, lacked a few key features. Stallman was refused access to the source code for the printer's program because the company who created the printer and its software did not want to allow anybody to see how it worked. They did this by having their developers sign a non-disclosure agreement.

Stallman had to make a decision to either become one of the developers who were forced not to help each other or to stop developing or to devise a way where he would be able to recreate a community where people helped each other to develop better applications. He realised that he would first need a free Operating System, as without an Operating System computers cannot function. This was a very brave step, designing a new Operating System is no small effort.

GNU is born

Stallman had made the decision to develop a free Operating System on 27 September 1983. He decided to design it so that it would be compatible with Unix, then the most stable and widely used Operating System, so that Unix users could easily use it, and so that applications could be easily transferred to the new Operating System (a process referred to as 'porting').

Following a Hacker tradition which uses recursive acronyms, the term GNU (pronounced "guh-noo") was born. This stands for *G*NU is *N*ot *U*nix.

Stallman started by developing a compiler, compilers are used to change the human-readable Source Code into Machine code. The Operating System needs this machine code to be able to run applications. This proved difficult to do and in reality it took him a few years to complete the compiler.

Stallman decided to work on a text editor, which he called GNU Emacs. Many people started to take an interest in Emacs and wanted to use it. He released GNU Emacs on a FTP server, but not everybody had access to the Internet, this was 1985 remember. To get his Emacs to the people who wanted to use it Stallman started a software distribution company that would mail people copies of the software for a small fee. This was a precursor to the many businesses that exist today that make a profit by redistributing Linux.

People started to join Richard Stallman in creating the GNU system in 1985, to fund their work they founded the Free Software Foundation, a tax-exempt charity that would create funds by distributing the software that the GNU had created.

By the time Linus Torvalds started working on his Operating System Kernel in 1991, the Free Software Federation had already written or helped to write a wide range of software distributed as Free Software.

The birth of the Linux kernel

Linus Torvalds was a student of the University of Helsinki when he announced on the 25th of August 1991 that he was busy developing a free Operating System.

At the time the only Operating System that made its source code available was MINIX. An Operating System developed by Professor Andrew S. Tanenbaum to teach his students the inner workings of an OS. MINIX was very limited and could only work on hardware based on the Intel 8086 framework. MINIX was also not Open Source, it had to be licensed, though the Source Code was available to licensed users. When Linus Torvalds started working on the kernel that would become Linux it was the start of the Internet boom, and Linus quickly got help from developers around the world, debugging the code and offering solutions to issues they found.

The Linux kernel was released under the GNU GPL License. Which allowed anybody to download the source files, modify them and use them in their own projects.

The Internet boom allowed many people to continue work on the project; new versions of the kernel were released often (sometimes even weekly). This had a number of benefits, perhaps the most notable is the fact that the kernel improved significantly in a very short period of time, another is that with this many releases it appealed to a wide range of users; those that wanted to be on the leading edge and worked on the development used the latest version of the kernel, whilst those want more stability used older versions. The Linux Kernel grew in popularity quickly.

The GNU's kernel

The GNU has been working on its own kernel for some time now. It is called the GNU HURD.

The official definition for HURD is:

> 'Hurd' stands for 'Hird of Unix-Replacing Daemons'. And, then, 'Hird' stands for 'Hurd of Interfaces Representing Depth'. We have here, to my knowledge, the first software to be named by a pair of mutually recursive acronyms.[18]

Its is believed that it will be released very soon (since it is Free Software you can already download and use it but it is not yet ready to be used in a production environment)

Benefits of the Open Source Development methods

When one examines the developmental history of the Linux kernel the point that stands out is the extraordinary improvements made to the Linux kernel in such a short time.

This success is attributed to the knowledge of the developers who contribute to the project, and the development model used by Linus Torvalds.

Proprietary software is usually developed by small teams working closely together and products are only released once the developers believe that they have found all the problems in the code, this results in long development times and, as we all know, code that still contains many problems.

The model used by the Open Source community is more open than that, it uses many developers, often people who have never met each other physically, and releases code often. This is done because Open Source developers depend on their users to help them improve the code.

[18] From:www.gnu.org [http://www.gnu.org/software/hurd/hurd.html]

The Cathedral and the Bazaar

The Cathedral and the Bazaar(CatB) is paper written by Eric S Raymond (ESR), which examines the differences between the development models used by the Open Source community (the Bazaar) and the one used by Proprietary software companies (the Cathedral). Raymond first presented The Cathedral and the Bazaar (CatB) on 21 May 1997 at the official "Linux Kongress" (sic). The latest revision was released on the 11th of September 2000.

> In The Cathedral and the Bazaar, Eric Raymond examines the Linux kernel development model and comes to the conclusion that not only does it work, but that it is perhaps the only economical way of developing large systems that satisfy most of the people. He also considers and responds to the arguments raised by people who prefer the traditional Cathedral style of development.
>
> —ESR; Cathedral and the Bazaar

This is Raymond's abstract of the work:

> I anatomize a successful open-source project, fetchmail, that was run as a deliberate test of the surprising theories about software engineering suggested by the history of Linux. I discuss these theories in terms of two fundamentally different development styles, the "cathedral" model of most of the commercial world versus the "bazaar" model of the Linux world. I show that these models derive from opposing assumptions about the nature of the software-debugging task. I then make a sustained argument from the Linux experience for the proposition that "Given enough eyeballs, all bugs are shallow", suggest productive analogies with other self-correcting systems of selfish agents, and conclude with some exploration of the implications of this insight for the future of software.
>
> —ESR; Cathedral and the Bazaar

Who is Eric Raymond?

Eric S. Raymond is the president of the Open Source Initiative (OSI)

Raymond was involved in Unix and Open Source development for the GNU before Linus Torvalds released the Linux kernel, which made him used to the Cathedral style of development, small teams working closely together on a project and only releasing the application once it was close to perfection.

Torvald's methodology of development (releasing early and often, delegating as much of the work as possible and being open to almost all suggestions) seemed

strange to Raymond.

> No quiet, reverent cathedral-building here rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles.
>
> —ESR; Cathedral and the Bazaar

Raymond wanted to learn why the model used in the development of Linux worked so well, and he worked hard to learn more about it. In 1996 he had the chance to apply Linus's methods in a project he had just started. He needed an email client that would allow him to automatically download email from the community Internet Service Provider he had helped to start. Raymond had tried a few of the existing client applications, but none did exactly what he wanted it to do, so he did what all good hackers do, he decided to develop a new POP client. This was the perfect opportunity for him to also test the bazaar style of development. The application that was developed is called fetchmail and is still used extensively today.

A Summary of "the Cathedral and the Bazaar"

In CatB Raymond lists 19 reasons why he believes the Bazaar development model works well.

“I will discuss these, as I understand them.” Riaan Bredenkamp

1. Every good work of software starts by scratching a developer's personal itch.

   Raymond needed a POP email client that would allow him to automatically fetch mail from his ISP (Internet Service Provider), the clients that were available did not have the necessary capabilities Raymond needed. In the Open Source world this is very true. If there were a need for something a developer would have all the resources needed to develop a better application. A developer is also sure to have the support of many other people who have felt the same need for a better application. Applications are developed for the love of the art, not for any other reasons.

2. Good programmers know what to write. Great ones know what to rewrite (and reuse).

   Because you are dealing with Open Source software the source code is always available. It would be senseless to re-design the wheel every time you need a mode of transportation, so why do it when you are developing an application? Raymond looked at 9 POP mail clients and chose 'fetchpop' by Seung-Hong

Oh, who included some of the changes that Raymond had written, in version 1.9 of fetchpop.

3. "Plan to throw one away; you will, anyhow." (Fred Brooks, The Mythical Man-Month, Chapter 11)

   Raymond wrote the code that allowed fetchpop to do what he wanted it to do, but was not satisfied with the total product. While searching for mailclients he could modify he had come across Carl Harris's popclient. Though fetchpop did what he wanted it to do Raymond had two reasons for switching to popclient, popclient supported multiple protocols including IMAP (Internet Mail Access Protocol), which is more powerful than POP3. He also had another more theoretical reason to change, and that was to throw the first code that he had written away. This was one of the ideas that were often used by the people developing the Linux kernel.

4. If you have the right attitude, interesting problems will find you.

   Carl Harris, the author of popclient, had lost interest in the project and he and Raymond decided that Raymond should take responsibility of popclient.

   Raymond suddenly was no longer writing a few modifications for an existing mail client, now he was maintaining a mail client and he was full of ideas that would lead him to make many changes to popclient.

5. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.

   It is important for developers to realise when it has become time for someone else to take responsibility for his or her project. Once Raymond had proved to Harris that he was the correct person for the job he graciously handed the reins over to Raymond. This attitude assures the continued development and growth of a project.

6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging

   When Raymond took over the popclient application, he did not only inherit the management of the code but also the users of popclient. In the Linux development model, users have the ability to be co-developers (if used correctly). This is one of the main reasons why the Linux kernel has been as successful as it has.

7. Release early. Release often. And listen to your customers.

   Previously most developers felt that this was a bad policy for bigger projects.

They felt that releasing buggy software would cause the users of the software to give up on the product.

Yet this was not the case with the Linux kernel. Linus Torvalds often released a new version of the Linux kernel more than once a day! This was what kept his users satisfied and stimulated. 'Stimulated by the prospect of having an ego-satisfying piece of the action, rewarded by the sight of constant (even daily) improvement in their work. ' ESR; CatB

8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.

   Raymond had dubbed this the 'Linus Law'. Raymond believes that this is the core difference between the cathedral and bazaar development models.

   In the cathedral model it often takes months for the developers to be satisfied that they had eliminated most of the problems in the program.

   In the Bazaar model, you have so many people looking and using the code that most bugs are found quickly. Even if this happens at the expense of having a major problem in a few of the released versions, the benefits of rapid development are still enough to justify this.

   For the users who did not want to use the latest version of the Linux kernel, Torvalds also made available the older versions in which most known problems were dealt with. This meant that a wide range of people would use the kernel, not just the few people that wanted to be on the bleeding edge of the technology.

9. Smart data structures and dumb code works a lot better than the other way around.

   Raymond started maintaining popclient by first rewriting it, he did this for two reasons;

   a. To understand how the application works;

   b. And also to change the way it was coded so that the data structures were more robust.

   In other words he redesigned the way that the different protocols were expressed in terms that the kernel and thus the hardware could understand.

10. If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.

Raymond had decided to develop his new mail client using the Linux kernel development model. He did this by doing the following:

> I released early and often (almost never less often than every ten days; during periods of intense development, once a day). I grew my beta list by adding to it everyone who contacted me about fetchmail. I sent chatty announcements to the beta list whenever I released, encouraging people to participate. And I listened to my beta-testers, polling them about design decisions and stroking them whenever they sent in patches and feedback.
>
> —ESR; CatB

Raymond was amazed at the response he got from the users of the application.

> I got bug reports of a quality most developers would kill for, often with good fixes attached. I got thoughtful criticism, I got fan mail, I got intelligent feature suggestions.
>
> —ESR; CatB

11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.

    One of the users of popclient sent Raymond some code that would allow it not just to be a local mail delivery agent (just fetch mail and make it available on a workstation), but also enable it to be a Mail Transport Agent (MTA). By using SMTP (Simple Mail Transfer Protocol) it would do the job better and giver it more capabilities. Thus one user's ideas allowed the project to grow fundamentally.

12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong. By using SMP and changing popclient to act as a Mail Transfer Agent rather than a Mail Delivery Agent Raymond could remove some of the most redundant features of popclient and make it easier to use and more stable.

    > ...the benefits proved huge. The [clumsiest] parts of the driver code vanished.
    >
    > —ESR; CatB

13. "Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away." (Antoine de Saint-Exup?)

    Raymond states that when your code evolves to be better and simpler, then you know it is better.

> There is a more general lesson in this story about how SMTP delivery came to fetchmail. It is not only debugging that is parallelizable (sic); development and (to a perhaps surprising extent) exploration of design space is, too. When your development mode is rapidly iterative, development and enhancement may become special cases of debugging-fixing 'bugs of omission' in the original capabilities or concept of the software.
>
> —ESR; CatB

Popclient has changed to such an extent that Raymond believed it was time to change its name, fetchmail was born.

14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.

    Raymond started to see that fetchmail could become what is termed a 'category killer', software that causes all others in that field to be forgotten. To achieve this, fetchmail would have to be able to do things he never set planned for it to do.

    > I'd have to write not just for my own needs, but also include and support features necessary to others but outside my orbit.
    >
    > —ESR; CatB

    Whilst at the same time making sure the program stayed simple and robust.

15. When writing gateway software of any kind, take pains to disturb the data stream as little as possible - and never throw away information unless the recipient forces you to!

    By following this rule Raymond was able to satisfy another demand from his users (8-bit MIME support). In the ASCII character set the eighth bit is not used, and another developer might have been tempted to use this bit to transport data internally in the program, Raymond was not, and thus was able to support 8-bit MIME without having to rewrite major parts of the code.

16. When your language is nowhere near Turing-complete, syntactic sugar can be your friend.

17. A security system is only as secure as its secret. Beware of pseudo-secrets.

    Some of the users asked Raymond to encrypt the password in the control file of the application, so that people who were looking at the control file would not be able to see the password in plain text. Raymond points out that anybody who has the permissions to read the control file would be able to find out, from the

code, which decoder to use to read the password. So security was not really enhanced, the user would just have been lulled into a false sense of security.

18. To solve an interesting problem, start by finding a problem that is interesting to you.

    It is also obvious that if you have developed something that is interesting to you and solves problems for you, that it would play the same role for other people.

19. Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

    > In The Mythical Man - Month, Fred Brooks observed that programmer time is not [interchangeable] ; adding developers to a late software project makes it later.
    >
    > —ESR; CatB

    This has become known as Brook's Law, but if it was true, how could the Linux kernel have been such a success?

    > Gerald Weinberg's classic The Psychology of Computer Programming supplied what, in hindsight, we can see as a vital correction to Brooks. In his discussion of "ego less programming", Weinberg observed that in shops where developers are not territorial about their code, and encourage other people to look for bugs and potential improvements in it, improvement happens dramatically faster than elsewhere"
    >
    > —ESR; CatB

    Clearly the Bazaar method needs to use this ego less method of development if it is to succeed. Something in which Linus Torvalds exceeds;

    > We may view Linus's method as a way to create an efficient market in egoboo[19] - to connect the selfishness of individual hackers as firmly as possible to difficult ends that can only be achieved by sustained cooperation.
    >
    > —ESR; CatB

Conclusion

Perhaps in the end the open-source culture will triumph not because

[19] 'Egoboo' is a word that ESR uses in Cathedral and the Bazaar. I believe he means that open source development is successful partly because the individuals who are working on the project enjoy having their ego stroked. And because anybody can see their code, everybody will know how good they are

> cooperation is morally right or software "hoarding" is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the closed - source world cannot win an evolutionary arms race with open - source communities that can put orders of magnitude more skilled time into a problem.
>
> —ESR; CatB

The book has been released under the Open Publication License v2.0 and can be read online on this [www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/] page

Why is Free Software not used extensively in the Enterprise environment?

Why has Free Software in general, and Linux specifically, not been as widely used in the enterprise environment as the proponents of it expected? Developers of proprietary software would have you believe that Linux is not suited to the corporate environment. But this is not true, the next section attempts to highlight why it is that Linux has not yet taken over from Microsoft, as the Operating System of choice for home and corporate users.

Introduction

Linux and Free Software must be a marketer's nightmare, Linux is perceived to be an Operating System used only by highly technical people, who hardly ever leave their homes or offices. Worse, the people who develop Linux call themselves hackers, so how do you sell a product to companies when people think that only 'uber'-geeks can use it, and that they use it to break into bank accounts via the Internet?[20] The truth of the matter is that home users and more importantly CEO's (Chief Executive Officer) and CTO's (Chief Technology Officers) have the perception that Linux is difficult to use and that would not be possible to use it in their environment.

When you examine how Microsoft has marketed their products and compare that to the way Linux as been marketed to the world, one begins to understand why Windows is the Operating System of choice, instead of Linux.

Traditionally Linux has been marketed to the business world, from the bottom up. Since it was only the technical people who knew about Linux, and how to use it, they were the people telling their bosses about how stable and cost-effective it is compared to the products offered by Sun, IBM or Microsoft. Unfortunately few

[20] The understanding of the man on the street of the term "hacker' is incorrect. Developers have been calling each other Hackers since the 1960's. In this context a Hacker refers to a developer who is able to improve a program in an intelligent and elegant manner. When using to people who gain unlawful entry into computer networks you should use the term "Cracker'.

technically minded people are also good businessmen, or know how to communicate their ideas to people who have the power to make decisions that will affect the company.

Microsoft, arguably the most successful software company around today, has marketed their products to the Chief Executive Officers, Chief Technology Officers and Chief Financial Officers. Not to the people who would use it, but rather the people who may not necessarily have the knowledge required to make a sound technical decision, but the people who are able to make financial decisions.

When deciding whether or not to use Linux in a business environment, one needs to make a distinction between an Operating System for a server, and an Operating System for a desktop. Whilst all Linux proponents would agree that it is very well suited to the server environment, some would say that when it comes to desktop systems, meant to be used as workstations, Linux may not yet be polished enough to replace Microsoft's products. Though, recent versions from SUSE and Red Hat are very close to being perfect for the desktop.

For Linux to gain more acceptance in the Business world, it will need to be marketed in the correct way.

CEO's would need to be made aware of why Linux is a viable option to use in the business environment, which is what the next section attempts to highlight. I will not be able to turn you into a marketer , or a business person, but I will attempt to list the reasons why Linux should be used.

Cost

Surely this is one of the main draw cards that Linux has over its competitors. During the IT industry boom in the late 1990', Information Technology seemed to promise unbelievable growth in profits and productivity. After the .com bust in 2000 many companies have slashed their IT budgets drastically. IT just did not deliver what it promised.

Today, businesses want even more out of the IT infrastructure, but they are more cautious when it comes to spending. Unlike the products from companies like IBM and SUN, Linux can run on almost any hardware architecture, you can use Linux to run your file server using the normal Pentium/AMD architecture. Of course it can run on other more obscure architectures, you can even run it on a Xbox gaming system. (Though that is not so strange once you know that the Xbox is just an IBM PC that is meant to be used for gaming exclusively. visit this [xbox-Linux.sourceforge.net] page for more information on how to install Linux on a XBox) What is impressive is that people are creating clustered computer systems from these 'hacked' Xboxes. They are using the Xbox, because it uses good-quality hardware, is relatively inexpensive and is very quiet.

There are Linux distributions like Red Hat Fedora, Debian, and Gentoo that you can

use completely free of charge, and there are distributions that require the user to purchase a license, for example Red Hat Enterprise 3 and SUSE Linux Enterprise 8. The advantage of buying a license is that you get support from the company who has created the Linux Distribution, including regular security updates and bug fixes. With the distributions that are free of charge, you depend on the community of users of that distribution for the security updates and bug fixes. Admittedly, this is a very enthusiastic community and these fixes are made available before most people know that they exist, but this is not a risk that many companies are willing to take. They would rather pay somebody for guaranteed service than depend on no-cost services.

Productivity

Free Software is renowned for its stability, which translates to better uptime (the time between rebooting the system). Many commercial web-hosting companies use Free Software to run their servers, and to deliver the pages to Internet users.

Security

Security is another reason why businesses would benefit from switching to Linux. Every year millions of dollars are lost worldwide by damage caused by Trojans, worms and viruses that affect Microsoft products. These programs exploit features in Microsoft that (it seems) Microsoft is unwilling to fix, since it would mean that Microsoft loses some of its ease of use. In Linux a much stricter security policy is implemented than on Microsoft Windows. In Linux the root user needs to allow any and all programs that want to run on the system. This can only be done by the root user (the administrator of a Linux machine).

In Microsoft systems, programs are allowed to run without any input from the user. In other words a malicious program can install itself on a Microsoft system, and run itself without the user of that system even knowing about it.

A classic joke: "Heard about the Linux virus? It works on the honor system. First it asks you to please e-mail it to all your friends, then it asks you to please log back in as root so it can tell you how to trash your system."[21]

Maturity

As discussed earlier, Linux is based on the Unix system, which has been used by enterprises since the early 1980's. With the release of version 2.6 of the Linux Kernel early in 2004 Linux has evolved even further.

(Joseph Pranevich has written an exhaustive analysis of what capabilities the latest Linux kernel bring to the Linux Operating System, read it at:

[21] As I am writing this, the MyDoom virus is being regarded to be the fastest spreading of all time. Three day after its release the mi2g Intelligence Unit (mi2g.net), a digital risk firm, has said that it has caused more than 20 billion dollars worth of damage. From this [http://thewhir.com/marketwatch/myd012904.cfm] page

http://www.kniggit.net/wwol26.html It is also included in the appendix Appendix B [167])

Support

Now that IBM and Novell have thrown their weight behind Linux, one can no longer say that there isn't a major company who will make support available for Linux servers and workstations. Many businesses would rather pay a license fee and be sure that support for their IT infrastructure is just a phone call away.

On the 13th of January 2004 Novell finalised its acquisition of SUSE Linux ,Press Release [http://www.novell.com/news/press/archive/2004/01/pr04003.html?sourceidint=susehomebottom_en-us] and this means that there is now a multi-billion dollar company offering support for Linux on an Enterprise level, from servers to workstations.

Does Linux meet industry standards i.e. POSIX, IEEE, X/OPEN ?

When choosing a Operating System the informed person would want to know it follows certain standards. The Portable Operating System Interface standard was created to ensure that Unix-like Operating Systems use applications that look and feel similar to those used on other POSIX compliant Operating Systems.

If an Operating System is POSIX compliant, you can be assure of the following:

1. It is an acceptable level of quality,
2. It will have the same look and feel as other products on the market, this makes training and support an easier task to source and implement,
3. It has industry input that has been built up over time and experience of other technical learning curves when building and supporting other operating systems.

The POSIX standard is maintained by the Portable Application Standards Committee (PASC) of the IEEE organisation. The standard is heavily influenced by Unix - and in the latest revision now merges with The Open Group's Base Specifications (LSB) which comprise the core of the Single Unix Specification.

The POSIX standard was developed so that people developing Operating Systems could reference one standard so as to ensure that different Operating Systems would be able to interoperate with each other. Linux is not completely compliant with POSIX, a draft document has been released by the Open Group detailing conflicts between POSIX and their Linux Standards Base (LSB). This can be viewed at: http://www.opengroup.org/personal/ajosey/tr28-07-2003.txt

The Open Group's LSB certification is another standard that has been set against which developers can test their products. Visit this [http://www.opengroup.org/lsb/cert/docs/faq.tpl] page to find out more about this certification. The LSB is a standard created to ensure greater conformity between the different Linux Distributions, whilst the POSIX standard is meant to ensure greater conformity between all Unix-like operating Systems.[22]

Exercises and Quiz

Please note that if you cannot answer these questions then maybe you should read the introduction again.

1. Why would it be essential that you have an understanding of the business benefits and aspects to Linux, Open Source and Free Software?

2. Why would the conventional business world frown upon using these products?

   a. Conventional business in a non computer-related sense (e.g insurance, furniture manufacturer).

   b. Conventional business in a computer related sense (e.g computer manufacturer, software development company).

3. Are Unix and Linux the same product?

4. Itemise a few of the players in the politics surrounding the use and development of Open Source and Free Software in a business related world. (Halloween papers?)

5. Do you understand how this course works and what equipment you will need to be able to complete this course?

[22]

Setup of Linux Emulator for Fundamentals Course

In order to make this course more accessable to students, we decided to provide a Virtual Linux Environment in which you could experiment. We built a mini-installation of Debian Linux within the Bochs IA-32 Emulator. Bochs can emulate the Intel x86 CPU, common I/O devices, and even a custom BIOS.

This means that you can have a fully working Linux system running on your Windows desktop machine. You should have access to the self-installing win32 executable, either on CD or via the course website.

You can download Bochs with the Debian GNU/Linux image mathew West created from this link [http://learnlinux.tsf.org.za/moodle/resources/bochs-2.1-debian.exe] .

Debian GNU/Linux, Bochs and NSIS are all open source products, licensed under the GPL.

Debian: http://www.debian.org/

Bochs: http://bochs.sourceforge.net/ http://bochs.sourceforge.net/

NSIS: http://nsis.sf.net/

Chapter 4. Essentials

What is Linux?

An operating system is composed of two major parts; these parts are known as the "kernel" and the "userland".

The kernel is responsible for handling communication between the physical hardware and the software running on the machine.

The "userland" is comprised of system utilities and user applications. These include editors, compilers and server daemons. System utilities allowing you to maintain, monitor and even upgrade your system are also included.

The phrase "Linux operating system" is a misnomer, as Linux is a kernel, and requires additional software in order to make it an operating system.

A Linux distribution is comprised of the Linux kernel, and a collection of "userland" software. The software is usually provided by the FSF[23] and GNU[24] organisations, as well as many private individuals. Some of it even originates from UCB's[25] BSD[26] Unix operating system.

There is some confusion over whether the word should be written as "Unix" or "Unix". Both forms are popular and are used interchangeably. Dennis Ritchie says that the all-caps spelling originated from CACM's 1974 paper, "The Unix Time-Sharing System". Apparently because "we had a new type setter and troff had just been invented and we were intoxicated by being able to produce small caps." Dennis Ritchie feels like it should be spelled "Unix", as it is a word and not an acronym. Therefore, this is the format that we will use in this document.

Some commercial Linux distributions even include commercially developed software, often unique to that particular distribution. An example of this would be SuSE Linux's "Openexchange" Server™.

There are many Linux distributions that are available. All of them use the Linux kernel, but they usually differ in what software is available as part of the "userland" how that software is managed and packaged.

