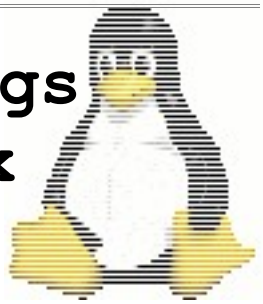
How to do Things with GNU/Linux



by Ed Holden
version 0.9.4 (rev. 20050919)

This guide is a general summary of using GNU/Linux systems, commonly called "Linux" after the popular kernel, and contains tutorials for setting up specific types of services. It is based upon the author's experiences using Red Hat Linux, Fedora Core and Debian-based distributions, but large portions of the guide are applicable to any GNU/Linux system, and some portions can be helpful with other UNIX or UNIX-like operating systems.

How to do Things with GNU/Linux

Table of Contents

0. Creative Commons License

This guide is licensed under a Creative Commons License. This means that the author retains most of the rights usually reserved under copyright, but under the terms of the license some additional rights are granted to the licensee.

The license is called the Attribution-NonCommercial license. It is available on the web here:

http://www.creativecommons.org/licenses/by-nc/1.0/

The upshot of this particular license is that you can copy, distribute, display, and perform this work as much as you'd like. In return, however, you must give the original author credit, and you may not use the work for commercial purposes unless you get the author's permission.

Attribution-NonCommercial 1.0

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

1. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
2. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License.
3. "Licensor" means the individual or entity that offers the Work under the terms of this License.
4. "Original Author" means the individual or entity who created the Work.
5. "Work" means the copyrightable work of authorship offered under the terms of this License.
6. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

1. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
2. to create and reproduce Derivative Works;
3. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;
4. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

1. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.
2. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
3. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

1. By offering the Work for public release under this License, Licensor represents and warrants that, to the best of Licensor's knowledge after reasonable inquiry:
1. Licensor has secured all rights in the Work necessary to grant the license rights hereunder and to permit the lawful exercise of the rights granted hereunder without You having any obligation to pay any royalties,

compulsory license fees, residuals or any other payments;

2. The Work does not infringe the copyright, trademark, publicity rights, common law rights or any other right of any third party or constitute defamation, invasion of privacy or other tortious injury to any third party.

2. EXCEPT AS EXPRESSLY STATED IN THIS LICENSE OR OTHERWISE AGREED IN WRITING OR REQUIRED BY APPLICABLE LAW, THE WORK IS LICENSED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES REGARDING THE CONTENTS OR ACCURACY OF THE WORK.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, AND EXCEPT FOR DAMAGES ARISING FROM LIABILITY TO A THIRD PARTY RESULTING FROM BREACH OF THE WARRANTIES IN SECTION 5, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

1. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

2. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

1. Introduction

GNU/Linux is a the fastest growing operating system in the world. It is commonly called "Linux" after the kernel that runs the system, but in fact the system is more than just a kernel. GNU/Linux is a vast collection of components independently developed by programmers all over the world; some work for companies that benefit from the open nature of the system, while others work independently.

The components of the system are a mixture of two types of software: *free software* and *open source software*. Free and open source software have much in common, so developers and advocates of each tend to cooperate. Both types require that the end user be allowed to access the source code that was used to create the software. This is vastly different from *proprietary software*, which is only distributed in a precompiled, binary format, or under a restrictive license.

No one owns free and open source software - the end user is just as much an owner of the software as the original developer, and the developer is often paid for the hours they code rather than for the number of copies that are later sold. The nice thing about this model is that when the author gets tired of his or her software another developer can adopt it and continue to improve it.

This guide focuses on the use of GNU/Linux for many common tasks. More specifically, it focuses on a couple common distributions of GNU/Linux on the x86 architecture called Fedora Core and Ubuntu. While these are popular distributions and x86 is the world's most prevalent hardware platform, there are many other distributions available and the OS has been ported to many other platforms.

The Audience of this Guide

The target audience of this guide is the vast hoard of geeks who know very little about GNU/Linux systems, or who are familiar with them but would like information on how to accomplish specific tasks. If you are interested in learning how to use a graphical user interface on a UNIX-like operating system, or how to use a free accounting program, or how to dial up to the Internet, this guide is probably not for you. It deals with lower-level workings and configuration of GNU/Linux systems, and is therefore of more interest to hobbyists and administrators.

The base system used to shape this guide is Fedora Core. This is a system based upon the popular Red Hat Linux, and is in fact sponsored by Red Hat, Inc. In 2003 Red Hat chose to discontinue their free version of the distribution and instead offer a pay version available only to subscribers. While this version was of high quality and came with support services, it opened a niche for a free hobbyist version of the system.

Enter Fedora. Fedora Core 1 was released in November of 2003 and features many improvements over its predecessor, Red Hat Linux 9. Fedora Core 2 introduced the new 2.6 series Linux kernel, updated desktop environments, SELinux for enhanced security ... and many growing pains. Both versions can be downloaded for free, along with updates, at *http://fedora.redhat.com*.

While Fedora is the basis of this guide, its usefulness is not limited to a single distribution: most tutorials can be accomplished similarly or even identically on other distributions.

The Benefits of Learning GNU/Linux

Despite the emphasis on geeks in the audience of this guide, there may be something for non-geeks to gain from this and other guides. GNU/Linux is a powerful system, and the time you invest in learning its intricacies can be a real benefit to you later on. As Mark Stone points out in his article "Real Desktop Linux,"

> Getting the benefit of Linux means embracing the idea that powerful capabilities come from an investment in *actually learning* the system [emphasis added]. Accepting and embracing this idea is, in fact, one of the biggest obstacles to Linux use in the enterprise because the Linux desktop, no matter how simplified, will always have greater power but also a substantial learning curve.

So while some other operating systems and software tools may be easy to master, the ones you'll find detailed in this guide are extremely powerful and customizable. Learning to edit with the vi text editor and string commands together with pipes and redirects may seem time-consuming, but in the long run you can save quite a lot of effort with mastery of a few common, open tools.

The Origins and Development of GNU/Linux

A complete GNU/Linux system has three major sources: UNIX, GNU and Linux.

Ken Thompson and Dennis Ritchie created UNIX at AT&T's Bell Laboratories, where they needed to implement a replacement for the older MULTICS system. UNIX was a multi-user, multitasking operating environment that could accomplish a number of simultaneous tasks requested by separate concurrent users. They later rewrote it entirely in C Language, allowing it to be ported to other platforms.

The programs that made up UNIX were developed under a guideline known as KISS, or "keep it simple, stupid." Each program was designed to accomplish a single task extremely well, and developers who needed to implement new features were encouraged to avoid building bloat into older software, and instead build new programs. The output of each program was made to be simple enough that it could be used as the input of another, so complex tasks were performed with many simple programs in conjunction, rather than with a single complex program. By these guidelines UNIX became a very modular operating system, and this trait was continued by later developers.

The second source of GNU/Linux is the Free Software Foundation's GNU Project, started by Richard M. Stallman in Cambridge, Massachusetts. Stallman, a former member of the MIT Artificial Intelligence Labs, had noticed a shift in software development during the 1970s and 1980s toward the use of proprietary software. Prior to this shift, disparate groups of developers freely shared software and helped each other to improve it by keeping the software's source code available to all. The activity was, as he phrases it, "as old as computers, just as sharing of recipes is as old as cooking." To Stallman, sharing software was not just polite, but also part of the process of learning how to develop software, and a prerequisite for ensuring that the software could be fixed and improved beyond the life (or attention span) of the original programmer.

With proprietary software this dynamic had been destroyed. "The rule made by the owners of proprietary software was, 'If you share with your neighbor, you are a pirate. If you want any changes, beg us to make them.'" Stallman experienced this firsthand when Xerox donated a new laser printer to the AI Labs. He and other lab members experienced problems with the printer's software, which didn't accurately recognize paper jams. When he contacted the author of the printer driver he was told that he couldn't have access to the source code. To fix the printer, he had to wait until the author decided to make an update.

Stallman started the GNU Project in 1984 with the aim of creating an entire operating system out of what he called *free software*. This software was to be free in the sense of "freedom," not necessarily free in terms of its cost. He started work on his GNU system (the name is a self-reflexive acronym for "GNU's Not UNIX") with many useful utilities, an early one being GNU C Compiler, or GCC. Because of this early focus the first people to reap the benefits of GNU were proprietary UNIX users, because GNU was modeled on UNIX and therefore UNIX was the testing ground for its components. All of this software was released under the terms of a license called the GNU General Public License, or GPL. Unlike a proprietary software license, the GPL is very easy to read and understand, and clearly outlines the freedoms end users are granted. It's worth a read, and if it's hard to find in your distribution you can always locate it at the Free Software Foundation's web page at *http://www.fsf.org/licenses/gpl.html*.

By the early 1990s the system lacked a core component: the GNU kernel. The planned kernel, known as the Hurd, was incomplete and would not be usable for over a decade, but without it the system was not truly free.

Enter Linux. The Linux kernel was written by Linus Torvalds, a student at the University of Helsinki, in 1991. His original goal for the kernel was modest: it was, in his words, "just a hobby." Linus had been dissatisfied with a UNIX-like system called Minix, written by Dutch professor Andrew Tanenbaum,

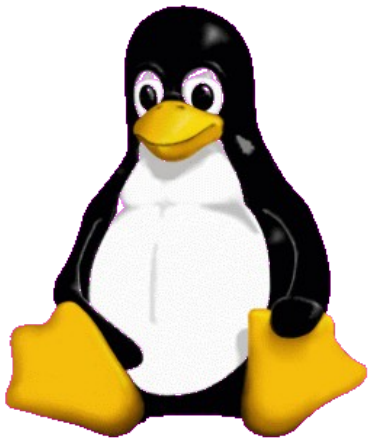
because he couldn't tweak it to his needs and Tanenbaum wasn't interested in developing it. So Linus decided to write his own system, which he named Linux. The online Minix user group was the first audience for the new kernel, and soon its popularity outgrew that of Minix. People started putting together GNU/Linux systems - that is, GNU systems running the Linux kernel - almost immediately, and the trend increased in popularity throughout the 1990s.

Many companies have arisen as distributors of GNU/Linux. Red Hat (which includes Fedora), Mandrake and SuSE are probably the most well-known distributions. The system gained increased credibility when it was embraced by IBM, which is now one of the most active corporate developers of the system. While it has mainly functioned as as an operating system for servers, a movement is afoot to bring the system to Linus Torvalds' intended arena: the desktop. In either scenario GNU/Linux is proving itself to be a stable, flexible and reliable system.

What's in a Name?

As mentioned earlier, most people call the operating system Linux, after the kernel. This is a perfectly good name for an operating system, but in recent years the Free Software Foundation has pushed for the use of the name GNU/Linux to describe the software. Their logic is that the system was developed to be GNU, and the Linux kernel is just one component of the overall system. Some have accused the Foundation of egotism, and admittedly the name Linux has more of a ring to it.

But there is another reason to use this term: clarity. In a 2003 lawsuit, the SCO Group accused IBM of adding proprietary UNIX code into Linux. This led to much discussion about the the alleged infringement, and many Linux kernel developers resolutely defended their code. But the early discussions were nonsensical because all parties, including IBM and SCO, used the word *Linux* to describe more than one thing. Discussions about the kernel and its surrounding components are far easier to understand when the kernel and operating system are distinguished, so it is primarily for clarity that I refer to the system as GNU/Linux and the kernel alone as Linux. I also refer to the Red Hat distribution as Red Hat Linux, as this is the product's name.



The chosen bird: Tux the Penguin

The Bird

Regarding the penguin: Tux is the symbol of Linux, chosen by Linus early in the kernel's development. Since people typically use the word Linux to describe more than one thing, Tux has generally come to embody the operating system as a whole.

The most famous version of the penguin was drawn by Larry Ewing using The GIMP image editor, and samples of his work are available at *http://www.isc.tamu.edu/~lewing/linux/*. No one is entirely clear on why Linus chose a penguin. He has offered many contradictory explanations for it, but the most enthusiastic one is the one in which he describes penguins as violently powerful creatures.

> Some people have told me they don't think a fat penguin really embodies the grace of Linux, which just tells me they have never seen a angry penguin charging at them in excess of 100mph. They'd be a lot more careful about what they say if they had.

'Nuff said.

2. Technical Specifications and Distribution Channels

Technical Capabilities

GNU/Linux is the world's most versatile operating system, mainly because its open nature has allowed many different parties to port it to other platforms. It is the only system that will power both a handheld computer and a large mainframe. It supports Symmetric Multiprocessing (SMP), is scalable to up to 32 CPUs (however no more than eight are recommended in the x86 architecture) and supports up to 1 GB of memory by default. The i686 version of the kernel can support up to 4 GB of RAM, and the "bigmem" kernel can do 64 GB - far more than the average user needs at the time of writing!

While the x86 platform is the most prevalent platform in the world, and the one on which GNU/Linux has the strongest foothold, it has also been ported to the IA-64 (64-bit Intel Itanium), the Digital/Compaq Alpha, the Sun SPARC, the IBM s/390, the PowerPC and the Motorola 68k. At the time of writing Red Hat has released a new version for the AMD x86-64, the 64-bit chip that could be the future of desktop computing. There is no limitation to the platforms the operating system can support, since the source code is available for anyone who wants to port it.

System Requirements

GNU/Linux can, depending upon the packages installed, use very little of a system's resources, and is therefore ideal for older computers that refuse to run bulkier, proprietary operating systems. (ToDo: specifics on hardware)

Distributions

GNU/Linux began its life as a very do-it-yourself platform. Early users had to download the source code of its various components and piece them together into a workable whole. This soon gave way to the emergence of *distributions*. A distribution is a pre-configured and generally pre-compiled collection of software that softens the DIY nature of the OS, making it easy for the non-guru to install and use.

There are many distributions of GNU/Linux available, and all of them share common features like the X Window system and popular libraries and utilities, but each differs somewhat in terms of the applications, configuration, program management and other features it presents. Some popular distributions are listed below:

- **Debian GNU/Linux** (*http://www.debian.org*) - Released in August of 1993 bu Ian Murdock, Debian was the first distribution endorsed by Richard Stallman and the Free Software Foundation. This popular distribution is used as a standard by HP and is a basis for many other distributions.
- **Slackware Linux** (*http://www.slackware.org*) - First released in April of 1993, Slackware is very popular amongst software hackers and is sometimes regarded as the most UNIX-like distribution.
- **Red Hat Enterprise Linux** (*http://www.redhat.com*) - The world's most popular Linux distribution. The company has forged many business alliances with hardware vendors, ensuring that they will remain a major player in the GNU/Linux marketplace for some time.
- **Fedora Core** (*http://fedora.redhat.com*) - The base system for Red Hat Enterprise Linux, this a free and unsupported version is released two to three times annually. It is aimed at hobbyists and has the same basic features as the Enterprise offering.
- **Linux Mandrake** (*http://www.mandrake.com*) - Based upon Red Hat's distribution, it boasts a slightly better user interface and installer, and is generally regarded as the best distribution for newbies.
- **SuSE Linux** (*http://www.suse.com*) - A very popular distribution in Europe, the German SuSE was purhased by Novell in 2003 and is regarded as an easy-to-use distribution, largely thanks to its YaST configuration system.
- **Gentoo Linux** (*http://www.gentoo.com*) - Gentoo features software packages that compile during installation, making it the only distribution that is customized for each machine on which it is run. The installation CD is just a set of basic packages, and the rest are downloaded for compilation during

the install process. Definitely the first of its kind.

- **Yellow Dog Linux** (*http://www.turbolinux.com*) - Based on Red Hat Linux, this distribution has been compiled and customized for the PowerPC processor architecture, and will work on most Macintosh computers.

Other Sources of Support

Because it is built through a community effort there are many sources of support for GNU/Linux online. The focal point for this support is the Linux Documentation Project (*http://www.tldp.org*). This site contains many How-To documents that can guide you through a variety of tasks.

Information on configuring the OS for use on a laptop is available at the Linux Laptop Pages (*http://www.linux-laptop.net*).

3. Basics of the Shell

A shell is an interface that translates human-readable commands into binary input. Users accustomed to graphical interfaces may view the text-based shell as an anachronism, however the UNIX shell is far more powerful than the old MS-DOS prompt with which many users are more familiar. Indeed, UNIX administrators favor the shell for its power and ease of remote management.

Like all UNIX components the shell is modular, so you can choose from any of a number of them. The original UNIX shell was the Bourne shell, which was located at `/bin/sh`. Because this was the only shell available for many years it became a necessary component of most systems, so in GNU/Linux distributions this file is still present as a symbolic link that points to the Bourne Again shell, or "Bash," located at `/bin/bash`. Bash was developed by the Free Software Foundation and is the default shell on most GNU/Linux distributions. It is fully backward-compatible with the Bourne shell.

Other popular shells are the C shell at `/bin/csh` (its more advanced successor is at `/bin/tcsh`). The C Shell was designed to emulate the structure of the C programming language, and `/bin/tcsh` is an enhanced version with command-line completion and filename editing. C is the default shell on most BSD-UNIX systems, including Mac OS X. The Korn shell at `/bin/ksh` includes features of both the Bash and C shells.

This guide focuses on Bash features because of the popularity of Bash among GNU/Linux users, though some features may apply to other shells.

About the Prompt and the Superuser

When you first log into a GNU/Linux system under the Bash shell you'll encounter a prompt, like the following one:

```
[wsmith@localhost /home/wsmith]$
```

There are four parts to this prompt: the first is your user name, the second is the computer's host name (which on a default configuration is simply localhost), the third is the present working directory, and the fourth is the "dollar sign" character, `$`. This character means that you are logged on as a normal user.

At times it may be necessary to log onto a system as the superuser, who is named *root* on most UNIX and UNIX-like operating systems. To change to the superuser, use the `su` (switch user) command, followed by a dash. The system will ask you for the superuser password before letting you log in.

```
[wsmith@localhost /home/wsmith]$ su -
Password:
[root@localhost /root]#
```

The only difference between the superuser prompt and that of a normal user is that the final character is a hash symbol, or `#`. In this guide sample commands will be shown after a `$` if they are carried out by a normal user and after a `#` if they are being run by the superuser.

Warning about the superuser

It is not recommended that you work as the superuser unless you are performing administrative tasks. The superuser has absolute power on a machine, so it's easy to cause accidental damage to your system.

Common Commands You Should Know

Most experienced users of GNU/Linux and other UNIX-like systems use the shell all the time. It is full-featured, stable and very. Very powerful. If you'd like to start experimenting with that power there are a few commands you should know, and we'll outline them here. This is by no means a comprehensive list of commands, as that is not the intention of this guide. But this should provide a good introduction to common UNIX commands, or a reference for users who are already familiar with them.

- **ls**

The most basic command is `ls`, which simply lists the files in the current directory. By default it lists filenames only, and it arranges them into multiple columns. No information about the files is displayed.

But ls is chock full of options, and this is probably a more informational use of the command:

```
$ ls -la
total 2601364
drwx------    4 eholden  eholden      4096 Oct  8 18:16 .
drwx------   34 eholden  eholden      4096 Oct  9 11:40 ..
-rw-rw-r--    1 eholden  eholden       389 Sep 19 16:01 auto.txt
-rw-r--r--    1 eholden  eholden       221 Sep 13  2002 CD%2fDVD-ROM
-rw-r--r--    1 eholden  eholden       831 Oct  8 18:16 .directory
-rw-r--r--    1 eholden  eholden       353 Aug 14 13:19 FTP Site
-rw-r--r--    1 eholden  eholden       151 Sep 13  2002 floppy
-rw-r--r--    1 eholden  eholden       232 Aug  5 10:50 Home
-rw-------    1 eholden  eholden  726974464 Jul 31 22:35 KNOPPIX_V3.2-2003-07-25-EN.iso
-rw-rw-r--    1 eholden  eholden    396202 Sep 10 18:08 mclean.ldif
-rw-------    1 eholden  eholden    280308 Oct  9 11:39 Notes - GNU-Linux
Administration.sxw
-rw-------    1 eholden  eholden  78347712 Oct  3 17:08
OOo_1.1.0_LinuxIntel_install.tar.gz
-rw-rw-r--    1 eholden  eholden      6384 Sep 16 14:19 rhce.sxw
-rw-r--r--    1 eholden  eholden       235 Jul 31 09:28 Shared
-rwxr-xr-x    1 eholden  eholden  668991488 Apr  1  2003 shrike-i386-disc1.iso
-rwxr-xr-x    1 eholden  eholden  677511168 Apr  1  2003 shrike-i386-disc2.iso
-rwxr-xr-x    1 eholden  eholden  508592128 Apr  1  2003 shrike-i386-disc3.iso
-rw-r--r--    1 eholden  eholden      3189 Aug  2 07:42 SysRq Key.txt
-rw-r--r--    1 eholden  eholden        88 Aug  4 14:23 Tasks.desktop
drwx------    5 eholden  eholden      4096 Oct  8 10:11 Trash
drwxr-xr-x    2 eholden  eholden      4096 Sep 25 11:07 .xvpics
```

The `-l` option lists files in "long" format, which shows you data like the UNIX file permissions, ownership, the size in bytes and the modification date and time. If the permissions start with a `d` then the file is actually a directory. The `-a` option lists all files; without it hidden files will not be listed. Hidden files start with a dot, so in this case we get to see the `.xvpics` and `.directory` files whereas they would normally be invisible.

- **pwd**

The pwd command stands for *present working* directory. It will echoes the directory in which you are currently working.

```
$ pwd
/home/wsmith/documents
```

- **cd**

Change directory - use this to change your present working directory. You can use an absolute path:

```
$ cd /home/wsmith/documents
```

Or if you're in an ideal place for it, change to a subdirectory:

```
$ pwd
/home/wsmith
$ cd documents
```

Or you can use `..` to change to a higher-level directory.

```
$ cd ..
```

Attach this to another directory name and you can go up to a parent directory then down from there into a subdirectory:

```
$ pwd
/home/wsmith/documents
$ cd ../moredocuments
$ pwd
/home/wsmith/moredocuments
```

- **cp**

Copy a file. For example, to back up the `passwd` file:

```
$ cp /etc/passwd passwd.backup
```

If you'd like to copy an entire directory, us the `-a` option for a recursive "archive" copy:

```
$ cp -a /etc /root/backups/
```

This particular option is not available in some UNIX-like systems, where you have to use `-r` and other options.

- **mv**

Move a file. For example, to take a file from the present working directory and move it down a couple directories:

```
$ mv document.txt  documents/business/
```

This command can also be used to rename files. In fact, UNIX-like systems have no "rename" command to speak of. Just move a file to rename it:

```
$ mv document.txt new-name.txt
```

- **mkdir**

Make a new directory.

- **rmdir**

Remove a directory

- **rm**

Remove a file. This is the most dangerous of the basic UNIX commands.

find
touch
cat - List the contents of a file (stands for "concatenate")
grep - Filer contents or results

sed - Find and replace. Stands for "stream editor," since it's designed to filter and alter input.
sed -e 's;/dev/null:/etc/vacation-only;/dev/null:/bin/false;g' passwd-20031009a > passwd-20031009b

(ToDo: any more basic commands and how to use them should be added, and the above should be completed.)