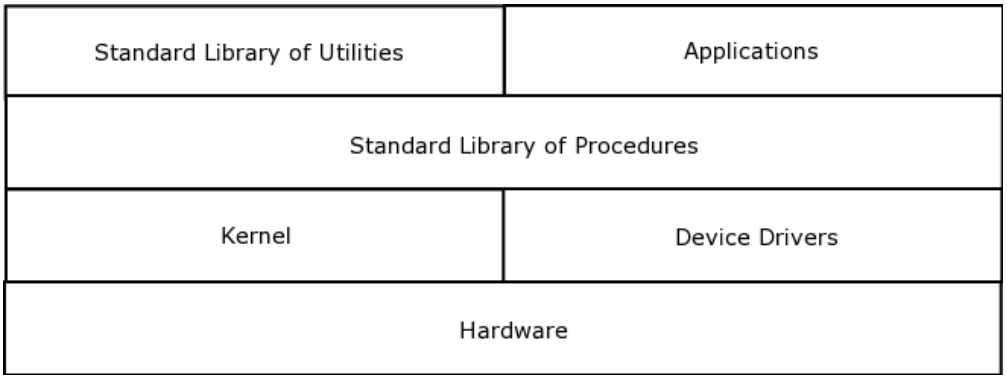
[23] Free Software Federation -- http://www.fsf.org/ [http://www.fsf.org/]
[24] GNU's Not Unix -- http://www.gnu.org/ [http://www.gnu.org/]
[25] University of California, Berkeley -- http://www.berkeley.edu/ [http://www.berkeley.edu/]
[26] Berkely Software Distribution -- http://en.wikipedia.org/wiki/Berkeley_Software_Distribution [http://en.wikipedia.org/wiki/Berkeley_Software_Distribution]

Unlike most Unix operating systems, which are based on previous versions of Unix, ultimately all leading back to the original "Unix System" from Bell Labs, the Linux kernel was written from scratch. However, Linux based operating systems follow and implement the Unix paradigm closely enough, that the bulk of this section of the course would apply to both Linux and Unix variants.

Structure of a Linux Based Operating System.

A Linux based operating system is structured in much the same way as other operating systems are structured.

Figure 4.1. Operating Systems Layers



Hardware

This is the physical equipment of which your computer is composed; this includes things like your keyboard and mouse, your video card and monitor, as well as your network card, if you have one. Other not-so-obvious pieces of hardware are your CPU and the RAM in your system.

Kernel

The Linux kernel acts as the interface between the hardware mentioned above, and the rest of the operating system. The Linux kernel also contains device drivers, usually ones, which are specific to the hardware peripherals that you are using.

The kernel is also responsible for handling things such as the allocation of resources (memory and CPU time), as well as keeping track of which applications are busy

with which files, as well as security; and what each user is allowed to do on the operating system.

Standard Library of Procedures

A Linux based operating system will have a standard library of procedures, which allows the "userland" software to communicate with the kernel. On most Linux based operating systems, this library is often called "libc".

Some examples may include calls to ask the kernel to open up a file for reading or writing, or to display text on the display, or even read in keystrokes from the keyboard.

Standard Utilities and User Applications

A Linux based system will usually come with a set of standard Unix-like utilities; these are usually simple commands that are used in day-to-day use of the operating system, as well as specific user applications and services. This is typically software that the GNU Project has written and published under their open source license, so that the software is available for everyone to freely copy, modify and redistribute.

Some examples would be the commands, which allow users to edit and manipulate files and directories, perform calculations and even do jobs like the backups of their data.

Lateral thinking with further details on the Operating System Simone Demblon

How These All Work Together

One of the benefits of Unix, and thus also of Linux, is the fact that it's designed to be a multi-user and multi-tasking operating system - in other words more than one user can be working on the same system at the same time - via different consoles, pseudo and dumb terminals, or perhaps even by scheduling some of their tasks to occur while they're not at their keyboard. This is an age where sharing information has become paramount and therefore this type of operating system can only be an advantage in a networked environment.

However, most PCs are single CPU systems, and, technically, the CPU cannot handle more than one task at a time - as is implied by the word "multi-tasking". The trick to multi-tasking is therefore part of the operating system, rather than the system hardware.

The kernel divides up the time alloted to tasks; these are called "time slices". The

kernel is responsible for running the tasks on the CPU, saving their state and removing them, and then replacing them with the next task for it's allocated "slice of time". This gives the impression that the system is performing many tasks concurrently, but it is in fact performing small parts of each task, one at a time, in quick succession.

The process whereby the kernel swaps tasks on and off the CPU is known as "context switching". Each task has its own environment, or context, which the kernel has to remember in order to fool the process that it is running on the CPU all on its own without any interruptions.

On machines with more than one CPU, a technique called Symmetric Multiprocessing (SMP) is used to do the time slicing over multiple CPU's. Obviously, with this system, the tasks are actually been done concurrently, although it is rare that a specific CPU is assigned to a single specific process.[27]

Process Flow:

When a user runs a standard utility or application, that software makes a call to the kernel, via the standard library of procedures, requesting system resources, such as the contents of files or the input being feed in via the keyboard. The kernel in turn then queries the actual hardware for this information, and then returns it, again via the standard library of procedures layer.

This layering is what provides the system with a level of stability. If your application crashes for some reason, it is seperated enough from the kernel to prevent it taking down the whole system.

Logging into a Linux System

Login

Once you have your Linux system up and running, you will be presented with a prompt asking for your username. This is often referred to as the login prompt.

```
Debian GNU/Linux
3.0 debian tty1
debian login:_
```

Once you've entered your username, you will be prompted for a password:

```
debian login: guest
```

[27] It is possible to do this though; such a technique is called "CPU affinity"

```
Password:_
```

Like Unix, Linux is case sensitive, so you need to make sure that both your username and password are entered in the correct case.

You will notice that your password is not echoed to the screen as you type it; this stops someone from being able to read over your shoulder and make a note of your password.

A good rule of thumb is to keep usernames in all lowercase, as this keeps things simple.

However, passwords should be made as difficult as possible to guess; preferably they should consist of both upper and lower case letters, as well as numbers and punctuation marks.

Traditional Unix systems have an 8 character limit on usernames and passwords. However, Linux based operating systems have a limit of 256 characters. Most Linux distributions can also be configured to operate in "legacy mode", using 8 character usernames and passwords, and so allow better interoperability with existing Unix installations.

Once you've typed in your password hit enter and you should be greeted with a welcome screen and you should be presented with a shell prompt and a flashing cursor.

If you're using the Virtual Linux Environment provided with this course, then your login name will be "student" and your password will be "student".

```
Debian GNU/Linux 3.0 debian tty1
debian login: student
Password:

Linux debian 2.2.20-idepci #1 Sat Apr 20 12:45:19 EST 2002 i686
    unknown Most of the programs included with the Debian GNU/Linux syst
    freely redistributable; exact redistribution terms for each program a
    described in the individual files in /usr/share/doc/*/copyright Debia
    GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted
    applicable law.
student@debian:~$ _
```

Once you've logged into the system for the first time, it is usually a good idea to set your password to something new, one that will be difficult for other people to guess.

The command to do this is "passwd" (short for "password"). This command should allow you to set your password on any Unix-like system.

You will be prompted for your old password, to ensure that it is really you at the keyboard, and you will then be prompted twice for your new password. This ensures that you don't make a typo!

```
debian login: student
    Password:
Linux debian 2.2.20-idepci
    #1 Sat Apr 20 12:45:19 EST 2002 i686 unknown Most of the programs included
    with the Debian GNU/Linux system are freely redistributable; the exact
    distribution terms for each program are described in the individual files
    in /usr/share/doc/*/copyright Debian GNU/Linux comes with ABSOLUTELY NO
    WARRANTY, to the extent permitted by applicable law.
student@debian:~$  passwd
Changing password for student (current) Unix password:
Enter new Unix password:
Retype new Unix password:
passwd: password updated successfully
student@debian:~$ _
```

Once you've successfully changed your password, you can type the 'exit' command to exit out of the session.

```
Debian GNU/Linux 3.0 debian tty1
debian login: student
    Password:
Linux debian 2.2.20-idepci
    #1 Sat Apr 20 12:45:19 EST 2002 i686 unknown Most of the programs included
    with the Debian GNU/Linux system are freely redistributable; the exact
    distribution terms for each program are described in the individual files
    in /usr/share/doc/*/copyright Debian GNU/Linux comes with ABSOLUTELY NO
    WARRANTY, to the extent permitted by applicable law.
student@debian:~$ passwd
Changing password for student
(current) Unix password:
Enter new Unix password:
Retype new Unix password:
passwd: password updated successfully
student@debian:~$ exit &lt;enter&gt;
```

The Password File

In the previous section, you saw that the system was able to validate your identity based on your username and password. In this section, we will look at the file which is commonly used to store this information.

One of the most important files on any Unix-like system is the password file; this

file is located in the "/etc/" directory, and is called "passwd".

The file originated on Unix 7th Edition, and maintains the same format to this day: 7 colon-delimited fields. These fields are, in order:

- username
- password placeholder
- user id
- group id
- GECOS field
- home directory
- shell

The following is an excerpt from the password file:

```
root:x:0:0:root:/root:/bin/bash
```

Table 4.1. /etc/passwd

user Name	Password Placeholder	User ID	Group ID	Gecos Field	Home Directory	Shell
root	x	0	0	root	/root	/bin/bash

Your "*user id*" is a numeric identifier, which the operating system uses to identify which files belong to you. The system always thinks of you in terms of a number! It uses the passwd file to convert the number into a more human-friendly form; your username. This *username* is a name that you have chosen or that has been given to you by the system administrator and is the name that you will use to log in to the system.

Your "group id" is very similar. A Unix group may contain none, one or more users, who will then be able to access the files and directories owned by that group, based on that groups permissions as discussed above. This is useful for sharing files between two people, as a file can only have one owner.

Most modern implementations make use of a concept called "User Private Groups" (UPG). This means that each user is assigned their own group, which is given the same name as their username. This user is the only member of that group.

The GECOS field was originally added to early Unix systems in order to enable interoperability with an operating system written by General Electric, called the General Electric Comprehensive Operating System (GECOS). Now the field is used to store *your full name, and possibly your room and telephone number.*

The final two fields are your *home directory*, where all your files are usually stored, as well as your choice of *command shell*.

On a traditional Unix system, an encrypted version of the password used to exist where the *password placeholder field* is now.

The password is encrypted with a one-way hash. This means that the password cannot be decrypted, but it does mean that people can try and guess your password.

The traditional encryption method was called the Data Encryption Standard (DES), but most recent versions of Unix, and most Linux distributions, default to using the MD5 (Message Digest 5) encryption method, which allows for much longer and difficult-to-compute passwords.

As computers became more and more powerful, it became feasible to try entire dictionaries of words to guess someone's password.

To counter this, the encrypted password field was moved into a separate file which only the superuser could read. Under Linux based operating systems, this file is called the shadow password file (/etc/shadow).

The superuser, or "root user" has complete control over the whole system, and is able to even override normal file permissions. Normally this login account is only used by the system administrator when doing system maintenance work.

The shadow password file contains the username and its associated encrypted password, as well as other fields which deal with password and account expiry.

The system uses the /etc/group file to determine the mapping of group names to group numbers, as well as to determine the members of each group.

The Shell Command Interpreter

The shell command interpreter is the command line interface between the user and the operating system. It is what you will be presented with once you have successfully logged into the system.

The shell allows you to enter commands that you would like to run, and also allows you to manage the jobs once they are running. The shell also enables you to make modifications to your requested commands.

Different Shell Command Interpreters

The Bourne-Again shell is not the only shell command interpreter available. Indeed, it is descended from the Bourne Shell (sh), written by Steve Bourne of Bell Labs. This shell is available on all Unix variants, and is the most suitable for writing portable shell scripts. This is discussed in depth in the Shell Scripting Course.

The default shell, which is provided with most Linux based systems is the Bourne-Again shell ("bash").

Other popular shells include the C Shell (csh), written at UCB, and so called because its Syntax is similar to that of the C language.

The TC Shell (tcsh) is an extension of the C shell.

A very popular shell on most commercial variants of Unix is the Korn Shell. Written by David Korn of Bell Labs, it includes features from both the Bourne shell and C shell.

Finally one of the most powerful and interesting shells although one that hasn't been standardised on any distribution that I've seen, is the Z shell. The zsh combines the best of what is available from the csh line of shell utilities as well as the best that is available from the bourne or bash line of shell utilities.

One relatively easy way to determine what shell it is that you are currently running is by looking at the prompt. If you are using a bourne shell or derivative, you will see a dollar sign. This is more than likely what you will see at your bash prompt at the moment. If however you are using csh, or tcsh you will see a percentage sign. If you are logged in as root or the superuser, irrespective of your shell, your prompt will normally always be a hash.

The bash prompt:

```
student@debian:~$ _
```

The tcsh prompt:

```
student@debian:~% _
```

The root prompt:

```
root@debian:~# _
```

The Command History within the shell

More modern shells allow you to access your history of commands, and automatically complete filenames and even correct mistyped commands.

- The up and down arrow keys will allow you to traverse the history of commands that you have typed in.
- The left and right arrow keys will allow you to navigate the cursor on the command which you are currently typing.
- The backspace and delete keys will delete the character behind the cursor, and underneath the cursor, respectively.
- The default mode at your bash prompt is the insert mode, which means that if you type a character somewhere on the line it will insert it at that position.
- If you then press the insert key it will then toggle the mode so that it will go into overwrite mode, which means that it will overwrite whatever character is directly underneath your cursor.
- Another useful feature is tab completion. What this means is that you can use the tab key in order to be able to complete a command, or be given a list of options of commands which match what you have typed so far.

Tab Completion:

```
student@debian:~$ pas &lt;tab&gt;
passwd paste
student@debian:~$ pas_
```

The shell is telling you that you need to make a choice between those two options. You can let it know which one by filling in one more letter ("s" or "t" in this case), and then pressing <tab> again.

```
student@debian:~$ pas &lt;tab&gt;
passwd paste
student@debian:~$ pass&lt;tab&gt;
student@debian:~$ passwd_
```

Now pressing <enter> will finally result in the command being executed.

- Another very useful feature of the bash shell, and one that I would recommend that you use often, is called "reverse case-insensitive history search". It will search through your history in reverse and case-insensitively (so it won't worry whether it was in upper or lowercase) for a command or even part of a command that you typed.

  In order to access this, you can use the shortcut key combination CTRL-R, followed by a command or a subsection of a command that you have typed and you will notice that it will go back in history to locate this command

```
student@debian:~$ &lt;ctrl-r&gt;
reverse-i-search)`p': passwd
```

The shell has now searched back into your command history for the letter "p", and the first command that has matched is "passwd", which you typed earlier. Pressing <enter> now will result in the command being executed. Alternatively, you can use the arrow keys to scroll through other options, or continue typing letters until you have a better match.

Configuring your shell environment

There are several files which will affect the behaviour of your bash shell:

- `/etc/profile`
- `/etc/bash.bashrc`
- `$HOME/.bashrc`
- `$HOME/.bash_profile`

The file where your history of commands is kept is called:

`$HOME/.bash_history`

Shell Command Processing

In this section, we will explain how the shell interprets the commands, which you give it to execute.

It is important to understand that the shell does actually interpret what you type to it.

Special Characters

What this means is that certain special characters will have to be interpreted or dealt with prior to the execution of the command.

An example is the asterisk or wildcard character. When a shell sees this character it will attempt to substitute any matching filenames in place of this wildcard character. It is important to note that this happens before the command is executed.

There are other special characters that will also be interpreted. Different shells interpret different characters in different ways. The most commonly interpreted characters are the asterisk, question mark, the various brackets, forward slashes and quotation marks.

We will learn the significance of each of these characters and the effect that they will have on the way that the shell executes your commands.

Once the shell has interpreted your command and run replacements where you have requested it, it will then check and see if the command is perhaps something that needs to be executed by the shell itself, in other words it is an internal command.

Internal Commands:

An internal command is a routine that is part of the shell itself and does not require the shell to open up an external file in order to execute it. Examples of internal commands are clear and history.

Shortcuts and Aliases:

As mentioned before, the bash shell allows you to have shortcuts, or aliases. These aliases are interpreted before executing the command as an internal or external one. For example, it is possible to configure bash to treat 'll' (two lowercase letter L's) as a shortcut to the ls (directory listing) command. This is made into a shortcut by using the alias command, which is a shell built-in command.

You can use the "type" shell built-in to determine if a command is an internal, external or an alias.

```
student@debian:~$  <indexterm><primary>type</primary>
                                                </indexterm>type type
type is a shell builtin
student@debian:~$  <indexterm><primary>type</primary>
                                                </indexterm>type passwd
passwd is /usr/bin/passwd
student@debian:~$
```

External Commands:

If the command is not an internal command, then it will be an external command. An external command is an executable file that exists somewhere on the system and that you are able to run. An example of an external command is passwd and the name of the shell itself, in our example: "/bin/bash".

If the command is not a shortcut, or an alias, and is not an internal command but is an external command, the file that is executed must be readable and executable by you.

You need to have the appropriate permissions in order to be able to run it. It also needs to exist in a directory that exists inside a directory that is inside your search path.

The search path lists all the directories in which the shell can find commands that you would like to be able to run.

On a usual Linux system, the /usr/bin and the /usr/local/bin directories will all be inside your path environment variable, as this is where the system stores your common executable files. These files are also known as binary files, as opposed to source files, hence the "bin" directory name.

So what the shell will do is that it will check in each of those directories for the existence of the command that you have typed. If it finds a file that matches the name and you are allowed to execute it, then the shell will request that the kernel load that programme into memory, and execute it; in other words, instantiate it as a running process.

Once the command execution is completed, it will return control back to the shell.

If the shell is not able to find a matching command to execute, in other words it not a shortcut, it's not an internal command and it cannot find an external programme in the path that you have provided, then it will give you an error message.

The advantage of internal commands is that they are less expensive for the kernel to execute, and so consume fewer system resources (CPU time and memory). The advantage of external commands is that they are able to be far more flexible.

For this reason, internal commands are usually often used, simple tasks; whereas external commands are often large applications.

The Shell Environment

There are several environment variables that influence the way that the shell operates and that can be used by commands that the shell executes.

These variables are set up for you during the boot-up of the system and during your log in process, mostly determined by your choice of shell. We discussed the files used by the bash shell earlier.

One variable that was mentioned before was the search path, this information is held in a environment variable called PATH.

Under the bash shell you can run the "set" command to list the current shell environment variables and their associated values. "set" is a shell built-in command.

```
student@debian:~$
<indexterm><primary>set</primary>
                                          </indexterm>set
BASH=/bin/bash BASH_VERSINFO=([0]=&amp;quot;2&amp;quot; [1]=&amp;quot;05a&amp;quot; [2]=&amp;quot;0
[4]=&amp;quot;release&amp;quot;[5]=&amp;quot;i386-pc-Linux-gnu&amp;quot;)
BASH_VERSION='2.05a.0(1)-release'
COLUMNS=80
DIRSTACK=()
EUID=1000
GROUP=student
GROUPS=()
HISTFILE=/home/student/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/home/student
HOST=debian
HOSTNAME=debian
HOSTTYPE=i386-Linux
HUSHLOGIN=FALSE
HZ=100 IFS=$' \t\n'
LINES=30
LOGNAME=student
MACHTYPE=i386
MAIL=/var/mail/student
MAILCHECK=60
OPTERR=1
OPTIND=1
OSTYPE=Linux
PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
PIPESTATUS=([0]=&amp;quot;0&amp;quot;)
PPID=225
PS1='\u@\h:\w\$ '
PS2='&amp;#62; ' PS4='+ '
PWD=/home/student
SHELL=/bin/bash
SHELLOPTS=braceexpand:hashall:histexpand:
monitor:history:interactive-comments:emacs
SHLVL=5
TERM=Linux
UID=1000
USER=student
```

```
VENDOR=intel
_=passwd
student@debian:~$ _
```

Setting A New Shell Variable Or Resetting An Existing Variable

Remember that these variables control your entire environment and as you will see they are pretty easy to change so if you are going to change one or part of your environment then please think it through to avoid problematic consequences.

If you wish to assign a value to an environment variable, the Syntax is:

```
student@debian:~$ VAR=value
```

Here, "VAR" is the variable name, and "value" is what we are storing inside of it.

Note that there is no space before or after the equal sign. The space would be assumed to be either part of the variable name or to be part of the value that you are assigning to the variable.

Exporting a variable value

Setting a variable using the Syntax above will cause the variable to only be available to the current shell instance. Usually, you want the variable to be propagated to any commands that you execute from the shell. To do this, you need to "export" the variable.

```
student@debian:~$ VAR=value
student@debian:~$ export VAR
```

You can also combine this into a single command:

```
student@debian:~$ export VAR=value
```

Let's have a look at an example of re-setting a variable and then exporting the value:

For this example we are going to use our login prompt variable called PS1, see the "set" command above.

```
student@debian:~$ PS1=&quot;newprompt $ &quot;
newprompt $ _
```

Enclose this in quotation marks to protect the spaces.

Now open a new shell and check for yourself that the prompt returns to the original prompt:

```
newprompt $ bash
student@debian:~$
```

Exit back to the original shell and export your variable PS1.

```
newprompt $ bash
student@debian:~$
student@debian:~$ exit
newprompt $ export PS1
newprompt $ bash
newprompt $
```

Now your new prompt has been exported to all subsequent shells.

Please note however that once you logout of this session your prompt will return to the default in your next log in. If you want to change this permanently change PS1 and export the value into one of your startup files. ($HOME/.bashrc or $HOME/.bash_profile)

The echo command

A useful command to query the contents of a single variable is the "echo" command.

This command simply displays any arguments that to pass to it on the screen. For example:

```
student@debian:~$ echo hello
hello
student@debian:~$
```

If you don't provide any arguments, then it will simply display a blank line. (This is

useful for providing spacing inside shell scripts.)

Do you remember that we mentioned that the shell actually interprets the commands that you give it before it executes them?

For example, in order to display the contents of a variable field as opposed to the variable name we could precede the variable name with a special character, in this case a dollar sign ($), and this will display the contents of that variable.

```
student@debian:~$ VAR=avalue      --Set the variable--
student@debian:~$ echo VAR        --To test our theory--
VAR                               --Displays the word not the value store
student@debian:~$ echo $VAR
avalue                            --Display the contents of the variable
student@debian:~$ _
```

Remembering to include the dollar sign ("$") before the variable name is very important, as illustrated below:

```
student@debian:~$<indexterm><primary>echo</primary>
                                                  </indexterm>echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
student@debian:~$<indexterm><primary>echo</primary>
                                                  </indexterm> echo PATH
PATH
```

Discussing system variables - TERM and PS1

Two other important bash shell environment variables: 'TERM' specifies the terminal type.

Depending how you log in to the system, the terminal type will either be "xterm" or "vt100".

The terminal type determines what the terminal capabilities are, such as can it do colour, does it have a speaker attached so that it can generate a beep or do you want the beeps to be visual flashes on the screen.

The terminal type also lets applications know if the terminal can handle certain types of characters. Very old terminals used to be only able to handle uppercase letters, so if you set the terminal type to one of those you would only be able to see uppercase characters on the screen.

Sometimes, being able to set the term environment variable is useful if you are connecting to a system that doesn't know where you are connecting from and you

wish to be able to specify the correct terminal type so that your display is correct.

A good terminal type to try if you're having problems with your terminal is vt100 as this works on almost all types of terminals.

Another important environment variable is PS1. This is the current prompt for the shell. The convention is that a C-style shell has a "%" prompt, where a Bourne-style shell will have a "$" prompt. The root user will have a "#" prompt; this allows you to easily see when you are a normal user or a user who has much more potentially destructive power!

Any changes that you make to the shell environment will be lost when you exit the current login session. If you wish to make the changes more permanent, you need to add the commands that you wish to run to either the system-wide /etc/bashrc file (the change would affect all bash users on the system), or to your own $HOME/.bashrc file (the changes would affect you only).

Using Shell Commands

Under Unix, and Linux, most commands are abbreviated using 2 to 4 characters; with the more often used commands being shorter and the less-often used ones being longer.

> "There are so many options to each command in Linux - get an overview of the command, that it exists at all as a tool for you to use. Working out how to use each and every nuance can be very confusing - you will get bogged down in detail instead of gaining a comprehensive overview.
>
> We are aiming for a lot of knowledge to be given and gained in this course and getting stuck on command details is not the main goal."
>
> — Simone Demblon

The man pages

All Unix and Unix-like systems come with online documentation. The most common form are man pages; man being short for "manual".

These pages are, however, not very good for teaching someone who is new to the system, but are very good as reference material.

They will document all the useful, and sometimes even obscure, switches and

features of the command line tools that you have at your disposal.

The man pages are divided into several numbered sections:

1 - General Commands
2 - System Calls
3 - Subroutines
4 - Special Files
5 - File Formats
6 - Games
7 - Macros and Conventions
8 - Maintenance Commands
9 - Kernel Interface

You will often see references such as "ls(1)"; this is referring to the page on "ls" in section 1 of the man pages.

You can use the " **man**" command to look up a page:

```
student@debian:~$ <indexterm><primary>man</primary>
                                                  </indexterm>man
man Reformatting man(1), please wait...
```

This will display the man page on the man command.

```
 man(1)
NAME
man - format and display the on-line manual pages
SYNOPSIS
  man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
  [-M pathlist] [-P pager] [-S section_list] [section] name ...

DESCRIPTION
    man formats and displays the on-line manual pages. If you
    specify sec- tion, man only looks in that section of the manual. name
    normally the name of the manual page, which is typically the name of
    command, function, or file. However, if name contains a slash (/) th
    interprets it as a file specification, so that you can do man ./foo.5
    even man /cd/foo/bar.1.gz. See below for a description of where man
    for the manual page files. [ ... ] September 2, 1995 man(1)
```

The "man" command will look for the first page it can find which matches what you've asked for. If you want to see a page in a specific section, you can specify it thus:

```
student@debian:~$<indexterm><primary>man</primary>
                                                  </indexterm> man 7 man
```

I recommend looking through the man page of each of the commands that follow, just to get a feel for what the commands can do, and what their various switches are.

You may have noticed that the man pages are displayed a page at a time - that is due to the influence of the "less" command that we will do in the next few pages of this course. This is a good example of the Unix paradigm where small tools are used together to make more complicated ones.

You can change which "pager" application is used with man(1) and other utilities by setting the PAGER environment variable in your shell.

You can use the "-k" switch to tell man to search for pages which contain specific keywords:

```
student@debian:~$ man -k bash
bash (1) - GNU Bourne-Again SHell
bashbug (1) - report a bug in bash
builtins (1) - bash built-in commands, see bash(1)
rbash (1) - restricted bash, see bash(1)
```

The "apropos" command is the functional equivalent of "man -k".

```
Syntax:
man [chapter] &lt;page&gt;
man -k keyword
apropos keyword
```

EXCERCISE:

Call up and peruse the man page for the "man" command. Now call up the man page for man, but in the "Macros" section of the manual.

The GNU Info pages

.The GNU Project distributes most of its software together with documentation in GNU Texinfo format.

This is another place where you should look for manuals and reference material for software on a Linux system.

You can access these pages by using the " **info**" command.

You may optionally specify which "info" page you wish to look at as a parameter. Unlike Linux man page, GNU Info pages allow you to use hyperlinks, much like

you are used to using in your web browser.