Wildcards

Sometimes you'll need to run a command on more than one file at a time, whether it is on a whole group of files or specific files in a given group. Wildcards allow you to tell the shell which files you want without listing them all.

The Splat Wildcard (*)

The * character - often called an asterisk, a star or a splat - is the most widely used wildcard. A * applies to all possible files. Imagine you have a directory filled with images and you want to delete them all. You'd use this command:

```
$ rm *
```

This is useful, but what if we only wanted to delete the JPEG images in the directory, and leave all the other files intact? Adding the filename extension to the splat fixes this:

```
$ rm *.jpg
```

Of course, you don't need to add the entire extension to the splat. If all of the other filenames are PNG images that end in `.png` you can type even less. This command deletes any file that end with the letter `g`:

```
$ rm *g
```

Of course the splat can represent any number of characters, so you could also delete all GIF images starting with the word titlebar like this

```
$ rm titlebar*.png
```

The Question Mark Wildcard (?)

The question mark represents only one character, useful when you know all of a filename except for one letter. If your directory of image files contains three files named `portrait11`, `portrait12` and `portrait16`, you could get them all with `portrait1?`. You could also use a group of question marks together; the following command will delete all files with four-character file extensions:

```
$ rm *.????
```

Square Brackets (the [and] symbols)

Square brackets represents a range or a choice of characters. If you have files named `portrait11`, `portrait12`, and `portrait13`, but you also have a couple dozen others up to `portrait37`, you can delete all of them by placing a range in brackets, as so:

```
$ rm portrait[11-37]
```

To use it to indicate a choice among characters, omit the dash:

```
$ rm portrait[458].jpg
```

The above command will delete `portrait4`, `portrait5` and `portrait8`. Separate with commas and you can combine ranges with choices:

```
$ rm [a-c,X-Z]*.png
```

This will delete files that have a .gif extension and have a first letter of a, b, c, X, Y, or Z.

The Editable Command Line

Movement

If you have to make a correction to an active command line there are a number of shortcut keys to help you navigate, some of which may come in handy if your terminal doesn't properly recognize arrow keys and backspaces. The following shortcuts use the Ctrl key to edit the command line:

Ctrl-a - Move the cursor to the beginning of the line
Ctrl-e - Move the cursor to the end of the line
Ctrl-w - Erase the previous word
Ctrl-u - Erase to beginning of the line

Tab Completion

A big time-saver is tab-completion. Pressing the tab key can complete filenames in the current directory or commands in your command path, expressed by the shell variable $PATH. Lif you type enough of the filename to make it unique the tab will save you typing out the whole command. For example:

```
$ chm(Tab) dat(Tab)
```

might be expanded to the following:

```
$ chmod data.txt
```

If you have more than one file that starts with the string "dat," tab completion will not work. However you can just hit tab twice, and Bash will print a list of all matching files and then return to your partially-completed command so that you can type more of the filename.

Aliases

You can use aliases to create your own custom shortcut commands. Some are already present for you in a typical GNU/Linux system, like in your home directory's .bashrc, .bash_profile, .profile and .aliases files. If you wanted to use the long listing format of the `ls` command by default you could add the following line to your .profile file:

```
alias ls='ls -l'
```

Now if you type the ls command bash will convert it to `ls -l` before executing it. This is especially useful for long, frequently-used commands.

Backticks

The backtick is one of the best kept secrets of the Bash shell, and is useful in scripting. It allows you to run a sub-command and use its output in-situ.

For example, imagine we have a file containing the names of three files. We'll catenate it to see its contents:

```
$ cat filenames.txt
name.txt
address.txt
telephone.txt
other-telephone.txt
```

Say we want to find a telephone number: we can see from the above output that our desired data is in either `telephone.txt` or `other-telephone.txt`. We could then find the number by viewing the content of both files. But we could skip a couple steps by having Bash catenate any file listed in `filenames.txt` that contains the word telephone.

```
$ cat `grep -l telephone filenames.txt`
(781) 555-1234
(781) 555-9865
(781) 555-2309
(617) 555-3866
```

In this example, we encapsulated the `grep` command in backticks, and its output was used in place of the command itself. So Bash filtered the main file and used both filenames containing the word telephone as source files for the `cat` command.

Pipes and Redirects

(ToDo: a section on pipes and redirects should go here.)

The Bash History and Associated Shortcuts

One of the nicest features of the Bash shell is its editable command line. Older shells did not allow the user to press the arrow keys to fix commands before executing them. But editing commands goes beyond using the left and right keys. Pressing the up and down keys allows the user to browse the history, which is a set of cached commands that make it easy to access frequently-used or hard-to-remember command sequences. The up arrow browses from newest to oldest, and the down arrow does the opposite.

Use the `history` command to create a numbered list of the commands you've entered, with the oldest at the top. As an example of a very short output:

```
$ history
1  cd /etc/mail
2  pico access
3  makemap hash /etc/mail/access < /etc/mail/access
4  cd /home/wsmith
```

If you're looking for a specific command and you remember a part of it, you can skip the above output, which is normally much longer, by piping to grep.

```
$ history | grep makemap
3  makemap hash /etc/mail/access < /etc/mail/access
```

You can then use the `!X` command, where X is the number of the command in the history. In the above example you could enter `!3` instead of "`makemap hash` ..."

Bash has many other built-in history utilities that can significantly speed up your use of the command line. The most basic is called bang-bang. A bang-bang is a pair of exclamation marks, or `!!`. `!!` tells the shell to repeat the last command that you entered. It also prints the command to the next line, so you know what command you ran.

This use of the `!` symbol can be taken a step further with the `!abc` command, where "abc" is a string of text from the beginning of a previous command. `!abc` will allow you to run the last command beginning with that sting, as long as you add enough text to make it unique. For example:

```
$ pwd
$ cd /home/wsmith
$ vi speeder-repair.txt
$ pwd
$ ps -aux
```

The command `pwd` was executed twice, and then `ps -aux`. If you type `!p` you will get the output of `ps -aux`, because this was the last executed command that started with "p." However, typing `!pw` gives Bash enough information to find `pwd` in the history.

To avoid the danger of executing the wrong command with the bang feature, append a `:p` onto your text string. This prints the command that would be executed instead of executing it, and also adds it to your history so that you can press the up arrow or `!!` to execute that command. For example:

```
$ !p:p
ps -aux
$ !pw:p
pwd
$ !!
pwd
/home/wsmith
```

If you want to search your history for a specific command without using the `history` command itself, try typing `^r` (Ctrl-r). This will initiate a reverse search, which will look through the history for whatever string you type. If you get close to your desired command, type ^r to browse through similar commands that contain the same string. When you find the command you want press ESC, or the left or right arrow key, to edit or execute the command.

The "bang dollar" string, or `!$`, can be used to signify the last string of text in the previous command. For example, if we look for a particular word in a file, we can then easily edit that file without typing its entire filename:

```
$ grep -i lorgana /etc/passwd
lorgana:x:520:520:Leia Organa,Alderann,,,:/home/lorgana:/bin/bash
$ vi !$
```

Bash will interpret `!$` as `/etc/passwd`, and allow you to edit the file. This is especially useful for long filenames, however keep in mind that it uses an entire "word," which is any sequence of characters containing no whitespace (spaces or tabs). The :p string can be appended to !$ to print out the command rather than execute it, just in case you're not sure.

Similar to bang dollar is "bang splat," or !*, which is interpreted as all of the arguments to the previous command rather than just the last one. This is useful if you accidentally type a command at a prompt that that already has some text after it, like in the following example where the user tried to switch to the superuser with the `su` command, then changed his or her mind but forgot to delete the command before entering a new command to change directory:

```
$ su cd /home/wsmith
su: user cd does not exist
$ !*
cd /home/wsmith
```

Circumflexes are a useful tool for correcting typos in previous commands, rather than retyping from scratch. Also called "hats," circumflexes use the ^ character that is the shift of the 6 key on most keyboards. A circumflex command is a single string of text following the formula `^original^new`. For example:

```
$ vi /etc/sysconfig/Network
$ ^N^n
vi /etc/sysconfig/network
```

Circumflexes can also be used to handle redundant commands that can't be handled with wildcards, hats will work great for you. For example:

```
$ chown wsmith.wsmith file1
$ ^file1^otherdata.txt
chown wsmith.wsmith otherdata.txt
```

More info at:
http://www.linuxorbit.com/modules.php?op=modload&name=Sections&file=index&req=viewarticle&artid=461&page=1

4. Editing Text: The Infamous Vi Editor

Why Vi?

Devoted GNU/Linux users find themselves needing to edit text files quite a lot. While word processing applications and graphical configuration tools get better all the time, the core behavior of the operating system is based upon UNIX - and UNIX is traditionally a text-based system. So while the average user can get by with a GUI, the true guru will need to fire up `vi` now and then.

`vi` is one of the oldest text editors for UNIX. It is also arguably the best. The unfortunate paradox is that while a `vi` master can exploit its features for extremely fast editing, it is a baffling editor for new users. Nonetheless it is a worthwhile study because some version of it can be found on almost every UNIX system. So while other text editors are easier to learn, `vi` will take you farther and help you work much faster.

Using Vi IMproved, the GNU Vi

The `vi` editor found on most GNU/Linux systems is Vi IMproved, or `vim`. For consistency with other UNIXes the `vi` command can be used to launch it. To open a text file with it, type `vi filename`.

When `vi` opens a file, the terminal displays the file's text, along with a blinking cursor at the top of the screen. At the bottom of the terminal is information about the file - its name, number of lines and number of characters.

```
This is a test
~
~
~
~
"textfile" 1L, 20C
```

Note the vertical strip of tilda (~) symbols. Each line after at the end of the text is represented by a tilda to indicate that it contains no information. This is to avoid confusion with blank lines, which contain no text but do make the file longer.

Insert Mode vs. Command Mode

The first thing that the average user will try to do is enter some text, and this is where vi becomes confusing. vi has two main modes: command mode and insert mode. Unlike most other editors `vi` starts in its command mode, so to edit text you have to enter insert mode by pressing `i`. To return to command mode, press Esc. Insert mode `vi` works like just about any other editor.

Saving and Quitting

When you have made changes to a file and wish to save it, press Esc to enter command mode. Then type `:w`. (Note that when you type the colon it appears at the bottom of the screen, where you can then type the w.) To quit **vi**, type `:q`. These commands can be used in conjunction, so `:wq` will write the file to disk and then quit. If you want to quit without saving changes, such as when you have made an error in typing, force the quit using `:q!`. The exclamation mark is necessary because **vi** will not otherwise quit unless you have saved the file. It can also be used to force a write if you are logged in as root and editing a read-only file, like this: `:w!`.

Command mode navigation

vi is known for the many cryptic shortcuts that make it very fast editor. To use them, enter the command mode by pressing Esc and then try the following cursor movement keys: h, j, k and l. These keys are in a

row, making them easy to use with the fingers of the right hand. The inner keys are for vertical navigation: j moves the cursor down a line and k moves up. The outer keys move the cursor left and right. While the arrow keys are easier to remember, these four letters are a handy tool to make text editing much faster.

Deletion and Replacement

The x key deletes the letter under the cursor, useful for fast deletions. The d key activates the delete command for longer deletions: dw to delete to the end of the current word, d$ to delete to the end of the line, and dd to delete the current line.

You can also specify a number of times to execute the command. 3dw will delete the next three words. 20dd will delete the next 20 lines.

Whenever you have removed characters you can get them back by pressing p to paste them. For lines, the p command will place lines *below* the one where the cursor is positioned.

If you only want to change one character and don't want to bother with going into insert mode, use the r command to replace it. Place your cursor on the character, press r and then the replacement character.

To replace longer strings of text, use c for the change command. This will delete the text you specify and then place you in insert mode to type the new text. The c command uses all of the same variables as the d command: cw will change until the end of the word, cw will change to the end of the line. (In fact, these variables can be used on their own as well - w and $ will move the cursor without changing any text.)

Searching for text

To search for a string of text, make sure you are in command mode and then press the / key. The / symbol will appear at the bottom of the screen, along with your cursor. Enter the text to be located and press Enter. **vi** will take you to the next instance of that string. To search for the same string again, press n.

vimtutor

The best way to learn **vi** is not to read a guide like this, but to actually edit files and practice its various commands. **Vi IMproved** comes with an excellent tutorial, and if it is installed correctly you can run the tutorial with the `vimtutor` command. The tutorial takes about half an hour but is well worth trying, because it presents the user with a file that contains errors that the student can correct in command mode.

Other, Easier Editors

If the above tutorial didn't convince you that **vi** is the right editor for you, there are many others. One of the most user-friendly is **pico**, which comes with the PINE e-mail client. PINE was once shipped with Red Hat Linux but was discontinued due to licensing problems, however you can download it from the University of Washington's website. The **emacs** editor, one of the Free Software Foundation's earliest components for the GNU system, is also popular. Graphical editors include **gedit**, which comes with the Gnome desktop, and **kate**, which comes with KDE. All of these alternatives are easy to use, but their main disadvantage is that any non-**vi** skills that you develop are not necessarily portable to other UNIX systems.

5. Installing GNU/Linux: Fedora's `anaconda`

In the early days of GNU/Linux the operating system was installed manually. This meant that a user had to create the filesystems using boot disks and then compile the Linux kernel and core utilities from source code Tar archives (or "tarballs" as they're affectionately called). This was a time-consuming process that was (and still is) rewarding for experts but not suitable for beginners.

Fortunately most modern distributions of the OS have implemented friendly installer programs; additionally the most popular distributions are "package" based. This means that different programs or groups of programs are installed as single files called "packages." These make choosing options during installation very easy, and you can be easily added, upgrade or remove packages from the OS at a later time. Fedora Core uses the RPM (RPM Package Manager) package format, and the OS is installed using a program called `anaconda`.

Getting the CDs

The Fedora Core installation CDs are available in most stores that sell software. They can also be found inside Linux guides sold in most technical bookstores, since publishers are free to redistribute the OS with their publications. But the CDs can also be obtained for free by downloading them over the Internet. Current ISO images are available at *ftp://download.fedora.redhat.com/pub/fedora/* and at many mirrors listed at *http://fedora.redhat.com/mirrors*. Some download sites feature not only the current release but also legacy and beta releases. Beware: the servers are often slow immediately after an new version of Fedora Core has been released.

Booting to CD or Creating Boot Disks

Newer PCs allow you to boot from a CD-ROM; if your PC supports this option you may have to enable it in your system BIOS, which can generally be found by pressing a setup key when the computer first starts to boot (it's possibly F1, F2, F10 or Del). You can then reboot directly to the Fedora Core Binary CD 1, which starts the installer. This is the best way to start your installation.

If you have an older machine and cannot boot to a CD, Fedora offers "first stage" installers on floppy disks. These are included on the binary CD 1 as disk images. In the CD's `/Fedora/images` directory you can use the file `bootdisk.img` to create a bootable diskette that will start the CD installer. In addition there are three other important files:

- `drvblock.img` - Contains supplemental block device drivers
- `drvnet.img` - Contains supplemental network drivers.
- `pcmciadd.img` - Contains PCMCIA device drivers.

The network installation disk

If you want to do a network installation you can create a disk based upon `drvnet.img` and install GNU/Linux using files stored on another machine. This may be worthwhile if you have to set up more than one machine, as a network installation can actually be faster than a CD installation. Insert a blank floppy disk, navigate to the CD's `/Fedora/images` directory and type this command:

```
$ cat bootdisk.img > /dev/fd0
```

To create a boot disk from within DOS or Windows, insert a floppy, navigate to the DOSUTILS directory on the CD and use this command:

```
D:\> rawrite.exe ..\fedora\images\bootdisk.img
```

After rebooting to the floppy the first stage installer should find the CD-ROM and launch anaconda, the second stage installer.

Choices in `anaconda`

Upon booting to the CD-ROM you will be invited to press enter for a graphical installation or to type *linux text* for a text-based installation. The graphical installer is recommended if it is supported by your machine. The text installation is intended for lower-resolution machines, but presents the same options as its graphical counterpart.

The `anaconda` installer will then present you with the option of testing your CD-ROM media. This is a good idea, especially if you downloaded and burned the three installation CDs. The integrity check will help prevent an interrupted installation due to bad media.

I/O Devices

The first options in the installer allow you to customize it for many languages and to specify your mouse and keyboard types. If you have a wheel mouse it is worthwhile to specify it here. It is also useful to have your mouse emulate a three-button mouse. UNIX graphical interfaces are traditionally used with three-button mice, and three-button emulation allows you to press both buttons of a two-button mouse to simulate the use of the middle button.

Install classes

`anaconda` allows you to choose to either to install or upgrade your system. An installation will perform a fresh install of the OS, whereas an upgrade will look for an existing OS and attempt to upgrade it. The upgrade is rather unique amongst operating systems because it is package-based, so the installer will modify each software package independently. This may result in a very short upgrade time, and allows you to upgrade from most previous versions of the distribution, even beta versions. Still, we tend to recommend doing fresh installations rather than upgrades, since a lot can change between versions.

A fresh installation presents three options: a Personal Desktop will give you an ideal system for a desktop or laptop, with basic productivity suite software. A Workstation will give you a desktop with common tools for software development. A Server will set up a system with many UNIX server utilities, such as Sendmail and X windows. Finally, a Custom installation allows you to control which packages are installed on your system.

If `anaconda` detects a PCMCIA controller it will also include a Laptop option, which will be generally the same as the Custom installation but with PCMCIA enabled by default.

Disk Partitioning

Partitions are divisions of a hard disk. It is sometimes useful to divide a single disk into sections and use each as a separate volume for storing data.

Only the Custom installation type will allow you to specify your disk partitioning preferences; other options use preconfigured partitioning schemes. For example, the Personal Desktop and Workstation installations create partitions for `/` and `/boot`, and if they find an existing GNU/Linux installation on the computer they will use space previously occupied by ext2 and ext3 partitions, erasing their existing contents. The Server will entirely erase and repartition *all* disks, creating separate partitions for `/`, `/boot`, `/home`, `/usr` and `/var`.

`anaconda` comes with two separate tools for disk partitioning, the advanced `fdisk` tool and the simpler Disk Druid. Disk Druid uses a graphical interface, and simplifies the creation of logical partitions within an extended partition. It also allows you to create partitions that will "grow" to the maximum available size when it writes the partition table after your settings are saved. And it lets you relabel your existing partitions, or even preserve them without reformatting if they contain data you'd like to keep.

`fdisk` is much more complex. It allows you to select from a wide range of partition types, customize your disk geometry, copy data and so forth. Because of its relative complexity it is recommended that

most users go with Disk Druid.

Partitions and the UNIX Hierarchical Filesystem

To understand how best to partition your disks it's important to understand the hierarchical filesystem of UNIX-like operating systems. On any UNIX-like system the top-level directory is called the *root directory*. It's represented by the / or slash symbol.

Under the root directory are a secondary level of directories, and a tertiary level afer that, spreading out like an inverted tree. When writing out a whole path you start out with the /, so the secondary level of directories includes names like /boot, /dev, /etc, /home, /opt, /proc /usr, and /var.

Partitions can be hidden within this structure: unlike Windows there is no C: drive, and unlike the Macintosh there is no Desktop. The volumes are "mounted" wherever you feel they are appropriate. Your inverted tree of directories will present the appearance of being a single disk, but in fact it may be made up of separate partitions from one or more disks, each mounted at a particular directory called a *mount point*. Thus /boot and /home may be on separate disks entirely.

To get an idea of how this is used in practice you can examine the directory structure of any UNIX-like system with the df command, which tells you how much disk space is free. To simplify the display we'll run this command with the -h argument to display the output in human-readable megabytes and gigabytes, rather than in kilobytes.

```
$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda8             8.5G  108M  7.9G   2% /
/dev/hda1              29M  6.2M   20M  23% /boot
/dev/hda2              10G  3.7G  6.3G  37% /dos
/dev/hda5              48G  818M   44G   2% /home
/dev/hda3             5.8G  2.0G  3.5G  36% /usr
/dev/hda6             589M   55M  504M  10% /var
```

The first partition listed is /dev/hda8. This is the eighth partition on the drive identified as /dev/hda, or the master drive on IDE channel 0. It is mounted at the very top of the heirarchy, at / . The other partitions are mounted as subdirectories under / .

To the end user this looks like a single filesystem, but in fact it is actually a number of different filesystems tacked together. You can even mount directories located on other computers.

Partitioning Limitations on the ix86 Architecture

The partition table of a hard disk on a 32-bit Intel-compatible system is limited to only four *primary partition* entries. This was once perfectly fine, but drives have a much higher capacity than when this limitation was created. This limit has therefore been circumvented for drives that require more than four partitions by the transformation on one primary partition into an *extended partition*, which can contain any number of **l***ogical partitions*.

On GNU/Linux the extended partition on the first IDE drive is always /dev/hda4, regardless of how many primary partitions exist. Logical partitions begin numbering at 5.

Partitioning Choices

It is useful to partition your disk. For one thing, recovery from a damaged system is a great deal easier if you have more than one filesystem because generally no more than one filesystem will be damaged. It is also easy to upgrade a system in which user data is separate from system files. From an administrator's standpoint, separate partitions allow you to implement user and group quotas to prevent disk space from being used too quickly. They also make the operating system a lot more flexible and easy to reconfigure.

How to partition your disk is one of the more involved choices in Fedora's `anaconda` setup. While `anaconda` provides an automatic partitioning tool that can make the choice for you, experienced users may want to be more deliberate. In the example above I chose to make the largest partition `/home`, because it is a desktop system in which I plan to store a lot of personal data. A mail server administrator may want to make `/var` much larger than `/home`, because `/var` is where mail spools and system logs are stored. The main thing to keep in mind when partitioning is that no matter how you set up your drive, you are probably wrong. There is no correct way to do it, and eventually you may discover that the extra 500 MB you put in `/usr` should probably have been in `/var`.

Some things to consider:

- The `/boot` directory, which contains boot loader data and the Linux kernel, should be near the beginning of the drive to avoid a 1024 cylinder limitation of some system BIOSes. The `/boot` partition need be no larger than 40 MB.
- Don't separate the following directories: `/bin`, `/dev`, `/etc`, `/lib` and `/sbin`. These must all be under `/` in the root filesystem because they are needed while the startup scripts run, and this happens *before* filesystems are mounted.
- Keep `/home` separate. If the machine is your own system, every time you rebuild it (e.g., if a new distribution is released) you can choose to leave the `/home` directory unformatted and keep your existing data and settings. If it is a shared system, a separate partition will prevent users from using up space used by the operating system.
- Keep `/var` separate. It contains logs and mail spools, which can also eat up disk space.
- Consider not partitioning the entire drive. If your machine has a large hard disk, like 40 to 80 GB, partitioning only the portion of it that you plan to use immediately will leave more space for later. When you've discovered how to employ that extra 30 GB you can use a tool like `fdisk` or parted to create or resize partitions to suit your needs.
- Partition with backups in mind. If you plan to back up a section of your filesystem as a whole, as is probably the case with `/home`, keep it separate. You an then use the `df` command to monitor the partition's size, and many archiving tools have options related to partition boundaries.