Inside the GNU info reader, you can use the arrow keys to move the cursor, and can use the <enter> key to select which hyperlink you wish to follow. Other useful keys are:

- q - quit
- n - next page
- p - previous page

```
Syntax:
info [page]
```

EXCERCISE:

Call up the info pages index and peruse its contents. Now call up the info page for the "bash" command.

Now call up the info page for the bash command, but this time do it directly from the command line.

the date command

What's the current date and time on the system?

```
student@debian:~$<indexterm><primary>date</primary>
                                                </indexterm> date
Thu Jan 15 16:05:07 SAST 2004
```

```
 Syntax:
 date
```

The cal command

Want to see a pretty calendar of the current month?

```
student@debian:~$ cal January 2004
      January 2004
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
```

```
18 19 20 21 22 23 24
25 26 27 28 29 30 31
 student@debian:~$ _
```

```
Syntax: cal [[month] year]
```

Exercise:

Now read the man page for the "cal" command. Can you figure out how to make it display a calendar listing for the entire year?

Try out the command "cal 2004" and see what happens.

the ls command

The "ls" command, short for "list directory contents", displays a list of files and directories within your current directory.

```
student@debian:~$ ls dataset
student@debian:~$ one.txt. two.txt
```

The "ls" command has several switches which modify its behaviour.

The "-l" (long) switch will display additional information about each of the items that it lists; these include the file permissions, owner, group, size and date of last modification.

```
student@debian:~$ ls -l
 total 2
drwxr-xr-x 2 student student 4096 Feb 19 03:10 dataset
drwxr-xr-x 2 student student 4096 Feb 19 03:10 dataset2
```

The "-a" (all) switch causes "ls" to display even hidden files. Under Linux, any file that begins with a period (.) is considered to be a hidden file. These files are not displayed in a "ls" listing unless the "-a" switch has been specified

```
student@debian:~$ ls -a
. .. .bash_history .bash_profile .bashrc dataset dataset 2
```

These files are often referred to as "dot files", and usually contain application configuration information particular to the user whose home directory they reside in.

The reason for hiding them is to free up your workspace from the clutter it creates, thus allowing you to more easily access your data files.

```
Syntax:
ls [-la]
```

Exercise:

Try combining the the flags and see what effect they have.

The pwd command

The "pwd" command, short for "print working directory" will print out the name of your current directory.

```
student@debian:~$ pwd
/home/student
```

if a command is a shell built in command, you may have to look at the man pages for the shell to find a write-up on the built-in command. (man bash)

```
Syntax:
pwd
```

Exercise:

Is the pwd command a shell built in, or an external command? Trick Question!

The cd command

You can use the "cd" (change directory) command to navigate your way around the filesystem.

There are two special types of directories.

The "." directory is an alias for your current directory, and the ".." is an alias for the parent to your current directory.

The "cd" command without any arguments will return you to your home directory from wherever you are.

```
student@debian:~$ pwd
/home/student
student@debian:~$ cd ..
student@debian:/home$ pwd
/home
student@debian:/home$ cd
student@debian:~$ pwd
/home/student student@debian:~$ _
```

You can specify the path for "cd" to change into as being either an "absolute" one, or a "relative" one.

An absolute (full) path begins with a slash ("/"), and indicates the absolute location of something on the filesystem, by specifying it from the root directory up.

A relative (partial) path does not begin with a slash, and merely indicates a location off the current branch of the filesystem. In other words, the new directory is being specified relative to the current one.

Relative path example:

```
student@debian:~$ cd ..
student@debian:/home$ ls -l
drwxrwxr-x 1 student group 16 student
student@debian:/home$ cd student
student@debian:~$ pwd
/home/student
```

Absolute path example:

```
student@debian:~$ pwd
/home/student
student@debian:~$ cd /tmp
student@debian:~$ pwd
/tmp
student@debian:~$ cd /home
student@debian:/home$ cd /home/student
student@debian:~$ pwd
/home/student
```

As you can see, both perform equivalent operations. However, relative paths are usually shorter to type

Question to make you think: Is the special ".." path a relative or an absolute one? What about the "." path?

A useful "cd" shortcut to learn is to use the dash (-) parameter.

This allows you to quickly change back to your most recent directory.

```
student@debian:~$ pwd
/home/student
student@debian:~$ cd /usr/local/src        --Using an absolute pathname
student@debian:/usr/local.src$ pwd
/usr/local/src
student@debian:/usr/local.src$ cd -
/home/student
student@debian:~$ pwd
/home/student
student@debian:~$ _
```

```
Syntax:
cd [path]
```

Exercises:

Use the cd, pwd and ls commands to explore the file system a bit.

The cat command

The "cat" command, short for concatenate, is most often used to display the contents of short text files.

```
student@debian:~$
<indexterm><primary/>
                                                </indexterm>cat /etc/pas
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync games:x:5:100:games:/usr/games:/bin/sh
man:x:6:100:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh
postgres:x:31:32:postgres:/var/lib/postgres:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
operator:x:37:37:Operator:/var:/bin/sh
list:x:38:38:SmartList:/var/list:/bin/sh irc:x:39:39:ircd:/var:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/home:/bin/sh guest:x:1000:1000:Debian
```

```
student:x:1000:1000:Student User,,,:/home/guest:/bin/bash
identd:x:100:65534::/var/run/identd:/bin/false
sshd:x:101:65534::/var/run/sshd:/bin/false
```

So why is it short for "concatenate"? Because it can output several files all at once:

```
student@debian:~$ pwd
/home/student
student@debian:~$ ls dataset
one.txt. two.txt
student@debian:~$ cd dataset
student@debian:~/dataset$ ls
 one.txt two.txt
student@debian:~/dataset$ cat one.txt
The coldsleep itself was dreamless. Three days ago they had
been getting ready to leave, and now they were here. Little
Jefri complained about missing all the action, but Johanna
Olsndot was glad she'd been asleep; she had known some of
the grownups on the other ship.

-- A Fire Upon the Deep, Vernor Vinge (pg 11)

student@debian:~/dataset$ cat two.txt
An hour's difference either way and Peregrine Wickwrack-
rum's life would have been very different.

  -- A Fire Upon the Deep, Vernor Vinge (pg 17)

student@debian:~/dataset$ cat one.txt two.txt
The coldsleep itself was dreamless.  Three days ago they had
been getting ready to leave, and now they were here.  Little
Jefri complained about missing all the action, but Johanna
Olsndot was glad she'd been asleep; she had known some of
the grownups on the other ship.

  -- A Fire Upon the Deep, Vernor Vinge (pg 11)

An hour's difference either way and Peregrine Wickwrack-
rum&#38;apos;s life would have been very different.

  -- A Fire Upon the Deep, Vernor Vinge (pg 17)

student@debian:~/dataset$ _
```

```
Syntax:
cat [file1 file2 ... ]
```

Exercise:

Now that you know how to navigate your way around the filesystem, and look at the

contents of files that you might find there, explore a bit more.

The more and less commands

What if you want to view a longer text file? You can use a utility called a pager to do this. Two of the most common ones available on Linux based operating systems are "more" and "less".

The "more" command is very simple, and only allows you to page down a line at a time using the <enter> key, or 24 lines at a time using the <spacebar> key. Hitting "q" will return you to the shell prompt.

The "less" command is the successor to "more", and has more features. You can use the arrow keys to navigate your way around the document.

You can also perform a search my pressing the "/" key, followed by the string which you wish to search for, and finally followed by the "<enter>" key.

To search for the next occurrence of the same string, you can simply press "/" followed by "<enter>". To search backwards from your current position, you can use the "?" instead of a "/".

Pressing the "h" key will display the help page for "less", which contains several more keystrokes which you may find useful.

```
Syntax:
less [file1 ...]
more [file1 ... ]
```

Now you have almost all the tools at your disposal to read all the documentation that comes with your Linux distribution!

The ps command

The "ps" command will give your a listing detailing your current "process status", listing your processes which are currently running on the system on your current terminal.

```
student@debian:~$ <indexterm><primary>ps</primary>
                                                </indexterm>ps
 PID  TTY          TIME  CMD
 2700 pts/1    00:00:00 bash
 3034 pts/1    00:00:00 ps
```

```
Syntax:
ps
```

Exercise:

1. Is the ps command a shell built-in?
2. How can you tell?

Files and Directories

Files under Linux

Each disk drive in a Unix or Unix-like system can contain one or more file systems. A file system consists of a number of cylinder groups, which in turn contain inodes and data blocks.

Each file system has its characteristics described by its "super-block", which in turn describes the cylinder groups. A copy of the super-block is made in each cylinder group, to protect against losing it.

A file is uniquely identified by its inode on the filesystem where it resides.

A data block is simply a set block of space on the disk in which the actual contents of files are stored; often more than one block is used to hold the data for a file.

Inodes

An inode is a data structure which holds information, or metadata, about a file on that filesystem.

You can use "ls" with the "-i" option to find a file's inode number:

```
student@debian:~/dataset$ ls -i
6553 one.txt 7427 two.txt
```

An inode itself contains the following information:

- the device where the inode resides

- locking information
- the addresses of the file's data blocks on disk (NOT the data blocks themselves)

Let's look at the directory listing below:

```
student@debian:~/dataset$ cd ..
student@debian:~$ ls -ila
total 8
4944 drwxr-xr-x    4 student  student      1024 Feb 19 03:45 .
4943 drwxrwsr-x    3 root     staff        1024 Jan 29 22:32 ..
5665 -rw-------    1 student  student       658 Feb 19 03:22 .bash_histo
4946 -rw-r--r--    1 student  student       509 Jan 25 12:27 .bash_profi
6540 -rw-r--r--    1 student  student      1093 Jan 25 12:27 .bashrc
6540 drwxr-xr-x    2 student  student      1024 Feb 19 03:45 dataset
7425 drwxr-xr-x    2 student  student      1024 Feb 19 03:45 dataset
```

The first line displayed is the total number of 512-byte blocks consumed by the files in the directory displayed.

As you can see from the remaining output, a file has several attributes, or metadata, associated with it. This data is stored in the inode and is, in field order:

File Mode

This is a ten character string which determines who is allowed access to the file. The string is composed of a single initial character, which determines the file type and a permission field.

File types

The following are the list of file types, and their associated character:

- - regular file
- d directory
- b block device
- c character device
- l symbolic link
- s socket link, also called a Unix networking socket
- p first-in first-out (FIFO) buffer, also called a named pipe

A regular file is the most common one that you will have to deal with; plain text files, configuration files, databases, archives, executables and even the kernel are all regular files.

A directory is a file, which contains zero or more other files names and their associated inode numbers.

You should only find character and block device files in your "/dev" directory. These files are used to allow communication between "userland" programs and the kernel. Character devices transfer data a single character at a time (eg, console, printer), while block devices transfer data in fixed-size chunks (eg, harddrive).

A symbolic link is a pointer to another file, and is therefore useful for creating shortcuts or aliases to files and directories.

A socket link file allows for two or more programs to communicate with each other. A common example of this is the system logging daemon (syslogd), which other programs communicate with via the "/dev/log" file. The logging deamon reads information out of the socket file, while other applications send information to it.

```
student@debian:~$ ls -l /dev/log
srw-rw-rw- 1 root root 0 Feb 25 11:03 /dev/log
```

A FIFO buffer is a file whose contents are read out in the order that they were written to the file.

A semaphore file, in Linux programming parlance, is simply a file used for two or more Linux processes to communicate with each other; also known as Inter-Process Communication, or IPC for short.

The simplest sort of semaphore is a binary one; in other words, it is either "on" (set to "1") or "off" (set to "0"). The semaphore file can be read and written to, this is based on its permissions, by the processes that wish to communicate with each other.

An example might be a database application, which will only make a copy of the database when the semaphore is set to "off", indicating that the database is not currently in use.

File Permissions

The 9 characters that follow the file type character are in fact three triplets, which affect who can do what with the file. The triplets are the permissions which affect: the file owner, the file group and everyone else.

The three possible permissions are, in order:

Table 4.2. File Permissions Table

r	read access	OR	-	not readable
w	write access	OR	-	not writeable
x	execute access	OR	-	not executable

The absence of a permission bit is indicated with a dash ("-").

The read and write permissions are self-explanatory for both the regular and directory files.

If the execute permission is set on a regular file, it means that the file can be executed; in other words, it's an application or program that may be run on the system.

If the execute permission is set on a directory, it means that the directory can be accessed (eg, using the "cd" command).

For more information on file permissions, see the section on "chmod" below.

Number of links

This is the number of links (see hard links below) that currently point to the file; when this number reaches zero, the filesystem makes the blocks containing the file contents available for use again. The most common scenario where this occurs is when the file is deleted.

Owner name

The person who owns the file. This information is stored as a numeric value on the filesystem, but is then looked up by tools such as "ls" from the /etc/passwd file, or equivalent file.

Group Name

The group whom owns the file. This information is stored as a numeric value on the filesystem, but is then looked up by tools such as "ls" from the /etc/group file, or equivalent information source.

A Unix group may contain none, one or more users, who will then be able to access the files and directories owned by that group, based on that groups permissions as discussed above. This is useful for sharing files between two people, as a file can only have one owner.

Number of bytes in the file

The size of the file, given in bytes.

Modification Time

The abbreviated Month Name, Day Of The Month, Hour and Minute the file was last modified.

If the modification time of the file is more than 6 months in the past or future, then the year of the last modification is displayed in place of the hour and minute fields.

File Name

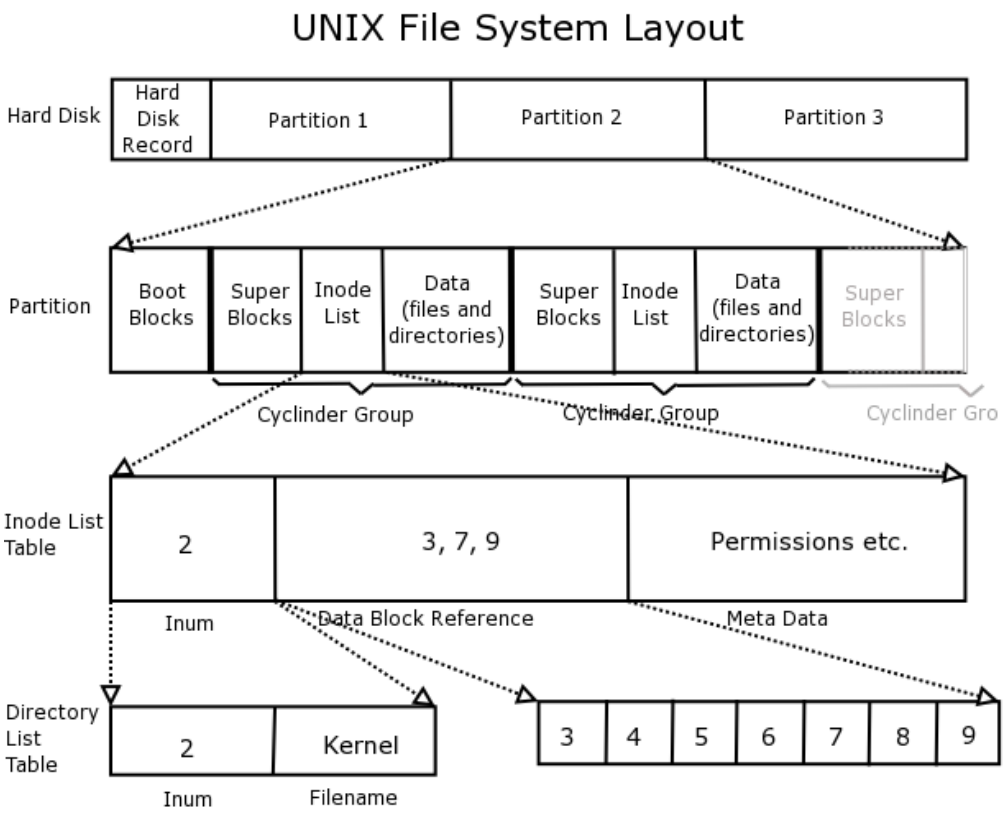
The File name is not stored in the inode!

File names under Linux are case-sensitive. They are limited to 255 characters in length and can contain uppercase, lowercase, numeric characters as well as escape characters.

Although it's a good idea to keep them generally all in lowercase avoiding use of escape characters where possible, so that the file names are easier for you to deal with in the shell.

The filename is held in the directory listing and referenced by the inode number. Look at the following diagram this should make it clearer.

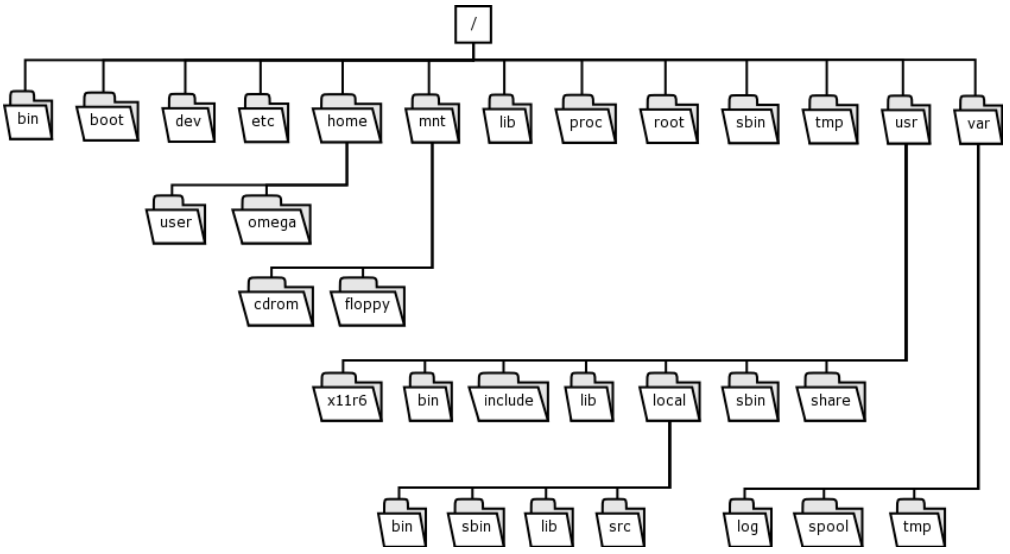
Figure 4.2. Filesytems, Cylinder, Inodes and Superblock Layouts



Linux FS Hierarchy

The Linux filesystem is broken up into a hierarchy similar to the one depicted below, of course you may not see this entire structure if you are working with the simulated Linux environment:

Figure 4.3. Debian Directory listing



The "/" directory is known as the root of the filesystem, or the root directory (not to be confused with the root user though).

The "/boot" directory contains all the files that Linux requires in order to bootstrap the system; this is typically just the Linux kernel and its associated driver modules.

The "/dev" directory contains all the device file nodes that the kernel and system would make use of.

The "/bin", "/sbin" and "/lib" directories contain critical binary (executable) files which are necessary to boot the system up into a usable state, as well as utilities to help repair the system should there be a problem.

The "/bin" directory contains user utilities which are fundamental to both single-user and multi-user environments. The "/sbin" directory contains system utilities.

The "/usr" directory was historically used to store "user" files, but its use has changed in time and is now used to store files which are used during everyday running of the machine, but which are not critical to booting the machine up. These utilities are similarly broken up into "/usr/sbin" for system utilities, and "/usr/bin" for normal user applications.

The "/etc" directory contains almost all of the system configuration files. This is probably the most important directory on the system; after an installation the default system configuration files are the ones that will be modified once you start setting up the system to suit your requirements.

The "/home" directory contains all the users data files.

The "/var" directory contains the user files that are continually changing.

The /usr directory contains the static user files.

The filesystem layout is documented in the Debian distribution in the hier(7) man page.

Exercise:

Read the file system hierarchy man page. Can you find the directory where this particular man page file itself resides?

Explanation of how to use /var and /usr efficiently:

One of the benefits of having a /var directory which contains all the files that are changing or which are variable, and having another directory called /usr where the files are static and they are only read, would be that if you wanted to create an incredibly secure system you could in fact mount your /usr directory read-only.

This would mean that even while the OS system is up and running, no one, not even the root user is allowed to modify any files in that directory.

However, because the system needs to be able to have read and write access to certain files in order to function, the /var partition would serve this purpose exclusively, allowing you to mount /usr as read-only.

So this means that you will be able to have a fully running machine doing all the things you would normally do except it will be virtually impossible for anybody to be able to place any Trojans or any other malicious binaries in your /usr directory.

Another benefit is that you can run diskless or almost diskless clients. Your /usr directory could for instance be mounted over the network from another machine. This means that you don't have to sacrifice all the disk space and instead you could rely on the network to provide your system with the needed binaries. This used to be very popular 5-10 years ago when disk space was quite a lot more expensive than it is today.

However thin-client technology, that seems to be making a comeback, could benefit quite a lot with being able to mount large applications from a remote file system that has a large amount of space, and the thin client could have no or very little disk space available. Examples of large applications are Open Office and Mozilla.

Editing Files under Linux

History

The editor we will be using is a text editor called "vim". Back in the days when Unix was growing up, people didn't even have a visual console to be able to see what was going on when they were running commands or editing files. They would use tele-type terminals.

The command that they used to edit files, was ed. ed is a very unfriendly editor, it is not as interactive as you would be used to using in most modern editors.

The "ed" application would simply give you a prompt and then you would have to tell ed, which line you want displayed. It would then display that line to you, that line only - you wouldn't be able to get an overall feel of what the text file looked like that you were editing.

If you wanted to insert a line, then you would tell ed that you wanted to insert a line at a particular position and then pipe the line that you wanted to insert and it would then insert it for you.

If you wanted to remove a word or delete a line, you would have to specify the location of the word or line respectively.

Obviously this is fine if you know the file or it is very short but if it's a large file or it's new to you then you're going to spend a long time in order to make any modifications to it.

Some time after this it became possible to get video display units, or visual consoles, which had the ability allow users to view more than a single line at a time. Around this time, "ed" developed into "ex", which was a more powerful version of the editor, but still limited to working on a single line at a time.

Unfortunately, these consoles were initially very slow, so screen output was still limited to the bare minimum.

Time progressed, and displays become faster; in fact, fast enough to be able to display an entire 25 lines without too much effort; and here we enter the era of the visual editor, also known as "vi".

Vi has 3 modes of operation: visual mode, insert/editing mode and an ex mode (where you actually access the original command line ed editor).

Those of you who have had to use edlin in DOS before will probably be able to relate to ex.

Under Linux, there is no vi, there is vi improved, or vim. vim is compatible with vi in almost all respects and has a vi compatibility mode available to complete the compatibility in any other respects. However vim expands greatly on the features of vi and allows you to do extra things such as Syntax highlighting, better integration with various other software packages. The ability to perform scripting functions on

your documents, etc.

Using vim

To start up vim, just type "vi", followed by the name of the file you want to edit.

On Debian systems you have the option of having both an open source version of vi (called nvi) and vim installed at the same time.

Debian will actually call "vim" when you execute the "vi" command if "vim" is installed, otherwise it will load "nvi".

```
student@debian:~$ cd dataset/
student@debian:~/dataset$ vi one.txt_
```

In the example above, we're opening the file "one.txt" for editing.

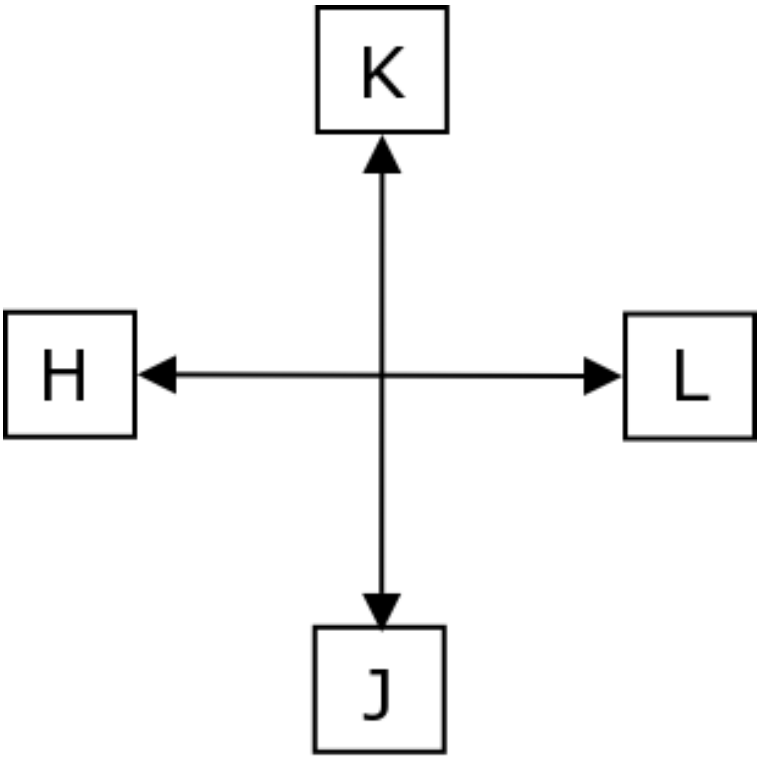
You may find the keys for vim to be tricky to learn, but, once mastered, they allow you to edit documents and configuration files with the minimum number of keystrokes, and with great speed.

Initially, vim will start in "visual mode". Once open the screen will be blank and probably have tildes running down the left hand side of the page indicating un-used lines.

Figure 4.4. Empty vi buffer

In this mode you can use the "h", "j", "k", and "l" keys to perform your movement functions.

Figure 4.5. Movement keys in vi



vim also allows you to use the cursor keys, so if you find the so-called VI movement keys daunting you can make use of these instead.

To switch to "insert mode", press the "i" key, notice that the letter "i" does not print on the screen - but now if you start typing your text will be shown on the screen.