Filesystems under GNU/Linux

The default filesystem format for Linux is ext3, which is based upon the older ext2 filesystem but has the added feature of journaling. Journaling filesystems keep a record of their operations so that they can be recovered quickly after a crash.

The kernel also supports the **minix** filesystem, which was designed for the Minix operating system that the Linux kernel was initially designed to replace, and the FAT32 filesystem used by Windows (called **vfat** in Linux).

Linux also has support for experimental filesystems still in testing and develipment. Thes include IBM's JFS, SGI's XFS and the new ReiserFS. ReiserFS, though still under development, has shown itself to be good for operations involving many small files, so it may be well-suited for partitions containing mail spools on an SMTP server.

And it doesn't end there. Many other filesystems are supported, including experimental support for Microsoft's NTFS filesystem. However, support for other filesystems dos not come built into Fedora Core's distribution. To get them you'll have to customize and recompile the Linux kernel source code, which will be dealt with in another chapter.

Software RAID

`anaconda` presents the option of implementing software RAID. This provides fault tolerance if you have extra disk space you're willing to devote to it. There are three levels of RAID available.

RAID 1 is often called disk mirroring. It is the oldest form of RAID because it is the simplest. With RAID 1 you create two identical partitions, ideally on separate physical disks, and your system writes

identical data to both partitions.

RAID 5 is the most common RAID. It involves three or more disks and saves parity data across some or all of them. This data can be used if a drive fails to reconstruct its contents on new media. Its only drawback is a small amount of overhead for calculating and writing the parity data.

RAID 0 allows disk striping without the use of parity data. This provides no fault tolerance but is extremely fast as a result of the coordinated use of multiple disks.

These RAID options can be enabled in Disk Druid partitioning setup. Disk Druid requires that /boot be stored on a RAID 1 partition, or on a normal non-RAID partition, because some of its files are location-sensitive.

Networking

`anaconda` gives the option of configuring each network interface card (NIC) separately. There are a number of options available, including whether the NIC should be activated during boot, and whether the NIC has a static IP address or one assigned dynamically via DHCP.

Firewall

After the network setup, `anaconda` will allow you to set up an `ipchains` firewall. Fedora Core provides three options for this type of firewall:

- **High** will block all connections except DHCP address assignment and DNS query replies.
- **Medium** will block connections to the privileged ports (i.e., all ports below 1024), the NFS server, the X Window server and the X Font server.
- **No Firewall** allows all network connections.

The High option is recommended if you plan to use the machine in an environment where you have access to the Internet, or if you will share the network with computers you do not control. The only drawback of such a strict firewall option is it makes setting up network services more challenging because each new service requires a modification of the `ipchains` firewall rules (discussed later on).

The **trusted devices** option allows you to select one or more NICs as "trusted," and all network traffic from a trusted device will ignore firewall rules.

The **allow incoming** section lets you customize the firewall for specific services. Some common services, such as SMTP mail and Secure Shell (SSH) are already listed. You can also list other services in the **other** box. When specifying the port number of a custom service you can use either the port number itself (e.g., `143:udp`) or a name taken from the `/etc/services` file (e.g., `pop3:tcp`).

Selecting Packages

If you selected a custom install earlier in the installation you will have the option to select packages. There are three ways to do this: you can select everything; you can select packages individually; and you can also install components that are comprised of a number of related packages. After you are done, anaconda uses the `/hdlib` resource on the CD-ROM to determine which packages it should install first, based upon CD location and dependencies.

Once you have selected your packages, installation will begin.

After installation you can view the anaconda log at `/root/install.log`, or view of the output of package installations at `/root/install.logl.syslog`.

Advanced Installation

`anaconda` is not just a GNU/Linux installer; it runs under an ISO-based Linux system, and as such it

has a number of logging, networking and display features that can enhance your installation of the OS. The installer features Linux's *virtual console* feature. This means that up to six virtual consoles are present during installation, and you can switch between them using the Alt and Function keys.

- Alt-F1: The console of the text-based installer.
- Alt-F2: A Bash shell for filesystem work and troubleshooting during installation.
- Alt-F3: Installer messages.
- Alt-F4: Linux Kernel messages.
- Alt-F5: Output of file system and boot loader configuration.
- Alt-F7: The console of the graphical installer.

The Headless Installation

The 2.2 series and later kernels allow you to perform a serial console installation. Thus, with neither keyboard nor mouse, you can use the following steps to perform an installation:

- Attach the target machine to the serial console via cable. The console may be a dumb terminal, or a GNU/Linux system running the `minicom` serial communications program).
- Create a bootable floppy from the `bootdisk.img` and, if doing a network installation, `drvnet.img` images.
- If your system will use a graphics subsystem, edit the diskette's `syslinux.cfg` file. Delete the lines starting with *prompt* and *delay*, and change the *default linux* line to *default linux console=ttySX,* where X is the serial port number.
- Set the serial parameters on the terminal to 9600,8,N,1.
- Boot the machine with the floppy, and perform the installation from the terminal.

If you think you will need to use a serial console to access this machine on a regular basis, you may want to enable this console during the installation. You can do this with Kickstart, the automated feature of the Fedora installer, by editing your Kickstart file's `%post` section (Kickstart will be described later on). Modify `/etc/inittab` to delete unused `mingetty` lines and add one for `/dev/ttySX`. Also add this console to `/etc/securetty`, or you won't be able to log onto the machine as root from this console.

Noprobe Mode

In general it is convenient that `anaconda` can detect hardware automatically, however in some cases this can cause the installer to encounter problems. Noprobe mode, formerly called "expert mode," allows you to specify your hardware manually. You can even set parameters, such as the I/O or IRQ port. Generally Noprobe mode is only useful with older ISA expansion cards.

Driver disks

Sometimes you may want to install GNU/Linux on a computer that has very recent hardware - hardware not supported by the current version of Anaconda. A driver disk allows you to add support for this hardware during installation. Four driver disks come with Fedora Core, and are in the `/Fedora/Images` directory on the first CD: `drvblock.img`, `drvnet.img`, `oldcdrom.img` and `pcmciadd.img`. But you can also use a custom driver disk, or one supplied by a hardware manufacturer.

Most devices can actually be added after the installation much more easily, so the driver disk is really only relevant if the device is needed during the installation itself. Network cards and SCSI adapters are frequently required for installation.

Driver disks included with Fedora Core are bootable floppies. Boot to them, and at the `boot:` prompt type either `linux expert` or `linux dd` to select the appropriate device.

Configuring your System after the Installation

Installing a system from scratch is the most obvious way to configure it, but this is generally meant to be a one-time step: after your system is built you'll inevitably want to reconfigure it, and you can't just run the installer each time. To change system settings you can run the following tools at any time:

- `system-config-display` lets you configure your X Window system. This includes your screen resolution and video card.
- `system-config-soundcard` lets you set up and test your sound card.
- Finally, the `setup` command opens a text-based interface to the above utilities and a few others. This is a convenient way to configure authentication and firewall settings.

System Configuration Files

A great deal of your system's configuration is stored in the `/etc/sysconfig` directory. The complete contents of this directory are outlined in the documentation, which is in the file `/usr/share/doc/initscripts*/sysconfig.txt`. But as a brief summary, the directory includes:

- `/etc/sysconfig/network` - Various networking settings, such as the default router and whether to activate networking on startup, as well as the hostname, NIS domain and other information.
- `/etc/sysconfig/network-scripts/` - This directory contains configuration settings for each network inerface.
- `/etc/sysconfig/clock/` - Time zone settings.
- `/etc/sysconfig/init/` - System startup options.

6. Working with File Systems

Disks and Partitions

Every hard disk has a 512-byte section of its first sector that is called the Master Boot Record (MBR). If a hard disk is identified by a computer's BIOS as the primary hard drive attached to the motherboard, the MBR of this disk is used to boot the computer.

The first 446 bytes of the MBR are executable code. This code is called the boot loader, and it is responsible for loading and transferring control to the operating system kernel - in this case, Linux.

The remainder of the MBR is comprised of a two byte header and four 16-byte table entries. These four entries define one to four separate hard disk partitions; as discussed earlier, any additional partitions have to be set up within the fourth primary partition, which becomes the extended partition. The Linux kernel supports a maximum or 63 partitions on each ATA disk, or 15 on each SCSI disk. However it does not have nodes for every possible device, so if you have a need for extra partitions you wil have to create them with the `mknod` command, which will be discussed later.

Boot Loaders

GNU/Linux has two frequently-used boot loaders. Until recently almost all systems used the LInux Loader, or LILO. However, Fedora Core and other distributions are now switching to **GRUB**, the GRand Unified Bootloader. GRUB is a versatile loader. It has a simple configuration file and supports a user-friendly graphical menu. It can also boot multiple operating systems.

Creating Partitions

The most common utility for partitioning a hard disk is `fdisk`. It is a simple, interactive text utility that lets you add or remove partitions, or set BSD partition labels. It does not let you resize or move existing partitions. It also does not function outside of its interactive mode; if you want to run a script that executes a simple partitioning command line you must use the flexible but less user-friendly `sfdisk`.

Disk Druid is a graphical utility that is incorporated into Fedora Core's `anaconda` installer. It has the same basic features as `fdisk`, but also lets you configure mount points and software RAID.

Parted is a GNU utility that lets you add, destroy, move, resize or copy partitions. It is interactive, much like fdisk, but only works for partitions containing FAT, FAT32 and ext2 filesystems.

Filesystems

A partition is not very useful on its own; within each partition is a filesystem that organizes data into files and directories. In all UNIX operating systems the various filesystems are mounted into a single hierarchical structure that spreads like an inverted tree from the root partition, or "/" (sometimes just called "slash").

Some filesystems, such as NFS network shares and the `/proc` filesystem, are virtual filesystems that are actually stored in RAM, rather than on a disk. Swap space is not a filesystem.

Each device on GNU/Linux is assinged a device node - a file that represents it. When writing data, for example, the device is something that you write *through*, with the end result that you write *to* the filesystem on the device itself. This is called *hardware abstraction*: the devices do not deal with data - rather, the filesystem does.

The Linux kernel accesses filesystems through the Virtual Filesystem (VFS), a single layer that passes the kernel's requests to the proper filesystem management code -be it for Extended, FAT, VFAT or other filesystems.

Making a Filesystem

You can create a filesystem in a hard disk partition with the `mkfs` command. By default it will create an ext2 filesystem, the Linux standard. But because GNU/Linux supports many filesystems, `mkfs` is a frontend for a suite of separate programs, each designed to create a different filesystem type. They can be invoked with the -t option:

```
# mkfs -t fstype
```

Using this structure `mkfs` will call `mkfs.ext2`, `mkfs.ext3`, `mkfs.vfat`, and so forth. Creating an Extended filesystem can also be achieved with `mke2fs`. There are many options available with Extended filesystems. For example:

```
# mke2fs -b 2048 -N 4096 -j /dev/hdb1
```

This example takes the first primary partition of the second IDE hard disk and makes an extended filesystem in it.

The `-b` option sets the size, in bytes, of each datablock. Since a datablock is the minimum amount of disk space a single file can occupy, this filesystem is optimized for files no less than 2 kilobytes in size. A datablock size of 512 bytes would be optimized for many small files.

The `-N` option reserves the number of inodes specified. An inode is just a record for a file, but each inode takes up 128 bytes. Thus if you create a 2 gigabyte filesystem but only plan to store one very large file in it, you could create the filesystem with a single inode and save a great deal of space. (Another useful option is the -i option, which sets the byte/inode ratio. It is less flexible, however, as the largest value it will accept is 8192.)

Finally the `-j` option adds a journal, making this an ext3 filesystem.

Working with Extended Filesystems

For many years ext2 was the standard GNU/Linux filesystem, and for compatibility the newer **ext3** format is nearly identical but contains a journal to allow faster recovery from a crash.

ext3 filesystems can be created natively, but they can also be created by adding a journal to an existing ext2 filesystem. To add a journal, follow these steps:

1.) Edit the `/etc/fstab` file, changing the appropriate references from ext2 to ext3.

2.) Create the journal with the `tune2fs` command:

```
# tune2fs -j /dev/partition
```

3.) The Linux kernel requires an initial ramdisk in order to access certain types of devices during boot. Create a new initial ramdisk that gives the kernel access to the ext3 module with the `mkinitrd` command:

```
# mkinitrd /boot/initrd-kernelversion-.img kernelversion
```

This ramdisk will contain the appropriate module because it configures itself partly from the `/etc/fstab` file.

4.) Reboot the machine, and check the `/proc/mount` file to ensure that the partitions are mounted as ext3.

There are three modes of filesystem jourals. The default is the ordered journal, which keeps track of meta-data. The journal mode keeps track of data as well. Writeback mode is less detailed but allows for

faster recovery. The journal mode can be set in `/etc/fstab` as a "data=*mode*" option.

Other Filesystems

GNU/Linux supports many different filesystems aside from Extended, including Microsoft's FAT and FAT32 filesystems and experimental read-only support for NTFS. Three popular journaling filesystems are JFS, ReiserFS and XFS.

JFS is a 64-bit filesystem developed by IBM. It has a minimum filesystem size of 16 MB, so it's not for everyone. But it has a maximum size of 16 petabytes (1,024 terabytes), and doesn't require inodes to be reserved until they are needed.

ReiserFS is an ambitious journaling filesystem developed by Hans Reiser at Namesys. It doesn't use an entire block for each file, instead packing small files together in a single block along with "tails" of large files. It is still under development, but has proven to be very efficient in filesystems involving many small files. It may therefore be ideal for a Linux `/var` partition containing frequently-updated mail spool files. Recent tests of version 4 of the filesystem show that it is the fastest available to Linux, however this speed comes at the expense of some CPU cycles.

XFS is a journaling filesystem developed by SGI. It is used as the default filesystem in SGI's IRIX operating system but is also available under the GPL for GNU/Linux. It is widely regarded as an extremely scalable solution that uses btrees to support large or sparse files.

Mounting filesystems

In order to access a filesystem's data it must be *mounted* onto the hierarchical filesystem, the inverted "tree" structure common amongst UNIX flavors. This tree structure provides many advantages, including a directory layout common to most UNIX-like operating systems and the flexibility of mounting multiple local and networked volumes at any point on the the inverted tree, independent of disks and drive letters and transparent to the end user.

You can use the **`mount`** command to connect a filesystem to the tree. The syntax is as follows:

```
mount options device mount-point
```

So if you were to mount mount a CD-ROM, which has a filesystem type called `iso9660`, we'd do this:

```
# mount -t iso9660 /dev/cdrom /mnt/cdrom
```

Your system will keep a record of available filesystems in the `/etc/mtab` file; there is no need to edit this file since it is maintained by the system.

The `/etc/fstab` file

To change your default filesystem layout and settings you can edit the `/etc/fstab` file. The `anaconda` installer will generate this file based upon your installation choices.

`/etc/fstab` is a very important file because it is used by the `/etc/rc.d/rc.sysinit` script that runs after you boot your machine. This process creates your filesystem hierarchy during system initialization by passing the options in the file to the `mount` command

A look at a sample `fstab` file should illustrate the types of data the system uses to keep track of your volumes:

```
# device or label   mount point  fs type  options              dump freq   fsck order
LABEL=/             /            ext3     defaults             1           1
LABEL=/boot         /boot        ext3     defaults             1           2
none                /dev/pts     devpts   gid=5,mode=620       0           0
LABEL=/home         /home        ext3     defaults             1           2
```

```
none                /proc        proc     defaults              0          0
none                /dev/shm     tmpfs    defaults              0          0
/dev/hda3           swap         swap     defaults              0          0
/dev/cdrom          /mnt/cdrom   iso9660  noauto,owner,kudzu,ro 0          0
/dev/fd0            /mnt/floppy  auto     noauto,owner,kudzu    0          0
/dev/hda1           /dos         vfat     umask=000             1          2
/SWAP               swap         swap     defaults              0          0
```

The first field is the device name or label. A name is just the name of a standard device node. For example, the first primary partition of the master IDE drive on IDE channel 0 would be `/dev/hda1`. But you can also use labels, which allow you to change the physical configuration of your machine without having to update the `/etc/fstab` file.

The second field is the mount point, which is the path in the filesystem used to access the device. in the example above, the first partition on the hard disk, `/dev/hda1`, is under `/dos`. It appears to be a directory, but in fact it's actually a whole vfat (Windows FAT32) filesystem.

The third field is the filesystem's type, and the fourth is a comma-separated list of options. The last two fields are the `dump` frequency and `fsck` order.

Probably the most useful field to learn and understand is the list of options, discussed in the context of the `mount` command later on in this chapter.

Filesystem Labels and Mounting Shortcuts

If you set up your system with ext2, ext3 or ReiserFS partitions, the `anaconda` installer will label them. This makes them easier to identify later on, and allows you to move devices around without editing fstab. As you can see in the example above, the `/`, `/boot`, and `/home` directories are labeled. This means that they can be easily referenced by labels instead of by device names, which is useful if you are rebuilding an existing system and wish to recover data or preserve entire partitions. It is also useful it you change the order of your drives.

You can manually mount filesystems based upon their label:

```
# mount LABEL=/home /home
```
or
```
# mount -L /home /home
```

In fact in this particular example you don't really need to be this specific because the filesystem is referenced in the `/etc/fstab` file and the Linux kernel is smart enough that it doesn't require both the mount point and the device name or label when the proper match is already outlined for it. You could therefore simplify it like this:

```
# mount /home
```

If you'd like to add labels to your system's partitions, or change the ones already in place, you can use the **`e2label`** command in this format:

```
# e2label devicename label
```

Be careful about changing labels haphazardly; if your `/etc/fstab` file uses labels you could end up with mis-mounted partitions and an unworking system.

Other `mount` and `/etc/fstab` Options

The `mount` command supports options that specify the filesystem type and control how the filesystem is mounted. The longer syntax of the `mount` is thus:

```
# mount -t filesystem-type -o options device mountpoint
```

The type can be ext2, ext3, vfat (Windows FAT32), iso9660 (CD-ROM), ReiserFS, or any of a dozen others, but it must always *precede* other options. This is because some options are type-specific. The options set by the superuser in `/etc/fstab` supersede options used in the `mount` command.

For ext2 and ext3 filesystems the most common options are these:

- **`async`** - File changes are managed asynchronously (default).
- **`auto`** - Allow the command `mount -a,` which mounts automatically (default).
- **`dev`** - Allow device files (default).
- **`exec`** - permit execution of binary files (default).
- **`gid=groupname`** - All files are owned by the specified group.
- **`loop`** - Mount the filesystem using a loopback device. Useful for mounting ISO images without having to burn them to CD-ROM first..
- **`noatime`** - Do not record last-accessed times for files and directories.
- **`noexec`** - The opposite of the `exec` option, this prevents the execution of binaries.
- **`nouser`** - Only the superuser can mount or unmount the filesystem (default).
- **`owner`** - The mount request and the device must be owned by the same user.
- **`rw`** - Read and write access is granted (default).
- **`suid`** - suid and sgid bits on files are honored (default).
- **`uid=username`** - All files are owned by the specified user.
- **`user`** - Any user can mount or unmount the filesystem, but only the user who mounted the it can unmount it.

The `owner` and `user` options automatically invoke the `nodev`, `noexec` and `nosuid` options for security reasons, but this can be overridden by specifying other options. Fedora Core's customizations also somewhat amend the ownership options: a user logged into the console is automatically the owner of CD-ROM and floppy devices, allowing easy local access to these media types.

Examples of `mount`

It might be useful on some systems to mount the home directories of users in a way that prohibits launching their own programs, so we can use `noexec`:

```
# mount -t ext2 -o noexec /home
```

We can mount a vfat partition for use by local users, but since it doesn't support UNIX file permissions we can specify ownership rather than allow it to default to root:

```
# mount -t vfat uid=500,gid=500 /dev/hda1 /dos
```

We can mount a CD-ROM ISO image as a read-only file system, using the `loop` option to utilize a loopback device. This lets us mount a file as a filesystem, and it looks exactly like the CD would look if we were to burn the ISO image and mount the disc:

```
# mount -t iso9660 -o ro,loop cdimage.iso /mnt/cdrom
```

Mounting Filesystems from a Network

Filesystems can be mounted remotely onto the GNU/Linux filesystem, where they behave exactly like local filesystems. This is particularly useful for shared network storage; the `/home` directory of a file server can be made available to many separate computers and act like a local home directory partition. There are two commonly used protocols for this file sharing: NFS and SMB.

NFS

NFS, short for Network File System, was developed by Sun Microsystems and is the most frequently-used protocol for sharing volumes between UNIX-like machines. You can mount an NFS share with the

`mount` command:

```
# mount servername:/shared/directory /mnt/nfsmount
```

Note that the `-t` option is omitted, since the `mount` command understands the syntax well enough to guess that the type is NFS.

You can set up your `/etc/fstab` file to mount an NFS share automatically:

```
servername:/shared/directory    /mnt/nfsmount    nfs    defaults 0 0
```

If you don't know the shares available on an NFS server you can get a list of them with the showmount command.

```
# showmount -e servername
```

The caveat of NFS file sharing is security: NFS permissions are based on UIDs and GIDs, as with any other UNIX filesystem, so it is important that an NFS client's users and groups have the same UIDs and GIDs as the appropriate users and groups on the NFS server. For example, if the local user wsmith with the UID 500 connects to a shared home directory on an NFS server where egoldstein also happens to have the UID 500, wsmith will be able to access egoldstein's files. The username is ignored. In environments where file security is a concern it is wise for the administrator of an NFS server to have control over client configurations as well.

SMB

Sometimes called CIFS, SMB the Windows filesharing protocol, but it is available as both client and server thanks to the Samba project (*http://www.samba.org*). Samba is generally used to mount Windows shares on UNIX machines or set up a UNIX file server that mimics the feature set of a Windows server. However may people use it between UNIX systems, because it avoids the problematic NFS use of UIDs and GIDs. Since security is based upon username and password, Samba is useful in environments where the administrator of the file server does not have complete control over the configurations of each client.

To mount an SMB share:

```
# mount //servername/shared-directory /mnt/smbmount
```

Like NFS, the mount command understands the syntax of the resource and treats it accordingly. In the case of SMB `mount` will pass the request to `/usr/sbin/smbmount`, which is part of Samba.

You can see the shares available on an SMB server with the `smbclient` command.

```
# smbclient -L servername -N
```

Unmounting Filesystems

The **`umount`** command is the opposite of the `mount` command. It is useful when unmounting removable media, such as CD-ROMs or floppies. It also comes in handy during filesystem maintenance. Its format is like this:

```
umount [options] device-or-mountpoint
```

Sometimes a filesystem may be in use when you wish to unmount it, either because it is someone's present working directory or because a file in it is in use. Under these circumstances unmounting is not allowed, however the `fuser` command can be helpful when a filesystem is locked.