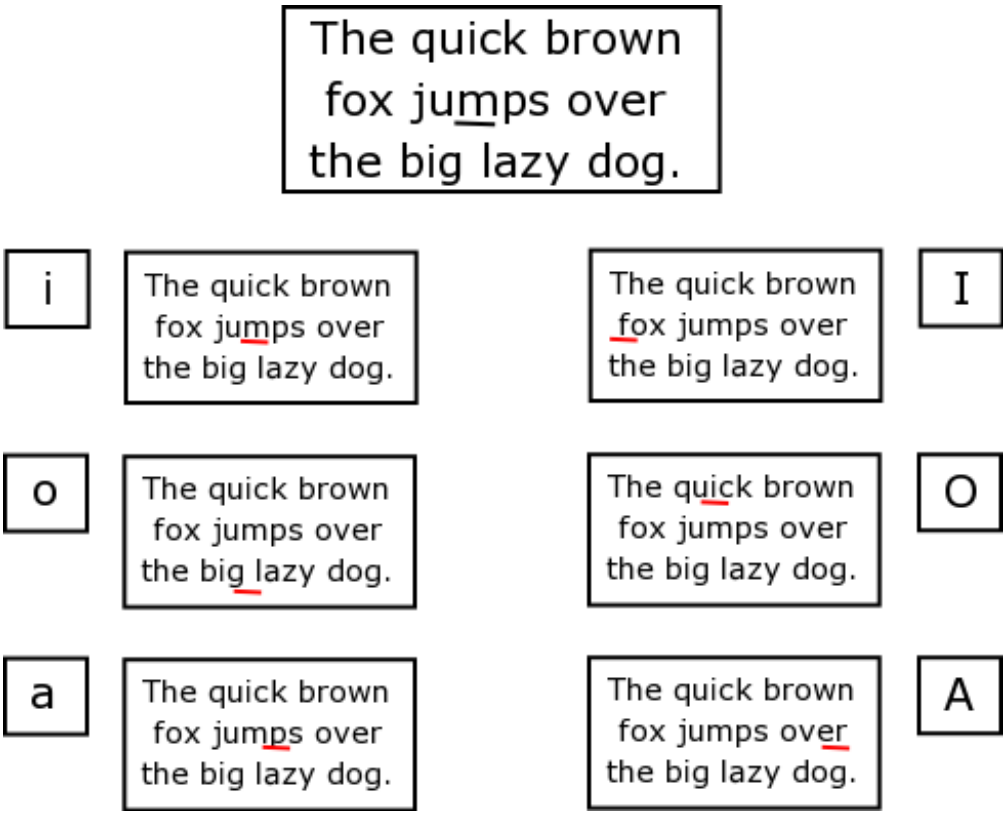
This will tell vim that you want to start inserting text at the current cursor position in the file that you are editing. You can tell when you are in insert mode as vim will display the word "INSERT" in the bottom left of the screen.

By pressing "i" you will enter insert mode, text will be inserted to the right of the current position of the cursor. To insert from the beginning of the line and get into insert mode at the same time, press "I" (shift + "i").

To insert text in the line above the cursor, press "O" (shift + "o"). To insert in the line below the cursor, press "o" . To append text just after the current position of the cursor press "a", to append text at the end of the currentl line, press "A" (shift + "a").

The figure below demonstrates how this affects the place where you will insert text. The first one indicates the position of the cursor, before entering into insert mode in these examples.

Figure 4.6. Different ways of getting into Insert Mode, and how that effects the place where text is inserted.



When you want to return to visual mode, press the "escape key" (Esc) on the keyboard. The word INSERT will disappear from the bottom of your screen.

You can delete characters while in visual mode by hitting the "x" key; this will delete the character currently above the cursor. (A capital "X" will work as a backspace key does deleting one character and moving backwards to the next character.)

The shortcut to delete a line is "dd"; hit the "d" twice in quick succession. (You can use "p" to paste the line afterwards). To delete two lines you would use the command "2dd".

If you wish to join the line below your current line to the end of the current line use "J" (that's a capital).

To insert a line below the one you are currently editing, you can press "o"; this will

also place you in insert mode on the newly created line. (Capital "O" will open a line above your current line and also put you into INSERT mode.)

To tell vim that you wish to save and exit, press "ZZ" (two capital "Z"'s in quick succession).

You can enter ex mode, which allows you to work as if you were in the "ex" editor, by hitting the ":" key when in visual mode. A colon will be printed at the base of the screen waiting for an ex command. To get back to your text press the escape key (Esc).

While in "ex" mode try some of the following suggestions:

- To access vim's built in help system, type "help" at the ex prompt (:) and ENTER.
- You can perform a search an replace in ex mode by using the following Syntax: %s/search/replace/g

  Try the example above in the sample text window, search globally for the occurrence(s) of "now" and change that to "not".
- You can also save ":w" and quit ":q" or save & quit in one command using ":wq" from vim while in ex mode. If you do this point remember to re-enter vim afterwards.

To recap there are three modes of operation:

1. Command mode

   Allows positioning and editing commands to perform functions. Entered via (Esc), from entry mode, or when you return from Last-line mode

2. Entry mode

   Allows you to enter text. Enter this mode via typing; A i I o O c C s S or R from command mode.

3. Last-line mode

   Initiated from command mode by entering advanced editing commands like : (a colon), / (a forward slash), ? (a question mark) and ! (a bang or exclamation mark).

```
Syntax:
vi &lt;filename&gt;
```

Exercise

Vim comes with an excellent tutorial session. It takes about 30 minutes to complete, but covers all the basics that you need to know and is very easy to follow.

To start the vim tutor, type the following:

```
student@debian:~$ vimtutor
```

Working with normal data files

Naming conventions (lowercase, etc.)

Linux filesystems are case sensitive, and support the full range of high and low ASCII characters; this makes it very powerful.

However, humans can only easily deal with a small number of this range of characters. For this reason, it's a good idea to keep the names of your files all lowercase, and to avoid spaces, brackets or any other "special" characters.

Important files, like "README" may be named with all capitals, so that they stand out better in a listing.

The "/" character is special in that it is used to denote a directory structure.

Unlike DOS or Windows, Linux has no strict concept of having file "extensions" and thus does not use them to determine what sorts of file it is. There is no "ls.exe", instead it's just "ls".

However, there is nothing to stop you creating a "readme.txt" file with a .txt extension; Linux just treats the period (".") as part of the filename instead of a special character. Indeed, you can have multiple periods; for example "backup.tar.gz".

So, let us go over the rules in a summary:

- Linux is case sensitive.
- We would advise that you use the lower-case alpha characters (a to z) and the numeric characters 0 to 9 when you name a file.

- A (.) in front of a filename means that it is a hidden file, whereas a (.) anywhere else in the filename is treated as a normal character by Linux.
- There are certain characters that will be interpreted by the shell and have a special function within the shell - do not use these in a filename. Some of these characters are:

  ; | < > lefttick " righttick $! % ( ) ^ \ [ ] & ? #
- Don't use control characters such as ^G (bell) or ^d (interrupt character), the space bar, the tab or the backspace. These all have special meaning to the shell.
- Observe the rules for the length of your filenames.

Using " file", magic

So how can you tell what sort of file a file is without actually looking inside if it doesn't have a handy extension?

The file command examines the contents of the file, and then compares it against a "magic" filter file, which tells the "file" command what sort of patterns would exist in what sort of file.

This is a powerful command, give it a try:

```
student@debian:~$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), stripped
student@debian:~$ file dataset
dataset: directory
student@debian:~$ file dataset/one.txt
dataset/one.txt: ASCII English text
student@debian:~$ file /etc/init.d/rc
/etc/init.d/rc: Bourne shell script text executable
```

```
Syntax:
file &lt;file&gt;
```

Exercise:

Use the file command on some of the files in the /dev directory.

File manipulation commands:

There are several command line tools available for manipulating files in a Linux environment:

touch

This command can be used to create an empty file, or update the modification time of an already existing file.

```
student@debian:~$ cd dataset/
student@debian:~/dataset$ ls
one.txt two.txt
student@debian:~/dataset$ touch testing
student@debian:~/dataset$ ls
one.txt testing two.txt
student@debian:~/dataset$ _
```

```
Syntax:
touch &lt;filename&gt;
```

Exercise:

Try using touch to update the modification time of an existing file. What command can you use to check that the time has in fact been changed?

mv, rm, cp

You can use the "mv" (move) command to move a file into another directory, or to change its name:

```
student@debian:~/dataset$ ls -i
6553 one.txt 7437 testing 7427 two.txt
student@debian:~/dataset$ mv testing testing.123
student@debian:~/dataset$ ls -i
6553 one.txt 7437 testing.123 7427 two.txt
student@debian:~/dataset$ _
```

Check the inode of the file above that you have just moved - what do you notice?

You should notice that the inode remains the same, therefore the only thin that has changed is the filename, which is held in the directory listing. Remember that the filename is ONLY held in the directory listing and not by the inode.

The mv, cp and rm commands all accept a "-i" switch. This makes the command "interactive", which means that the command will ask you to confirm an operation if

it is a potentionally destructive one. For example, if you copy one file over another, or try to delete a file.

```
Syntax:
mv [-i] &lt;oldfilename&gt; &lt;newfilename&gt;
mv [-i] &lt;filename&gt; &lt;directory&gt;
```

You can use the "cp" (copy) command to make a copy of a file - in other words to make an identical copy of the datablocks but at another address, using a different inode number:

```
student@debian:~/dataset$ ls -i
6553 one.txt 7437 testing.123 7427 two.txt
student@debian:~/dataset$ cp testing.123 testing.456
student@debian:~/dataset$ ls -i
6553 one.txt 7437 testing.123  7438 testing.456 7427 two.txt
```

In the example above using the cp command what happens with the inodes for testing.123 and test.456 - are they the same?

```
Syntax:
cp [-i] &lt;file1&gt; &lt;file2&gt;
```

Exercise:

How would you copy the file testing.456 into your home directory - using partial pathnames?

Once you've made a copy of the file, check the new file's inode. Is it different to the original?

What would the inode number be if you had moved the file instead of copying it?

You can delete files with the "rm" (short for remove) command.

```
student@debian:~/dataset$ ls
one.txt testing.123 two.txt
student@debian:~/dataset$ rm testing.123
student@debian:~/dataset$ ls
one.txt two.txt testing.456
student@debian:~/dataset$
```

You can use the "-r" flag with "rm" to recursively delete files and directories. Use

this option with extreme caution!

```
Syntax:
rm [-ir] &lt;filename&gt;
```

Exercise:

Delete the previous "testing.456" file that we created with touch, and then copied and/or moved into your home directory.

mkdir, rmdir

To create and remove directories, you can use the mkdir (make directory) and rmdir (remove directory) commands:

```
student@debian:~/dataset$ ls
 one.txt two.txt
student@debian:~/dataset$ mkdir testdir
student@debian:~/dataset$ cd testdir
student@debian:~/dataset/testdir$ ls
student@debian:~/dataset/testdir$ cd ..
student@debian:~/dataset$ rmdir testdir
student@debian:~/dataset$ ls
one.txt two.txt
```

Exercise:

Create a directory called "test". Now try the following command inside your "dataset" directory:

cp one.txt test

What happens to test? Does it become overwritten with "one.txt"? Or does something else happen?

grep

This is a very powerful command and we will only cover the basics here.

The "grep" command will search through a file for a certain pattern, and then display lines, which contain the matching pattern:

```
student@debian:~/dataset$ grep &quot;action&quot; one.txt
Jefri complained about missing all the action, but Johanna
```

You don't have to put the search pattern in double quotation marks, but it does help to avoid confusion, especially with more complicated patterns, so it's a good habit to get into.

You can also use grep on an input stream; see the pipe ("|") section further on in the course.

The command also has the following flags:

```
 -n displays the line numbers of the lines which match
 -v inverts the match, ie, displays non-matching lines
 -i makes the match case-insensitive
```

```
student@debian:~/dataset$ grep &amp;quot;action&amp;quot; one.txt
 Jefri complained about missing all the action, but Johanna
student@debian:~/dataset$ grep -vi &amp;quot;action&amp;quot; one.txt
The coldsleep itself was dreamless.  Three days ago they had
been getting ready to leave, and now they were here.  Little
Olsndot was glad she&amp;#38;apos;d been asleep; she had known some of
the grownups on the other ship.

  -- A Fire Upon the Deep, Vernor Vinge (pg 11)


student@debian:~/dataset$ _
```

```
Syntax:
grep [-nvi] &amp;lt;filename1&amp;gt; &amp;lt;filename...&amp;gt;
```

Exercise:

Practice using the grep command using the switches above. Can you find all the lines in the file called "two.txt" which contain a the letter "l".

find

The find command is powerful, but is sometimes tricky to get ones head around.

```
Syntax:
find from-directory [-options matchspec] [and-action]

From-directory:
/ to specify a directory name
. current directory
```

Remember though that you would have to have permissions to look in the directory(ies) that you specify to search through.

Options and matchspec:

1. name "foo.txt"

   This option matches all file names matching the pattern, wildcards such as "*" and "?" can be used, but then the name should be enclosed in quotes

2. type f, d, c, b, l, s, etc.

   Matches all files of a certain type, e.g. f=regular file, d=directory file, c=character device file, b=block device file, etcetera

3. user username

   Matches all files belonging to the specified user

4. group groupname

   Matches all the files belonging to a certain group

You could negate an option using the exclamation mark ("!"), but check first on escaping the meaning of certain characters to the shell (especially the bash shell).

And-action:

1. print

   To print the results to the terminal screen; this is the default action.

2. exec command {} \;

   If finding a match, then execute the command on that file. The Syntax for this is important, the curly braces will hold the name of the file that has been found in order that the command can execute. The semicolon means separate the commands so if there is more than one match found each will be dealt with. The backslash escapes the meaning of the semicolon to the shell and keeps it as a semicolon.

3. ok command {} \;

   If finding a match, then ask if it is OK to execute the command on that file. The Syntax for this is important, the curly braces will hold the name of the file that

has been found in order that the command can execute. The semicolon means separate the commands so if there is more than one match found each will be dealt with. The backslash escapes the meaning of the semicolon to the shell and keeps it as a semicolon.

```
student@debian:~$ find . -name &quot;one.txt&quot;
./dataset/one.txt
student@debian:~$ find . -name &quot;one.txt&quot; -ok rm {} \;
&lt; rm ... ./dataset/one.txt &#62; ? n
student@debian:~$ find . -name &quot;one.txt&quot; -exec cat {} \;
The coldsleep itself was dreamless.  Three days ago they had
been getting ready to leave, and now they were here.  Little
Jefri complained about missing all the action, but Johanna
Olsndot was glad she&#38;apos;d been asleep; she had known some of
the grownups on the other ship.

  -- A Fire Upon the Deep, Vernor Vinge (pg 11)
```

Let's just analyse the second example to see exactly what happens:

- It does find the file, we know that from our first example.
- Now it has to execute a command on that file:

  -ok rm {foo.txt};

- Do not remove this file so say NO when it asks Y/N? to removing that file, and enter.
- If there had been two files with the name found (foo.txt and tmp/foo.txt) then each would have been put into the same command sequence as follows:

  -ok rm {foo.txt}; -ok rm {tmp/foo.txt};

  Each time you would have been asked if you wanted to remove the file and each time you would give your answer Y/N and enter.

```
student@debian:~$ find dataset -name &quot;*.txt&quot; -exec ls -l {} \;
-rw-r--r--    1 student  student       321 Feb 19 03:10 dataset/one.txt
-rw-r--r--    1 student  student       150 Feb 19 03:45 dataset/two.txt
```

This command does a long listing on all files in the "dataset" directory which end in

".txt"

```
 student@debian:~$ find . \! -name &quot;*.txt&quot;
.
./.bashrc
./.bash_profile
./.bash_history
./dataset
./dataset2
./dataset2/relay01.dat
./dataset2/relay02.dat
./dataset2/relay03.dat
```

You can also use the "-o" option to cause the options to be logically ordered:

```
student@debian:~$ find . -name &quot;one.*&quot; -o -name &quot;two.*&quot;
./dataset/one.txt
./dataset/two.txt
```

As you can see, this finds all files in the current directory which begin in either "one." *OR* "two.".

```
student@debian:~$ find . -name &quot;*.txt&quot; -user student
foo.txt
```

head and tail

The head and tail commands can be used to inspect the first 10 or the last 10 lines of a file, respectively. You can specify a "-#" parameter to change the number of lines that are displayed.

```
student@debian:~/dataset$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
games:x:5:100:games:/usr/games:/bin/sh
man:x:6:100:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
student@debian:~/dataset$ head one.txt
The coldsleep itself was dreamless.  Three days ago they had
been getting ready to leave, and now they were here.  Little
Jefri complained about missing all the action, but Johanna
```

```
Olsndot was glad she&#38;apos;d been asleep; she had known some of
the grownups on the other ship.

  -- A Fire Upon the Deep, Vernor Vinge (pg 11)

student@debian:~/dataset$ _
```

```
 Syntax:
head [-#] &lt;file1&gt; &lt;file#&gt;
tail [-#] &lt;file1&gt; &lt;file#&gt;
```

Exercise:

Can you head or tail more than one file using a single command? How?

wc

The "wc" or word count command does a word count on a file, and displays the following information:

lines, words, characters

You can also use the -l, -w and -c switches to limit the command to only displaying some of these.

```
student@debian:~/dataset$ wc one.txt
8      57     321 one.txt
student@debian:~/dataset$ wc -l one.txt
8 one.txt
student@debian:~/dataset$ _
```

```
Syntax:
wc [-lwc] &lt;file&gt; &lt;file1&gt;
l -- displays lines in file
w -- displays words in file
c -- displays characters in file
```

Exercise:

Can you run wc on more than one file at once? Can you combine the commands switches?

gzip, bzip2

The "gzip" command can be used to reduce the size of a file using adaptive Lempel-Ziv, while "bzip2" uses the Burrows-Wheeler block sorting text compression algorithm together with Huffman coding.[28]

"gzip" is the most popular compression utility on the Linux platform; files compressed with it end in a ".gz" extension - it tends to give better compression that "compress".

"bzip2" is more recent, and requires more memory to compress a file, but usually gives a better compression ratio than "gzip".

```
student@debian:~/dataset$ ls -l one*
-rw-r--r-- 1 student student 321 Feb 19 03:10 one.txt
student@debian:~/dataset$ gzip one.txt
student@debian:~/dataset$ ls -l one*
-rw-r--r-- 1 student student 247 Feb 19 03:10 one.txt.gz
student@debian:~/dataset$
```

You'll note that the command automatically renames the file to ".gz" and that the file size has decreased.

The commands each have a related command, which is the same name, but prefixed with an "un"- to reverse the process:

```
student@debian:~/dataset$ ls -l one*
-rw-r--r-- 1 student student 247 Feb 19 03:10 one.txt.gz
student@debian:~/dataset$  gunzip one.txt.gz
student@debian:~/dataset$ ls -l one*
-rw-r--r-- 1 student student 321 Feb 19 03:10 one.txt
```

```
Syntax:
gzip &lt;file&gt;
gunzip &lt;file.gz&gt;
```

Wildcards

A wildcard is a pattern-matching character. It is useful in the following ways:

1. An asterisk (*) will match zero, one or more characters.
2. A question mark will match any single character (?)

[28] See http://www.data-compression.com/lempelziv.html for an explanation on how the Lempel-Ziv encoding algorithm works, very interesting.

3. Putting options into square brackets means either-or e.g. [ab] either an "a" or a "b"

4. To use the square brackets with a dash option means a range e.g. [0-9] a single character of any of the range 0 through to 9 (0,1,2,3,4,5,6,7,8 or 9)

5. If wanting to select a range within the alphabet e.g. [a-z] would mean a single character that might match any of the letters in the entire lowercase alphabet.

6. Could use the brackets to specify letters and numbers as follows: [a-e1-5] would match a single character that is either a letter in the range a to e or numerical 1 to 5. (a,b,c,d,e,1,2,3,4 or 5)

7. Using a bang at the beginning of the expression in the square brackets would negate the expression. [!ab] meaning a single character that is not an "a" or a "b".

8. Another interesting variation would be the following: [1-57] which would mean any single character in the range of numbers 1 through 5 or the number 7. (1,2,3,4,5 or 7)

Examples:

Here's how you long list all the files in your "dataset2" directory ending with .txt:

```
student@debian:~/dataset2$ ls -l *.txt
```

Here's how you list all the files starting with o and ending with xt:

```
student@debian:~/dataset2$ ls o*txt
-rw-r--r--     1 student student     0 feb 19 04:59  org.txt
```

The following command will copy all files starting with "b" through to "f" and ending with anything (*) from your dataset2 directory to your previous dataset directory. The [b-f] option indicates one single digit or character.

```
student@debian:~/dataset2$ cp [b-f]*  /tmp/
student@debian:~/dataset2$
```

If you have the following files in your directory:

```
foot
```

```
foo2
foo.txt
filer
file
files
greper
grep202
fast
slower
peanuts100
12.bed
camping.tent
```

These files are not in your directories, it is a rhetorical question only - question yourself as to what will happen in each of these examples before just accepting the answer as a given:

```
rm  file*                          Which files will be removed?
filer
files
file (an asterisk means zero characters as well)

cp  foo*  tmp                            Which files will be copied?
foot
foo2
foo.txt

mv  f[a-z]?.*]  tmp                Which files will be moved?
        foo.txt
```

Starts with a "f" then a single character of lowercase alphabet,any single character with the "?", then a full-stop, then zero one or more characters.

```
cp  [!ft]  tmp
greper
grep202
slower
peanuts100
12.bed
camping.tent

Any file name that does not have an &quot;f&quot; or a &quot;t&quot; in it.

cp  [!f][l-z]??[1-5][0-9][1-5]  tmp/
grep202
```

The filename starts with any single character except an "f", then a single character between "l" and "z" (l,m,n,o,p,q,r,s,t,u,v,w,x,y or z), then two single characters (any characters), then a number between 1 and 5, then a number between 0 and 9, the last character is again a number between 1 and 5.

Exercise:

Ok, now that you've worked out the answers above using just your head, use the "touch" command to actually create the files, and perform the commands on them. Do the actual results match those you had worked out?

links

symbolic links versus hard links

Unix filesystems (including BSD's UFS and Linux's ext2fs, ext3fs and XFS) support file links. Links are used to have one or more copies of a file in many places at once, without duplicating the actual data of the file for each copy. This is often used to save space, or to help when moving data from one location in the filesystem to another.

Each link points to the original copy of the file; the way in which the link points to a file determines whether the link is a "hard" link or a "soft" link, also known as a "symbolic" link.

Hard links

Hard links are created with the "ln" command:

```
student@debian:~/dataset$ ls -li
total 8
1101067 -rw-r--r--    1 student  student       321 Feb 19 03:10 one.txt
1101076 -rw-r--r--    1 student  student       150 Feb 19 03:45 two.txt
student@debian:~/dataset$ ln one.txt chapterone.txt
student@debian:~/dataset$ ls -li
total 12
1101067 -rw-r--r--    2 student  student       321 Feb 19 03:10 chapteron
1101067 -rw-r--r--    2 student  student       321 Feb 19 03:10 one.txt
1101076 -rw-r--r--    1 student  student       150 Feb 19 03:45 two.txt
student@debian:~/dataset$ _
```

You'll notice that the inode number for both files are the same; this is how hard links work. The actual contents of the file, its data blocks, have not been duplicated, merely its directory entry.

You'll see that if you edit one file, the other file's contents are also changed.

Hard links share an inode, and therefore they can only be created on the same filesystem where the original file exists.

Removing a file, which is a hard link doesn't delete the file it only removes the link. The file itself is only removed once all the filesystem link entries that point to it are removed.

Then the inode becomes a shadow inode and is zero-ised in the directory listing. [29]

```
student@debian:~/dataset$ ls -li
total 12
1101067 -rw-r--r--    2 student  student       321 Feb 19 03:10 chapterone.txt
1101067 -rw-r--r--    2 student  student       321 Feb 19 03:10 one.txt
1101076 -rw-r--r--    1 student  student       150 Feb 19 03:45 two.txt
student@debian:~/dataset$ rm one.txt
student@debian:~/dataset$ ls -li
total 8
1101067 -rw-r--r--    1 student  student       321 Feb 19 03:10 chapterone.txt
1101076 -rw-r--r--    1 student  student       150 Feb 19 03:45 two.txt
```

In the example above, we've deleted the original file, but you can see that the actual contents are still preserved, as we still have another directory entry for the same inode number.

Exercise:

Now use "cp" to make a backup copy of "chapterone.txt", and then delete it. Now use "ls" to check what's changed with the inode numbers. Change "chapterone.txt" back to "one.txt" when you're finished.

Symbolic Link

A symbolic link is a pointer to another file path; you use "ln" in conjunction with the "-s" switch to create a symbolic link:

```
student@debian:~/dataset$ ls -il
total 8
1101067 -rw-r--r--    1 student  student   321 Feb 19 03:10 one.txt
1101076 -rw-r--r--    1 student  student   150 Feb 19 03:45 two.txt
student@debian:~/dataset$ ln -s two.txt chaptertwo.txt
student@debian:~/dataset$ ls -il
total 8
1101075 lrwxrwxrwx    1 student  student   7 Feb 19 05:08
                       chaptertwo.txt -&#62; two.txt
1101067 -rw-r--r--    1 student  student   321 Feb 19 03:10 one.txt
```

[29] You could see this if piping a directory listing that you know has shadow inodes through a hex dump tool.

```
1101076 -rw-r--r--    1 student  student   150 Feb 19 03:45 two.txt
```

Symbolic links can traverse different filesystems, and so are often useful when shuffling data off a full disk onto a new one, while still preserving the directory path to the original file.

You'll notice that a symbolic link is given it's own inode number, unlike hard links, which share another filename's inode number.

An example of when you'd use this in real life: you have a big database in "/home/database", and you want to move it onto a partition mounted on "/scratch" which has a lot more free space. The problem is that the database software has been configured to use "/home/database" to access its files.

What you can do is stop the database, move the "database" directory out of "/home" and into "/scratch", and then set a symbolic link called "database" in "/home" to point to the real "database" directory in "/scratch".

Now when you restart your database, it will still be able to find its files where it expects them!

File permissions/security

We learnt about file mode permissions earlier. Now we're going to look at some of the tools that can be used to manipulate them.

As a reminder the permissions field was the 9 digits after the file type field. In a long listing of your files above the permission mode was set be default system wide and user wide. This is set up for you by a parameter called umask which we will discuss later on.

Let us look at the permission values and see what they mean on files and directory files - this is a revision section. Although the permissions have obvious effects for regular files, directories and their associated permissions can sometimes be confusing.

A directory for which you are allowed "read" access can have its contents listed by you. Using, say, "ls".

However, you are only allowed to change into ("cd") a directory for which you have the "execute" permission, also called the "directory search bit".

If we breakdown the permissions field for a regular file as follows:

Table 4.3. File Permissions example 1

File type	Owner	Group	Other/public or everyone else on the system
-	rwx	rwx	rwx

This would mean that the file is a regular file and that anyone using the system can access the file read it, write and change it and execute it as a program (if relevant as not all regular files are programs.)

Again:

Table 4.4. File Permissions example 2

File type	Owner	Group	Other/public or everyone else on the system
-	rw-	r--	---

This would mean that the file is a regular file and the owner can read and write to the file, the group can only read the file and anyone else on the system has no permissions on that file at all.