To list all open resources on a locked device, use the verbose option and the mount point.

```
# fuser -v /home
```

This will include any related process IDs. To kill the processes use the kill option.

```
# fuser -km /home
```

Another useful command for this and similar situations is **`lsof`**. This lists all open files belonging to all active processes. In the case of unmounting, however, `fuser` is probably a better option because it is more concise.

Occasionally you might want to unmount your filesystem solely for the purpose of remounting it again. For example, you may have added newly-modified options to the `/etc/fstab` file and want to try them out. To avoid killing processes use the `mount` command with the `remount` option.

```
# mount -o remount /home
```

You can add comma-separated options to this if you'd like to grant temporary options to a filesystems. For example:

```
# mount -o remount,ro /home
```

This would quickly change /home to a read-only filesystem.

The Automounter

The `automount` daemon is launched from the `autofs` script. It is designed to allow flexible access to non-removable media for end users who do not have root access to a system.

Any user can mount a CD-ROM or a floppy disk, but it is sometimes useful for other filesystems to unmount between uses. `automount`, when configured properly, can mount a filesystem as soon as it is requested and then unmount it after a specified interval of inactivity. This is particularly useful for network shares.

`automount` is configured with the `/etc/auto.master` file. Each line in the file references a directory, a series of mount options and then the name of the device or resource. For example:

```
/misc       -fstype=ext3              /dev/hda7
```

In this case if a user uses the `ls` or `cd` command in a way that requires the presence of the /misc filesystem, `automount` mounts it and keeps it mounted during use, as well as the default 60 seconds after activity has stopped.

Virtual Memory (move elsewhere?)

A computer's RAM is a severely volatile resource. It is entirely electronic, so if you pull the plug on your computer everything in memory will cease to exist. But this electronic nature of RAM is also its strength: it is extremely fast compared to hard disks.

Still, the amount of memory on your system is finite - probably an order of magnitude smaller than the amount of disk space you have. If you utilize all of your RAM, further processes will fail, so *virtual memory* is available as a disk-based supplement to your RAM.

Virtual memory saves inactive data from memory into *swap space*, and then retrieves it as needed. This adds overhead to your processes: the more virtual memory you have to use, the slower your machine will become. But it helps to prevent your computer from running out of memory entirely.

Swap space can be stored in a file or on a partition. To make a swap partition, create a partition of type 0x82 (your partitioning utility should provide a way to st its type. Then use the `mkswap` command:

```
# mkswap -v1 /dev/hda7
```

This creates a special signature on the filesystem. To use the new swap partition add a record for it in your `/etc/fstab` file like this:

```
/dev/hda7    swap    swap    defaults    0 0
```

You can then activate the swap space using the swapon command. The -a option activates the space, and the -s option to display a summary of swap usage.

```
# swapon -a
# swapon -s
Filename                        Type            Size    Used    Priority
/dev/hda7                       partition       612320  30816   -1
```

To create a swap file, use this variation on the data dump command, using your own values for the filename and the size (kilobytes):

```
# dd if=/dev/zero of=filename bs=1024 count=KB
```

Run the same `mkswap` command used above to create the swap signature and activate it using `swapon`. To activate it automatically create or modify a startup script (e.g., /etc/rc.d/rc.local) to run `swapon`.

You can deactivate your swap usage with the `swapoff` command.

Filesystem Maintenance

Filesystems are subject to occasional corruption, which can lead to data loss. This is particularly likely when a system is shut down improperly, resulting in incomplete disk operations.

The **`fsck`** command is a frontend for a suite of filesystem utilities. If you are checking an ext2 or ext3 filesystem the command invokes the `e2fsck` utility. To run the utility and force the check:

```
# fsck -f devicename
```

During the system initialization the `rc.sysinit` file runs `fsck` on any filesystems that have been marked as corrupt, or if they have data in the journal. `fsck` will also attempt to repair them, and if it succeeds the boot process will resume. If it fails, `rc.sysinit` will launch a superuser login and tell you to run `fsck` manually. You should not run `fsck` on a filesystem that is mounted in read-write mode.

If `fsck` finds problems and repairs them, it may unlink data blocks from their inodes. If this happens the file will not be lost, but rather re-linked to a file in the `lost+found` directory in the root directory of the filesystem. The file will be named after the number of its previous inode.

`tune2fs` can be used to modify attributes of a filesystem. You could use it to change the maximum count before the boot `fsck` is forced, causing a complete filesystem check upon each boot. Or you could use the -j option to add a journal, converting an ext2 filesystem to ext3.

The **`dumpe2fs`** command lists information about the filesystem.

The **`resize2fs`** command can be used to change the size of an ext2 or ext3 filesystem. This does not adjust the size of a partition; this must be done first with a partitioning utility like `fdisk`. Try the `parted` utility, which can resize both a filesystem and its partition simultaneously.

Adding a New Drive

Adding a new disk drive to a GNU/Linux system can be confusing to Windows and Macintosh users. Rather than create a new desktop volume or drive letter, the system will merely recognize the new drive at boot and ignore it. As long as it is installed properly with the proper jumper settings, and as long as it is

recognized by the system BIOS, the boot process will make a record of the new drive in `/var/log/dmesg`.

Once the drive is installed you can use `fdisk` or another disk utility to create one or more paritions. If a partition will be used for swap space, use `fdisk` to change the partition type to 0x82, the swap standard. Then use `mkfs` to create any filesystems, or `mkswap` to mark the swap space. Remember that it is useful to label filesystems, which you can do with the -L option of `mkfs`.

You must then decide what data you wish to keep in your new filesystems. If it is for personal data you may want to mount it at /home. If you are operating a POP3 mail server you may want to mount the new space at `/var`, `/var/spool`, or `/var/spool/mail`. Or you can create an entirely new mount point. If you plan to use an existing directory as your mount point you may want to mount your new filesystem in a temporary location so that you can copy data from the existing directory. For example:

```
# mkdir /tempdir
# mount /dev/hdb1 /tempdir
# cp -a /home/* /tempdir
# rm -rf /home/*
# umount /dev/hdb1
# mount /dev/hdb1 /home
```

You will probably want to perform this type of operation in single-user mode (runlevel 1), which will be discussed later. In this example your new partition becomes your home directory, replacing a normal directory in the root filesystem.

Finally, add entries for your filesystems to the `/etc/fstab` file. This is optional, but useful, and necessary if you'd like the filesystem to mount automatically during boot.

Managing Filesystem Usage

It is important to periodically check on the use of disk space on a system. Otherwise you may run out, resulting in failed processes. The `df` and `du` commands can help to monitor this.

`df` lists the amount of space on a given filesystem, or on all filesystems. It defaults to kilobytes, so for larger filesystems it is useful to use the -h option for "human-readable" output. For example:

```
# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda6             4.6G  2.3G  2.1G  52% /
/dev/hda2              36M  9.2M   24M  28% /boot
/dev/hda5             4.8G  139M  4.4G   3% /home
/dev/hda1             8.9G  4.7G  4.1G  53% /dos
```

To display the disk usage for a specific filesystem, simply specify its device name or mount point after the options. Use the -i option to find out the number of inodes used and free.

The **`du`** command displays the directory structure and the number of data blocks assigned to each directory. As with `df` you can either get the directory structure of the whole system, or of a partition you specify.

7. Hardware on GNU/Linux

Linux Kernel Device Drivers

Devices on a GNU/Linux system are generally managed through the use of device drivers. These can be implemented "statically" - that is, as components of the Linux kernel - or as modules that are loaded when needed.

Static drivers are loaded during the boot process using boot parameters supplied by the boot loader (GRUB and LILO are the most popular loaders), but if you wish to modify your boot loader by passing customized parameters to the kernel during boot you can read the in-depth explanation in the `bootparam` man page.

Modular drivers can be loaded into the running kernel without rebooting using the `insmod` command. They can also be configured with the `modprobe` command.

Devices, like almost everything else in a UNIX operating system, are expressed as files; most of them are stored in the /dev directory in the form of special files called "nodes." (Network interfaces use a network interface abstration, and therefore do not follow this rule.) Thus almost every device on a GNU/Linux system can be found somewhere under /dev.

Detecting Hardware with Kudzu

During boot the kernel will generate messages that scroll up the screen and record your detected hardware. These messages are cataloged in the continuously-overwritten log file `/var/log/dmesg`, and can be viewed by either viewing this file or using the `/bin/dmesg` command. Additionally you can use the hwbrowser command to open a graphical hardware utility.

Kudzu is a utility named after a southeastern American weed that keeps a detailed database of the system's hardware, including the hardware's configuration, at `/etc/sysconfig/hwconf`. Kudzu runs during boot to compare the database to the detected hardware. If hardware has been added or removed Kudzu will offer an automatic configuration or a configuration utility. Its catalog of known hardware is stored in the directory `/usr/share/hwdata/`.

Additionally, one can initiate a Kudzu hardware scan at any time by starting its service with the following command:

```
# service kudzu start
```

Hardware Details in the /proc Filesystem

Like the /dev directory, the /proc directory contains files that are not really typical files per se, but rather contain dynamic information used by the kernel to keep track of running processes and hardware. /proc contains a great deal of information, some of which can be viewed with special commands. One example is the `lsusb` command, which lists all connected USB devices.

Device Nodes

Most Linux devices are accessed as special files called device nodes, which are conventionally kept in the /dev/ directory. A device file does not use any space on the filesystem; it is only an access point to the device driver, so you can access devices using their nodes. For example, you can send a list of the users on your system to a floppy disk by listing the contents of the /etc/passwd file and redirecting the output to the floppy device node:

```
# cat /etc/passwd > /dev/fd0
```

Or direct it to a parallel port printer instead:

```
# cat/etc/passwd > dev/lp0
```

In either case the user list is not sent to a file, but to the device. The printer will print it. The floppy will store the information as raw data, without encapsulating it in a file.

There are two types of devices: character (stream-oriented, such as a command-line terminal) and block (random access, such as a disk). Each device has a major and a minor number: the major number refers to a device in the kernel, and the minor number is a driver parameter.

For example, the first partition of the master ATA disk drive on IDE channel 0 (which on a DOS-based system would be mapped as "drive c") looks like this when the `/dev` directory is listed with the `ls -l` command:

```
brw-rw----    1 root     disk       3,   1 Jul 23 14:50 hda1
```

The device is a block device (as denoted by the leading "b" in the permissions at the start of the line). It has the major number 3 and the minor number 1.

The main terminal `tty0`, on the other hand, is a character device with the major number 4 and the minor number 0.

```
crw--w----    1 root     root       4,   0 Jul 23 14:50 tty0
```

If you ever need to create a device node, such as in a situation where you have accidentally deleted one, you can do so with the `mknod` command. The following example creates a node for the above hard disk drive; the syntax specifies the device name and type, and the major and minor numbers:

```
# mknod hda1 b 3 1
```

Some devices are just symbolic links to other device nodes. For example, a CD-ROM drive is really just an IDE drive that reads CD-ROMS instead of magnetic media, so the device `/dev/cdrom` is just a link to `/dev/hdXX`.

Block Devices

By default the kernel has support for standard floppy drives and IDE controllers. SCSI controllers are supported in modules. If Kudzu or the OS installer detects a SCSI controller it will place an alias to it in `/etc/modules.conf` called `scsi_hostadaptor`. If the installer detects a SCSI device it is important for that device to be available at boot time, so it will create a file named `/boot/initrd-(version).img` that is used during the boot process.

The `/proc/ide` and `/proc/scsi` directories contain information about detected drives and controllers.

PCI Devices

The command `/sbin/lspci` will give a detailed list of devices on the PCI bus, a bus present in almost all x86 compatible machines. PCI devices are easily configurable as Plug and Play, however PnP ISA devices are also usable under Linux and can be found under `/proc/isapnp`.

USB and Firewire

When the kernel detects hotswappable devices that use USB or IEEE 1394 (Firewire) interfaces, it notifies running processes using the `/sbin/hotplug` program. Modules for these devices are loaded by scripts in the `/etc/hotplug` directory. The `/sbin/lsusb` command will list all detected USB devices.

PCMCIA Cards

PCMCIA support is initiated via the `/etc/rc.d/init.d/pcmcia` init script. The kernel has modular support for PC cards, and configures them in `/etc/sysconfig/pcmcia`. The card manager daemon, `/sbin/cardmgr`, monitors the swapping of PCMCIA devices, but they can also be configured with the `/sbin/cartctl` utility.

Graphics and the Consoles

The kernel image contains support for standard SVGA text consoles. During boot, the kernel initializes a number of virtual consoles between which you can switch using the Ctrl-Alt-Function keys. The graphical X Window environment runs in virtual console 7. You could, for example, switch between console 1 and console 7 with Ctrl-Alt-F1 and Ctrl-Alt-F7, which would switch you between text- and graphics-based consoles. This is useful when troubleshooting your X Windows environment.

To configure the virtual consoles use the `/sbin/setterm` command.

The kernel provides experimental support for graphical environments with framebuffer kernel modules. These provide an interface to many graphics cards, and can configured with the `/sbin/fbset` command and the `/etc/fb.modes` file, both of which are available in the fbset RPM package.

However the more traditional, and by far the most popular, method of implementing a graphical environment is to use an X Window server. Fedora Core and most other distributions use an X server provided by X.Org, and this is described later in this guide.

Ports

Details on the various serial ports in the system are in /proc/tty/driver/serial, and settings can be configured through the `/sbin/setserial` command. At startup this command is usually run via the `/etc/rc.serial` script. Serial devices will often use device names that are symbolic links, rather than real nodes. A modems, for example, may use `/dev/modem`.

Parallel support is supplied via a kernel module.

8. System Initialization

Introduction

Understanding what a GNU/Linux system does each time it boots is helpful in understanding the operating system as a whole, as it encompasses many of the basics of filesystem mounting, the functions of the kernel, and system services that are critical in understanding how GNU/Linux works.

It is also a unique and somewhat holistic glimpse into the operating system's simplicity. It was once said that UNIX was so simple it took a genius to understand its simplicity; this is a good illustration of the underlying simplicity of GNU/Linux, but in fact understanding it does not really require genius at all.

There are four primary elements to the boot process: the BIOS, the boot loader, the Linux kernel and `init`.

The BIOS

The BIOS is the Basic Input/Output System - the most basic point of contact between hardware and software on Intel x86-compatible computers. Without the BIOS your system would not be able to boot, and your operating system would be unable to interface with your hardware.

The first thing the BIOS does after your system is powered on is a power-on self-test, or *POST*. After a successful test it will look for peripheral devices and a data source from which to boot. It loads and saves its hardware configuration using the CMOS (Complimentary Metal Oxide Semiconductor), a small battery-powered chip on your systems motherboard. The CMOS uses the battery to retain its settings even when your system is powered down. Any changes to your BIOS settings are kept in the CMOS.

The BIOS selects a boot device from the list of local media devices, whether it be a CD-ROM drive, a hard disk or some other device. It does this partly by using your CMOS settings, and partly by prioritizing devices that are available. Older systems give floppy devices a high priority; newer systems may default to the CD-ROM or hard disk. This can be customized in the BIOS.

The Boot Loader

After POST the BIOS reads the first 512 bytes of the hard disk (if this is the chosen boot device). This 512 bytes is called the Master Boot Record (MBR). This contains your system's boot loader.

The most widely-used boot loader, **GRUB** (GRand Unified Bootloader), supports a number of advanced features. It can read standard Linux filesystems, so it can access its configuration file at `/boot/grub/grub.conf` during boot. Because of this feature you don't have to run a utility to create the boot loader in the MBR each time you change its configuration; just edit the file. GRUB can be set up to boot many different operating systems, or to multiple versions of the Linux kernel.

GRUB features an advanced graphical boot menu, making it very easy to use once configured.

The following is a typical GRUB configuration file:

```
default=0
timeout=10
splashimage=(hd0,1)/grub/splash.xpm.gz
title GNU/Linux (Fedora Core 1) (2.4.22-1.2115.nptl)
        root (hd0,1)
        kernel /vmlinuz-2.4.22-1.2115.nptl ro root=LABEL=/ hdc=ide-scsi
        initrd /initrd-2.4.22-1.2115.nptl .img
title GNU/Linux (customized) (2.4.22-custom)
        root (hd0,1)
        kernel /vmlinuz-2.4.22-custom ro root=LABEL=/ hdc=ide-scsi
        initrd /initrd-2.4.22-custom.img
```

```
title Windows 2000 Professional
        rootnoverify (hd0,0)
        chainloader +1
```

This boot configuration allows you to boot three different operating systems. The first two use the Linux kernel; one a default installation of Fedora Core and the second a modified kernel.

The first line of each Linux loader is the location of the kernel (root, in this case not necessarily referring to the ultimate system root directory). Note the hard disk notation: unlike the Linux kernel device names, GRUB recognizes devices in the format (hd**x**,**y**) where x is the device number and y is the partition number, each starting at zero. The second line is the kernel, and the third line is the initial ramdisk.

The third option in this example file is an alternative OS, and it requires somewhat different options. For more information on booting alternative systems from GRUB see the manual page.

Older GNU/Linux systems used **LILO** (the LInux LOader). LILO is a dependable but not very feature-rich boot loader that loads the Linux kernel based upon its physical location on the disk. If you change your configuration or move your kernel on a LILO-loaded system, use the **`lilo`** command to reinstall the loader in the MBR. LILO can also be configured to use a graphical menu, but you can press Ctrl-x to switch to a text prompt.

The Kernel

When the kernel loads, device drivers compiled into it will try to locate their devices. If successful the driver will initialize, and will output a message. Drivers that are compiled as modules instead of as part of the kernel must be included in the initial ram disk, which is mounted to make these modules available. Then the kernel mounts the root filesystem in read-only mode.

Once the root filesystem is readable the kernel starts **`init`**, the parent of all system process, and pass control to it. `init` immediately starts to output messages to the screen; if you can't read fast you can check out these boot messages in the log file `/var/log/dmesg`.

`init`

`init` is the parent of all processes, and always has the process ID (PID) of 1. It gets its configuration from the file `/etc/inittab`. This file outlines how to boot this system for each **run level**, and also which run level should be the default. When switching from a graphical system that uses the X-Window system to a standard UNIX console-based system you are switching from run level 5 to run level 3, and `init` makes this change by consulting `/etc/inittab`.

In run level 5 `init` chooses a display manager, such as Gnome, KDE or a plain X-Windows desktop. The choice of desktops is is stored in the `/etc/sysconfig/desktop` file, and the choice of desktops determines the login manager used to authenticate users when the `/etc/X11/prefdm` script is run. Choices include gdm (Gnome), kdm (KDE) or xdm (X-Windows), though any of these login managers can be used to launch a different desktop.

`init` also provides a text login by spawning six instances of `/sbin/mingetty`., which is a small program that opens and initializes the tty line, reads a login name, and invokes login.

Run levels

Run levels define which services should be running, so switching from one run level to another starts and stops services. `init` defines the run levels 0-9, `S` and `emergency`, though traditionally only 0-6 are used. The various run levels each run their own set of processes, so for example run level 5 launches the X Window system, whereas run level 3 does not, so when you switch from run level 5 to run level 3 the X server is stopped. Here is an overview of the various run levels:

0- Halts the system.

1 - Single user mode, plus runs the run level 1 scripts.
S, s, or single - Single user modes, without the scripts; useful in troubleshooting damaged systems.
2 - Multi-user mode, without NFS file sharing.
3 - Multi-user mode.
4 - (unused)
5 - Graphical (X-Window) mode.
6 - Reboots the system

You do not need to shut down and restart to change run levels, and knowing how to change run levels can be useful. For example, if you are in run level 5 and are having trouble with the X Window system you can simply switch to run level 3, a standard text-based console mode, like this:

```
# init 3
```

You can then switch back to run level 5 to re-launch the X Window system and start a new graphical session:

```
# init 5
```

Or if you have to perform work on a mail server that runs in run level 3, you can temporarily switch to run level 1 to stop the mail server and other multi-user services while you do your work, then switch back to run level 3 when you are finished. The `id:?:initdefault` line in the `/etc/inittab` file sets the default run level in the second field (between the two colons). If no run level is selected you will be prompted for a run level.

Run levels 4 and 7-9 are unused, but can be configured for special uses. More information on the run levels and how they can be used is in the next section on services.

`/etc/rc.d/rc.local`

When your system enters a new run level the very last script it runs is `rc.local`. This file is not often used nowadays, but it is a common place to put custom scripts you want to run whenever a run level starts. It is also ignored by Fedora Core installers during upgrades, so it is a safer place for customizations than the other, more frequently used scripts.

Regarding the Virtual Consoles

Fedora Core provides six working *virtual consoles*. These text-based consoles, which all run at the same time, each have their own login prompt and are similar to VT-100 terminals on older UNIX systems.

You can switch between virtual consoles by using the keyboard combination `Ctrl-Alt-Fkey`. For example, `Ctrl-Alt-F1` switches to virtual console one, `Ctrl-Alt-F2` to console two, and so forth. The X-Window system runs in virtual console seven.

If you're using multiple text-based consoles you can switch between them in the aforementioned fashion, or use `Alt-RightArrow` and `Alt-LeftArrow` (this doesn't work in console seven when X is running). To scroll up and down and see previous commands, use `Shift-PageUp` and `Shift-PageDown`.

The six working consoles are spawns of the `getty` process, which provides the login prompt and allows input at a shell. The `gettys` are called in a section of the `/etc/inittab` file, shown here:

```
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

You can add additional consoles or shut off unneeded ones by editing this file. But though the consoles that do not have `getty` processes do not allow input, they can still be used for output. For example the superuser can output the contents of a file to it:

```
# cat /var/log/messages > /dev/tty8
```

This can also be useful when working with the system log daemon, which can be configured to output errors to a virtual console.

Graphical boot

Fedora Core 1 and other recent distributions support graphical booting. After the kernel loads and some basic services start, a simple X server will launch and display the progress of your system's initialization, along with a nice image. This serves no purpose at all except that it looks a lot nicer than the output of service initialization scripts, which is what you normally see while booting a GNU/Linux system.

Because graphical booting is pointless Fedora Core 1 has it disabled by default, however you can activate it if you think you'd like your boot process to take a few extra seconds. First you must install the RHGB (Red Hat Graphical Boot) package. Once it's installed, find the `GRAPHICAL` line in the `/etc/sysconfig/init` file and set its value to `yes` (you can add this line if it is not already present in the file). Then edit your boot loader configuration, which is in `/boot/grub/grub/conf`. Find that stanza for the installation you'd like to boot graphically and add the string `rhgb` to the end of the `kernel` line. Reboot and enjoy.

9. Service Types and Management

There is more than one type of service on a GNU/Linux system, and each type has its own method of launching and running. It is useful to understand what the attributes of each type are, and which services each is typically used to run.

Some services are designed to run continuously in the background, awaiting stimulus to provide its service. These are called *daemons*. When a daemon process receives the necessary stimulus (e.g., an HTTP request for a web server), it generally forks a copy of itself that performs the service while the original daemon awaits further requests. There are two main types of daemons: *standalone* and *transient*.

Services Controlled by `init`

Any service controlled by `init`, the master process, is considered a standalone daemon. These will be respawned automatically if they crash. Such services tend to be locally-oriented and often non-TCP/IP related: for example, login managers like the GeTTYs that manage console-based logins are run by init. So is the X Window system, which runs in run level 5 (though it is network-aware). If the X Window server or a GeTTY process crashes, `init` will respawn it immediately.

You can change these services by editing the `/etc/inittab` file. If you'd like your changes to go immediately into effect, use the command `init q` to force `init` to recheck the `inittab` file.

System V Daemons

The System V daemons, which often have names that end in "d," are also considered standalone daemons, much like those launched by `init`. However System V daemons are are designed to run, or to *not* run, in *run levels* set by `init`. As described in the previous section, a run level is simply a numerical value from 0 to 9. When you change run level with the `init` command, the system sees which services should run in the new run level and starts or stops services accordingly.

The System V scripts, which are used to start, stop, restart or reload services, are kept in subdirectories of the `/etc/rc.d` directory. Each subdirectory represents the configuration of a run level; for example, a detailed listing of `/etc/rc.d/rc5.d` shows the services called when you switch to run level 5. The following file listing for the script for the X Font Server:

```
lrwxrwxrwx    1 root     root           13 Oct  1 16:50 S90xfs -> ../init.d/xfs
```

As you can see from the arrow at the end of the listing, this script is actually a symbolic link to the file `/etc/rc.d/init.d/xfs`. It is in the `init.d` directory that we find the actual System V scripts, and all of the script links in the run level directories point to `init.d` to start or stop services.

In the above example, the link to the master `xfs` script begins with the letter S and a two-digit code. The S indicates that the `xfs` script should be run with the "start" argument. The two digit number is the order in which it should process in relation to other "start" scripts.

Now let's take a look at a service that *stops* in run level 5.

```
lrwxrwxrwx    1 root     root           15 Oct  1 17:24 K15httpd -> ../init.d/httpd
```

This link to the Apache Web Server `httpd` script begins with the letter K for "kill." Whenever the runlevel changes, `init` launches the appropriate scripts as outlined in the file `/etc/rc.d/rc`. Basically this "master" run level script sends the `stop` argument to all links that begin with a K and then sends the `start` argument to all links that begin with an S.

`xinetd` Services

Finally there are xinetd services. These are started by the xinetd "master daemon," which is a System V service used to manage other services. While System V scripts run in the background all the time, an xinetd service will launch only when needed, at the bequest of xinetd. If you set up a POP3 mail server you'll have to configure xinetd to launch the POP3 daemon whenever it gets a connection on port 110.

xinetd does not allow connections blindly. Rather, it uses libwrap to examine incoming requests and compare them to entries in the hosts.allow and hosts.deny files located in /etc (this functionality, called TCP wrappers, will be discussed in the Security chapter later in this guide). The /etc/xinetd.conf file contains the default settings for all xinetd-managed services. The top of the file, with a little added notation, looks like this:

```
defaults
{
    # The maximum instances of the service that can exist at a time
       instances               = 60
    # The type of logging that should be done, in the format "SYSLOG facility level,"
    # so in this case the "authpriv" facility indicates that log entries should go to
    # the /var/log/secure log file, as configured in /etc/syslog.conf.
       log_type                = SYSLOG authpriv
    # What should be noted in the logs when a connection succeeds.  There are three
    # options, and they can all be listed sequentially.  DURATION logs how long the
    # session lasted, HOST logs the IP address from which the request came and PID
    # logs the process ID of the server.
       log_on_success          = HOST PID
    # What should be noted about failures, same options.
       log_on_failure          = HOST
    # Connections per second.  In this case, if over 25, stop handling requests for
    # 30 seconds.  Helps to avoid denial of service attacks.
       cps                     = 25 30
}

includedir /etc/xinetd.d
```

In this file you can also keep custom settings for specific services in separate stanzas, but as the last line above suggests, service-specific text files can be stored in the directory /etc/xinetd.d/. For example, here is /etc/xinetd.d/ipop3, which controls the options under which the POP3 mail service is launched. Again, notations are included to help decode the options.

```
service pop3
{
    # Characteristics of the service - stream, dgram and so forth.
       socket_type             = stream
    # Other characteristics, such as single or multi-threaded.
       wait                    = no
    # User account under which the service should run.
       user                    = root
    # Path of the server executable.
       server                  = /usr/sbin/ipop3d
    # Same options for successes and failures as those used above.
       log_on_success  += HOST DURATION
       log_on_failure  += HOST
    # Disabling lets you turn off a service without having to delete this file.
       disable                 = no
}
```

When a request comes in on port 110, xinetd uses the /etc/services file to determine that this is a POP3 request, and launches the /usr/sbin/ipop3d service under these options. By default most xinetd services have the disable option set to yes.

Let's look at these options more specifically:

(ToDo: options in xinetd)

Turning Services On or Off

GNU/Linux provides numerous utilities for controlling which services are running, and which one start by default. The two most important ones are `service` and `chkconfig`, which are each outlined below.

Other commands that can also prove useful are `serviceconf` and `ntsysv`. `serviceconf` provides a graphical utility for changing the default behaviors of standalone daemons in each run level. `ntsysv` does much the same, but as a console-based utility instead of a graphical one.

Service operations with the `service` command

The `service` command will start, stop, restart or display the status of any System V-launched service. For example.

```
# service httpd stop
Stopping httpd:                        [  OK  ]
# service httpd start
Starting httpd:                        [  OK  ]
# service httpd restart
Stopping httpd:                        [  OK  ]
Starting httpd:                        [  OK  ]
# service httpd status
httpd (pid 1574 1573 1572 1571 1570 1569 1568 1567 1564) is running...
```

Configuring Service Defaults with `chkconfig`

The `chkconfig` command lists or sets the default behavior for both System V and `xinetd`-managed daemons. `chkconfig` does not change the current state of any services; rather, it allows you to change whether services are configured as on or off.

The most basic use of the command is for a list of services configured to start on your system.

```
# chkconfig --list
```

This will list System V services by run level, and `xinetd` services as simply on or off. A good practice on a newly-installed system is to use this command and pipe it to `grep`, thus showing you only active services.

```
# chkconfig | grep on
```

Scanning through this list will give you a good idea of whether your system may be configured to start services you do not need. If it is, this is a waste of your computer's resources and possibly poses a security risk as well. You can use `chkconfig` to reconfigure these services to be inactive.

Using `chkconfig` on System V Services

For System V services, `chkconfig` creates symbolic links in the various runlevel directories, and by doing so sets the services that are started in each run level. For example, suppose you wanted to know whether your web server would automatically start `httpd` when you enter run level 3.

```
# chkconfig httpd --list
httpd      0:off   1:off   2:off    3:off    4:off    5:off    6:off
```

Using `chkconfig` with the `--list` argument reveals that the answer is no, `httpd` is off in all run levels. In other words, the links to `httpd`'s System V script in every run level's directory are set to stop the service, not start it. But you can change this with `chkconfig`:

```
# chkconfig httpd on
# chkconfig httpd --list
httpd      0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

Now the list shows that the service will start in runlevels 2, 3, 4, and 5 - though it will still stop in the other runlevels because "off" indicates that kill links are still present there.

How does it decide which run levels should run the service? It's doesn't choose them randomly: `chkconfig` knows by looking in the service's System V script for a special reference near the beginning. For the `lpd` service this looks like this:

```
chkconfig: 2345 60 60
```

This string tells `chkconfig` three things: first the run levels in which the service should be on, then the default start order, and finally default stop order.

Command guide: `chkconfig`

`chkconfig (service) <on off reset>`
Configures a service to be on or off, or resets its value to the default

`chkconfig --level X <on off reset>`
For System V services only, customizes a service for run level X.

`chkconfig (service) --list`
Lists service configuration.

`chkconfig (service) --add`
Adds a service to chkconfig control (creates links in every runlevel).

`chkconfig (service) --del`
Removes a service from chkconfig control (deletes links in every runlevel).

Once you have reconfigured a System V service you must use the `service` command to start or stop it. `chkconfig` does not stop or start System V services.

Using `chkconfig` on `xinetd` Services

`chkconfig` works a bit differently for `xinetd` services. By using the `on` and `off` arguments with various `xinetd` services, you add those services to `xinetd`'s list of services for which it should respond to incoming requests. chkconfig does this by simply changing the service's file in /etc/xinetd.d. In the service's file will be a line like this:

```
disable = no
```

chkconfig changes this value from no to yes, or vice-versa. This affects `xinetd`-managed services immediately; there is no need to restart `xinetd`.

Shutting Down or Rebooting

There are a number of commands used to shut the system down or reboot it. `halt` simply shuts down your system. `init 0` starts run level 0, which shuts down the system; `init 6` starts run level 6, or a reboot. `poweroff` and `reboot` do as their names suggest. But probably the most useful one is `shutdown`, which supports a number of indispensable options. The most common, to send all processes the termination signal and shut down immediately:

```
# shutdown -h now
```

Or, to reboot your system:

```
# shutdown -r now
```

Or to reboot in five minutes:

```
# shutdown -r +5
```

Or to wait until 11:59 PM and then reboot:

```
# shutdown -r 23:59
```

To cancel a shutdown, use the `shutdown` command with the -c option.

You can also reboot your system with the `Ctrl-Alt-Del` key combination, but only at a virtual console. While Fedora Core catches this keystroke and issues a reboot command, this can be customized in `/etc/inittab`.

While desktops and laptops have their own practicalities, you will rarely have to shut down a GNU/Linux server unless you need to do hardware work or upgrade the kernel or distribution.

10. TCP/IP Networking with GNU/Linux

Network Devices

Because of the wide variety of network devices available it is impractical to compile support for all of them into the Linux kernel, so Fedora Core and most other distributors offer them as modules so that Linux can adapt to many hardware environments. If networking is enabled in the `/etc/sysconfig/network` file, the appropriate drivers are loaded during boot and aliases to them are set in the `/etc/modules.conf` file. For example, the generic 3Com network driver `3c59x` may be aliased as `eth0`, creating a logical device name that is common to all GNU/Linux systems, regardless of the specific hardware used on each one.

The logical names of ethernet drivers are `eth0` for the first interface, `eth1` for the second, and so on.

To view the adapter's hardware address and other information use the `ifconfig` command. You can also use the `dmesg` command and the `/var/log/dmesg` file to see what hardware has been recognized by your system.

Address Resolution Protocol

Address Resolution Protocol (ARP) is a feature of TCP/IP that maps the MAC addresses of network cards to IP addresses. MAC address are generally built into network hardware, whereas IP addresses are assigned with software; hence the need to map between the two. Once an IP address for a computer on the local network is mapped, future data packets intended for that IP address can be sent directly to the MAC address.

The `arp` command can be used to view the ARP table with the `-a` option. It can also be used to add or delete entries manually.

`mii-tool`

The `mii-tool` command allows you to view information about the Media Independent Interface unit. This unit negotiates the speed and duplex of a network connection on a multi-speed device. So for example you could find out if a network card is running at 10 or 100 Mbps with the `-v` option (ed- add example), and force 100 Mbps with the `--force` option.

```
# ifdown eth0
# mii-tool -v --force 100BaseTX-FD eth0
# ifup eth0
```

The value "100BaseTX-FD" is 100Mbps at full duplex. To reset the speed and duplex through automatic negotiation by the card:

```
# ifdown eth0
# mii-tool -vr eth0
# ifup eth0
```

`ifconfig`

You can display or configure your network interfaces with `ifconfig`. It displays your interface type, hardware (MAC) address, statistics and resources. If you use the command on a system with a single ethernet adapter, and you don't include any arguments, it will display both the `eth0` adapter and an interface called `lo`, which is the local loopback. So to view only a specific adapter's information, specify it:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:20:AF:A2:BD:34
          inet addr:192.168.1.102  Bcast:192.168.1.255  Mask:255.255.255.0
```

```
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:20722 errors:0 dropped:0 overruns:0 frame:0
TX packets:17052 errors:0 dropped:0 overruns:0 carrier:0
collisions:756 txqueuelen:100
RX bytes:23470797 (22.3 Mb)  TX bytes:2091848 (1.9 Mb)
Interrupt:10 Base address:0x220
```

ifcalc

When setting your network adapter's settings using `ifconfig` or a GUI utility it can be difficult to determine your broadcast and netmask settings, particularly if your network uses classless routing. `ifcalc` is a tool designed to compute some of these these values if you already have others available.

ifup and ifdown

The command `ifup eth0` will activate your interface, and `ifdown eth0` will deactivate it. This is sometimes useful when you'd like to refresh your adapter's settings.

Configuration files

While some general networking configuration is kept in the file `/etc/sysconfig/network`, settings specific to adapters are each kept in their own adapter-specific file in the `/etc/sysconfig/network-scripts` directory. The files start with the characters `ifcfg-` and end with the interface name. As an example, my `ifcfg-eth0` file looks like this:

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp
```

This tells the system the device's name, activates it on boot and sets it to use DHCP to get its configuration dynamically. The DHCP configuration is then kept elsewhere.

If the system had a statically-assigned IP address it might look more like this:

```
DEVICE=eth0
IPADDR=192.168.1.100
NETMASK=255.255.255.0
BOOTPROTO=static
ONBOOT=yes
```

Network Configuration Utilities

Editing these text files and resetting adapters can be cumbersome. Because of this there are a number of utilities to help you set your configuration.

`netconfig` is a text-based tool, and is therefore the most versatile. It lets you set up your first ethernet adapter, or a different adapter with the "`--device` *`devicename`*" option. `netconfig` is most frequently accessed as a component of the `setup` utility, which also lets you set your firewall configuration, choose your mouse and keyboard type, and configure other services. It is also called by `kudzu` when a new network adapter is detected.

For a graphical network utility you can try Fedora Core's Network Administration Tool, `neat`, which also goes by the longer name of `system-config-network`. `neat` is a nice GUI interface that lets you configure multiple adapters. It is unique in that it creates a separate set of configuration files from the common GNU/Linux networking files at `/etc/sysconfig/network-scripts`. The additional files, located in `/etc/sysconfig/networking/profiles`, allow `neat` to remember multiple network profiles, which can be handy for moving a computer between different networks.

Global Network Settings

The /etc/sysconfig/network file contains global parameters for networking. You can turn networking on or off, set the hostname, define the global default gateway or join an NIS domain.

Working with multiple adapters

Many systems have multiple network cards. This is particularly true for systems that are used as firewalls or routers, but is also useful for providing redundancy on a dedicated server.

Adding a new network card is not very difficult once you've physically installed it; in fact, if you use kudzu on your system the card may simply configure itself when you reboot. But if you have to add a card manually, start by adding an alias line in /etc/modules.conf. For example, alias eth1 3c509 would set up a 3Com 3c509 series card as the second network adapter. Different types of network cards will require different strings for the final portion of the line; look for yours online or in Fedora Core's documentation.

Once you've established the card as a module you can create a configuration file for the interface in /etc/sysconfig/network-scripts, either by hand-editing it, copying another adapter's file and modifying it, or generating a file automatically by running neat or netconfig.

Binding multiple IP addresses on one adapter

It is often necessary to have more than one IP address assigned to a single network card. This isn't a necessity for mosts systems, but web servers that run SSL-encrypted secure sites have to use unique IP addresses. This causes conflicts when a single web server must host more than one SSL-encrypted web site.

You solution for binding multiple addresses it to create a file for each address. However a better way to go about it is to create a range file. For example, a range file named ifconfig-eth0-range0 containing this:

```
IPADDR_START=192.168.1.50
IPADDR_END=192.168.1.65
CLONENUM_START=0
```

would support the 16 IP addresses specified. You can have multiple files for many different ranges, but you can't bind a card outside a class C network block. This means that a maximum of 256 addresses are available for any given card. In the example above, many more ranges can be added in the 192.168.1 network block, but none in 192.168.2.

DHCP

DHCP is the Dynamic Host Control Protocol. It allows a network adapter to obtain its configuration from a central server automatically. This configuration can include the IP address default gateway, DNS servers, and other information. This configuration is kept in a "lease," allowing the DHCP server to gradually reclaim disused IP addresses.

Older Red Hat distributions used the dhcpd daemon to obtain DHCP configuration. This daemon kept its lease information in /etc/dhcpd/dhcpd-(interface).info. It kept one such file for each DHCP-enabled adapter.

As of Red Hat Linux 8.0 dhclient (from the dhcp RPM package) is now the default DHCP client. Like dhcpd, dhclient maintains one file for each adapter, however it stores them in /var/lib/dhcp. For example, in the file dhclient-eth0.leases you might find something like the following:

```
lease {
  interface "eth0";
  fixed-address 192.168.1.102;
  option subnet-mask 255.255.255.0;
```

```
  option routers 192.168.1.1;
  option dhcp-lease-time 86400;
  option dhcp-message-type 5;
  option domain-name-servers 63.240.76.19,204.127.198.19;
  option dhcp-server-identifier 192.168.1.1;
  option domain-name "ne2.client2.attbi.com";
  renew 3 2002/12/11 14:14:37;
  rebind 4 2002/12/12 01:05:26;
  expire 4 2002/12/12 04:05:26;
}
lease {
  interface "eth0";
  fixed-address 192.168.1.102;
  option subnet-mask 255.255.255.0;
  option dhcp-lease-time 86400;
  option routers 192.168.1.1;
  option dhcp-message-type 5;
  option dhcp-server-identifier 192.168.1.1;
  option domain-name-servers 63.240.76.19,204.127.198.19;
  option domain-name "ne2.client2.attbi.com";
  renew 5 2002/12/13 14:06:28;
  rebind 6 2002/12/14 00:34:00;
  expire 6 2002/12/14 03:34:00;
```

This outlines the two separate leases for the adapter `eth0`, both of which authorized the adapter to use the same IP address: 192.168.1.102.

`dhclient` is called called by the `ifup` and `ifdown` commands, which can be used to force a renewal of the lease.

End User Network Management

On a server it would be foolish to allow normal users to change a system's network configuration, so by default Fedora Core only allows this to be done by the superuser. But for home systems it can be useful to relax this policy a bit.

In the `/etc/syconfig/network-scripts/` directory edit the file `ifcfg-(interface)`, adding the line `USERCTL=yes`. Users can then run the `ifup` and `ifdown` commands, which call the suid-root program `usernetctl`.

Default route

If your system determines a host to be on a different network, it will send all traffic intended for that host to the default gateway to be processed. The default gateway is therefore generally a router, and it is responsible for routing packets beyond the local network.

Thee are two types of default gateways. A *global* default gateway is defined by the `GATEWAY=` line in `/etc/sysconfig/network`. If you have more than one network card you can still use a global default gateway, but you can also omit the global configuration line mentioned above and add a line to each `/etc/sysconfig/network-scripts/ifconfig-(interface)` file. This will set up a different rule for each adapter, which is useful if each adapter is on a different network. This will not work if a global route is specified.

Static IP Routing

The Linux kernel creates network routes for all connected networks, but sometimes it is useful to specify a non-intuitive route for network traffic. For example, a static route would be useful if you wanted to secure private e-mail by routing traffic between two mail servers on different networks over a private line rather than over the public Internet.

Static routes are set in `/etc/sysconfig/static-routes`. Entries follow this format:

```
<device> host|net <route command arguments>
```

For, for example, you could route all traffic to the server at 192.168.1.37 via your second network interface:

```
eth1 host 192.168.1.37 eth1
```

In this case the route is brought up along with the device. To have the device calculated at run time, use the word "any" in place of the first device name.

Tweaking the ARP Cache

Ideally packets are sent to hosts on the local network via their hardware (MAC) addresses. Systems translate between IP addresses and MAC addresses using the address resolution protocol (ARP), which keeps a cache of that can be manipulated with the `arp` command. Used without options `arp` simply prints the ARP cache.

IP Forwarding

IP forwarding is a feature required if you want to set up a GNU/Linux system as a router or firewall. Using network address translation (NAT) you can use a single public IP address for an entire private subnet. Since IP forwarding is the routing of packets from one network to another through your machine, it requires that you have two network adapters.

Forwarding is turned off by default, but it can be turned on without rebooting. Change the contents of the file `/proc/sys/net/ipv4/ip_forward` from `0` to `1`. For example:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

You could also use a text editor. This change will remain in effect until the system is rebooted. To make the change stick, use the `sysctl` utility (discussed later).

Name Resolution

To find out your hostname you can use the `hostname` command, which simply obtains it from the `/etc/sysconfig/network` file.

The `/etc/hosts` file contains the names of other hosts on your network, and allows you to resolve those hosts to IP addresses. The widespread use of hosts files was eliminated by the creation of DNS many years ago, but for small networks it is sometimes less cumbersome to create a common `/etc/hosts` file for all of your computers than to organize a DNS server. As configured in `/etc/rsswitch.conf`, the hosts file is checked before DNS by default.

A typical `/etc/hosts` file looks like this:

```
127.0.0.1    localhost.localdomain localhost
192.168.1.12 thiscomputer.domain.com
192.168.1.42 adams.domain.com
192.168.1.37 bogey.domain.com
```

By default an `/etc/hosts` file should have the equivalent of the first two lines: that is, the entries for the local loopback address 127.0.0.1 and the local hostname and IP address.

DNS Configuration

The DNS is responsible for mapping hostnames to IP addresses. Your DNS configuration is kept in `/etc/resolv.conf`. As an example of this file:

```
; generated by /sbin/dhclient-script
search domain.com
nameserver 63.240.76.19
```

```
nameserver 204.127.198.19
```

The first line (aside from the line with the comment) allows you to refer to local network hosts by their hostnames rather than by their fully-qualified domain names (FQDNs). The subsequent `nameserver` lines set your DNS servers, which are queried in the order listed.

DNS Troubleshooting

A number of utilities are available for troubleshooting and testing DNS. The `dig` command allows you to obtain the output of a variety of different name server queries. For example, to search for information about the domain called domain.com:

```
$ dig domain.com
```

To specify which nameserver you query:

```
$ dig @nameserver.domain.com domain.com
```

Following the domain name you can also specify a type to get more specific information: `A`, `MX`, `CNAME`, etc. One useful option is `axfr`, which is designed to facilitate a zone transfer to another nameserver. Adding this option to the end of the command gives a complete listing of the specified nameserver's zone file for the specified domain.

The `host` command is a non-interactive utility for performing DNS lookups - generally, to convert names to IP addresses and vice versa. To get an IP address:

```
$ host example1.domain.com
example1.domain.com has address 192.168.1.33
$ host 192.168.1.33
192.168.1.33.in-addr.arpa domain name pointer example1.domain.com.
```

Network Diagnostics

A few commands are available for general networking diagnostics.