Now with a directory file:

Table 4.5. File Permissions example 3

File type	Owner	Group	Other/public or everyone else on the system
d	rwx	r-x	r-x

Here the owner of the directory file can list the contents of the directory, change (edit and save) files within the directory or make new files, remove files, they can also cd into the directory or perform a find command that can search that directory with no permission errors.

The group and general public can list the contents of the directory and cd into the

directory but cannot change (edit and save), remove, or write a new file into that directory.

Something that you should maybe beware of is that you may be getting permission errors on a directory, you have checked that directory and you have full permissions. Look again at the parent directory you probably do not have permissions on that directory and that will stop you doing what you want or need to do even in the child directory.

chmod

You can use this command to change the mode of the file;

```
Syntax:
chmod mode file-name(s)
```

Octal mode:

The octal format is supposedly a more difficult method (some find it easier than the symbolic method), but it is the way in which the modes are actually stored by the operating system, and is also the mode more widely used in documentation and in script files, and so is useful to know.

Each permission bit a number associated with it:

```
 r = 4    w = 2    x = 1
```

These numbers are then added together to form the set of modes that you want, for example if you want "rw-" permissions set then:

```
 r + w = rw     4 + 2 = 6
```

There is a grouping of three sets of permissions and the octal method expresses all three fields (owner, group and public). Thus, a mode of "660" means that the user, and group, have "rw" access, and everyone else has no access (-rw-rw----).

The first digit ("6") is the mode, which applies to the user (rw-), the second digit ("6") applies to the group (rw-) and the third digit ("0") applies to everyone else (---).

```
 student@debian:~/dataset$ ls -l
one.txt -rw-r--r-- 1 student student 321 Feb 19 03:10 one.txt
student@debian:~/dataset$ chmod 660 one.txt
student@debian:~/dataset$ ls -l one.txt
-rw-rw---- 1 student student 321 Feb 19 03:10 one.txt
```

Symbolic mode:

You must use one character out of each column to form a triple, with no spaces between the three characters.

```
Syntax:
chmod permission-mode filename
```

Table 4.6. Symbolic File Permission switches

Owners	Add, Remove or Set	Permission
u owner permissions	+ adds the permission	r read
g group permissions	- removes the permission	w write
o other or world permissions	= sets the permission	x execute
a all of the above		

You can use a comma (",") to separate operands, but don't use any spaces!

```
student@debian:~$ ls -l
total 8
drwxr-xr-x    2 student  student      1024 Feb 19 05:08 dataset
drwxr-xr-x    2 student  student      1024 Feb 19 05:01 dataset2
student@debian:~$ chmod ug=rw,o= dataset2
student@debian:~$ ls -l
total 8
drwxr-xr-x    2 student  student      1024 Feb 19 05:08 dataset
drw-rw----    2 student  student      1024 Feb 19 05:01 dataset2
student@debian:~$ chmod u=rx,g-w,o+r dataset
student@debian:~$ ls -l
total 8
dr-xr-xr-x    2 student  student      1024 Feb 19 05:08 dataset
drw-rw----    2 student  student      1024 Feb 19 05:01 dataset2
student@debian:~$ chmod a+rx,u=w dataset
student@debian:~$ ls -l
total 8
d-w-r-xr-x    2 student  student      4096 Feb 19 05:08 dataset
drw-rw----    2 student  student      4096 Feb 19 05:01 dataset2
student@debian:~$ cd dataset
bash: cd: dataset: Permission denied
student@debian:~$ cd dataset2
bash: cd: dataset2: Permission denied
```

You cannot change into either of the directories because the owner (student) does not have "execute" or "search bit" access to either of them.

```
student@debian:~$ chmod u+x dataset*
student@debian:~$ ls -l
total 8
d-wxr-xr-x    2 student  student      4096 Feb 19 05:08 dataset
drwxrw----    2 student  student      4096 Feb 19 05:01 dataset2
student@debian:~$ cd dataset
student@debian:~/dataset$ ls
ls: .: Permission denied
student@debian:~/dataset$ cd ..
```

Now we've given ourselves back search bit access, but we still don't have read access to "dataset", which means that while we can "cd" into it, we cannot get a listing of its contents!

Exercise:

Can you still "cat" files inside dataset, even though you only have "x" and not "r"?

Correct the permissions on the directories to what they should be.

chown and chgrp

Only the root user may use the "chown" command; we will cover this command in detail in the System Administration section.

You can change both the owner and the group by using the following Syntax:

```
Syntax:
chown user:group &lt;file&gt;
```

This changes the user and group associated with the file.

A normal user may change the group to which a file belongs, provided that they are a member of that group and also own the file, by using the chgrp command.

umask

The umask determines what the default permissions will be on a file that is created, either system-wide or user based if specified in the home directory log in files. When using the bash shell, this is a builtin command.

It understands octal and symbolic representations of permissions.

To see what the current umask is, just type "umask":

```
student@debian:~$ umask
0022
```

```
student@debian:~$  umask -S
u=rwx,g=rx,o=rx
```

As you can see, the octal values for the umask are not the same as those for chmod.

In the umask above (0022), the first "0" we will not explain right now - suffice it to say that this relates to setting an additional permission bit (SUID, SGID or Sticky Bit).

In umask the permission mode is calculated by subtracting from a full permission set of read write and execute permission bits as follows: (r + w + x = 4 + 2 + 1 = 7)

A value of zero (0) in the umask means then full permissions (7 - 0 = 7).

A value of 2 in the umask means read (4) and execute (1) permissions (7 - 2 = 5).

To change the current umask:

```
student@debian:~$ touch file1
student@debian:~$ ls -l file1
-rw-r--r--  1 student  student  0 Feb 19 02:39 file1
student@debian:~$ umask u=rwx,g=rx,o=           (OR umask 0027)
student@debian:~$ touch file2
student@debian:~$ ls -l
-rw-r-----  1 student  student  0 Feb 19 02:39 file2
```

You'll notice that umask is clever enough to only apply the +x (execute) bit to directories, and not regular files as above. Regular files will not be executable by default.

```
student@debian:~$ mkdir dir1
student@debian:~$ ls -l file1 dir1
drwxr-x---  2 student  student   512 Jan 14 02:40 dir1
-rw-r--r--  1 student  student     0 Jan 14 02:39 file1
```

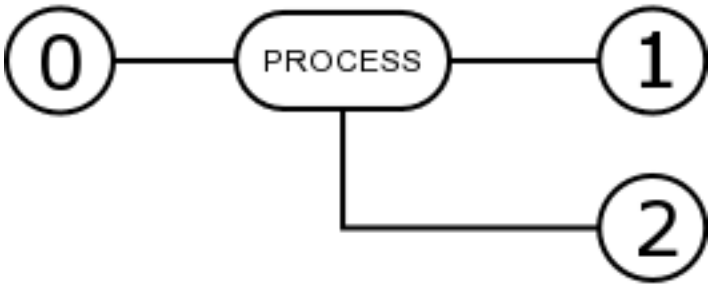
File Redirection, Named and un-named pipes

In the Linux shell, each process has three file handles (also called file descriptors, or fd's for short) associated with it.

- Standard input, or stdin -- numbered as file descriptor "0". This is where the process receives its input from; this is usually the keyboard.

- Standard output, or stdout -- numbered as file descriptor "1". This is where the process sends its output to; this is usually the console or terminal screen.
- Standard error, or stderr -- numbered as file descriptor "2". This is where the process sends its error messages; this is also usually the console or terminal screen.

Figure 4.7. stdin, stdout, stderr



You can tell the Linux shell to change any or all of these on a per-command basis, by using pipes ("|") and redirections ("<" and ">").

These features are useful if you want, for example to get the output of a command into a file for later perusal, or if you want to string multiple commands together in order to achieve the desired result.

Remember, Linux is made up of lots of small building blocks that can be put together in order to make something more complicated; the pipes and redirections are what you can use to help join the blocks together.

stdin

The standard input is usually taken from what you type in at the keyboard. However we could take the output from a file.

We would use the less than sign (<) to redirect input.

```
command 0&lt; filename or command &lt; filename
```

stdout

If we want to store the output of a command in a file we would use standard output

and redirect it. Normally by default the output goes to the screen. . If the file doesn't already exist, it will be created.

We use the greater than sign (>) to redirect standard output.

```
 command 1&#62; filename or command &#62; filename
```

stderr

It is important to think of output from a command and errors produced by a command as separate actions. We might want to store the output of a command in a file and leave the errors to display on the screen. Or we might want to store the error messages in a file.

```
       command 2&#62; filename
```

In this instance the number "2" preceding the redirection sign is not optional.

```
       student@debian:~$ touch file1 file2
```

Make sure that file1 and file2 exist in your directory, file3 should not exist.

```
student@debian:~$ ls file1 file2
       file1 file2
```

Standard output goes to the screen and there are no error messages.

```
student@debian:~$ ls file1 file2 file 3
       file3 ls: file3: No such file or directory file1 file2
```

File3 does not exist so a message is printed to standard error. The directory listing for file1 and file2 is printed to standard output.

```
student@debian:~$ ls file1 file2
       file3 &#62; stdout.txt ls: file3: No such file or directory
```

Redirect standard output to a file called stdout.txt, standard error is kept as the default display to the screen.

```
student@debian:~$ ls file1 file2
        file3 2&#62; stderr.txt file1 file2
```

Redirect standard error to a file called stderr.txt, standard output is left to the default setting of the screen.

```
student@debian:~$ ls file1 file2
        file3 &#62; stdout.txt 2&#62; stderr.txt student@debian:~$ _
```

Redirect standard output and standard error and nothing is displayed on the screen.

```
student@debian:~$ cat stdout.txt
        file1 file2  student@debian:~$ cat stderr.txt ls:
        file3: No such file or directory
```

Check what is in each relevant file.

For standard input we can demonstrate how it works however at this stage of your course it is harder to describe a really useful example.

A simple example would be:

```
student@debian:~$ cat <
        stdout.txt file1 file2
```

When we know a little more we could do something more sophisticated, like run a program that normally requires input from a user at a keyboard - but run it after hours in the background with no user present to press the relevant key strokes.

Appending to a file

You can use a double redirection (">>") to append to a file instead of overwriting it. If the file doesn't already exist, it will be created.

```
student@debian:~$ ls output.txt
ls: output.txt: No such file or directory
```

Make sure that the file does not already exist in your home directory.

```
 student@debian:~$ echo &quot;test&quot; &#62;&#62; output.txt
 student@debian:~$ cat output.txt
 test
```

```
 student@debian:~$ echo &quot;test again&quot; &#62;&#62; output.txt
student@debian:~$ cat output.txt
```

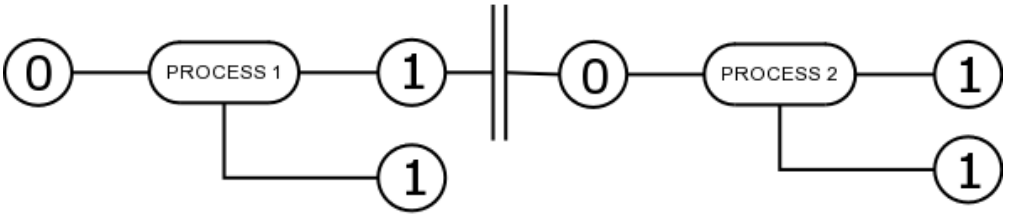
```
test test again
```

The above two steps will prove that the append function actually does create the file if it does not already exist.

Piping

A pipe ("|") directs the stdout from one process to the stdin of another:

Figure 4.8. piping from one process to another



Note that stderr is _not_ passed through the pipe!

```
student@debian:~$ ls dataset2 | grep &quot;txt&quot;
fight.txt
flight.txt
org.txt
singularity.txt
three.txt
vernor.txt
vinge.txt
student@debian:~$ _
```

This type of pipe is called an "un-named pipe". The un-named pipe uses a temporary buffer type of action to pass the stream of data through.

You can also use "named pipes". We briefly mentioned them earlier in this course; they are also known as FIFO buffers.

Named pipes work exactly the same as unnamed pipes, where the stdout from one process is fed into the stdin for another process.

You can use the **mkfifo** command to create such a file, and you can then simply use the redirection symbols to put data in, or take it out. As the name implies, the data is read First In, First Out.

```
student@debian:~$ mkfifo foop
```

```
student@debian:~$ ls -l foop
prw-r--r-- 1 mwest mwest 0 Jan 13 23:26 foop|
student@debian:~$ echo &quot;testing the named pipe&quot; &#62; foop
```

Ok, now go and open up another terminal. You can do this by holding down the **Alt** key and pressing **F2**. Once there, log in, and then type the following:

```
student@debian:~$ cat < foop
testing the named pipe
student@debian:~$ _
```

If you switch back to your original terminal window (press alt-f1), you should see that your shell has now returned back to your prompt. This is because your previous "cat" command took the contents out of the pipe, and so your original "echo" command was able to complete.

Check if the file called "foop" still exists as a named pipe in your directory.

Experiment with this one a bit, it can be quite fun!

Other commands

There are a few other commands, which you may find useful:

clear - clears the screen

uname - this displays information about the system the most common switch used with this is "-a"

```
student@debian:~$  uname -a
Linux debian 2.2.20 #1 Sat Apr 20 12:45:19 EST 2002 i586 unknown
```

```
   Linux debian 2.2.20 #1 Sat Apr 20 12:45:19 EST 2002 i586 unknown
   1.    2.      3.       4.         5.                        6.
```

1. the OS name, could be Linux, FreeBSD, Solaris, etc.
2. the hostname
3. the version of the kernel currently running
4. how many times this kernel had been compiled

5. the compilation date

6. the architecture it was compiled for

last - indicates last logins of users, and on which terminals, it reports on a file called wtmp.

```
student@debian:~$ last
student  pts/0        192.168.0.5      Thu Feb 19 03:01   still logged in
root     pts/0        192.168.0.5      Thu Feb 19 02:58 - 03:00  (00:02)
student  tty2                          Thu Feb 19 02:56   still logged in
reboot   system boot  2.2.20-idepci    Thu Feb 19 02:56          (02:15)
student  pts/0        192.168.0.5      Thu Jan 29 22:32 - 23:57  (01:24)
root     pts/0        192.168.0.5      Thu Jan 29 22:32 - 22:32  (00:00)
student  tty1        192.168.0.5      Thu Jan 29 22:30 - 22:32  (00:02)
root     tty1                          Thu Jan 29 22:15 - down   (01:42)
reboot   system boot  2.2.20-idepci    Thu Jan 29 22:14          (01:42)
root     tty1                          Sun Jan 25 12:28 - 12:29  (00:01)
reboot   system boot  2.2.20-idepci    Sun Jan 25 14:26          (4+09:31)

wtmp begins Sun Jan 25 14:26:47 2004
```

tty - tells you which terminal you are currently on

```
student@debian:~$ tty
/dev/tty1
```

Appendix A. Linux Professional Institute (LPI) Certification

Introduction

Visit this site [http://www.lpi.org/en/lpic.html]for the latest information. The following section will describe the certification methods and pre-requisite knowledge that is needed to gain the LPI certifications.

Junior Level Administration (LPIC1)

Status: Available since January 2000, latest revision done in March 2003 Pre-requisite Knowledge: none Requirements: Pass exam 101 and 102 Job description of a person with this certification:

- Work at the Linux command line
- Perform easy maintenance tasks: help out users, add users to a larger system, backup & restore, shutdown & reboot
- Install and configure a workstation (including X) and connect it to a LAN, or a stand-alone PC via modem to the Internet.

LPI exam 101 Details

Topic 101 Hardware and Architecture

Table A.1. LPI exam 101: Hardware and Architecture

Weight	Title	Description	Key Files, terms and utillities
1	Fundamental BIOS Settings	Candidates should be able to configure fundamental system hardware by making the correct settings in the system BIOS. This objective includes a	/proc/ioports /proc/interrupts /proc/dma /proc/pci

		proper understanding of BIOS configuration issues such as the use of LBA on IDE hard disks larger than 1024 cylinders, enabling or disabling integrated peripherals, as well as configuring systems with (or without) external peripherals such as keyboards. It also includes the correct setting for IRQ, DMA and I/O addresses for all BIOS administrated ports and settings for error handling.	
1	Configure Modem and Sound cards	Ensure devices meet compatibility requirements (particularly that the modem is NOT a win-modem), verify that both the modem and sound card are using unique and correct IRQ's, I/O, and DMA addresses, if the sound card is PnP install and run sndconfig and isapnp, configure modem for outbound dial-up, configure modem for outbound PPP | SLIP | CSLIP connection, set serial port for 115.2 Kbps	Not applicable

1	Setup SCSI Devices	Candidates should be able to configure SCSI devices using the SCSI BIOS as well as the necessary Linux tools. They also should be able to differentiate between the various types of SCSI. This objective includes manipulating the SCSI BIOS to detect used and available SCSI IDs and setting the correct ID number for different devices especially the boot device. It also includes managing the settings in the computer's BIOS to determine the desired boot sequence if both SCSI and IDE drives are used.	SCSI ID /proc/scsi/scsi_info
1	Configure Communication Devices	Candidates should be able to install and configure different internal and external communication devices like modems, ISDN adapters, and DSL switches. This objective includes verification of compatibility requirements (especially important if that modem is a winmodem),	/proc/dma /proc/interrupts /proc/ioports setserial(8)

		necessary hardware settings for internal devices (IRQs, DMAs, I/O ports), and loading and configuring suitable device drivers. It also includes communication device and interface configuration requirements, such as the right serial port for 115.2 Kbps, and the correct modem settings for outbound PPP connection(s).	
1	Configure USB devices	Candidates should be able to activate USB support, use and configure different USB devices. This objective includes the correct selection of the USB chipset and the corresponding module. It also includes the knowledge of the basic architecture of the layer model of USB as well as the different modules used in the different layers. Key files, terms, and utilities include:	lspci(8) usb-uhci.o usb-ohci.o /etc/usbmgr/ usbmodules /etc/hotplug
3	Setup different PC expansion cards	Candidates should be able to configure various cards for the various expansion slots. They should know	/proc/dma /proc/interrupts /proc/ioports /proc/pci pnpdump(8) isapnp(8) lspci(8)

		the differences between ISA and PCI cards with respect to configuration issues. This objective includes the correct settings of IRQs, DMAs and I/O Ports of the cards, especially to avoid conflicts between devices. It also includes using isapnp if the card is an ISA PnP device.	

Topic 102 Linux Installation & Package Management

Table A.2. LPI exam 101: Linux Installation & Package Management

Weight	Title	Description	Key Files, terms and utillities
1	Install a boot manager	Candidate should be able to select, install, and configure a boot manager. This objective includes providing alternative boot locations and backup boot options (for example, using a boot floppy).	/etc/lilo.conf /boot/grub/grub.conf lilo grub-install MBR superblock first stage boot loader
3	Manage shared libraries	Candidates should be able to determine the shared libraries that executable programs depend on and install them when necessary.	ldd ldconfig /etc/ld.so.conf LD_LIBRARY_PATH

		Candidates should be able to state where system libraries are kept.	
5	Design hard disk layout	Candidates should be able to design a disk partitioning scheme for a Linux system. This objective includes allocating filesystems or swap space to separate partitions or disks, and tailoring the design to the intended use of the system. It also includes placing /boot on a partition that conforms with the BIOS' requirements for booting.	/ (root) filesystem /var filesystem /home filesystem swap space mount points partitions cylinder 1024
5	Make and install programs from source	Candidates should be able to build and install an executable program from source. This objective includes being able to unpack a file of sources. Candidates should be able to make simple customizations to the Makefile, for example changing paths or adding extra include directories.	gunzip gzip bzip2 tar configure make
8	Use Debian package management	Candidates should be able to perform package management skills using the Debian	unpack configure /etc/dpkg/dpkg.cfg /var/lib/dpkg/* /etc/apt/apt.conf /etc/apt/sources.list

		package manager. This objective includes being able to use command-line and interactive tools to install, upgrade, or uninstall packages, as well as find packages containing specific files or software (such packages might or might not be installed). This objective also includes being able to obtain package information like version, content, dependencies, package integrity and installation status (whether or not the package is installed).	dpkg dselect dpkg-reconfigure apt-get alien
8	Use Red Hat Package Manager (RPM)	Candidates should be able to perform package management under Linux distributions that use RPMs for package distribution. This objective includes being able to install, re-install, upgrade, and remove packages, as well as obtain status and version information on packages. This objective also includes obtaining package information such as version, status,	/etc/rpmrc /usr/lib/rpm/* rpm grep

		dependencies, integrity, and signatures. Candidates should be able to determine what files a package provides, as well as find which package a specific file comes from.	

Topic 103: GNU & Unix Commands

Table A.3. LPI exam 101: GNU & Unix Commands

Weight	Title	Description	Key Files, terms and utillities
1	Perform basic file editing operations using vi	Candidates should be able to edit text files using vi. This objective includes vi navigation, basic vi nodes, inserting, editing, deleting, copying, and finding text.	vi /, ? h,j,k,l G, H, L i, c, d, dd, p, o, a ZZ, :w!, :q!, :e! :!
3	Modify process execution priorities	Candidates should should be able to manage process execution priorities. Tasks include running a program with higher or lower priority, determining the priority of a process and changing the priority of a running process.	nice ps renice top
3	Perform basic file management	Candidates should be able to use the basic Unix commands to copy,	cp find mkdir mv ls rm rmdir touch file globbing

		move, and remove files and directories. Tasks include advanced file management operations such as copying multiple files recursively, removing directories recursively, and moving files that meet a wildcard pattern. This includes using simple and advanced wildcard specifications to refer to files, as well as using find to locate and act on files based on type, size, or time.	
3	Search text files using regular expressions	Candidates should be able to manipulate files and text data using regular expressions. This objective includes creating simple regular expressions containing several notational elements. It also includes using regular expression tools to perform searches through a filesystem or file content.	grep regexp sed
5	Work on the command line	Candidates should be able to Interact with shells and commands using the command line. This includes	. bash echo env exec export man pwd set unset ~/.bash_history ~/.profile

		typing valid commands and command sequences, defining, referencing and exporting environment variables, using command history and editing facilities, invoking commands in the path and outside the path, using command substitution, applying commands recursively through a directory tree and using man to find out about commands.	
5	Use streams, pipes, and redirects	Candidates should be able to redirect streams and connect them in order to efficiently process textual data. Tasks include redirecting standard input, standard output, and standard error, piping the output of one command to the input of another command, using the output of one command as arguments to another command and sending output to both stdout and a file.	tee xargs < << >> | ' '
5	Create, monitor, and kill processes	Candidates should be able to manage processes. This	& bg fg jobs kill nohup ps top

		includes knowing how to run jobs in the foreground and background, bring a job from the background to the foreground and vice versa, start a process that will run without being connected to a terminal and signal a program to continue running after logout. Tasks also include monitoring active processes, selecting and sorting processes for display, sending signals to processes, killing processes and identifying and killing X applications that did not terminate after the X session closed.	
6	Process text streams using filters	Candidates should should be able to apply filters to text streams. Tasks include sending text files and output streams through text utility filters to modify the output, and using standard Unix commands found in the GNU textutils package.	cat cut expand fmt head join nl od paste pr sed sort split tac tail tr unexpand uniq wc

Topic 104: Devices, Linux Filesystems, Filesystem

Hierarchy Standard

Table A.4. LPI exam 101: Devices, Linux Filesystems, Filesystem Hierarchy Standard

Weight	Title	Description	Key Files, terms and utillities
1	Create and change hard and symbolic links	Candidates should be able to create and manage hard and symbolic links to a file. This objective includes the ability to create and identify links, copy files through links, and use linked files to support system administration tasks.	ln
1	Manage file ownership	Candidates should be able to control user and group ownership of files. This objective includes the ability to change the user and group owner of a file as well as the default group owner for new files. Key files, terms, and utilities include:	chmod chown chgrp
3	Create partitions and filesystems	Candidates should be able to configure disk partitions and then create filesystems on media such as hard disks. This objective includes using various mkfs commands to set up partitions to various	fdisk mkfs

		filesystems, including ext2, ext3, reiserfs, vfat, and xfs.	
3	Maintain the integrity of filesystems	Candidates should be able to verify the integrity of filesystems, monitor free space and inodes, and repair simple filesystem problems. This objective includes the commands required to maintain a standard filesystem, as well as the extra data associated with a journaling filesystem.	du df fsck e2fsck mke2fs debugfs dumpe2fs tune2fs
2	Control mounting and unmounting filesystems	Candidates should be able to configure the mounting of a filesystem. This objective includes the ability to manually mount and unmount filesystems, configure filesystem mounting on bootup, and configure user mountable removeable filesystems such as tape drives, floppies, and CDs.	/etc/fstab mount umount
3	Managing disk quota	Candidates should be able to manage disk quotas for users. This objective includes	quota edquota repquota

		setting up a disk quota for a filesystem, editing, checking, and generating user quota reports. quotaon	
5	Use file permissions to control access to files	Candidates should be able to control file access through permissions. This objective includes access permissions on regular and special files as well as directories. Also included are access modes such as suid, sgid, and the sticky bit, the use of the group field to grant file access to workgroups, the immutable flag, and the default file creation mode.	chmod umask chattr
5	Find system files and place files in the correct location	Candidates should be thouroughly familiar with the Filesystem Hierarchy Standard, including typical file locations and directory classifications. This objective includes the ability to find files and commands on a Linux system.	find locate slocate updatedb whereis which /etc/updatedb.conf

Topic 110: The X Window System

Table A.5. LPI exam 101: The X Window System

Weight	Title	Description	Key Files, terms and utillities
3	Setup a display manager	Candidate should be able setup and customize a Display manager. This objective includes turning the display manager on or off and changing the display manager greeting. This objective includes changing default bitplanes for the display manager. It also includes configuring display managers for use by X-stations. This objective covers the display managers XDM (X Display Manger), GDM (Gnome Display Manager) and KDM (KDE Display Manager).	/etc/inittab /etc/X11/xdm/* /etc/X11/kdm/* /etc/X11/gdm/*
5	Install & Configure XFree86	Candidate should be able to configure and install X and an X font server. This objective includes verifying that the video card and monitor are supported by an X server, as well as customizing and tuning X for the videocard and monitor. It also includes installing an X font server, installing fonts, and configuring X to use the font server	XF86Setup xf86config xvidtune /etc/X11/XF86Config .Xresources

		(may require a manual edit of /etc/X11/XF86Config in the "Files" section).	
5	Install & Customize a Window Manager Environment	Candidate should be able to customize a system-wide desktop environment and/or window manager, to demonstrate an understanding of customization procedures for window manager menus and/or desktop panel menus. This objective includes selecting and configuring the desired x-terminal (xterm, rxvt, aterm etc.), verifying and resolving library dependency issues for X applications, exporting X-display to a client workstation	.xinitrc .Xdefaults xhost DISPLAY environment variable

LPI Exam 102

Topic 105: Kernel

Table A.6. LPI Exam 102: The kernel

Weight	Title	Description	Key Files, terms and utillities
3	Reconfigure, build, and install a custom kernel and kernel	Candidates should be able to customize, build,	/usr/src/Linux/* /usr/src/Linux/.config /lib/modules/kernel-version/*

	modules	and install a kernel and kernel loadable modules from source This objective includes customizing the current kernel configuration, building a new kernel, and building kernel modules as appropriate. It also includes installing the new kernel as well as any modules, and ensuring that the boot manager can locate the new kernel and associated files (generally located under /boot, see objective 1.102.2 for more details about boot manager configuration).	/boot/* make make targets: config, menuconfig, xconfig, oldconfig, modules, install, modules_install, depmod
4	Manage/Query kernel and kernel modules at runtime	Candidates should be able to manage and/or query a kernel and kernel loadable modules. This objective includes using command-line utilities to get information about the currently running kernel and kernel modules. It also includes manually loading and unloading modules as appropriate. It also includes being able	/lib/modules/kernel-version/modu /etc/modules.conf & /etc/conf.modules depmod insmod lsmod rmmod modinfo modprobe uname

		to determine when modules can be unloaded and what parameters a module accepts. Candidates should be able to configure the system to load modules by names other than their file name.	