`ping` sends ICMP packets to a host specified in the command, and the remote host will reply if the network is functioning properly. By default `ping` sends 64-byte packets, and it will continue to do so until the user cancels the command with `Ctrl-C`. For example:

```
[eholden@localhost eholden]$ ping example1.domain.com
PING www.google.com (192.168.1.33) from 192.168.1.102 : 56(84) bytes of data.
64 bytes from www.google.com (192.168.1.33): icmp_seq=1 ttl=44 time=90.0 ms
64 bytes from www.google.com (192.168.1.33): icmp_seq=2 ttl=44 time=89.5 ms
64 bytes from www.google.com (192.168.1.33): icmp_seq=3 ttl=44 time=89.0 ms
64 bytes from www.google.com (192.168.1.33): icmp_seq=4 ttl=44 time=91.0 ms

--- example1.domain.com ping statistics ---
4 packets transmitted, 4 received, 0% loss, time 3009ms
rtt min/avg/max/mdev = 89.056/89.912/91.049/0.739 m
```

The `traceroute` command attempts to display the route between hosts, and is generally used between hosts separated by one or more routers.

```
$ traceroute www.domain.com
```

Finally, `netstat` is a multi-purpose tool designed to print network connections, routing tables, interface statistics, masquerade connections, and multicast membership. `netstat -r` is the same as the route command, and it prints the Linux kernel routing table. Another good implementation is `netstat -anA inet`, which will display statistics on all active network connections.

11. User and Group Management

Setting up users and groups with limited privileges is the most basic ingredient in a good security policy. On home systems this is optional; however in a corporate environment it is imperative that users be limited in what they can do on a multiuser machine.

It is also important that users protect access to their accounts with strong passwords. In GNU/Linux, user accounts are stored in the `/etc/passwd` file and passwords are kept in `/etc/shadow`.

The Password File

The Password file at `/etc/passwd` contains user account information. Each user has his or her own line; a typical line looks like this:

```
wsmith:x:500:500:winston smith,Lars Homestead,x2822,,:/home/wsmith:/bin/bash
```

This string of text contains the following fields, delimited by colons:

- **Username**.
- **Password**, encoded on old systems but represented by an "x" on newer systems where shadow passwords are enabled (see `/etc/shadow`, below).
- **User ID** (UID) - a unique security ID for the user. A common convention, and one followed in Fedora Core, is for UIDs to begin numbering at 500, with lower UIDs used by system accounts. Any user account that has a UID of 0 is a superuser, and has full administrative privileges on the system. Generally only the account named root has this UID.
- **Group ID** (GID) - The default group to which the user belongs. This is usually the user's private group, which has a name identical to the username. It maps to a group name in the `/etc/group` file.
- **Comment** - Optional information about the user, with fields separated by commas.
- **Home Directory** - This is the user's personal space, where all personal configuration files are sent by default, and into which the user is deposited upon logging in.
- **Shell** - On most Linux distributions this is usually the Bourne Again Shell (/bin/bash). Other options include the C shell (/bin/csh or /bin/tcsh), the Bourne shell (/bin/sh) or the Korn shell (/bin/ksh). To prevent a user from accessing a shell this can be set to /bin/false.

The password file has the permissions 644, or rw-r--r--. It is readable by anyone who has access to the system, but only writable by root. This allows system services to access it.

The Shadow File

It is recommended that any GNU/Linux system be set up to use Shadow passwords. This places the encoded passwords of users listed in `/etc/passwd` in a file called `/etc/shadow`. Unlike the Password file, the Shadow file is readable only by root, and users are authenticated by a password authentication system. A typical shadow entry looks like this:

```
wsmith:$1$vWJqqSzw$nYdxmvcOUBN8owar7cB910:11856::::::
```

This string of text contains the following fields, delimited by colons:

- **Username** - Maps to a username in `/etc/passwd`.
- **Encoded password**
- **Last password change date** - Measured in days since January 1, 1970. Set this to zero if you'd like to force a user to change his or her password at next login.
- **Minimum days value** - How long the user must keep a new password before it can be changed again, which prevents a user from changing a password and then changing it back to the old value. A value of zero disables this feature.
- **Maximum days value** - The number of days that a user is allowed to keep the same password, and is

often set to a very large value to effectively remove the limit.
- **Warn days value** - How many days in advance the user will be notified of an upcoming password expiration.
- **Inactive days value** - The number of days after the password expires that the account will be automatically disabled if the password hasn't changed.
- **Expiry date** - The date on which the account expires and is automatically disabled, regardless of the password status.

The Group file

`/etc/group` contains information about security groups. A typical group entry looks like this:

```
wsmith::500:wsmith
```

This string of text contains the following fields, delimited by colons:

- **Group name** - This can map directly to a username if the group is a personal group, or can be something new, such as a department name.
- **Password** - Not used.
- **Group ID** (GID) - Must be unique. For personal groups this will map to the GID in a user account.
- **Users** - As many as you want, separated by commas.

Adding, Changing and Deleting users

Users can be added by editing the various security files discussed above, however, simple commands are also available to automate user maintenance. To add new users, use the **`useradd`** command:

```
# useradd wsmith
```

This can be enhanced by adding arguments that will populate the appropriate fields. This example specifies the UID (and if available the corresponding GID), adds a comment and assigns a shell:

```
# useradd -u 510 -c "winston smith,Lars Homestead,x2822,," -s /bin/bash wsmith
```

To change a user's settings, use the **`usermod`** command. This changes the user's shell to the C shell:

```
# usermod -s /bin/tcsh wsmith
```

To remove a user, run **`userdel`**. If used with the `-r` option the user's home directory and mail spool will be removed along with his or her security information.

Passwords

When setting up your system it is generally a good idea to use MD5 passwords, which allow passwords up to 256 characters in length. Without this option your passwords cannot exceed eight characters.

To set a user's password you can use the **`passwd`** command:

```
# passwd wsmith
Changing password for user wsmith.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

By default password never expire, but forcing users to change passwords periodically is a good security policy. The password aging settings are stored in `/etc/shadow` in the fields after the encoded password, but you can use the **`chage`** command to automate aging. Useful options:

-m	minimum days between password changes.
-M	maximum days between password changes.

-I	number of days inactive since password expired before locking account.
-E (date)	expiration date.
-W	number of days before a password expires to warn the user.

Authentication Options

Your system can authenticate in more ways than solely from the built-in security files, and can itself be an authentication authority for a network. You can set up a variety of systems during installation, or later on by using the `authconfig` utility (also part of the `setup` utility) Common options:

- **NIS** - Network Information Services.
- **LDAP** - Leightweight Directory Access Protocol.
- **Kerberos** - An MIT authentication standard.
- **Hesiod** -
- **SMB** - Now called CIFS, this is the Windows networking protocol.

These schemes are centralized, and allow you to set up one or more authoritative servers for authenticating users.

Group Administration

Groups can be useful in setting up security policies for a group of users with similar needs. Groups are stored in the `/etc/group` file.

To add a group, use the `groupadd` command. To remove it, use `groupdel`. To change the name of a group or change its group ID (GID), use the `groupmod` command. It is also easy to modify the Group file manually.

When using Shadow password the group passwords are kept in the `/etc/gshadow` file.

Switching user accounts

The `su` command, or "switch user," is used to change to another user account. Unless you are the superuser, you will have to enter the password of the account to which you are switching. Used with a - (dash) symbol, the command will also switch to the target user's environment variables, such as the command path and default directory. For example:

```
[wsmith@localhost wsmith]$ su - lorgana
Password:
[lorgana@localhost lorgana]$
```

Note that the directory also changed; omit the dash and you will remain in the original directory.

`su` is most frequently used by system administrators who want to switch to the root account temporarily. In fact, when used without a username as in "`su -`" the command will switch to the superuser.

File ownership

When file is created it becomes owned by the user who creates it, as well as by that owner's default group as defined in the `/etc/passwd` file.

To see the ownership of a file use the `ls` command with the `-l` argument to list the file in long form:

```
$ ls -l
-rw-rw-r--    1 wsmith wsmith         1583 Nov 19 22:54 textfile.txt
-rw-rw-r--    1 wsmith wsmith         1583 Nov 19 22:54 imagefile.png
-rw-rw-r--    1 wsmith wsmith         1583 Nov 19 22:54 soundfile.ogg
```

The opening string of dashes and letters are file permissions, which are discussed below. The two names

listed after the permissions are the user and group respectively, so all three files in the user wsmith's data directory are owned by that user and the user's group.

The superuser can change the ownership of a file with the `chown` command. For example, if the superuser creates a `.forward` file to forward incoming mail for a user, the file can then be given to that user (actually necessary for mail forwarding):

```
# touch .forward
# chown wsmith .forward
```

Of course the superuser should probably have changed both the user and the group, and both can be done simultaneously:

```
# chown lskywaker.wsmith .forward
```

Similarly you can use the `chgrp` command to change only the owning group.

Sometimes it is useful to have the files created by one user be accessible to an entire group - for example, in a shared directory on a file server users might want to access each others files for a group projects. In that case you can use an SGID bit, an enhanced permission discussed later in this guide.

File permissions

Every file has permissions built into it. These permissions are divided into three *access levels*: those of the owner ("user"), the owning group ("group") and the rest of the world ("other"), generally abbreviated "u", "g" and "o". Additionally there are three *access modes* of access for files: read, write and execute - abbreviated "r", "w" and "x". The combination of access levels and access modes is called the *file mode*.

For normal files the effects of the file mode are pretty straightforward. Let's look at an example by calling the `ls -l` command.

```
$ ls -l /etc/passwd
-rw-r--r--    1 root     root         1583 Nov 19 22:54 passwd
```

At the beginning of the output line is a series of ten characters, the first of which is occupied by a dash. This means that `/etc/passwd` is a regular file (this space can be occupied by other characters if the file is of another type, and we'll examine this more later). The other nine spaces are related to the file mode. To decipher the permissions, try comparing them to full permissions:

	User / Group / Other
In this example:	r w - r - - r - -
Full permissions:	r w x r w x r w x

So the user who owns the file, in this case root, has read and write permissions (rw), and both the group and all other users have read-only permissions (r). This makes sense, as the `/etc/passwd` file contains user account information and should only be changed by the superuser.

Permissions are applied to a user using the following steps:

- First, if a user trying to access a given file is the file's owner, then the user-access level applies. The rest of the permissions are *completely ignored*.
- If the user is not the owner, but *is* a member of the owning group, then group-level access applies and the rest of the permissions are ignored.
- If the user is neither the owner nor a member of the group, other-level access applies and user and group permissions are ignored.

By this scheme if a file has the permissions `---r-xr-x`, the owner of the file cannot read it even if he or she is a member of the owning group, because user-level access takes precedence over group-level access.

Directory permissions

A directory is just a file that contains records of the sub-files it contains. If you list one you'll see that its first character is a d, indicating that it is a directory:

```
drwxr-xr-x   60 root     root         4096 Dec  8 16:33 etc/
```

Because directories are files and contain information about other files, the permissions on a directory can have unusual effects that users should take the time to understand. The effects actually make sense when you consider that a directory is not a "container," just a file.

Read permissions on a directory allow you to read the directory. This allows you to use the `ls` command to view the directory's contents.

Write permissions on directories can be very dangerous and are important to understand. They allow a user to write to the directory file itself - that is, to add or remove file entries. So if a directory has world-writable permissions *any* user can delete *any* file in it, even if someone else owns the file. Use directory write permissions as sparingly as possible.

Execute permissions are more benign. They allow a user to change to a directory with the `cd` command, making it the present working directory.

Adding permissions

To add permissions use the `chmod` command. You can use the u, g and o abbreviations for the file's user-level and the r, w and x abbreviations for the access mode. The + and - characters add or subtract permissions. So if you have written a shell script in a shared directory and you want to let anyone execute it, you would use change the mode with the o+w argument.

```
$ ls -l script.sh
-rwxr--r--    1 wsmith wsmith      1520 Nov 11 21:54 script.sh
$ chmod o+w script.sh
$ ls -l script.sh
-rwxr--r-x    1 wsmith wsmith      1520 Nov 11 21:54 script.sh
```

Numerical permissions

To speed up file mode changes UNIX systems recognize numeric permissions. Like alphabetical permissions, numerical permissions follow the order "user-group-other," however they use numbers for each access mode: 4 for read, 2 for write and 1 for execute. Adding the numbers together for each user level gives you the file mode. Full-access is granted by the number 7, which is 4+2+1. 4 is read-only addess. Other examples:

```
$ chmod 777 textfile.txt
$ ls -l textfile.txt
-rwxrrwxrwx    1 wsmith wsmith      1520 Nov 11 21:54 textfile.txt

$ chmod 644 textfile.txt
$ ls -l textfile.txt
-rw-r--xr--    1 wsmith wsmith      1520 Nov 11 21:54 textfile.txt
```

So if you remember the numbers 4-2-1, numerical permissions will save you a lot of time.

Executables with s-bits: the SUID and SGID

When you start a process, such as the PINE e-mail program or `vi` text editor, that process runs with your user ID. Some processes, however, may require enhanced permissions that are unavailable to a normal user. A good example is the `passwd` command, which changes a user's password. This command can change entries in the `/etc/passwd` and/or `/etc/shadow` file, which are owned by root and unwritable by other users. For this reason the `passwd` command must run as the superuser. Let's take a

look at the command's permissions to find out how; it's located in `/usr/bin`.

```
$ ls -l /usr/bin/passwd
-r-s--x--x    1 root     root        15368 May 28  2002 passwd
```

Note that in the user access level the execute bit is hidden by the a lowercase s. This is an SUID bit, which allows the user who launches the command to run as root. And SGID bit allows the command to be launched as a member of the owning group.

When doing a list of the file, an s-bit will be capitalized if the execute bit is off. If the execute bit is in the s-bit will be lowercase (as above) to indicate that though the x is not visible, it is still there.

To add an s-bit, use the abbreviation s:

```
# chmod u+s filename
```

Only the superuser can grant or revoke an s-bit.

Other security implications of s-bits

In the above example the `passwd` command has security features that prevent it from being used to make arbitrary changes to the system's security files. It is aware of the fact that all users (except the superuser) should be allowed to change his or her security information *only*. Most programs lack this awareness and rely on basic system security to prevent malicious use. Such programs can be dangerous with an SUID or SGID bit; for example, if the `vi` editor had an SUID bit, a user could use it to edit any file on the system because the `vi` command would run as root. For this reason s-bits should be avoided on normal files unless there is a compelling reason to use them.

On directories there are practical, everyday reasons to use s-bits. An s-bit on a directory has an effect on the ownership of files created in that directory. For example, let's say a shared directory used by members of a group called "accounting" has its SGID bit set. Since the directory's owning group is "accounting," the SGID bit makes all files created in the directory owned by the accounting group. The files are still owned by the users that created them, but the group can freely share them. For this reason SGID bits are used frequently on UNIX file servers.

Sticky bits

As mentioned above, in a normal directory with write-permissions for either the group-level or other-level, an user with access to the directory can delete any file in that directory, even if it is owned by someone else, because deleting a file is simply a write to the directory file.

A sticky bit is a precaution that allows files to be created in the directory, but prevents deletions. A good example of a common use for the sticky bit is the `/tmp` directory:

```
drwxrwxrwt   11 root     root         4096 Dec  8 17:37 tmp/
```

So any user on the system can create files in this directory - a requirement because any user can run processes that might create temporary files - but only the owner of the temporary file can delete it, preventing users from sabotaging each other's processes.

To add a sticky bit, use a "t" abbreviation in the `chmod` command:

```
# chmod o+t /export/shared/important-files
```

The same rules of capitalization apply to the t/T as to the s/S, so in the above example the execute bit is present for the `/tmp` directory.

Numerical s-bits

To change s-bits in file permissions you don't have to rely on abbreviations like u+s, g-s, etc. You can expand on the numerical permissions outlined above. In a four-digit number, the first digit is used to change the s-bits of the file. 4 adds an SUID bit for the user access-level, 2 adds an SGID bit to group, and 1 adds a sticky bit to other. So the number 7777 leads to permissions of `rwsrwsrwt`. The number 2770 gives you an ideal shared data directory: `rwxrws---`.

The `umask` variable

By default, all files are created with the permissions 666, allowing all users to read and write to the file unless the permissions are specifically changed. Similarly, directories are created with 777 permissions. In practice this is rarely a useful set of permissions, so the `umask` variable changes this default. Fedora Core's default `umask` is 002, which subtracts 2 (write) from the "other" access-level. This results in permissions of 664 for files and 775 for directories.

The `umask` is set in the `/etc/bashrc` script, which runs at startup as an integrated part of most other scripts (ed - **check this** ... and where are other variables kept?). You can also change your `umask` with the `umask` command.

```
$ umask 022
```

The above example results in default file permissions of 644 for files and 755 for directories.

Extended Filesystem Attributes

If you use the ext2 or ext3 filesystem you can take advantage of special file attributes that go beyond the basic UNIX permissions. You can set them with the `chattr` command, whose syntax looks like this:

```
chattr (+ or - or =)attribute file
```

For any file or directory, try the following attributes:

- `A` When the file is modified, the *atime* (last access time) will not change.
- `a` The file may only be opened in append mode, and not deleted (set by superuser only).
- `c` Stores the file in compressed mode, and uncompresses the file when accessed.
- `d` The `dump` command will skip the file when backing up.
- `i` The file is immutable and can't be changed, deleted or renamed (set by superuser only).
- `j` In an ext3 filesystem the data and meta data will be recorded in the journal even if the journal mode is set to `ordered` or `writeback`.
- `S` Changes to the file are written synchronously to the filesystem.
- `s` When the file is deleted the system deletes it securely by overwriting it with zeros.
- `u` When the file is deleted its contents are saved, allowing for an undelete.

So if you wanted to protect a log file form having lines deleted from it, but still allow new lines to be added, you could do this:

```
# chattr +a /var/log/secure
```

To check on attributes, use the `lsattr` command.

Private Groups

Whenever a new user is created on a system he or she is added to a new group of the same name. This is called the user's *private group*. A private group is useful because it allows the user's files to be owned by a group that has only one member - the user. The file does not have to be given to a common group containing many users.

While this is a secure way of dealing with group permissions, it has a user-behavior drawback: that is,

users will often give files or directories world-accessibility when they want to share them, rather than simply changing the group ownership to a common group.

The "Skeleton" Directory Template

The `/etc/skel` directory is a "skeleton" directory. It is a template for the home directories of all new users, so changes to the contents of `/etc/skel` will affect what files new users see when they log in. This includes files that contain environment variables and command aliases for that user, but it also includes normal files that do not affect the environment. For example, if you wanted to give new users a guide to the `vi` text editor you could put it in a text file in the `/etc/skel` directory and it would end up in all newly-created home directories.

Another use for skeleton directory is to give different resources to different types of user. Using the `useradd` command with the `-k` option lets you select an alternate directory - perhaps one containing instructions for using the `pico` editor rather than `vi`.

User Environment

Every time a user logs in a series of scripts are called to set up the user's environment. The first is `/etc/profile`. This file sets basic variables like the HISTSIZE, which is how many commands the Bash history will remember (1000 by Fedora Core's default); MAIL, which is the location of the user's incoming mail spool file. These are system-wide variables, and when changed affect all users. The `/etc/profile` script also runs scripts in the `/etc/profile.d` directory.

After system-wide profile scripts run, the user's local `.bash_profile` file runs. This script runs basic environment and startup functions, and also calls the `.bashrc` file.

.bashrc, located in each user's home directory, contains user-specific variables and aliases. So if you wanted to use the single-character command `l` as a quick alias for a more tedious command like "`ls -alp | more`" you can put an alias for it in your `~/.bashrc` file, like this:

```
alias l="ls -lap | more"
```

The next time you log in this alias will be available. You can also make your changes take place immediately by typing `source .bashrc`. Note that this file is not a list of variables and aliases - it is a script that is run whenever you open an interactive shell, and it is run by `~/.bash_profile` whenever you first log in. `alias` is just a command, and it can be run on its own without the scripts.

By default the `.bashrc` script calls the global file `/etc/bashrc`, which contains system-wide aliases and functions. So if the system administrator wants to make the above alias available to all users it can be placed in `/etc/bashrc`.

Filesystem Quotas

Quotas are a service of the Linux kernel, and they can be used to limit a user or group's disk usage space. They operate on a per-filesystem basis, so you must manage different quota systems for different partitions. This can actually be useful if you want different rules for different partitions. For example an IMAP mail server could limit `/var` to 50 MB per user, which would limit the size of each user's inbox, while limiting `/home` to 200 MB for other IMAP mail folders. Fedora Core supplies a `quota` RPM that contains many utilities to help administer quotas.

Preparing the filesystem

You should mount any filesystem on which you want to maintain quotas with the `usrquota` and `grpquota` options enabled. This can be done in `/etc/fstab`. For example, if you wanted to control mail spool sizes on a POP mail server, you might enable quotas on the `/var` partition. This would affect

files in /var/spool/mail, where inboxes are stored. Since you care only about individual users we'll omit the grpquota optionfor this example.

The edited line in /etc/fstab would look like this:

```
LABEL=/var     /var     ext3     usrquota  1  2
```

Then you should remount the filesystem so the setting takes effect.

```
# mount -o remount /var
```

Quota initialization

In the topmost directory of the partition, in this case /var, you must create a database for your user quota called aquota.user (if you wanted to do group quotas as well the group equivalent would be named aquota.group). This file contains information about the filesystem and its usage.

This is where the quotacheck command comes in. quotacheck is mainly for refreshing the database if it becomes out of sync with the filesystem, such as after a system crash, but with the -c option it will ignore the absence of a database and create a brand new one.

Ideally you'll want an unchanging (and therefore read-only) filesystem, so try to pick a time when the filesystem is not being used to run the command. But if you can't find a time when it isn't being used you can force the check even while data is being written by adding the -m option, like this:

```
# quotacheck -cm /var
```

You can use quotacheck again if you ever want to recreate the database from scratch.

Enabling or disabling quotas

You can turn quotas on or off with the quotaon and quotaoff commands.

```
# quotaoff /var
# quotaon /var
```

If you use quotas on multiple partitions you can use quotaon or quotaoff with the -a option to enable or disable quotas on all filesystems in /etc/fstab for which quotas are enabled.

Editing quotas for users

To set quotas for a user, use the edquota command.

```
# edquota wsmith
```

The edquota command uses your default editor as a text interface for setting options, like this:

```
Disk quotas for user wsmith (uid 500):
  Filesystem       blocks       soft       hard     inodes     soft     hard
  /dev/hda5       7424024          0          0      23063        0        0
```

After the partition number are six fields; the first three are for the number of blocks the user can use and the second three are for the number of inodes. It's generally much easier to manage a block usage and leave the inode quotas set to 0 (no quota) because blocks are generally 1 KB in size and can be easily translated into human-comprehensible storage limitations.[1] A user with a limit of 10,240 blocks can store

1 Inode quotas are more useful for limiting the number of *files* a user can have, regardless of the size of the files. On some filesystems where a lot of inodes are required this might be useful to prevent the system from running out of inodes. For example if you store e-mail for hundreds of users in Maildir format an Inode quota will limit the number of directories each e-mail message will occupy its

up to 10 MB of data.

The number of blocks or inodes listed is the number currently in use, assuming the database is up-to-date. The *soft* limit is the limit at which a warning will be given to the user that they are coming close to reaching their maximum alloted storage. The *hard* limit is the limit at which the user will be prevented from adding any more data (in the case of blocks) or files (in the case of inodes).

After setting a user's quota and saving the database it will be implemented immediately.

Saving yourself a lot of time with templates

Because the `edquota` command must be performed for each individual user it can be helpful to establish a template for disk quotas. So once you've created settings for your first user you can use that user as a prototype for other users with the `-p` option.