Topic 106: Boot, Initialization, Shutdown and Runlevels

Table A.7. LPI Exam 102: Boot, Initialization, Shutdown and Runlevels

Weight	Title	Description	Key Files, terms and utillities
3	Boot the system	Candidates should be able to guide the system through the booting process. This includes giving commands to the boot loader and giving options to the kernel at boot time, and checking the events in the log files.	/var/log/messages /etc/conf.modules or /etc/modules.conf dmesg LILO GRUB
3	Change runlevels and shutdown or reboot system	Candidates should be able to manage the runlevel of the system. This objective includes changing to single user mode, shutdown or rebooting the system. Candidates should be able to alert users before switching runlevel,	/etc/inittab shutdown init

		and properly terminate processes. This objective also includes setting the default runlevel.	

Topic 107: Printing

Table A.8. LPI Exam 102: Printing

Weight	Title	Description	Key Files, terms and utillities
1	Manage printers and print queues	Candidates should be able to manage print queues and user print jobs. This objective includes monitoring print server and user print queues and troubleshooting general printing problems.	/etc/printcap lpc lpq lprm lp
1	Print files	Candidates should be able to manage print queues and manipulate print jobs. This objective includes adding and removing jobs from configured printer queues and converting text files to postscript for printing.	lpr lpq mpage
1	Install and configure local and remote printers	Candidate should be able to install a printer daemon, install and configure a print filter (e.g.: apsfilter, magicfilter). This objective includes making local and	/etc/printcap /etc/apsfilter/* /var/lib/apsfilter/*/ /etc/magicfilter/*/ /var/spool/lpd/*/ lpd

		remote printers accessible for a Linux system, including postscript, non-postscript, and Samba printers.	

Topic 108: Documentation

Table A.9. LPI Exam 102: Documentation

Weight	Title	Description	Key Files, terms and utillities
3	Find Linux documentation on the Internet	Candidates should be able to find and use Linux documentation. This objective includes using Linux documentation at sources such as the Linux Documentation Project (LDP), vendor and third-party websites, newsgroups, newsgroup archives, and mailing lists.	Not applicable
4	Use and manage local system documentation	Candidates should be able to use and administer the man facility and the material in /usr/share/doc/. This objective includes finding relevant man pages, searching man page sections, finding	MANPATH man apropos whatis

		commands and man pages related to them, and configuring access to man sources and the man system. It also includes using system documentation stored in /usr/share/doc/ and determining what documentation to keep in /usr/share/doc/.	
1	Notify users on system-related issues	Candidates should be able to notify the users about current issues related to the system. This objective includes automating the communication process, e.g. through logon messages.	/etc/issue /etc/issue.net /etc/motd

Topic 109:

Table A.10. LPI Exam 102: Shells, Scripting, Programming and Compiling

Weight	Title	Description	Key Files, terms and utillities
3	Customize or write simple scripts	Candidate should be able to customize existing scripts, or write simple new (ba)sh scripts. This objective includes using standard sh Syntax (loops,	while for test chmod

		tests), using command substitution, testing command return values, testing of file status, and conditional mailing to the superuser. This objective also includes making sure the correct interpreter is called on the first (#!) line of scripts. This objective also includes managing location, ownership, execution and suid-rights of scripts.	
5	Customize and use the shell environment	Candidate should be able to customize shell environments to meet users' needs. This objective includes setting environment variables (e.g. PATH) at login or when spawning a new shell. It also includes writing bash functions for frequently used sequences of commands.	~/.bash_profile ~/.bash_login ~/.profile ~/.bashrc ~/.bash_logout ~/.inputrc function (Bash built-in command) export env set (Bash built-in command) unset (Bash built-in command)

Topic 111: Administrative Tasks

Table A.11. LPI Exam 102: Administrative Tasks

Weight	Title	Description	Key Files, terms and utillities

3	Configure and use system log files to meet administrative and security needs	Candidate should be able to configure system logs. This objective includes managing the type and level of information logged, manually scanning log files for notable activity, monitoring log files, arranging for automatic rotation and archiving of logs and tracking down problems noted in logs.	/etc/syslog.conf /var/log/* logrotate tail -f
3	Tune the user environment and system environment variables	Candidate should be able to modify global and user profiles. This includes setting environment variables, maintaining skel directories for new user accounts and setting command search path with the proper directory.	/etc/profile /etc/skel env export set unset
4	Manage users and group accounts and related system files	Candidate should be able to add, remove, suspend and change user accounts. Tasks include to add and remove groups, to change user/group info in passwd/group databases. The objective also includes creating special purpose and limited accounts. Key files, terms, and utilities	/etc/passwd /etc/shadow /etc/group /etc/gshadow chage gpasswd groupadd groupdel groupmod grpconv grpunconv passwd pwconv pwunconv useradd userdel usermod

		include:	
4	Automate system administration tasks by scheduling jobs to run in the future	Candidate should be able to use cron or anacron to run jobs at regular intervals and to use at to run jobs at a specific time. Task include managing cron and at jobs and configuring user access to cron and at services.	/etc/anacrontab /etc/at.deny /etc/at.allow /etc/crontab /etc/cron.allow /etc/cron.deny /var/spool/cron/* at atq atrm crontab
4	Maintain an effective data backup strategy	Candidate should be able to plan a backup strategy and backup filesystems automatically to various media. Tasks include dumping a raw device to a file or vice versa, performing partial and manual backups, verifying the integrity of backup files and partially or fully restoring backups.	cpio dd dump restore tar
4	Maintain system time	Candidate should be able to properly maintain the system time and synchronize the clock over NTP. Tasks include setting the system date and time, setting the BIOS clock to the correct time in UTC, configuring the correct timezone for the system and configuring the	/usr/share/zoneinfo /etc/timezone /etc/localtime /etc/ntp.conf /etc/ntp.drift date hwclock ntpd ntpdate

		system to correct clock drift to match NTP clock.	

Topic 112:

Table A.12. LPI Exam 102: Networking Fundamentals

Weight	Title	Description	Key Files, terms and utillities
3	Configure Linux as a PPP client	Candidates should understand the basics of the PPP protocol and be able to configure and use PPP for outbound connections. This objective includes the definition of the chat sequence to connect (given a login example) and the setup commands to be run automatically when a PPP connection is made. It also includes initialisation and termination of a PPP connection, with a modem, ISDN or ADSL and setting PPP to automatically reconnect if disconnected.	/etc/ppp/options.* /etc/ppp/peers/* /etc/wvdial.conf /etc/ppp/ip-up /etc/ppp/ip-down wvdial pppd
4	Fundamentals of TCP/IP	Candidates should demonstrate a proper understanding of network fundamentals. This	/etc/services ftp telnet host ping dig traceroute whois

		objective includes the understanding of IP-addresses, network masks and what they mean (i.e. determine a network and broadcast address for a host based on its subnet mask in "dotted quad" or abbreviated notation or determine the network address, broadcast address and netmask when given an IP-address and number of bits). It also covers the understanding of the network classes and classless subnets (CIDR) and the reserved addresses for private network use. It includes the understanding of the function and application of a default route. It also includes the understanding of basic internet protocols (IP, ICMP, TCP, UDP) and the more common TCP and UDP ports (20, 21, 23, 25, 53, 80, 110, 119, 139, 143, 161).	
7	TCP/IP configuration and troubleshooting	Candidates should be able to view, change and verify configuration settings and	/etc/HOSTNAME or /etc/hostname /etc/hosts /etc/networks /etc/host.conf

		operational status for various network interfaces. This objective includes manual and automatic configuration of interfaces and routing tables. This especially means to add, start, stop, restart, delete or reconfigure network interfaces. It also means to change, view or configure the routing table and to correct an improperly set default route manually. Candidates should be able to configure Linux as a DHCP client and a TCP/IP host and to debug problems associated with the network configuration.	/etc/resolv.conf /etc/nsswitch.conf ifconfig route dhcpcd, dhcpclient, pump host hostname (domainname, dnsdomainname) netstat ping traceroute tcpdump the network scripts run during system initialization.

Topic 113: Networking Services

Table A.13. LPI Exam 102: Networking Services

Weight	Title	Description	Key Files, terms and utillities
4	Setup and configure basic DNS services	Candidate should be able to configure hostname lookups and troubleshoot problems with local caching-only name server. Requires an understanding of	/etc/hosts /etc/resolv.conf /etc/nsswitch.conf /etc/named.boot (v.4) or /etc/named.conf (v.8) named

		the domain registration and DNS translation process. Requires understanding key differences in configuration files for bind 4 and bind 8.	
4	Configure and manage inetd, xinetd, and related services	Candidates should be able to configure which services are available through inetd, use tcpwrappers to allow or deny services on a host-by-host basis, manually start, stop, and restart internet services, configure basic network services including telnet and ftp. Set a service to run as another user instead of the default in inetd.conf.	/etc/inetd.conf /etc/hosts.allow /etc/hosts.deny /etc/services /etc/xinetd.conf /etc/xinetd.log
4	Operate and perform basic configuration of sendmail	Candidate should be able to modify simple parameters in sendmail configuration files (including the "Smart Host" parameter, if necessary), create mail aliases, manage the mail queue, start and stop sendmail, configure mail forwarding and perform basic troubleshooting of sendmail. The objective includes	/etc/aliases or /etc/mail/aliases /etc/mail/* ~/.forward mailq sendmail newaliases

		checking for and closing open relay on the mailserver. It does not include advanced custom configuration of Sendmail. Key files, terms, and utilities include:	
4	Operate and perform basic configuration of Apache	Candidates should be able to modify simple parameters in Apache configuration files, start, stop, and restart httpd, arrange for automatic restarting of httpd upon boot. Does not include advanced custom configuration of Apache.	httpd.conf apachectl httpd
4	Properly manage the NFS, smb, and nmb daemons	Candidate should know how to mount remote filesystems using NFS, configure NFS for exporting local filesystems, start, stop, and restart the NFS server. Install and configure Samba using the included GUI tools or direct edit of the /etc/smb.conf file (Note: this deliberately excludes advanced NT domain issues but includes simple sharing of home directories and printers, as well as correctly setting the nmbd as a WINS	/etc/exports /etc/fstab /etc/smb.conf mount umount

		client).	
4	Set up secure shell (OpenSSH)	The candidate should be able to obtain and configure OpenSSH. This objective includes basic OpenSSH installation and troubleshooting, as well as configuring sshd to start at system boot..	/etc/hosts.allow /etc/hosts.deny /etc/nologin /etc/ssh/sshd_config /etc/ssh_known_hosts /etc/sshrc sshd ssh-keygen

Topic 114: Security

Table A.14. LPI Exam 102: Security

Weight	Title	Description	Key Files, terms and utillities
3	Setup host security	Candidate should know how to set up a basic level of host security. Tasks include syslog configuration, shadowed passwords, set up of a mail alias for root's mail and turning of all network services not in use.	/etc/inetd.conf or /etc/inet.d/* /etc/nologin /etc/passwd /etc/shadow /etc/syslog.conf
4	Perform security administration tasks	Candidates should know how to review system configuration to ensure host security in accordance with local security policies. This objective includes how to configure TCP wrappers, find	/proc/net/ip_fwchains /proc/net/ip_fwnames /proc/net/ip_masquerade find ipchains passwd socket iptables

		files with SUID/SGID bit set, verify packages, set or change user passwords and password aging information, update binaries as recommended by CERT, BUGTRAQ, and/or distribution's security alerts. Includes basic knowledge of ipchains and iptables.	
1	Setup user level security	Candidate should be able to configure user level security. Tasks include limits on user logins, processes, and memory usage. Key files, terms, and utilities include:	quota usermod

Intermediate Level Administration (LPIC2)

Status: Available now; published November 29, 2001 Pre-Requisites: Completion of LPIC Level 1 Requirements: Passing Exams 201 and 202 # Overview of Tasks: To pass Level 2 someone should be able to

- Administer a small to medium-sized site
- Plan, implement, maintain, keep consistent, secure, and troubleshoot a small mixed (MS, Linux) network, including a:
    - LAN server (samba)
    - Internet Gateway (firewall, proxy, mail, news)
    - InternetServer(webserver,FTPserver)

- Supervise assistants
- Advise management on automation and purchases

LPI Exam 201

Topic 201: Linux Kernel

Table A.15. LPI Exam 201: The Linux Kernel

Weight	Title	Description	Key Files, terms and utillities
1	Kernel Components	Candidates should be able to utilize kernel components that are necessary to specific hardware, hardware drivers, system resources and requirements. This objective includes implementing different types of kernel images, identifying stable and development kernels and patches, as well as using kernel modules.	zImage bzImage
1	Compiling a kernel	Candidates should be able to properly compile a kernel to include or disable specific features of the Linux kernel as necessary. This objective includes compiling and recompiling the Linux kernel as needed, implementing	/usr/src/Linux/ /etc/lilo.conf make options (config, xconfig, menuconfig, oldconfig, mrproper zImage, bzImage, modules, modules_install) mkinitrd (both Red Hat and Debian based) make

		updates and noting changes in a new kernel, creating a system initrd image, and installing new kernels.	
1	Customizing a kernel	Candidates should be able to customize a kernel for specific system requirements by patching, compiling, and editing configuration files as required. This objective includes being able to assess requirements for a kernel compile versus a kernel patch as well as build and configure kernel modules.	/usr/src/Linux /proc/sys/kernel/ /etc/conf.modules, /etc/modules.conf patch make modprobe insmod, lsmod kerneld kmod
2	Patching a kernel	Candidates should be able to properly patch a kernel for various purposes including to implement kernel updates, to implement bug fixes, and to add support for new hardware. This objective also includes being able to properly remove kernel patches from existing production kernels.	Makefile patch gzip bzip

Topic 202:

Table A.16. LPI Exam 201: System Startup

Weight	Title	Description	Key Files, terms and utillities
2	Customizing system startup and boot processes	Candidates should be able to edit appropriate system startup scripts to customize standard system run levels and boot processes. This objective includes interacting with run levels and creating custom initrd images as needed.	/etc/init.d/ /etc/inittab /etc/rc.d/ mkinitrd (both Red Hat and Debian scripts)
3	System recovery	Candidates should be able to properly manipulate a Linux system during both the boot process and during recovery mode. This objective includes using both the init utility and init= kernel options.	inittab LILO init mount fsck

Topic 203: Filesystem

Table A.17. LPI Exam 201: Filesystem

Weight	Title	Description	Key Files, terms and utillities
3	Operating	the Linux filesystem Candidates should be able to properly configure and navigate the standard Linux	/etc/fstab /etc/mtab /proc/mounts mount and umount sync swapon swapoff

		filesystem. This objective includes configuring and mounting various filesystem types. Also included, is manipulating filesystems to adjust for disk space requirements or device additions.	
3	Creating and configuring filesystem options	Candidates should be able to configure automount filesystems. This objective includes configuring automount for network and device filesystems. Also included is creating non ext2 filesystems for devices such as CD-ROMs.	/etc/auto.master /etc/auto.[dir] mkisofs dd mke2fs
4	Maintaining a Linux filesystem	Candidates should be able to properly maintain a Linux filesystem using system utilities. This objective includes manipulating a standard ext2 filesystem.	fsck (fsck.ext2) badblocks mke2fs dumpe2fs debuge2fs tune2fs

Topic 204: Hardware

Table A.18. LPI Exam 201: Hardware

Weight	Title	Description	Key Files, terms and utillities
1	Configuring PCMCIA devices	Candidates should be able to configure	/etc/pcmcia/ *.opts cardctl cardmgr

		a Linux installation to include PCMCIA support. This objective includes configuring PCMCIA devices, such as ethernet adapters, to autodetect when inserted.	
2	Configuring RAID	Candidates should be able to configure and implement software RAID. This objective includes using mkraid tools and configuring RAID 0, 1, and 5.	/etc/raidtab mkraid
2	Software and kernel configuration	Candidates should be able to configure kernel options to support various hardware devices including UDMA66 drives and IDE CD burners. This objective includes using LVM (Logical Volume Manager) to manage hard disk drives and particitions as well as software tools to interact with hard disk settings.	/proc/interrupts hdparm tune2fs sysctl
3	Adding new hardware	Candidates should be able to configure internal and external devices for a system including new hard disks, dumb terminal devices, serial UPS devices, multi-port	/proc/bus/usb XFree86 modprobe lsmod lsdev lspci setserial usbview

		serial cards, and LCD panels.	

Topic 209: File and Service Sharing

Table A.19. LPI Exam 201: File and Service Sharing

Weight	Title	Description	Key Files, terms and utillities
5	Configuring a samba server	The candidate should be able to set up a Samba server for various clients. This objective includes setting up a login script for Samba clients, and setting up an nmbd WINS server. Also included is to change the workgroup in which a server participates, define a shared directory in smb.conf, define a shared printer in smb.conf, use nmblookup to test WINS server functionality, and use the smbmount command to mount an SMB share on a Linux client.	smbd, nmbd smbstatus, smbtestparm, smbpasswd, nmblookup smb.conf, lmhosts
3	Configuring an NFS server	The candidate should be able to create an exports file and specify filesystems to be exported. This objective includes editing exports file	/etc/exports exportfs showmount nfsstat

		entries to restrict access to certain hosts, subnets or netgroups. Also included is to specify mount options in the exports file, configure user ID mapping, mount an NFS filesystem on a client, using mount options to specify soft or hard and background retries, signal handling, locking, and block size. The candidate should also be able to configure tcpwrappers to further secure NFS.	

Topic 211: System Maintenance

Table A.20. LPI Exam 201: System Maintenance

Weight	Title	Description	Key Files, terms and utillities
1	System logging	The candidate should be able to configure syslogd to act as a central network log server. This objective also includes configuring syslogd to send log output to a central log server, logging remote connections, and using grep and other text utils to automate log	syslog.conf /etc/hosts sysklogd

		analysis.	
1	Packaging software	The candidate should be able to build a package. This objective includes building (or rebuilding) both RPM and DEB packaged software.	/debian/rules SPEC file format rpm
2	Backup operations	The candidate should be able to create an offsite backup storage plan.	Not applicable

Topic 213: System Customization and Automation

Table A.21. LPI Exam 201: System Customization and Automation

Weight	Title	Description	Key Files, terms and utillities
3	Automating tasks using scripts	The candidate should be able to write simple Perl scripts that make use of modules where appropriate, use the Perl taint mode to secure data, and install Perl modules from CPAN. This objective includes using sed and awk in scripts, and using scripts to check for process execution and generate alerts by email or pager if a process dies. Candidates should be able to write and schedule automatic	perl -MCPAN -e shell bash, awk, sed crontab at

		execution of scripts to parse logs for alerts and email them to administrators, synchronize files across machines using rsync, monitor files for changes and generate email alerts, and write a script that notifies administrators when specified users log in or out.	

Topic 214: Troubleshooting

Table A.22. LPI Exam 201: Troubleshooting

Weight	Title	Description	Key Files, terms and utillities
1	Creating recovery disks	Candidate should be able to: create both a standard bootdisk for system entrance, and a recovery disk for system repair.	/etc/fstab /etc/inittab Any standard editor Familiarity with the location and contents of the LDP Bootdisk-HOWTO /usr/sbin/rdev /bin/cat /bin/mount (includes -o loop switch) /sbin/lilo /bin/dd /sbin/mke2fs /usr/sbin/chroot
1	Identifying boot stages	Candidate should be able to: determine, from bootup text, the 4 stages of boot sequence and distinguish between	boot loader start and hand off to kernel kernel loading hardware initializiation and setup daemon initialization and

		each.	setup
1	Troubleshooting LILO	Candidate should be able to: determine specific stage failures and corrective techniques.	/boot/boot.b Know meaning of L, LI, LIL, LILO, and scrolling 010101 errrors Know the different LILO install locations, MBR, /dev/fd0, or primary/extended partition. Know significance of /boot/boot.### files
1	General troubleshooting	A candidate should be able to recognize and identify boot loader and kernel specific stages and utilize kernel boot messages to diagnose kernel errors. This objective includes being able to identify and correct common hardware issues, and be able to determine if the problem is hardware or software.	/proc filesystem Various system and daemon log files in /var/log/ /, /boot, and /lib/modules screen output during bootup kernel syslog entries in system logs (if entry is able to be gained) location of system kernel and attending modules dmesg /sbin/lspci /usr/bin/lsdev /sbin/lsmod /sbin/modprobe /sbin/insmod /bin/uname strace strings ltrace lsof
1	Troubleshooting system resources	A candidate should be able to identify, diagnose and repair local system environment.	/etc/profile && /etc/profile.d/ /etc/init.d/ /etc/rc.* /etc/sysctl.conf /etc/bashrc /etc/ld.so.conf (or other appropriate global shell configuration files) Core system variables Any standard editor

			/bin/ln /bin/rm /sbin/ldconfig /sbin/sysctl
1	Troubleshooting environment configurations	A candidate should be able to identify common local system and user environment configuration issues and common repair techniques.	/etc/inittab /etc/rc.local /etc/rc.boot /var/spool/cron/crontabs/ /etc/`shell_name`.conf /etc/login.defs /etc/syslog.conf /etc/passwd /etc/shadow /etc/group /etc/profile /sbin/init /usr/sbin/cron /usr/bin/crontab

Exam 202

Topic XXX: Networking

Table A.23. Exam 202: Networking

Weight	Title	Description	Key Files, terms and utillities
5	Basic networking configuration	Modified: 2001-August-24 Maintainer: Kara Pritchard Weight: 5 Description: The candidate should be able to configure a network device to be able to connect to a local network and a wide-area network. This objective includes being able to communicate between various subnets within a single network,	/sbin/route /sbin/ifconfig /sbin/arp /usr/sbin/arpwatch /etc/

		configure dialup access using mgetty, configure dialup access using a modem or ISDN, configure authentication protocols such as PAP and CHAP, and configure TCP/IP logging.	
3	Advanced Network Configuration and Troubleshooting	The candidate should be able to configure a network device to implement various network authentication schemes. This objective includes configuring a multi-homed network device, configuring a virtual private network and resolving networking and communication problems.	/sbin/route /sbin/route /sbin/ifconfig /bin/netstat /bin/ping /sbin/arp /usr/sbin/tcpdump /usr/sbin/lsof /usr/bin/nc

Topic 206: Mail & News

Table A.24. Exam 202: Mail & news

Weight	Title	Description	Key Files, terms and utillities
1	Serving news	Candidates should be able to install and configure news servers using inn. This objective includes customizing and	innd

		monitoring served newsgroups.	
1	Configuring mailing lists	Install and maintain mailing lists using majordomo. Monitor majordomo problems by viewing majordomo logs.	Majordomo2
3	Managing Mail Traffic	Candidates shold be able to implement client mail management software to filter, sort, and monitor incoming user mail. This objective includes using software such as procmail on both server and client side.	procmail .procmailrc
Using Sendmail	4	Candidates should be able to manage a Sendmail configuration including email aliases, mail quotas, and virtual mail domains. This objective includes configuring internal mail relays and monitoring SMTP servers.	/etc/aliases sendmail.cw virtusertable genericstable

Topic 207 DNS

Table A.25. Exam 202: DNS

Weight	Title	Description	Key Files, terms and utillities
2	Basic BIND 8	The candidate	/etc/named.conf

	configuration	should be able to configure BIND to function as a caching-only DNS server. This objective includes the ability to convert a BIND 4.9 named.boot file to the BIND 8.x named.conf format, and reload the DNS by using kill or ndc. This objective also includes configuring logging and options such as directoryh location for zone files.	/usr/sbin/ndc /usr/sbin/named-bootconf kill
3	Create and maintain DNS zones	The candidate should be able to create a zone file for a forward or reverse zone or root level server. This objective includes setting appropriate values for the SOA resource record, NS records, and MX records. Also included is adding hosts with A resource records and CNAME records as appropriate, adding hosts to reverse zones with PTR records, and adding the zone to the /etc/named.conf file using the zone statement with appropriate type, file and masters values. A candidate	contents of /var/named zone file Syntax resource record formats dig nslookup host

		should also be able to delegate a zone to another DNS server.	
3	Securing a DNS server	The candidate should be able to configure BIND to run as a non-root user, and configure BIND to run in a chroot jail. This objective includes configuring DNSSEC statements such as key and trusted-keys to prevent domain spoofing. Also included is the ability to configure a split DNS configuration using the forwarders statement, and specifying a non-standard version number string in response to queries.	SysV init files or rc.local /etc/named.conf /etc/passwd dnskeygen

Topic 208 Web Services

Table A.26. Exam 202: Web Services

Weight	Title	Description	Key Files, terms and utillities
2	Implementing a web server	Candidates should be able to install and configure an Apache web server. This objective includes monitoring Apache load and	access.log .htaccess httpd.conf mod_auth htpasswd htgroup

		performance, restricting client user access, configuring mod_perl and PHP support, and setting up client user authentication. Also included is configuring Apache server options such as maximum requests, minimum and maximim servers, and clients.	
2	Maintaining a web server	Candidates should be able to configure Apache to use virtual hosts for websites without dedicated IP addresses. This objective also includes creating an SSL certification for Apache and defining SSL definitions in configuration files using OpenSSL. Also included is customizing file access by implementing redirect statements in Apache's configuration files.	httpd.conf
2	Implementing a proxy server	Candidates should be able to install and configure a proxy server using Squid. This objective includes impelementing access policies, setting up authentication, and	squid.conf acl http_access

		utilizing memory usage.	

Topic 210 Network Client Management

Table A.27. Exam 202: Network Client Management

Weight	Title	Description	Key Files, terms and utillities
1	NIS configuration	The candidate should be able to configure an NIS server and create NIS maps for major configuration files. This objective includes configuring a system as a NIS client, setting up an NIS slave server, and configuring ability to search local files, DNS, NIS, etc. in nsswitch.conf.	nisupdate, ypbind, ypcat, ypmatch, ypserv, ypswitch, yppasswd, yppoll, yppush, ypwhich, rpcinfo nis.conf, nsswitch.conf, ypserv.conf Contents of /etc/nis/: netgroup, nicknames, securenets Makefile
1	LDAP configuration	The candidate should be able to configure an LDAP server. This objective includes configuring a directory hierarchy, adding group, hosts, services and other data to the hierarchy. Also included is importing items from LDIF files and add items with a management tool, as well as adding users to the	slapd slapd.conf

		directory and change their passwords.	
2	DHCP configuration	The candidate should be able to configure a DHCP server and set default options, create a subnet, and create a dynamically-allocated range. This objective includes adding a static host, setting options for a single host, and adding bootp hosts. Also included is to configure a DHCP relay agent, and reload the DHCP server after making changes.	dhcpd.conf dhcpd.leases
2	PAM authentication	The candidate should be able to configure PAM to support authentication via traditional /etc/passwd, shadow passwords, NIS, or LDAP.	/etc/pam.d pam.conf

Topic 212 System Security

Table A.28. Exam 202: System Security

Weight	Title	Description	Key Files, terms and utillities
1	TCP_wrappers	The candidate should be able to configure tcpwrappers to allow connections	inetd.conf, tcpd hosts.allow, hosts.deny xinetd

		to specified servers from only certain hosts or subnets.	
2	Securing FTP servers	The candidate should be able to configure an anonymous download FTP server. This objective includes configuring an FTP server to allow anonymous uploads, listing additional precautions to be taken if anonymous uploads are permitted, configuring guest users and groups with chroot jail, and configuring ftpaccess to deny access to named users or groups.	ftpaccess, ftpusers, ftpgroups /etc/passwd chroot
2	Configuring a router	The candidate should be able to configure ipchains and iptables to perform IP masquerading, and state the significance of Network Address Translation and Private Network Addresses in protecting a network. This objective includes configuring port redirection, listing filtering rules, and writing rules that accept or block datagrams based	/proc/sys/net/ipv4 /etc/services ipchains iptables routed

		upon source or destination protocol, port and address. Also included is saving and reloading filtering configurations, using settings in /proc/sys/net/ipv4 to respond to DOS attacks, using /proc/sys/net/ipv4/ip_forward to turn IP forwarding on and off, and usingtools such as PortSentry to block port scans and vulnerability probes.	
2	Secure shell (OpenSSH)	The candidate should be able to configure sshd to allow or deny root logins, enable or disable X forwarding. This objective includes generating server keys, generating a user's public/private key pair, adding a public key to a user's authorized_keys file, and configuring ssh-agent for all users. Candidates should also be able to configure port forwarding to tunnel an application protocol over ssh, configure ssh to support the ssh protocol	ssh, sshd /etc/ssh/sshd_config ~/.ssh/identity.pub and identity, ~/.ssh/authorized_keys .shosts, .rhosts

		versions 1 and 2, disable non-root logins during system maintenance, configure trusted clients for ssh logins without a password, and make multiple connections from multiple hosts to guard against loss of connection to remote host following configuration changes.	
3	Security tasks	The candidate should be able to install and configure kerberos and perform basic security auditing of source code. This objective includes arranging to receive security alerts from Bugtraq, CERT, CIAC or other sources, being able to test for open mail relays and anonymous FTP servers, installing and configuring an intrusion detection system such as snort or Tripwire. Candidates should also be able to update the IDS configuration as new vulnerabilities are discovered and apply security patches and	Tripwire telnet nmap

		bugfixes.	

Topic 214 Network Troubleshooting

Table A.29. Exam 202: Network Troubleshooting

Weight	Title	Description	Key Files, terms and utillities
1	Troubleshooting network issues	A candidates should be able to identify and correct common network setup issues to include knowledge of locations for basic configuration files and commands.	/sbin/ifconfig /sbin/route /bin/netstat /etc/network || /etc/sysconfig/network-scripts/ system log files such as /var/log/syslog && /var/log/messages /bin/ping /etc/resolv.conf /etc/hosts /etc/hosts.allow && /etc/hosts.deny /etc/hostname || /etc/HOSTNAME /sbin/hostname /usr/sbin/traceroute /usr/bin/nslookup /usr/bin/dig /bin/dmesg host

This page intentionally left blank

Appendix B. Linux kernel version 2.6

The Wonderful World of Linux

The following is a article written by Joseph Pranevich when version 2.6 of the Linux Kernel was released. It details the new capabilities of the kernel. The material is used by kind permission of Joseph Pranevich, the latest version of the article can be found at: http://kniggit.net/wwol26.html. This material is not covered by the creative Commons License agreement for this manual. To use this material outside the context of this manual, please contact jpranevich <at> kniggit.net. (Mail address disguised to avoid web-crawlers looking for email addresses to send spam to)

Joseph Pranevich - jpranevich <at> kniggit.net

Although it seems like only yesterday that we were booting up our first Linux 2.4 systems, time has ticked by and the kernel development team has just released the 2.6 kernel to the public. This document is intended as a general overview of the features in the new kernel release, with a heavy bias toward i386 Linux. Please also be aware that some of the "new" features discussed here may have been back-ported to Linux 2.4 after first appearing in Linux 2.6, either officially or by a distribution vendor. I have also included information on a handful of cases where a new feature originated during the maintenance cycle of Linux 2.4, and those will be marked as appropriate in the text.

At present, this document has been translated into ten languages. Please see the "Translations" section at the very bottom for more information.

The Story So Far...

The Linux kernel project was started in 1991 by Linus Torvalds as a Minix-like Operating System for his 386. (Linus had originally wanted to name the project Freax, but the now-familiar name is the one that stuck.) The first official release of Linux 1.0 was in March 1994, but it supported only single-processor i386 machines. Just a year later, Linux 1.2 was released (March 1995) and was the first version with support for different hardware platforms (specifically: Alpha, Sparc, and Mips), but still only single-processor models. Linux 2.0 arrived in June of 1996 and also included support for a number of new architectures, but more importantly brought Linux into the world of multi-processor machines (SMP). After 2.0, subsequent major releases have been somewhat slower in coming (Linux 2.2 in January 1999 and 2.4 in January 2001), each revision expanding Linux's support for new hardware and system types as well as boosting scalability. (Linux 2.4 was also notable in

being the release that really broke Linux into the desktop space with kernel support for ISA Plug-and-Play, USB, PC Card support, and other additions.) Linux 2.6, released 12/17/03, stands not only to build on these features, but also to be another "major leap" with improved support for both significantly larger systems and significantly smaller ones (PDAs and other devices.)

Core Hardware Support

One of the most important strengths of Linux-powered operating systems is their flexibility and their ability to support a wide range of hardware platforms. While this document is geared specifically to the uses of Linux on PC-derived hardware types, the Linux 2.6 kernel has made some remarkable improvements in this area that deserve to be pointed out.

Scaling Down -- Linux for Embedded Systems

One of the two most fundamental changes to Linux in 2.6 comes through the acceptance and merging of much of the uCLinux project into the mainstream kernel. The uCLinux project (possibly pronounced "you-see-Linux", but more properly spelled with the Greek character "mu") is the Linux for Microcontrollers project. This variant of Linux has already been a major driver of support for Linux in the embedded market, and its inclusion in the official release should encourage further development in this space. Unlike the "normal" Linux ports that we are generally accustomed to, embedded ports do not have all the features that we associate with the kernel, due to hardware limitations. The primary difference is that these ports feature processors that do not feature an MMU. ("memory management unit" - what makes a protected-mode OS "protected") While these are generally true multitasking Linux systems, they are missing memory protection and other related features. (Without memory protection, it is possible for a wayward process to read the data of, or even crash, other processes on the system.) This may make them unusable for a multi-user system, but an excellent choice for a low-cost PDA or dedicated device. It is difficult to over-emphasize this architecture shift in Linux 2.6; all versions of Linux up to this point were derived (however indirectly) from the limitations inherent with Linus' initial work on his Intel 80386.

There are several new lines of embedded processors supported by Linux 2.6, including Hitachi's H8/300 series, the NEC v850 processor, and Motorola's line of embedded m68k processors. Motorola's offerings are the most familiar to the average Linux user as they are the guts underneath Palm Pilots starting with the first (the Palm 1000), up until the Palm III. Other models go by names such as Dragonball and ColdFire and are included on systems and evaluation boards manufactured by Motorola, Lineo, Arcturus, and others. Sadly, support for other, older m68k processors without MMUs (such as the 68000s used in early

Macintoshes) is not yet covered by the new release but it is highly possible that "hobbyist" projects of the future will seek to support Linux on these and other antique system.

Although not a part of the uCLinux merge, the new revision of Linux also include support for Axis Communications' ETRAX CRIS ("Code Reduced Instruction Set") processors. (Actual support for this processor arrived as a feature during the 2.4 kernel's maintenance cycle - - well after the 2.4.0 release- - so it deserves a brief mention.) These are embedded processor, but with MMUs, that is primarily used in network hardware. Related support for MMU-less variants of these processors has not yet been accepted into the kernel, but are being worked on by outside projects.

In addition to pure hardware support, there have been a number of other wins through the integration of the embedded work into the mainline kernel. While most of these changes are under the hood, changes such as ability to build a system completely without swap support add to the overall robustness of the OS.

Scaling Up -- NUMA and Bigger Iron

The second of the two most fundamental changes in Linux 2.6 happens to work in the other direction: to make Linux a more acceptable kernel on larger and larger servers. (Some of these larger servers will be i386 based, and some not.) The big change in this respect is Linux's new support for NUMA servers. NUMA (or "Non-Uniform Memory Access") is a step beyond SMP in the multi-processing world and is a major leap forward for efficiency on systems that have many processors. Current multiprocessing systems were designed with many of the same limitations as their uniprocessor counterparts, especially as only a single pool of memory is expected to serve all processors. On a many-processor system, there is a major performance bottleneck due to the extremely high contention rate between the multiple cpus onto the single memory bus. NUMA servers get around that difficulty by introducing the concept that, for a specific processor, some memory is closer than others. One easy way (and not terribly technically incorrect) to imagine this is that you have a system built with separate cards, each containing CPUs, memory, and possibly other components (I/O, etc.) There are many of these cards in a system and while they can all talk to each other, it's pretty clear that the CPUs will have the easiest time talking to the local memory (the memory on the cpu card rather than on a separate card.)You can imagine the new NUMA architecture being somewhat similar to a very tight-knit cluster at the lowest levels of hardware.

To properly support these new NUMA machines, Linux had to adapt in several respects to make the new model efficient. To start with, an internal topology API was created to actually let the kernel internals understand one processor or one memory pool's relations to I/O devices and each other. Derived from that, the Linux process scheduler now is capable of understanding these relationships and will attempt to optimize tasks for best use of local resources. Additionally, many NUMA machines are built in such a way that they have "holes" in the linear memory space

"between" nodes. The new kernel is able to deal with those discontinuous cases in a reasonable way. There are many other internal changes which were made to allow Linux to support these new high-end machines, and this is definitely an area of growth for the kernel as a whole. However, this is an area where Linux is very rapidly growing and maturing and much work remains to be done to make the most efficient use of resources possible. Over the course of the next year, we can expect to see many more improvements in Linux's support for these very high-end systems.

Sub-architecture Support

While not quite as core as the two previous changes, the new revision of the kernel also includes a new concept called a "Sub-architecture" which further extends Linux's reach into new hardware types. Previously, Linux often had the underlying assumption that processor types and hardware types went hand in hand. That is, that i386-descendant processors are only used on PC/AT-descendant servers. In Linux 2.4, this assumption was broken for i386 with the addition of support for SGI's Visual Workstation, a "legacy-less" platform running with an Intel chip. (And in fact, it was broken long before on many other architectures. For example, m68k has long supported Amigas, Macintoshes, and other platforms.) The big change in Linux 2.6 is that this feature and concept was standardized so that all architectures handle this in a similar and saner way that allows for more clear separation of the components that need to be separated.

With this standardization comes two new platforms to support for i386. The first is NCR's Voyager architecture. This is a SMP system (developed before the now-standard Intel MP specification) supporting 486-686 processors in up to 32x configurations. The actual number of configurations that were sold with this architecture is relatively small, and not all machines are supported yet. (The oldest ones are unsupported.) The second architecture supported is the more widespread PC-9800 platform developed by NEC into the (almost) dominant PC platform in Japan until relatively recently. The original PC-9800 machines shipped with an 8086 processor and the line eventually evolved and matured (in parallel with the AT-descendants) until they featured Pentium-class processors and SMP support. (Of course, the support for Linux is limited to 386 or better.) Although completely unknown in the US, versions of Microsoft products up until Windows 95 were ported to run on this hardware. The line has been officially discontinued by the manufacturer in favor of more "standard" PCs.

By formalizing Linux's support for these "slightly different" hardware types, this will more easily allow the kernel to be ported to other systems, such as dedicated storage hardware and other components that use industry-dominant processor types. To be absolutely clear though, one should not take this subdivision too far. These Sub-architecture have been separated because very low-level components of the system (such as IRQ routing) are slightly or radically different. This is quite different than running Linux on an X-Box, for example, where relatively little other than hardware drivers and some quirks separate the system from being a "generic"

i386 system. Support for the X-Box would not be a Sub-architecture.

Hyperthreading

Another major hardware advancement supported under Linux 2.6 is hyperthreading. This is the ability, currently only built into modern Pentium 4 processors but applicable elsewhere, allows a single physical processor to masquerade (at the hardware level) as two or more processors. This allows for performance boosts in some circumstances, but also adds scheduling complexity and other issues. Key in the kernel's improved support for this feature is that the scheduler now knows how to recognize and optimize processor loads across both real and virtual processors within a machine. In previous versions of Linux, it was quite possible to overwork a single processor because it was not possible to factor in the workload as a whole. One of the great things to note about this feature is that Linux was ahead of the market curve on supporting this new hardware feature transparently and intelligently. (Windows 2000 servers can see the additional faux-processors, but does not recognize them as virtual. Thus, you also require additional CPU licenses to take advantage of the feature. It was not until the Windows XP release that Microsoft completely supported this feature.)

Linux Internals

Scalability Improvements

In addition to the previously described generic features such as NUMA and hyperthreading, Linux 2.6 also has other changes for Intel servers at the top of the food chain. First and foremost is improved support for other new Intel hardware features including Intel's PAE ("Physical Address Extension") which allows most newer 32-bit x86 systems to access up to 64GB of RAM, but in a paged mode. In addition, IRQ balancing has been significantly improved on multiprocessor systems through major improvements to Linux's APIC support.

In addition to just supporting new hardware features, internal limits have been also increased when possible. For example, the number of unique users and groups on a Linux system has been bumped from 65,000 to over 4 billion. (16-bit to 32-bit), making Linux more practical on large file and authentication servers. Similarly, The number of PIDs (Process IDs) before wraparound has been bumped up from 32,000 to 1 billion, improving application starting performance on very busy or very long-lived systems. Although the maximum number of open files has not been increased, Linux with the 2.6 kernel will no longer require you to set what the limit is in advance; this number will self-scale. And finally, Linux 2.6 will include improved 64-bit support on block devices that support it, even on 32-bit platforms such as i386. This allows for filesystems up to 16TB on common hardware.

Another major scalability improvement in Linux 2.6 is that the kernel itself can now

not only support more types of devices, but also support more devices of a single type. Under all iterations of Linux (and indeed, most Unix-derived operating systems), users and applications running on a system communicate with the attached hardware using numbered device nodes. (The entries in the "/dev" directory.) These device nodes were limited to 255 "major" devices (generally, one type of device gets one or more device nodes) and 255 "minor" numbers (generally, specific devices of that type.) For example, the "/dev/sda2" device (the second partition on the first detected SCSI disk), gets a major number of 8, common for all SCSI devices, and a minor number of 2 to indicate the second partition. Different device types allocate their major and minor numbers differently, so it can't be said with assurance how many devices you can have on a Linux system. Unfortunately, this system breaks down badly on large systems where it would be possible, for example, to have many more than 255 of any specific device in a system. (Think large storage arrays, print farms, etc.) Under Linux 2.6, these limitations have been eased to allow for 4095 major device types and a more than a million subdevices per type. This increase should be more than adequate to support high-end systems for the time being.

Interactivity and Responsiveness

In addition to just scaling up, another priority with the new release has been to make the system more responsive. This is useful not only for the general desktop user (who always likes to see things respond quickly), but also to more time-critical applications where absolute preciseness is required to achieve the desired effect. Despite these changes, Linux 2.6 will not be a "hard" Real Time OS, which has very strict requirements for absolutely ensuring that actions happen predictably, but the overall responsiveness improvements should appeal to all classes of Linux users. (That said, there are external projects which have unofficial patches to provide RTOS functionality. Those projects could conceivably be made official in the next major release.)

One of the key improvements in Linux 2.6, is that the kernel is finally preemptible. In all previous versions of Linux, the kernel itself cannot be interrupted while it is processing. (On a system with multiple processors, this was true on a per-CPU basis.) Under Linux 2.6, the kernel now can be interrupted mid-task, so that other applications can continue to run even when something low-level and complicated is going on in the background. Of course, there are still times when the kernel cannot be interrupted in its processing. In reality, most users never saw these delays, which are rarely over small fractions of a second. Despite that, many users may notice an improvement in interactive performance with this feature enabled; things like user input will "feel" faster, even when the system is bogged down.

Linux's Input/Output (I/O) subsystems has also undergone major changes to allow them to be more responsive under all sorts of workloads. These changes include a complete rewrite of the I/O scheduler, the code of the kernel that determines what processes get to read from devices and when. The newly rewritten layer is now better capable of ensuring that no processes get stuck waiting in line for too long,

while still allowing for the older optimizations which made sure that reading data still happens in the most efficient way for the underlying hardware.

On the application software side, another change that will help make Linux programs more responsive (if they use the feature) is support for new "futexes" (or "Fast User-Space Mutexes") Futexes are a way in which multiple processes or threads can serialize events so that they don't trample on each other (a "race condition"). Unlike the traditional mutex operations that most threading libraries support, this concept is partially kernel based (but only in the contention case) and it also supports setting priorities to allow applications or threads of higher priority access to the contested resource first. By allowing a program to prioritize waiting tasks, applications can be made to be more responsive in timing-critical areas.

In addition to all of the above, there have been a number of other smaller changes which will improve interactivity and performance in many cases. These include more removals of the "Big Kernel Lock" (non-fine-grained locks which were used in the early days' of Linux's support for multiple processors), optimizations of filesystem readahead, writeback, and manipulating small files, and other similar changes.

Other Improvements

Linux, like the Open Source movement in general, has always been a flag-bearer for the benefits of open standards. Another major change in the 2.6 release, is that the kernel's internal threading infrastructure has been rewritten to allow the Native POSIX Thread Library (NPTL) to run on top of it. This can be a major performance boost for Pentium Pro and better processors in heavily threaded applications, and many of the top players in the "enterprise" space have been clamoring for it. (In fact, RedHat has already backported the support to Linux 2.4 and includes it starting with RedHat 9 and Advanced Server 3.0) This change includes new concepts to the Linux thread space including thread groups, local memory for individual threads, POSIX-style signals, and other changes. One of the major drawbacks is that applications (such as some versions of Sun Java) not written to spec that rely on old Linux-isms will break with the new support enabled. As the benefits overwhelm the cost (and with so many large players in the game), it's clear that most important applications will support the changes before too long after the new kernel is released.

Module Subsystem and the Unified Device Model

Increasingly in modern operating systems, the device handling subsystems have taken on new prominence as they are forced to deal with a myriad of internal and external bus types and more devices by more vendors than you can shake a stick at. It should come as no surprise then, that the upcoming upgrade to the Linux kernel

will include improved support both in its module loader, but also in its internal understanding of the hardware itself. These changes range from the purely cosmetic (driver modules now use a ".ko" extension, for "kernel object", instead of just ".o") to a complete overhaul of the unified device model. Throughout all of these changes is an emphasis on stability and better grasp of the limitations of the previous revision.

Strictly in the module (driver) subsystem, there are a handful of major changes to improve stability. The process for unloading modules have been changed somewhat to reduce cases where it is possible for modules to be used while they are still being unloaded, often causing a crash. For systems where this problem cannot be risked, it is now even possible to disable unloading of modules altogether. Additionally, there has been extensive effort to standardize the process by which modules determine and announce what hardware they support. Under previous versions of Linux, the module would "know" what devices it supported, but this information was not generally available outside of the module itself. This change will allow hardware management software, such as RedHat's "kudzu", to make intelligent choices even on hardware that would not otherwise recognize. Of course, in the event that you know better than the current version of the driver what it supports, it is still possible to force a driver to try to work on a specific device.

Outside of just module loading, the device model itself has undergone significant changes in the updated kernel release. Unlike the module loader, which just has to concern itself with detecting the resource requirements of incoming hardware, the device model is a deeper concept which must be completely responsible for all of the hardware in the system. Linux versions 2.2 and earlier had only the barest support for a unified device model, preferring instead to leave almost all knowledge of the hardware solely at the module level. This worked fine, but in order to use all of the features of modern hardware (especially ACPI), a system needs to know more than just what resources a device uses: it needs to know things like what bus it is connected to, what subdevices it has, what its power state is, whether it can be reconfigured to use other resources in the event of contention, and even to know about devices that haven't had modules loaded for them yet. Linux 2.4 expanded on this foundation to become the first edition to unify the interfaces for PCI, PC Card, and ISA Plug-and-Play busses into a single device structure with a common interface. Linux 2.6, through its new kernel object ("kobject") subsystem, takes this support to a new level by not only expanding to know about all devices in a system, but also to provide a centralized interface for the important little details like reference counting, power management, and exports to user-space.

Now that an extensive amount of hardware information is available within the kernel, this has allowed Linux to better support modern laptop and desktop features that require a much more in-depth knowledge of hardware. The most readily apparent application is this is the increasing proliferation of so called "hot plug" devices like PC Cards, USB and Firewire devices, and hot-plug PCI. While it's hard to think back that far now, Linux didn't offer true support for any of these devices until the 2.2 kernel. Given that hot-plugging is the rule these days and not the

exception, it is fitting that the new device infrastructure essentially eliminates the differences between a hot-plug and a legacy device. Since the kernel subsystem does not directly differentiate between a device discovered at boot time from one discovered later, much of the infrastructure for dealing with pluggable devices has been simplified. A second up and coming driver of this newly rewritten subsystem is for improved support of modern power management. The new power management standard in recent years, called ACPI for "Advanced Configuration and Power Interface", was first supported in rough form for the previous version of the kernel. Unlike old-fashioned APM ("Advanced Power Management"), OSes run on systems with this new interface are required to individually tell all compatible devices that they need to change their power states. Without a centralized understanding of hardware, it would be impossible for the kernel to know what devices it needs to coordinate with and in what order. Although these are just two obvious examples, there are clearly other areas (such as hardware auditing and monitoring) that will benefit from a centralized picture of the world.

The final, but possibly the most obvious, ramification of the new centralized infrastructure is the creation of a new "system" filesystem (to join 'proc' for processes, 'devfs' for devices, and 'devpts' for Unix98 pseudo-terminals) called 'sysfs'. This filesystem (intended to be mounted on '/sys') is a visible representation of the device tree as the kernel sees it (with some exceptions). This representation generally includes a number of known attributes of the detected devices, including the name of the device, its IRQ and DMA resources, power status, and that sort of thing. However, one aspect of this change that may be confusing on the short term is that many of the device-specific uses of the "/proc/sys" directory may be moved into this new filesystem. This change has not (yet) been applied consistently, so there may continue to be an adjustment period.

System Hardware Support

As Linux has moved forward over the years and into the mainstream, each new iteration of the kernel appeared to be leaps and bounds better than the previous in terms of what types of devices it could support-- both in terms of emerging technologies (USB in 2.4) and older "legacy" technologies (MCA in 2.2). As we arrive at the 2.6 however, the number of major devices that Linux does not support is relatively small. There are few, if any, major branches of the PC hardware universe yet to conquer. It is for that reason that most (but certainly not all) of improvements in i386 hardware support have been to add robustness rather than new features.
Internal Devices

Arguably as important as the processor type, the underling bus(es) in a system are the glue that holds things together. The PC world has been blessed with no shortness of these bus technologies, from the oldest ISA (found in the original IBM PC) to modern external serial and wireless busses. Linux has always been quick to adapt to a new bus and device type as they have become popular with consumer devices, but significantly less quick adapting to technologies that get relatively little use.

Improvements in Linux's support for internal devices are really spread across the board. One specific example where Linux is playing "catch up" is support for the old ISA ("Industry Standard Architecture") Plug-and-Play extensions. Linux didn't offer any built-in support for PnP at all until the 2.4 release. This support has been rounded out with the upcoming kernel to include full PnP BIOS support, a device name database, and other compatibility changes. The sum of all of those modifications, is that now Linux is now a "true" Plug-and-Play OS and may be set as such in a compatible machine's BIOS. Other legacy busses such as MCA ("Microchannel Architecture") and EISA ("Extended ISA") have also been wrapped into the new device model and feature device naming databases. On a more modern front Linux 2.6 brings to the table a number of incremental improvements to its PCI ("Peripheral Component Interconnect") subsystem including improved Hot-Plug PCI and power management, support for multiple AGPs ("accelerated graphics ports" -- a separate high-speed extension to the PCI bus), and other changes. And finally, in addition to all of the "real" busses, Linux 2.6 has internally added the concept of a "legacy" bus that is specific to each architecture and contains all of the assumed devices that you would expect to find. On a PC, for example, this may include on-board serial, parallel, and PS/2 ports-- devices that exist but are not enumerated by any real busses on the system. This support may require more complicated work (such as querying firmware) on some platforms, but in general this is just a wrapper to ensure that all devices are handled in a standard way in the new driver paradigm.