```
# edquota -p wsmith hsolo
```

This sets the new user `hsolo` to have the same quota policy as `wsmith`. If you have hundreds or thousands of users it's probably worth scripting this so that the command can be applied to everyone with little effort.

Sending warnings

If users work on the system where the quotas are maintained, such as on a terminal, then a warning will appear on the user's terminal automatically if a soft or hard quota is reached. For hard quotas this warning will be accompanied by a sudden inability to save data.

But not everyone has terminal access. On a mail server or file server users will need warnings sent to them via e-mail, so you can accomplish this with the `warnquota` command. `warnquota` sends warnings to *all* users who have exceeded their soft or hard limits, not just the culprits of your choice, so it is useful to run it as a `cron` job every minute, or perhaps every few minutes.

In such a configuration you may want the soft and hard limits to be far enough apart that the user will get the soft limit warning *before* his or her hard limit is reached. Particularly in the case of a mail server, the soft quota warning should be sent before the hard quota is reached - otherwise the account won't be able to receive the warning message.

If you get this working you might find that the e-mail warnings are a bit cryptic, at least to the average user. A mail server user may not know that `/dev/hdb2` is the `/var` partition, nor what a `/var` partition is anyway. So the `warnquota` command's output can be customized in `/etc/warnquota.conf`, where you can change the e-mail text to contain information that might be more valuable to users, like "you have too much mail and should delete some of it or risk not being able to receive any more."

Quota reporting

Users may want to occasionally inspect their use of storage to ensure that they are not consuming to much of it. This is a rarity, since only experienced users tend to understand that data takes up space, and that *more* data takes up *more* space. But for those experienced users there is the `quota` command, which will report on quotas and storage use on a filesystem.

```
$ quota
Disk quotas for user wsmith (uid 500):
  Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
    /dev/hda5 7425000* 7424065 7425000           23075       0       0
```

For the administrator there is the `repquota` command, which outputs information about users with

own file, and therefore its own inode. In such a scenario it might be advisable to limit inode use, but in this example we'll manage a user's block use.

quotas, even those who have not yet reached their quotas.

```
# repquota /home
*** Report for user quotas on device /dev/hda5
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root       --      20       0       0              2     0     0
wsmith     -- 7424064 7424065 7425000          23074     0     0
hsolo      --    3420       0       0            452     0     0
```

Only the superuser can use the `repquota` command because a normal user cannot view the quota or another user.

12. Using NIS for Authentication

While a home user can survive using local authentication - that is, user accounts stored locally on his or her machine - environments with many different computers often need a more robust and centralized authentication system. There are two commonly-used network authentication methods for GNU/Linux: NIS and LDAP. Both allow centralized administration of users and groups. Of the two, NIS is by far the easier to configure. For this reason LDAP has its own chapter, and this chapter is dedicated solely to setting up an NIS server.

Authentication With NIS

Network Information Systems (NIS) was developed by Sun Microsystems for its Solaris operating systems. Its original name was Yellow Pages, so it's sometimes called YP. However British Telecom owns the trademark on the name Yellow Pages in the United Kingdom so the system was renamed NIS. Nonetheless the daemons and clients still use the initials YP.

The NIS daemon serves as a central database of user information, providing the same information that the `passwd`, `shadow`, `group` and `gshadow` files serve on a standalone GNU/Linux system. Client machines "bind" to the NIS server and use this database to authenticate. The collection of the NIS clients and the server is called a domain.

An NIS domain should not be confused with a Windows domain, in which a Windows or Samba domain controller performs authentication for the user (and sometimes for the machine). An NIS server provides its database of all users to the client, and the client uses the database to perform its own authentication.

Limitations

NIS is unfortunately very insecure. It transfers the user list over the network, so an insecure network would allow a malicious person to sniff out these passwords. And users on NIS clients can view the entire user database with the `ypcat` utility. In either case the passwords are encrypted, but a password cracking program can eventually decrypt any UNIX password. In short, NIS is not nearly as secure as keeping passwords in the `/etc/shadow` file, which is readable only by the superuser.

The NIS Server

The NIS daemon is called `ypserv`. It runs using `portmap`, the RPC (remote procedure call) daemon that assigns TCP/IP ports dynamically. `portmap` must be running in order to run the NIS server, and you can use the `rpcinfo` command to manually query an NIS server. On the local machine you'd do this:

```
$ rpcinfo -p 127.0.0.1
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100004    2   udp    956  ypserv
    100004    1   udp    956  ypserv
    100004    2   tcp    959  ypserv
    100004    1   tcp    959  ypserv
```

Each service running via RPC will be listed with a program ID, version number, protocol (TCP or UDP), port number and name. You can get the status of a given program.

```
$ rpcinfo -u 127.0.0.1 100004 2
program 100004 version 2 ready and waiting
```

The NIS Client

The NIS client is `ypbind`. To configure your system as an NIS client the easiest option is to use

`authconfig`, part of the `setup` program.

Make sure that the `/etc/nsswitch` file is correctly configured with the lookup order, that it can broadcast a request for its domain's server or get the server's hostname or IP address from `/etc/yp.conf`.

Client Tools

Once you are bound to the domain, there are a number of tools you can use to query the NIS server.

- `ypcat` - lets you print out an NIS database. For example, `ypcat passwd` outputs the password database.
- `ypchfn` - Change NIS finger information (like the `chfn` command).
- `ypchsh` - Change NIS user's shell (like the `chsh` command).
- `yppasswd` - Change NIS user's password.
- `ypwhich` - identifies which master or slave server is being queried.

The `/etc/nsswitch` file

The `/etc/nsswitch` file should be configured to look for certain essential services via NIS rather than via local files. Generally these services perform a lookup in this order:

```
files nisplus nis
```

Other options include `ldap`, `dns` and `db`. You may want to tweak your `nsswitch` file appropriately. In the above configuration a local user named `wsmith` would take precedence over an NIS user of the same name, which could lead to conflicts if you have local user accounts. You can edit this file directly, or use the `authconfig` utility.

NIS Servers

NIS uses a flat namespace, unlike LDAP and DNS. There is no sub-domain concept, and a slave server has exactly the same information as its master.

Slave servers do nothing but mirror the data in the master. Their purpose is to provide load balancing and/or redundancy.

Configuring an NIS Master Server

To run an NIS server, make sure that the `ypserv` and `yppasswd` daemons are installed and configgred to run by default. Both are System V services. Edit the `/etc/sysconfig/network` file to include this line:

```
NISDOMAIN=domainname
```

Your domain name does not have to be the same as your DNS domain name; in fact it's a good idea to keep them different for security, because any client that knows your NIS domain name can bind to it. It also helps you to keep the two domains separate in your head.

Edit `/var/yp/securenets` to include only the networks you want the NIS server to acknowledge.

Edit `/var/yp/Makefile` to set these variables:

```
NOPUSH=true
MERGE_GROUP=false
```

`NOPUSH` should be set to `false` if you plan to have one or more slave servers. `MERGE_GROUP` should

be true if you have group passwords in a `gshadow` file.

Then, while still in `/var/yp`, run the `make` command like this:

```
# make passwd shadow group
```

This assumes that you are using NIS only for authentication. You can also add many other options to share with NIS clients, such as host, which replaces the `/etc/host` file (and could substitute for local DNS).

Make sure you start the `ypserv` and `yppasswd` daemons. You can then update the NIS database with the contents of the local `passwd`, `group` and `shadow` files whenever you'd like with this command:

```
# make -C /var/yp
```

NIS passwords are safe when you preform these updates because changes to NIS passwords update the local shadow file, keeping both in sync.

Configuring an NIS Slave Server

The first thing you need to do if you're implementing an NIS slave is to add its name to the master server's list at `/var/yp/ypservers`. You might have to create this file. The syntax is simple: just list your slave servers.

Now you must configure your master server to recognize this file. By default the Makefile in `/var/yp/` is set up to generate a master server with no slaves. You should open the Makefile and find the NOPUSH line, changing it to look like this:

```
NOPUSH=false
```

Now it will push, so the master server will send its databases to the slave..

Slave Server Settings

To accept pushes from the master server you'll have to configure `portmap` to allow connections from the it. `portmap` uses TCP wrappers, which can be configured by adding entries in the `hosts.allow` and `hosts.deny` files. While the uses of TCP wrappers can't be discussed in depth here, this is the basic syntax you probably want in the `/etc/hosts.allow` file:

```
#  allow access to portmap
portmap:     master-name

#  allow access to ypserv
ypserv:      master-name
```

Once this setting is functioning on the slave server you should install `ypserv` and issue this command:

```
# /usr/lib/yp/ypinit -s master-name
```

Substitute the name of your master server appropriately. In the output of this command you'll probably see several messages that say the following:

```
Trying ypxfrd...not running
```

This happens because the slave server can't force the master to send it the current maps. You haven't configured the master server to do this, so that's not surprising. If you want the master to deliver its maps on demand you need it to run the `rpc.ypxfrd` daemon. This not always necessary, since we're using the push method instead. Whenever an update is performed the master will send its maps to the NIS slaves using the `yppush` program.

If you want your NIS slave to download the maps, which might be useful if your master ever happens to push the most recent maps at a time when the slave is down, you need to start `rpc.ypxfrd` on the master server and configure it to run all the time, like this:

```
# service ypxfrd startup
# chkconfig ypxfrd on
```

You can then use `cron` to schedule your slave server to run a transfer once per hour, once per day or twice per day using scripts found in the directory `/usr/lib/yp/`.

The final step is to make your slave server use itself as an NIS server. Edit `/etc/yp.conf` on the slave and set it to use 127.0.0.1 for NIS lookups. You can leave the existing server there as well; just add a new line like this:

```
# NIS Server configuration in /etc/yp.conf
domain domain-name server master-name
domain domain-name server 127.0.0.1
```

So your master will be the default server, but if it is absent the slave will use itself.

Troubelshooting

An NIS slave server can be hard to get working right. For one thing, the master server's name must be identical to that found in the output from the command `ypwhich -m`. It's important to keep this consistent in your various commands and configuration files.

Also, in some cases it helps to customize the `/usr/lib/yp/ypinit` script. In keeping with the above suggestion, sometimes it is important to switch these two lines in the script:

```
#  maps=`ypwhich -m | egrep $MASTER$| awk '{ printf("%s ",$1) }' -`
maps=`$YPBINDIR/yphelper --maps $MASTER`
```

In other words, delete the hash symbol from the beginning of the first line and put one on the beginning of the second line before you run the command `/usr/lib/yp/ypinit -s mastername`.

Debugging NIS

NIS uses `portmap`, the RPC daemon, to dynamically assign a TCP/IP port number. Because of this many problems with NIS can be caused by `portmap`, so you can troubleshoot NIS with the `rpcinfo` command.

First make sure `portmap` is running. Check its status.

```
# service portmap status
```

If it's not running, stop all RPC services, such as `ypserv`, and start `portmap`:

```
# service portmap start
```

Then start all RPC services again and try running the `rpcinfo` command:

```
# rpcinfo -p localhost
```

Using the Automounter with NIS

Your NIS server should ideally make its `/home` directory available to NIS users. This way all clients will share not only authentication, but also user-specific settings, such as the GNOME or KDE desktop.

To facilitate this, add the following line to the NIS client's `/etc/auto.master` file:

```
/home    /etc/auto.home    --timeout 60
```

Next create your `auto.home` file. Add this line to it:

```
*     -rw,soft,intr   nis-server-name:/home/&
```

If the client's home directory is a separately mounted partition, unmount it.

On the server, make sure `/home` is shared by adding this line to `/etc/exports` and starting NFS:

```
/home   client-ip-address(rw)
```

The home directory should export to the correct location using the automounter.

The LDAP Alternative

Lightweight Directory Access Protocol (LDAP) is a proposed directory standard for the Internet, and has endless applications. You can use it as an address book for e-mail clients, store user information and photos, or use it as a back end to a web-based phone book.

LDAP authentication and other applications of the LDAP daemon are discussed in Chapter 30. Because of the extensibility of the protocol and its enhanced security options it is often used in place of NIS.

alternatives

In most distributions you will find multiple programs that fulfill the same basic functions. In some cases, such as with text editors or music CD player GUIs, you can simply choose one over the other. But in some cases it is important to have only one of a given type of service available at a given time. For example, Fedora Core comes with the `sendmail` and `postfix` message transfer agents, but since you can only have one MTA on a given e-mail server you'll want to make sure that all other programs that call the local MTA call the one you've chosen to use. Enter `alternatives`.

Under `alternatives` each type of service has a generic name that is actually a symbolic link to a file in the `/etc/alternatives` directory. For example the print command `/usr/bin/lpr` is a symlink to `/etc/alternatives/print`. This target file is *also* a symlink, and it points to whichever print manager you prefer, LPRng or CUPS.

Links in the `/etc/alternatives` directory are organized into groups, with the "master" link pointing to the primary program for that group (e.g., `sendmail` for the MTA group), and "slave" links that follow the lead when a change is made to the master link. Thus if you set `sendmail` as your MTA, the MTA group's slave links will update along with it. This will ensure that you get the correct mail queue and alias programs for `sendmail`, as well as the right UNIX manuals.

Currently the only link groups available for alternatives are `mta` and `print`. When a link group is set up it is in *automatic mode*. The links have a default priority, and the link with the highest priority is activated first. Changing the link takes you into *manual* mode, in which you set your own preference.

To display the default print server you'd type this:

```
# alternatives --display print
```

You can then set alternatives to choose your print manager automatically based upon the priority of selections in the print group:

```
# alternatives --auto print
```

You can select from a list of available programs with the `--config` argument:

```
# alternatives --config print
```

Of you can choose a different print manager, such as CUPS, manually:

```
# alternatives --set print /usr/bin/lpr.cups
```

Printing on GNU/Linux

Fedora Core comes with two print management systems: LPRng and CUPS. Both have System V and BSD interfaces, making them compatible with print commands from UNIXes past and present. However each system has its own configuration, so if you set up a printer in CUPS you will have to configure it again if you switch to LPRng, and vice-versa.

To start a print job you can use the `lpr` or `lp` commands. To view the print queue status you can use `lpq` or `lpstat`. To cancel a job you can type `lprm` or `cancel`. And because of the `alternatives` feature, these commands will call whichever print manager you're using as your default.

The print queue is an environment variable, $PRINTER, and it can be changed with the `lpr -P (printer-name)` or `lp -d (printer-name)` commands.

LPRng

LPRng is a basic print management utility, generally selected by default. It offers dynamic job redirection and support for setting up a pool of printers.

The LPRng program accepts jobs with the `lpr` command, then hands the job to the `lpd` daemon. This daemon then sends the job to the printer, or to a remote `lpd` service it the printer is on another machine. To send to a remote printer you can set your print queue with the `lpr -P` command and append an `@hostname` to the print queue name.

The `lpd.conf` file controls the `lpd` daemon. The `lpd.perms` file allows you to regulate access to the printing system.

Configuring LPRng Printers

LPRng's printer configuration is kept in the `/etc/printcap` file, which is generated automatically by the `printconf` command. This command is also accessible through the graphical `printconf-gui` utility. `printconf` is a useful utility that includes multiple types of print queues, as well as `foomatic` filters (which are a set of compressed PostScript printing drivers) for using to different printer models. A text user interface is also available via `printconf-tui`.

The `printcap` file is is regenerated from the `printconf` database each time the `lpd` startup script is run. To make custom changes you can hand edit the file `/etc/printcap.local`. The contents of this file are included verbatim in the `printcap` file whenever it is regenerated by restarting `lpd`.

The `lpd` service can be restarted with the command `service lpd restart`. Additionally, the `printconf-gui` utility contains a command to restart `lpd`.

To check your `/etc/printcap` file for errors you can run the `checkpc` command.

Managing LPRng Queues

LPRng's queue maintenance tool is `lpc`. It accepts a number of commands.

- `status` displays the status of all queues.
- `hold`, `release` and `topq` allow management of specific jobs.
- `redirect` sends a job to another print queue.
- `start` or `stop` turns the queue on or off.
- `hold` and `release` allow you to temporarily keep a job from printing.
- `topq` sends a job to the top of the queue.

CUPS: The Common UNIX Printing System

CUPS is a new advanced printing system for UNIX. It features support for the Internet Printing Protocol (IPP) and a web-based configuration utility. It stores its log files in the Common Log Format, which can be analyzed by most log analysis tools designed to study web logs.

CUPS has two configuration files. The file `/etc/cups/cupsd.conf` is used to configure the daemon itself, and looks remarkably like an Apache web server configuration file. The file `/etc/cups/printers.conf` contains information on your CUPS printers.

Only the superuser can manage CUPS printers and operation. To add other CUPS administrators, edit the main configuration file `/etc/cups/cupsd.conf` by uncommenting the `SystemGroup` line and adding other users.

Managing CUPS: the Web and Command Line Interfaces

After CUPS is running you can open the configuration page in any browser. Just go to *http://127.0.0.1:631* to view the CUPS web page. (Note: your system's firewall must be configured to allow this, even if your the service is running locally).

You don't need to manage CUPS from the web interface; it has a command line interface as well. The `lpadmin` command can help you manage your printers.

To add a printer:

```
# lpadmin -p (printer-name) -E -v (device-name)
```

To set it as your default printer:

```
# lpadmin -d (printer-name)
```

To remove it:

```
# lpadmin -x (printer-name)
```

14. Automating System Tasks

Using `cron`

The `cron` daemon is a utility that runs scheduled tasks. These tasks can be anything you can express on the command line or in a script, from a periodic system tape backup to a regeneration of a web page's search engine database. `cron` accomplishes this by checking once every single minute for jobs to run. A job is a single-line command, but administrators often use `cron` to call shell scripts, allowing `cron` to run much more complex jobs.

User `crontab` files

To edit your recurring `cron` events, use the `crontab` command to load a specially-formatted crontab file into `cron`'s to-do list.

```
$ crontab filename
```

The crontab file is a simple text file that has one line for each recurring event. For example:

```
0 9 * * * mail -s "Daily log" wsmith@example.com < /var/log/maillog
```

Let's examine this job in detail. The first group of characters is the schedule, with the * (asterisk) symbol used as a wildcard to indicate limitless repetition. A crontab line with only * symbols will run the specified command every minute, but in this case the command does the following:

0 - The minute of the hour (0-59), so this command only runs on the hour itself.
9 - The hour of the day (0-23). So this command runs at 9am.
* - The day of the month. So this command runs every day.
* - The month of the year; again, every month.
* - The day of the week (0-6, with 0 as Sunday), so it runs all week long.

Taken as a whole, every day at 9am the `cron` daemon will send an e-mail to wsmith@example.com with the subject "Daily log," and the body of the e-mail will be the current mail server log file.

Or if the recipient of this daily e-mail wants to receive it after he has finished his coffee, and doesn't need to see the e-mails on weekends, it can be changed thusly:

```
30 9 * * 1,2,3,4,5 mail -s "Daily log" wsmith@example.com < /var/log/maillog
```

As you can see the e-mail has now been delayed to send at 9:30am and the * referring to the day of the week is now a list separated by commas, with no spaces, of Monday through Friday.

Scheduling can get even more detailed than this. For example, time designations can be divided with the / symbol. So a `*/5` in the minute position will make the job run on any minute divisible by 5 (in this case 9:05, 9:10, 9:15, 9:20 and so forth).

Managing crontabs

The crontab command can also be used to manage cron jobs. The most common arguments are:

`-l` - List the crontab.
`-r` - Remove the crontab.
`-e` - Edit the crontab.
`-u` *`username`* - Install a crontab for another user (superuser only).

System crontabs

The system has its own crontab, which also runs from the `crond` daemon, but which has a different format than user crontabs like those described above. The master crontab is `/etc/crontab`, and by default it looks like this:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

The sixth field, after the specified date fields, is the username. In each of the four examples in this default version of the file the run-parts script is run. Located in `/usr/bin`, `run-parts` runs everything in the specified directory. As you can see from deciphering the date fields, these scripts run either hourly, daily, weekly or monthly depending upon the directory. These are generally system tasks, which you can customize. For example, the `cron.daily` directory contains scripts to clean up the `/tmp` directory.

The `/etc/cron.d` directory can contain additional system crontab files.

If the `/etc/cron.allow` file exists then you must be listed in that file to use `cron`. If the `/etc/cron.deny` file exists but the `/etc/cron.allow` file does not, then you must be listed in the deny file to be denied the use of `cron`.

Using `at`

Use the `at` utility to schedule a single job for a particular time. Unlike `cron`, it cannot be used to schedule recurring events.

The `at` command with a time argument will give you an `at>` prompt on which to type the command you'd like executed at that time. If you press Enter you'll get another line for further commands to be executed. When you're done with the last command press enter, and at the next `at>` prompt type Ctrl-D.

The time argument is extremely versatile. For example, all of the following are valid times:

```
$ at now + 10 minutes
$ at 3:30pm January 4
$ at 10 am Sunday
$ at noon Saturday + 5 minutes
```

To check on your scheduled jobs use the `atq` command. This will only show you your own running jobs, although the superuser can specify a username after the command. To cancel an at job, use the `atrm` and the job number, which can be found using `atq`.

So for example:

```
$ at now + 20 minutes
at> shutdown -r now
at>
job 1 at 2002-12-20 10:05
$ atq
1       2002-12-20 10:05 a wsmith
$ atrm 1
```

If the `/etc/at.allow` file exists then you must be listed in that file to use at. If the `/etc/at.deny` file exists but the `/etc/at.allow` file does not, then you must be listed in the deny file to be denied the use of `at`.

Using anacron

anacron is a scheduler, much like cron and at. However unlike the other schedulers anacron is not a daemon; generally it runs at system startup. And it is imprecise. It does not allow you to specify a time or date on which to run a job. Instead it lets you specify minimum intervals between jobs. This is useful because the cron and at daemons require your system to be running in order for them to function properly. If you use cron or at to schedule a job to run at 12:30pm on Saturday and your system is down at that time, the job will simply fail to run. cron and at do not care about jobs scheduled in the past, whether they ran properly or not.

With anacron you can set a monthly job to run if more than 30 days have passed. This makes it useful for systems that shut down frequently, such as desktop and laptop systems.

The anacrontab file

When anacron runs it parses the anacrontab file. This file is similar to a crontab file in its format. Here is an example anacrontab file:

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

1       65      cron.daily              run-parts /etc/cron.daily
7       70      cron.weekly             run-parts /etc/cron.weekly
30      75      cron.monthly            run-parts /etc/cron.monthly
```

As with cron, each line is a job. The first number in a job is the delay in days. The second number is the number of minutes to wait before running the job. The third field is the label for the command, which is used to name the timestamp file anacron uses to determine when it last ran a given job. As you can see by the contents of the default file, Fedora Core sets anacron to run system cron jobs. For example, if more than one day has passed since cron.daily ran, anacron waits 65 minutes and then runs it.