External Devices

While it is true that the older-style internal device busses have not seen many new features during the most recent development cycle, the same cannot be said for hot new external hardware. Possibly the most important change in this space is the new support for USB 2.0 devices. These devices, commonly referred to as "high speed" USB devices, support device bandwidth of up to 480 megabits per second, compared to 12 mbit/sec of current USB. A related new standard, USB On-the-Go (or USB OTG), a point-to-point variant on the USB protocol for connecting devices directly together (for example, to connect a digital camera to a printer without having a PC in the middle) is not currently supported in Linux 2.6. (Patches for this feature are available, but not yet rolled into the official release.) In addition to device support, much of the way USB devices have been internally enumerated has been revised so that it is now possible to have many more devices of the same type all accessible from within Linux. In addition to the large changes, there has been an emphasis placed in this development cycle on simplification, stability, and compatibility that should improve the support of USB devices for all Linux users.

On the complete opposite end of the field, Linux 2.6 for the first time includes support that allows a Linux-powered machine to be a USB device, rather than a USB host. This would allow, for example, your Linux-powered PDA to be plugged into your PC and to have both ends of the line speaking the proper protocol. Much of this support is new, but this is an essential direction for Linux to move into for embedded devices.

Wireless Devices

Wireless technology has really taken off within the public in the past several years. It often seems as if cords (except power... maybe?) will be a thing of the past within a handful of years. Wireless devices encompass both networking devices (the most common currently) and also more generic devices such as PDAs, etc.

In the wireless networking space, devices can generally be divided into long range (for example, AX.25 over amateur radio devices) and short range (usually 802.11, but some older protocols exist.) Support for both of these has been a hallmark of Linux since the early days (v1.2) and both of these subsystems have been updated during development of 2.6. The largest change here is that major components of the short range subsystems for the various supported cards and protocols has been merged into a single "wireless" subsystem and API. This merge resolves a number of minor incompatibilities in the way different devices have been handled and strengthens Linux's support for the subsystem by making a central set of userspace tools that will work with all supported devices. In addition to just standardization, Linux 2.6 introduces a number of overall improvements including better capability to notify in the event of a state change (such as a device that has a "roaming" state) and a change to TCP to better handle periodic delay spikes which occur with wireless devices. Due to the immediate desire to better support wireless devices in the current Linux 2.4 kernel, many of these changes have already been back-ported and are available for use.

In the generic wireless devices space, there have been similar major advancements. IrDA (the infrared protocol named for the Infrared Data Associates group) has received some advancements since the last major release such as power management and integration into the new kernel driver model. The real advancements however have been made in providing Linux support for Bluetooth devices. Bluetooth is a new wireless protocol that is designed to be short range and low on power consumption, but does not have the line of sight limitations that IrDA has. Bluetooth as a protocol is designed to go "anywhere" and has been implemented in devices like PDAs, cell phones, printers, and more bizarre things such as automotive equipment. The protocol itself is made up of two different data link types: SCO, or "Synchronous Connection Oriented", for lossy audio applications; and L2CAP, or "Logical Link Control and Adaptation Protocol", for a more robust connection supporting retransmits, etc. The L2CAP protocol further supports various sub-protocols (including RFCOMM for point-to-point networking and BNEP for Ethernet-like networking.) Linux's support for the things that Bluetooth can do continues to grow and we can expect this to mature significantly once more devices are in the hands of the consumers. It should also be mentioned that initial support for Bluetooth has been integrated into later editions of the 2.4 kernel.

Block Device Support

Storage Busses

Dedicated storage busses, such as IDE/ATA ("Integrated Drive Electronics/Advanced Technology Attachment") and SCSI ("Small Computer System Interface"), have also received a major update during the 2.6 cycle. The most major changes are centered around the IDE subsystem which has been rewritten (and rewritten again) during the development of the new kernel, resolving many scalability problems and other limitations. For example, IDE CD/RW drives can now be written to directly through the real IDE disk driver, a much cleaner implementation than before. (Previously, it was required to also use a special SCSI-emulating driver which was confusing and often difficult.) In addition, new support has been added for high-speed Serial ATA (S-ATA) devices, which have transfer rates exceeding 150 MB/sec. On the SCSI side, there have also been many small improvements scattered around the system both for wider support and scalability. One specific improvement for older systems is that Linux now supports SCSI-2 multipath devices that have more than 2 LUNs on a device. (SCSI-2 is the previous version of the SCSI device standard, circa 1994.) Another important change is that Linux can now fall back to test media changing like Microsoft Windows does, to be more compatible with devices that do not completely follow the specification. As these technologies have stabilized over time, so too has Linux's support for them.

Although not a storage bus in itself, Linux now includes support for accessing a newer machine's EDD ("Enhanced Disk Device") BIOS directly to see how the server views its own disk devices. The EDD BIOS includes information on all of the storage busses which are attached to the system that the BIOS knows about (including both IDE and SCSI.) In addition to just getting configuration and other information out of the attached devices, this provides several other advantages. For example, this new interface allows Linux to know what disk device the system was booted from, which is useful on newer systems where it is often not obvious. This allows intelligent installation programs to consider that information when trying to determine where to put the Linux boot loader, for example.

In addition to all of these changes, it should be stressed again that all of the bus device types (hardware, wireless, and storage) have been integrated into Linux's new device model subsystem. In some cases, these changes are purely cosmetic. In other cases, there are more significant changes involved (in some cases for example, even logic for how devices are detected needed to be modified.)

Filesystems

The most obvious use of a block device on a Linux (or any other) system is by creating a filesystem on it, and Linux's support for filesystems have been vastly improved since Linux 2.4 in a number of respects. Key among these changes include support for extended attributes and POSIX-style access controls.

When dealing strictly with conventional Linux filesystems, the extended filesystems

(either "ext2" or "ext3") are the systems most associated with a core Linux system. (ReiserFS is the third most common option.) As these are the filesystems that users care about the most, they have also been the most improved during the development of Linux 2.6. Principal among these changes is support for "extended attributes", or metadata that can be embedded inside the filesystem itself for a specific file. Some of these extended attributes will be used by the system and readable and writable by root only. Many other operating systems, such as Windows and the MacOS, already make heavy use of these kinds of attributes. Unfortunately, the Unix legacy of operating systems have not generally included good support for these attributes and many user-space utilities (such as 'tar') will need to be updated before they will save and restore this additional information. The first real use of the new extended attribute subsystem is to implement POSIX access control lists, a superset of standard Unix permissions that allows for more fine-grained control. In addition to these changes for ext3, there are several other smaller changes: the journal commit time for the filesystem can now be tuned to be more suited for laptop users (which might have to spin up the drive if it were in a power save mode.), default mount options can now also be stored within the filesystem itself (so that you don't need to pass them at mount time), and you can now mark a directory as "indexed" to speed up searches of files in the directory.

In addition to the classic Linux filesystems, the new kernel offers full support for the new (on Linux) XFS filesystem. This filesystem is derived from and is block-level compatible with the XFS filesystem used by default on Irix systems. Like the extended filesystems and Reiser, it can be used as a root-disk filesystem and even supports the newer features such as extended attributes and ACLs. Many distributions are beginning to offer support for this filesystem on their Linux 2.4-based distributions, but it remains to be seen yet what place this filesystem will have in the already crowded pantheon of Unix-style filesystems under Linux.

Outside of those, Linux has also made a number of improvements both inside and outside the filesystem layer to improve compatibility with the dominant PC operating systems. To begin with, Linux 2.6 now supports Windows' Logical Disk Manager (aka "Dynamic Disks"). This is the new partition table scheme that Windows 2000 and later have adopted to allow for easier resizing and creation of multiple partitions. (Of course, it is not likely that Linux systems will be using this scheme for new installations anytime soon.) Linux 2.6 also features improved (and rewritten) support for the NTFS filesystem and it is now possible to mount a NTFS volume read/write. (Writing support is still experimental and is gradually being improved; it may or may not be enabled for the final kernel release.) And finally, Linux's support for FAT12 (the DOS filesystem used on really old systems and floppy disks) has been improved to work around bugs present in some MP3 players which use that format. Although not as dominant in the marketplace, Linux has also improved compatibility with OS/2 by adding extended attribute support into the HPFS filesystem. Like previous releases, the new additions to Linux 2.6 demonstrate the importance of playing well with others and reinforces Linux's position as a "Swiss Army Knife" operating system.

In addition to these changes, there have been a large number of more scattered changes in Linux's filesystem support. Quota support has been rewritten to allow for the larger number of users supported on a system. Individual directories can now be marked as synchronous so that all changes (additional files, etc.) will be atomic. (This is most useful for mail systems and directory-based databases, in addition to slightly better recovery in the event of a disk failure.) Transparent compression (a Linux-only extension) has been added to the ISO9660 filesystem (the filesystem used on CD-ROMs.) And finally, a new memory-based filesystem ("hugetlbfs") has been created exclusively to better support shared memory databases.

Input / Output Support

On the more "external" side of any computer system is the input and output devices, the bits that never quite seem as important as they are. These include the obvious things like mice and keyboards, sound and video cards, and less obvious things like joysticks and accessibility devices. Many of Linux's end-user subsystems have been expanded during the 2.6 development cycle, but support for most of the common devices were already pretty mature. Largely, Linux 2.6's improved support for these devices are derived directly from the more general improvments with external bus support, such as the ability to use Bluetooth wireless keyboards and similar. There are however a number of areas where Linux has made larger improvements.

Human Interface Devices

One major internal change in Linux 2.6 is the reworking of much of the human interface layer. The human interface layer is the center of the user experience of a Linux system, including the video output, mice, and keyboards. In the new version of the kernel, this layer has been reworked and modularized to a much greater extent than ever before. It is now possible to create a completely "headless" Linux system without any included support for a display or anything. The primary benefit of this modularity may be for embedded developers making devices that can only be administrated over the network or serial, but end-users benefit as many of the underlying assumptions about devices and architectures has been modularized out. For example, it was previously always assumed that if you had a PC that you would need support for a standard AT (i8042) keyboard controller; the new version of Linux removes this requirement so that unnecessary code can be kept out of legacy-less systems.

Support to Linux's handling of monitor output has also received a number of changes, although most of these are useful only in configurations that make use of the kernel's internal framebuffer console subsystem. (Most Intel Linux boxes are not configured this way, but that is not the case for many other architectures.) In my personal opinion, the best feature is that the boot logo (a cute penguin, if you've never seen it) now supports resolutions up to 24bpp. That aside, other new features for the console include resizing and rotating (for PDAs and similar) and expanded

acceleration support for more hardware. And finally, Linux has now included kernel support for querying VESA ("Video Electronics Standard Association") monitors for capability information, although XFree86 and most distributions installation systems already have covered this detail in user-space.

In addition to the big changes, Linux 2.6 also includes a number of smaller changes for human interaction. Touch screens, for example, are now supported. The mouse and keyboard drivers have also been updated and standardized to only export a single device node ("/dev/input/mouse0", for example) regardless of the underlying hardware or protocol. Bizarre mice (with multiple scroll wheels, for example) are now also supported. PC keyboard key mappings have also been updated to follow the Windows "standard" for extended keys. Joystick support has also been improved thanks not only to the addition of many new drivers (including the X Box gamepad), but also to include newer features such as force-feedback. And finally (but not least important), the new release also includes support for the Tieman Voyager braille TTY device to allow blind users better access to Linux. (This feature is important enough that it has been back-ported to Linux 2.4 already.)

As a side note, Linux has also changed the "system request" interface to better support systems without a local keyboard. The system request ("sysrq") interface is a method for systems administrators at the local console to get debugging information, force a system reboot, remount filesystems read-only, and do other wizardly things. Since Linux 2.6 now supports a completely headless system, it is now also possible to trigger these events using the /proc filesystem. (Of course, if your system hangs and you need to force it to do things, this may not be of much help to you.)

Audio & Multimedia

One of the most anticipated new features of Linux 2.6 for desktop users is the inclusion of ALSA (the "Advanced Linux Sound Architecture") in lieu of the older sound system. The older system, known as OSS for "Open Sound System", has served Linux since the early days but had many architectural limitations. The first major improvement with the new system is that it has been designed from the start to be completely thread and SMP-safe, fixing problems with many of the old drivers where they would not work properly outside the expected "desktop-means-single-cpu paradigm." More importantly, the drivers have been designed to be modular from the start (users of older versions of Linux will remember that modularity was retro-fitted onto the sound system around Linux 2.2), and that this allows for improved support for systems with multiple sound cards, including multiple types of sound cards. Regardless of how pretty the internals are, the system would not be an improvement for users if it did not have neat new whiz-bang features, and the new sound system has many of those. Key among them are support for newer hardware (including USB audio and MIDI devices), full-duplex playback and recording, hardware and non-interleaved mixing, support for "merging" sound devices, and other things. Whether you are an audiophile or just someone that likes to play MP3s, Linux's improved sound support should be a

welcome step forward.

Beyond simple audio these days, what users want is support for the really fancy hardware like webcams, radio and TV adapters, and digital video recorders. In all three cases, Linux's support has been improved with the 2.6 release. While Linux has supported (to a greater or lesser extent) radio cards (often through userspace) for many iterations, support for television tuners and video cameras was only added within the last one or two major revisions. That subsystem, known as Video4Linux (V4L), has received a major upgrade during the work on the new edition of the kernel including both an API cleanup and support for more functionality on the cards. The new API is not compatible with the previous one and applications supporting it will need to upgrade with the kernel. And on a completely new track, Linux 2.6 includes the first built-in support for Digital Video Broadcasting (DVB) hardware. This type of hardware, common in set-top boxes, can be used to make a Linux server into a Tivo-like device, with the appropriate software.

Software Improvements

Networking

Leading-edge networking infrastructure has always been one of Linux's prime assets. Linux as an OS already supports most of the world's dominant network protocols including TCP/IP (v4 and v6), AppleTalk, IPX, and others. (The only unsupported one that comes to mind is IBM/Microsoft's obsolete and tangled NetBEUI protocol.) Like many of the changes in the other subsystems, most networking hardware changes with Linux 2.6 are under the hood and not immediately obvious. This includes low-level changes to take advantage of the device model and updates to many of the device drivers. For example, Linux now includes a separate MII (Media Independent Interface, or IEEE 802.3u) subsystem which is used by a number of the network device drivers. This new subsystem replaces many instances where each driver was handling that device's MII support in slightly different ways and with duplicated code and effort. Other changes include major ISDN updates and other things.

On the software side, one of the most major changes is Linux's new support for the IPsec protocols. IPsec, or IP Security, is a collection of protocols for IPv4 ("normal" IP) and IPv6 that allow for cryptographic security at the network protocol level. And since the security is at the protocol level, applications do not have to be explicitly aware of it. This is similar to SSL and other tunneling/security protocols, but at a much lower level. Currently supported in-kernel encryption includes various flavors of SHA ("secure hash algorithm"), DES ("data encryption standard"), and others.

Elsewhere on the protocol side, Linux has improved its support for multicast networking. Multicast networks are networks where a single sent packet is intended to be received by multiple computers. (Compare to traditional point-to-point

networks where you are only speaking to one at a time.) Primarily, this functionality is used by messaging systems (such as Tibco) and audio/video conferencing software. Linux 2.6 improves on this by now supporting several new SSM (Source Specific Multicast) protocols, including MLDv2 (Multicast Listener Discovery) and IGMPv3 (Internet Group Messaging Protocol.) These are standard protocols that are supported by most high-end networking hardware vendors, such as Cisco.

Linux 2.6 also has broken out a separate LLC stack. LLC, or Logical Link Control protocol (IEEE 802.2), is a low-level protocol that is used beneath several common higher-level network protocols such as Microsoft's NetBeui, IPX, and AppleTalk. As part of this change-over, the IPX, AppleTalk, and Token Ring drivers have been rewritten to take advantage of the new common subsystem. In addition, an outside source has put together a working NetBEUI stack and it remains to be seen whether it will ever be integrated into the stock kernel.

In addition to these changes, there have been a number of smaller changes. IPv6 has received some major changes and it can now also run on Token Ring networks. Linux's NAT/masquerading support has been extended to better handle protocols that require multiple connections (H.323, PPTP, etc.) On the Linux-as-a-router front, support for configuring VLANs on Linux has been made no longer "experimental".

Network Filesystems

Overlaid on top of Linux's robust support for network protocols is Linux's equally robust support for network filesystems. Mounting (and sometimes exporting) a network filesystem is one of the very few high-level network operations that the kernel cares about directly. (The most obvious other, the "network block device", did not receive many changes for 2.6 and is generally used in specialized applications where you end up doing something filesystem-like with it anyway.) All other network operations are content to be relegated to user-space and outside the domain of the kernel developers.

In the Linux and Unix-clone world, the most common of the network filesystems is the aptly named Network File System, or NFS. NFS is a complicated file sharing protocol that has deep roots in Unix (and especially Sun Solaris' excellent implementation). The primary transport protocol can utilize either TCP or UDP, but several additional sub-protocols are also required, each of which also run on top of the separate RPC ("remote procedure call") protocol. These include the separate "mount" protocol for authentication and NLM ("network lock manager") for file locking. (The common implementation is also tied closely to other common RPC-based protocols, including NIS-- "network information service"-- for authentication. NIS and its progeny are not commonly used for authentication on Linux machines due to fundamental insecurities.) It is perhaps because of this complexity that NFS has not been widely adapted as an "Internet" protocol.

In Linux 2.6, this core Linux filesystem received many updated and improvements.

The largest of these improvements is that Linux now experimentally supports the new and not widely adopted NFSv4 protocol version for both its client and server implementations. (Previous versions of Linux included support for only the v2 and v3 versions of the protocol.) The new version supports stronger and more secure authentication (with cryptography), more intelligent locking, support for pseudo-filesystems, and other changes. Not all of the new NFSv4 features have been implemented in Linux yet, but the support is relatively stable and could be used for some production applications. In addition, Linux's NFS server implementation has been improved to be more scalable (up to 64 times as many concurrent users and a larger request queues), to be more complete (by supporting serving over TCP, in addition to UDP), to be more robust (individual filesystems drivers can adapt the way files on those systems are exported to suit their particularities), and more easily maintainable (management though a new 'nfsd' filesystem, instead of system calls.) There have also been may other under the hood changes, including separating lockd and nfsd, and support for zero-copy networking on supported interfaces. NFS has also been made somewhat easier to secure by allowing the kernel lockd port numbers to be assigned by the administrator. The NFS client side has also benefited from a number of improvements to the implementation of the underlying RPC protocol including a caching infrastructure, connection control over UDP, and other improvements for TCP. Linux's support for using NFS-shared volumes as the root filesystem (for disk-less systems) has also been improved as the kernel now supports NFS over TCP for that purpose.

In addition to improving support for the Unix-style network filesystems, Linux 2.6 also delivers many improvements to Windows-style network filesystems. The standard shared filesystem for Windows servers (as well as OS/2 and other operating systems) has been the SMB ("server message block") protocol and the Linux kernel has had excellent client support of the SMB protocol for many revisions. Windows 2000 however standardized on an upgraded superset of the SMB protocol, known as CIFS ("common Internet filesystem.") The intention of this major update was to streamline and refine certain aspects of SMB which had at that point become a complete mess. (The protocol itself was loosely defined and often extended to the point that there were cases even where the Win95/98/ME version was incompatible with the WinNT/Win2k version.) CIFS delivered on that intention and added UNICODE support, improved file locking, hard linking, eliminated the last vestiges of NetBIOS dependencies, and added a few other features for Windows users. Since Linux users do not like to be kept in the dark for long, Linux 2.6 now includes completely rewritten support for mounting CIFS filesystems natively. Linux 2.6 also now includes support for the SMB-Unix extensions to the SMB and CIFS protocols which allows Linux to access non-Windows file types (such as device nodes and symbolic links) on SMB servers which support it (such as Samba.) Although not as commonly seen today, Linux has not completely forgotten about the Novell NetWare users. Linux 2.6 now allows Linux clients to mount up to the maximum of 256 shares on a single NetWare volume using its built in NCP ("NetWare Core Protocol") filesystem driver.

Linux 2.6 also includes improved support for the relatively new domain of

distributed network filesystems, systems where files on a single logical volume can be scattered across multiple nodes. In addition to the CODA filesystem introduced in Linux 2.4, Linux now includes some support for two other distributed filesystems: AFS and InterMezzo. AFS, the Andrew filesystem (so named because it was originally developed at CMU), is presently very limited and restricted to read-only operations. (A more feature complete version of AFS is available outside the kernel-proper.) The second newly supported filesystem, InterMezzo (also developed at CMU), is also newly supported under Linux 2.6 and it allows for more advanced features such as disconnect operation (so you work on locally cached files) and is suitable for high-availability applications where you need to guarantee that storage is never unavailable (or faked, when down). It also has applications for keeping data in sync between multiple computers, such as a laptop or PDA and a desktop computer. Many of the projects providing support for these new types of filesystems are initially developed on Linux, putting Linux well ahead of the curve in support for these new features.

Miscellaneous Features

Security

Another of the big changes in Linux 2.6 that does not receive enough attention is the wealth of new security-related changes. Most fundamentally, the entirety of kernel-based security (powers of the super user under a Unix-like operating system) has been modularized out to be one out of a potential number of alternate security modules. (At present however, the only offered security model is the default one and an example how to make your own.) As part of this change, all parts of the kernel have now been updated to use "capabilities" as the basis of fine-grained user access, rather than the old "superuser" system. Nearly all Linux systems will continue to have a "root" account which has complete access, but this allows for a Linux-like system to be created which does not have this underlying assumption. Another security-related change is that binary modules (for example, drivers shipped by a hardware manufacturer) can no longer "overload" system calls with their own and can no longer see and modify the system call table. This significantly restricts the amount of access that non-open source modules can do in the kernel and possibly closes some legal loopholes around the GPL. The final change that is somewhat security-related is that Linux with the new kernel is now able to use hardware random number generators (such as those present in some new processors), rather than relying on a (admittedly quite good) entropy pool based on random hardware fluctuations. Virtualizing Linux

One of the most interesting new features in Linux 2.6 is its inclusion of a "user-mode" architecture. This is essentially a port (like to a different hardware family) of Linux to itself, allowing for a completely virtualized Linux-on-Linux environment to be run. The new instance of Linux runs as if it was a normal application. "Inside" the application, you can configure fake network interfaces,

filesystems, and other devices through special drivers which communicate up to the host copy of Linux in a secure way. This has proved quite useful, both for development purposes (profiling, etc.) as well as for security analysis and honeypots. While most users will never need this kind of support, it is an incredibly "cool" feature to have running on your box. (Impress your friends!)

Laptops

In addition to all of the other general purpose support described above (improved APM and ACPI, wireless support improvements, etc.) Linux also includes two other hard-to-classify features that will best assist laptop users. The first is that the new edition of the kernel now supports software-suspend-to-disk functionality for the Linux user on the go. This feature still has some bugs to iron out, but is looking solid for many configurations. The new version also supports the ability of modern mobile processors to change speed (and, in effect, power requirements) based on whether your system is plugged in or not.

Configuration Management

Linux 2.6 includes another feature which might seem minor to some, but will both greatly assist developers' abilities to debug kernel problems of end-users as well as make it easier for individual administrators to know configuration details about multiple systems. In short, the kernel now supports adding full configuration information into the kernel file itself. This information would include details such as what configuration options were selected, what compiler was used, and other details which would help someone reproduce a similar kernel if the need arose. This information would also be exposed to users via the /proc interface.

Legacy Support

Although Linux 2.6 is a major upgrade, the difference to user-mode applications will be nearly non-existent. The one major exception to this rule appears to be threading: some applications may do things that worked under 2.4 or 2.2 but are no longer allowed. Those applications should be the exception to the rule however. Of course, low-level applications such as module utilities will definitely not work. Additionally, some of the files and formats in the /proc and /dev directories have changed and any applications that have dependencies on this may not function correctly. (This is especially true as more things shift over to the new "/sys" virtual filesystem. In the "/dev" case, backwards-compatible device names can easily be configured.)

In addition to those standard disclaimers, there are a number of other smaller changes which may affect some environments. First, very old swap files (from Linux 2.0 or earlier) will need to be reformatted before they can be used with 2.6. (Since swap files do not contain any permanent data, that should not be a problem for any user.) The kHTTPd daemon which allowed the kernel to serve web pages directly

has also been removed as most of the performance bottlenecks that prevented Apache, Zeus, et. al. from reaching kernel speeds have been resolved. Autodetection of DOS/Windows "disk managers" such as OnTrack and EzDrive for large harddisk support with older BIOSes has been removed. And finally, support for using a special kernel-included boot sector for booting off of a floppy disk has also been removed; you now need to use SysLinux instead.

Stuff At The Bottom

This document was assembled primarily from long hours looking at BitKeeper changelogs, looking at and playing with the source, reading mailing list posts, and lots and lots of Google and Lycos searches for documentation about this and that. As such, there are likely places where something could have been missed or misunderstood, and places where something could have been backed out after the fact. (I have been especially careful of the two versions of IDE support that were worked on during this time period, but there are other examples.) As a bit of my research was done by looking at the web pages of various kernel projects, I have had to be careful that the independent projects weren't farther ahead with features than were accepted into the mainline Linux code. If you see any errors in this document or want to email me to ask me how my day is going, you can do so at jpranevich <at> kniggit.net.

The newest version of this document can always be found at http://kniggit.net/wwol26.html

Translations

Not an English speaker? This document (or an older revision) has been translated into a handful of other languages.

Bulgarian - http://kniggit.net/wwol26bg.html (Ivan Dimov)

Chinese -http://www-900.ibm.com/developerWorks/cn/Linux/kernel/l-kernel26/index.shtml (Stone Wang, et. al.)

Czech - http://www.Linuxzone.cz/index.phtml?ids=10&idc=782 (David Haring)

French - http://dsoulayrol.free.fr/articles/wonderful_2.6.html (David Soulayrol)

Hungarian - http://free.srv.hu/b/e/behun/pn/modules.php?op=modload&name=News&file=index&catid=&top (Ervin Novak) (Not yet completed.)

Italian - http://www.opensp.org/tutorial/vedi.php?appartenenza=42&pagine=1 (Giulio Ciuffi Vampa)

Portuguese (BR) - http://geocities.yahoo.com.br/cesarakg/wwol26-ptBR.html (Cesar A. K. Grossmann)

Russian - http://www.opennet.ru/base/sys/Linux26_intro.txt.html (Sergey Prokopenko)

Spanish - http://www.escomposLinux.org/wwol26/wwol26.html (Alex Fernandez)

An abridged version also appeared in German in the 09/2003 issue of LanLine magazine. I believe that an unabridged edition may be floating around also, but I am uncertain of the link. If you know of any additional translations to add to this list, please let me know.

Index

U

W