Only the superuser can use anacron.

Using tmpwatch

The tmpwatch command cleans old files out of directories, and is generally used to keep the /tmp directory clean. This directory contains temporary files created by running programs and services, but often these files are not deleted by their parent programs when they are no longer needed. Left unchecked the /tmp directory would simply grow perpetually.

Generally you don't have to run tmpwatch because Fedora Core includes it in the daily system cron job. The script, located at /etc/cron.daily/tmpwatch, cleans out files in /tmp that are more than 240 hours old, and files in /var/tmp that are more than 720 hours old.

The System Logs

All modern operating systems keep a running log of their operations. This is useful for troubleshooting after a fault, or for the general keeping of records during normal operation. The GNU/Linux system logger daemon is syslogd. Additionally, klogd is a kernel logger that sends messages from the Linux kernel to syslogd.

System logs are simply an output of program messages into a basic text file, however they can also be dealt with in other ways. A message can be broadcast to users, printed on the console or sent to other servers to be logged elsewhere.

All system log messages have a severity level. This allows you to filter messages and log only very severe

ones to your log files. Out of the box, Fedora Core systems broadcast emergency messages to all users (including messages like system shutdowns). Most other messages are saved to log files in `/var/log`, many into the general-purpose `/var/log/messages` file. This includes post-boot kernel messages. Any messages during boot in this file are from non-kernel sources.

`klogd` writes kernel messages in a ring buffer that is 8196 bytes in size, and it is constantly overwritten. The `dmesg` file always contains messages from the most recent boot, and these messages are taken from `klogd`. In fact its contents are identical to the output of the `dmesg` command.

Log Entry Format

System log entries are written to a single line and have four fields, like this:

```
Dec 30 20:12:13 localhost apmd[630]: Now using AC Power
```

This message is from the Advanced Power Management daemon (`apmd`) running on a laptop system. It's announcing that it has switched from using the internal battery to AC power. In this line you can pretty easily see the four fields:

- The date and time (Dec 30th at 8:12 PM).
- The hostname of the system where the message originated (localhost).
- The name of the application where the message came from (`apmd`) plus its process ID.
- The actual message details. These can go on for several lines if you view a log in an editor that wraps lines. For example, in `maillog` some lines must include the sender of a message, the sending mail server, the recipient, any error messages and other details.

The `syslog` Configuration File

The configuration file for the system log is `/etc/syslog.conf`. It has an extremely simple format:

```
source.severity destination
```

A source is a class of programs, and more than one program can be considered in the same class. Both a POP mail server for downloading mail and an MTA such as Sendmail count as `mail` log sources.

Severity measures the importance of the messages. The severity specified in the `syslog.conf` file is interpreted by `syslog` as meaning that it should record messages of the specified severity, plus any messages of higher severity.

Let's look in the file. This line, for example, logs all `mail` messages, regardless of their severity, to the mail log file:

```
mail.*       /var/log/maillog
```

This one logs all emergency messages from any source as a broadcast:

```
*.emerg      *
```

In addition to the above configuration file settings, the `/etc/sysconfig/syslog` file contains switches used when starting the `syslogd` and `klogd` services during startup.

System Log Files

Fedora Core maintains a number of different log files.

- `/var/log/dmesg` - Kernel boot messeges
- `/var/log/messages` - System error messages, including non-kernel boot messages
- `/var/log/maillog` - Mail-related messages

- `/var/log/secure` - Security messages such as logins and access of secure services
- `/var/log/xferlog` - FTP daemon log

Log Rotation

The `logrotate` service maintains logs files, keeping them from growing too large. A cron job in `/etc/cron.daily` runs it each day.

When a log, such as `/var/log/maillog`, is a week old, `logrotate` renames it. In this example it becomes `maillog.1` and a brand new `maillog` file is created as the "live" log file. A week later maillog.1 is renamed `maillog.2` and the current `maillog` is renamed `maillog.1`. And so forth.

General `logrotate` settings are kept in `/etc/logrotate.conf`. This includes how often logs are rotated into new files (weekly by default), how many rotated logs are maintained at a time (four by default) and a reference to the directory `/etc/logrotate.d`, which contains service-specific log settings and is generally populated by files deposited by RPM package installations.

If you find that your log files are becoming very large you may want to increase the rotation frequency or the number of logs kept in rotation.

Monitoring Your Logs

System logs are most useful when you actually read them. Unfortunately this can involve miles of text, so filtering your logs to find the most important messages is vital to a healthy system.

By default the Fedora Core distribution includes `logwatch`, which runs each day in the `/etc/cron.daily` job and can be used to notify you of specified error messages. It is configured in the file `/etc/log.d/conf/logwatch.conf`. The default settings for `logwatch` call for a low level of detail and a report to be mailed to the superuser each night.

You can also keep an active eye on your log files as they change by using the `tail` command with the `-f` argument. For example, you could monitor the `secure` and `messages` logs in real time:

```
# tail -f /var/log/secure -f /var/log/messages
```

Managing Running Processes

A number of tools are available to display running processes and manage your system resources. Some tools that let you view and manipulate processes are listed below.

- `top` - Displays a snapshot of running processes that is updated every five seconds
- `gtop` - A graphical version of top
- `kill` - kills a process specified by its process ID; add the `-9` option to guarantee that the process ends
- `renice` - Changes the priority of a process, where "niceness" leads to fewer resources available to a process and less nice processes get done faster

And here are some tools for managing resource utilization:

- `free` - Summarizes system memory useage
- `vmsat` - Summarizes virtual memory statistics
- `procinfo` - Displays `/proc` filesystem information
- `iostat` - Displays resource utilization

15. System Backups

Backups are a severely important ingredient if you'd like a reliable system. Computers crash, buildings collapse and power surges ... well, happen. If you experience a crash but do not have a useful backup the consequences can be dire.

Backups can be done to many types of media. The most common backup media is the tape device., but we'll also go over CD-ROM backups later on.

Tape Backups

GNU/Linux supports a variety of tape devices, such as IDE and SCSI drives and drives that use the floppy controller. IDE devices follow the device node naming schema `/dev/(n)htX`, where the optional `n` indicates that the device is non-rewinding and the `X` is the device number. So a typical non-rewinding tape drive would be `/dev/nht0`. SCSI tape devices follow the schema `/dev/(n)stX`, and floppy tape devices are `/dev/(n)rftX`.

Rewinding and non-rewinding device nodes refer to the same devices, however the use of one node or the other affects how the system treats the tape after use. The use of a non-rewinding name such as `/dev/nht0` will cause the system to leave the tape as-is after writing. The use of a rewinding name such as `/dev/ht0` will cause the tape to be rewound after each use.

Positioning and tape marks

Tapes are hard to understand to beginners, but the best way to illustrate how they work is to contrast them with a filesystem. In a filesystem like the one on your hard disk, files are stored all over the place and managed by a table. You can move files around as much as you'd like because a move does not really create a new file; it just updates the table.

Tape devices, on the other hand, are less flexible. On a tape, files are stored in a purely linear fashion. Once a file is on the tape you cannot move it. Tapes are both written and read in a linear fashion as well. This is because tape drives read and write using a *tape head*. This part of the device can only access a single point on the tape at a time. When you read or write to a tape, the device will begin reading or writing from the point on the tape where the head happens to be, so before writing you should always make sure that no data on the tape will be overwritten unless it is no longer needed. If you want to keep your old files and and add new data to a tape you should fast forward the tape to the end of the existing entries, rewind over the final tape mark and then start writing.

Positioning and Tape Marks

The tape marks tell where files begin and end. Because you have no filesystem, the files have no names. This is why people use utilities like `tar` to create archives. The archives cannot have file names, but the files in them can.

To understand tape marks, think of your data as a series of data records indicated by the letter `d`. Between these we can place tape marks, represented by the letter `m`. The tape might look symbolically like this:

```
dddddddd m ddddd m dd m ddddd m ddddddddd m
```

Tape Device Control with `mt`

The `mt` command can be used to control a tape device. This means rewinding the tape, ejecting it, or moving forward or backward to the position at which you'd like to start reading or writing.

Here are some useful commands to use on your tape:

To rewind the tape:

```
# mt -f /dev/ht0 rewind
```

To eject it:

```
# mt -f /dev/ht0 eject
```

Rewind and then eject:

```
# mt -f /dev/ht0 rewoff
```

To erase the entire tape:

```
# mt -f /dev/ht0 erase
```

To position the tape ahead 15 files:

```
# mt -f /dev/ht0 fsf 100
```

To position the tape ahead 14 files:

```
# mt -f /dev/ht0 bsf 14
```

To reset the tape's tension (something you have to do for travan tape drives):

```
# mt -f /dev/ht0 retension
```

To write a tape mark at the current tape position:

```
# mt -f /dev/nst0 eof
```

(ToDo: what is weof?)

To print the status of the tape unit:

```
# mt -f /dev/nst0 status
```

Tape archive with `tar`

`tar` (an abbreviation of "tape archive") was originally designed for writing data to tape media. Because it can create archive files on tape, many people use it to create archive files on a normal filesystem. To use `tar`, the basic syntax to remember is the following:

- `c` creates an archive
- `x` extracts an archive
- `z` invokes `gzip` to compress or decompress the archive
- `p` maintains the filesystem permissions
- `f` is the file (or device) where the archive resides
- `v` causes verbose output

So for example you could create an archive of your entire /home directory:

```
# tar cf home.tar /home
```

Or if it contains a lot of data you can compress it:

```
# tar czf home.tar.gz /home
```

To save it to a SCSI tape device:

```
# tar czf /dev/st0 /home
```

To extract it later:

```
# tar xzf /dev/st0
```

You can point `tar` to any writable device, not just tapes. To save to a Jaz or Zip drive simply use that device's node name instead.

To view the contents of an archive use the `t` option like this:

```
# tar tzf /dev/st0
```

Extracting specific files

If you find the file you need on your archive media (or your archive file) you can extract it by specifying it at the end of the command. Be sure to omit the leading slash. So if you wanted to get a file from your home directory you might do this:

```
# tar xzf /dev/st0 home/wsmith/filename.doc
```

The file will be written to the present working directory, and the complete path will be recreated. So in this case the directory `home` will be created if it is not already present, and under that the directory `wsmith` will be created, and `tar` will put the file into that. Logically, then, if you launch the above command from the root directory of your system the file will be written to its original location, overwriting any file with the same name. For this reason it is a good idea to extract files into a safe subdirectory and then move them as needed.

`dump` and `restore` for extended filesystems

`dump` and `restore` can aid you even further in creating backups. Like `tar` they can read and write data to tape, however they are written for use with Extended filesystems, so you can't take advantage of them unless you use ext2 or ext3.

The advantage of using `dump` and `restore` is that they can perform both full and incremental backups, allowing you to back up only data that has changed since the last full backup. It does this by keeping a log of its activity at `/etc/dumpdates`.

Using `dump`

When dump is performed it reads the `/etc/fstab` file to determine whether it should back up each filesystem. It does this by consulting the `dump` frequency field, which is the second-to-last field (the first single-digit number at the end of a partition's line, after the options).

Or you can bypass this automation and specify the directory in the command, like this:

```
# dump -0u -f /dev/st0 /home
```

This example would do a full backup of `/home`. The `-u` option specifies that dump should update the `/etc/dumpdates` file. The `0` is the dump level, and it specifies a full backup. Dump levels range from 0-9. While 0 guarantees a full backup, any higher number will back up data since the last lower-numbered backup. So for example:

- If you do a level 0 backup on Monday, it will archive the whole filesystem.
- If you do a level 9 backup on Tuesday it will archive any data that has changed since Monday.
- If you do a level 5 backup on Wednesday it will again archive data that has changed since Monday.

- If you do a level 9 backup on Thursday it will archive data that has changed since Wednesday, since that was a level 5 backup, and is therefore the most recent lower-numbered dump.
- If you do a level 4 backup on Friday it will back up files changed since Monday again.

Using `restore`

To recover data backed up with `dump`, make a new filesystem, mount it, `cd` into it and run the `restore` utility:

```
# restore -rf /dev/st0
```

`cpio`

The `cpio` utility is similar to `tar`. It copies files into or out of an archive file, either in `tar` or `cpio` format. In *copy-out mode* it reads a list of files and archives them to the specified device or file. For example:

```
# find /home | cpio -ocv > /dev/st0
```

This lists all files in the `/home` directory and its subdirectories and pipes the output to `cpio`, which writes them to the SCSI tape drive. The `o` option is for copy-out mode, the `c` option is for the newer `cpio` data format and the `v` is for verbose output.

In *copy-in* mode it exports the contents of a file or device:

```
# cpio -icdv < /dev/st0
```

In this case the `i` option is for copy-in mode and the `d` tells `cpio` to create directories as needed.

Remote backups

`dump` and `tar` can be used remotely. This is useful when only one machine has a tape drive. They do this by invoking `rmt` for remote management.

```
# dump -0uf username@server-name:/dev/st0 /home
```

This sends the local `/home` directory to the specified server under the specified user's account (though the `username@` part is optional).

Change the `RSH` environment variable to `ssh` and this process will be done securely.

CD-ROM Backups

If you have a CD Writer and want to make a data CD, there are a few good GUI tools available, including the KonCD utility for KDE. But it is worth learning the `cdrecord` command for basic recording operations at the command line.

The first thing you need to do to write a CD is to identify your CD writer's output address. To do this, run the `cdrecord` command with the `-scanbus` option. You must do this as the superuser.

```
# cdrecord -scanbus
Cdrecord 1.10 (i686-pc-linux-gnu) Copyright (C) 1995-2001 J?rg Schilling
Linux sg driver version: 3.1.24
Using libscg version 'schily-0.5'
scsibus0:
        0,0,0     0) 'ATAPI   ' 'CD-R/RW 8X4X32  ' '5.DW' Removable CD-ROM
        0,1,0     1) *
        0,2,0     2) *
        0,3,0     3) *
        0,4,0     4) *
```

```
0,5,0     5) *
0,6,0     6) *
0,7,0     7) *
```

In this example `cdrecord` found the writer at the output address 0,0,0. You can now use this information to burn an ISO image to CD, which you can do as a normal user:

```
$ cdrecord -v speed=8 dev=0,0,0 -data cdimage.iso
```

In the most recent versions of Fedora Core, this command is much less complicated, as cdrecord has been updated to support more friendly device names. On Fedora Core 2 you can do this:

```
$ cdrecord dev=/dev/cdwriter -data cdimage.iso
```

Fedora Core 2 also has experimental support for DVD writers.

Making an ISO image

Of course you need an ISO image to burn first. You can make these at the command line as well using the `mkisofs` command:

```
$ mkisofs -v -o file.iso file-or-directory
```

Multiple directories can be specified in the command. Once the ISO image file is ready it might be a very large file, depending upon the amount of data in the file or directory you are converting to ISO. Keep in mind that you only want to burn 650 or 700 MB at a time, depending upon the type of media, or else the image will not fit.

If you'd like to check out your image before you burn it you can actually mount it, and Linux will treat it like a real CD-ROM. Just use the `loop` option:

```
# mount -o loop image.iso /mnt/cdrom
```

Once you're satisfied with your image you can burn it.

16. Working With System Software: RPM and DEB Packages

In the early days of UNIX and GNU/Linux all software was added by compiling it and configuring it manually. Many people still prefer this method, and for some software it is advisable to install by compiling source code. But to make things quicker and easier for day-to-day work developers have created utilities to provide software in "packages" containing pre-compiled binaries and complete configuration files. Red Hat's contribution is called RPM (which was originally called Red Hat Package Manager but was later renamed RPM Package Manager).

RPM is central to many distributions, including Mandrake and SuSE. Its primary function is to install, remove or upgrade your software.

The Debian package management system is called DPKG, and it allows you to install DEB packages. These are analogous to RPMs, and there are great tools available for working with these files. Distributions that use DEB packages include Debian itself, Ubuntu, SimplyMEPIS and most other Debian-derived distributions.

Let's start with RPM.

RPM Packages

RPM Packages are distributed in RPM files. These files are archives filled with compressed files and dependency information. RPM maintains a database of installed software in the `/var/lib/rpm/` directory, and this database stores package information. For example, the names of all files owned by the GNOME desktop environment are stored in the database, allowing you to query the database and see what changes have occurred to GNOME's files.

A package name, by convention, includes the software's name, its version number, its release number (after a dash) and its architecture (`i386`, `i586` and `i686` for Intel-compatible PCs). A typical filename might be `ytalk-3.1.1-12.i386.rpm`. Other architectures include `alpha` for the Digital Alpha, `ia64` for the Intel Itanium, `s390` for the IMB S/390, `ppc` for the PowerPC and `noarch` for code that is architecture-independent, such as images or shell scripts.

Source code is distributed in files with an architecture of `src`. A source code package does not modify the RPM database, but rather unpacks into `/usr/src/`.

Adding New Software

To install a new RPM onto your system, use the `rpm` command with the `-i` argument.

```
# rpm -i ytalk-3.1.1-12.i386.rpm
```

Better yet, to use verbose mode and add hashes to indicate progress, you could use this variation::

```
# rpm -ivh ytalk-3.1.1-12.i386.rpm
```

When you launch the installation command, `rpm` will access the database in `/var/lib/rpm` to ensure that all prerequisites are met, and that the installation will not overwrite any existing files that are needed by another RPM-installed package. To avoid these checks, which can often prevent installation, use options like the `--force`, or `--nodeps` to ignore checks for dependencies, or `--replacefiles` to overwrite files that may already exist, like configuration files (not recommended if you're upgrading and need to keep old settings).

You can also add the "verbose" option with `-v`. This prints out progress bars.

Removing Software

The `-e` (erase) option will remove software from your system. It is important to use the name of the package, not the name of the file used to install the package, like this:

```
# rpm -e  ytalk-3.1.1-12
```

If you are not sure of a package's name you can query for it. RPM querying is discussed further below.

Upgrading Software

The `-U` option can be used to upgrade RPM software. Software upgraded by RPM is remove from the system and replaced with the newer version. The old configuration files are saved with a `.rpmsave` extension.

To upgrade a package, or install it if it's not there yet, you'd do this:

```
# rpm -Uvh ytalk-3.1.1-12.i386.rpm
```

Never upgrade the kernel!

Never, ever upgrade the Linux kernel. While it is worth updating your RPMs, particularly the kernel, the upgrade option can break your system if you use it on the kernel because it will remove the previous kernel and all its files. Install new kernels with `rpm -i`. Multiple kernel versions can happily coexist, and the installation will update your GRUB boot loader configuration, giving you a choice the next time you boot. If things go awry you'll be glad to have your previous version of Linux as a choice. (LILO users will have to update their configuration manually, since kernel RPMs do not update LILO.)

Freshening Software

Freshening, called with the -F option, is a great deal like upgrading except that an upgrade will install packages regardless of whether an existing version is installed, whereas freshening ignores packages you do not currently have.

This makes freshening useful when you want to apply errata, or updated packages, on your system. Errata are incremental updates that fix bugs and security problems, and the Fedora Core teamposts them online frequently. To apply all errata that apply to your system, use this FTP-enabled command:

```
# rpm -Fvh ftp://mirror.site.url/pub/fedora/linux/core/updates/1/i386/*.rpm
```

Querying Packages

RPM provides the -q querying option to help you manage packages installed on your system. There are two types of options with queries: options that help you find a package installed in your system, and options that give you information about those packages.

Finding a Package

The most basic query is to simply acquire a list of all packages installed on your system with the -a option:

```
# rpm -qa
```

But if you're looking for the version and name of a specific package, such as the Pine e-mail program, query for it by name:

```
# rpm -q mozilla
mozilla-1.4.1-17
```

If you're not sure of a package's exact name you can take the output of the `-a` option and filter it by piping it to the `grep` command:

```
# rpm -qa | grep vi
vim-common-6.1-14
vim-minimal-6.1-14
vim-enhanced-6.1-14
```

If you have a file that you think may be important and want to check the RPM database to see if it is owned by a package or else safe to delete, you can use the `-f` option:

```
# rpm -qf /usr/lib/mozilla-1.4.1/libgkgfx.so
mozilla-1.4.1-17.i386.rpm
```

If the file is a general file used by many packages, such as a library, you can query that file to see which packages are dependent upon it, as with the Kerberos 5 library:

```
# rpm -q --whatrequires libkrb5.so.3
```

Or you can query a package file rather than the database with the `-p` option.

```
# rpm -qp wine-20020605-2.i386.rpm
```

Finding Information on a Package

Using the steps above can specify a package, either by name or by criteria. At the end of the RPM query you can ask for information about the package(s). For information about a package, use the `-i` option.

```
# rpm -qi mozilla
```

Or list all of the files the `mozilla` package provides with the `-l` option.

```
# rpm -ql mozilla
```

To get a list of prerequisites for a package, use the `--requires` argument. For example, if you are going to install the latest Wine package you can first check to see its requirements:

```
# rpm -q wine --requires
```

Or see what scripts are launched when the package is added or removed:

```
# rpm -q wine --scripts
```

You can also see what has changed for a package since it was originally installed - i.e., if it has been upgraded.

```
# rpm -q wine --changelog
```

And using the query options in conjunction you can find out a lot of useful information that often eludes GNU/Linux newbies. For example, to find information about the package that owns the file `/usr/lib/mozilla-1.4.1/libgkgfx.so` (which happens to be Mozilla)

```
# rpm -qf /usr/lib/mozilla-1.4.1/libgkgfx.so -i
```

To find out what files you need to install a package on your filesystem:

```
# rpm -qp wine-20020605-2.i386.rpm --requires
```

This dumps more information than you're likely to need:

```
# rpm -qp --dump pkg
```

From left to right, the output fields are the path, size, mtime, md5sum, mode, owner, group, isconfig,

isdoc, rdev, and symlink.

The Complete RPM Database

Fedora Core provides the complete database of RPM packages for each release of its distribution. Installing the `rpmdb-redhat` package will give you information on all packages, even ones not currently installed on your system. This may be useful if you're missing a file that is a prerequisite for software you'd like to install. If you find yourself in this sort of predicament and you use Fedora Core, you can run `rpm` with the argument `--redhatprovides`. This will tell you information about which Fedora Core package you should install to get the required file.

```
# rpm --redhatprovides libkrb5.so.3
```

Verification of RPMs

RPM stores information about file sizes, permissions, ownership and other package attributes. The `-V` option can be used to check packages and ensure they are unchanged; this is valuable for security and troubleshooting.

For a single package you would do this:

```
# rpm -V mozilla
```

Or to verify everything in your system you'd add the `-a` option:

```
# rpm -Va
```

Verification error codes

Verification will produce a list files that have changed since the package was created. The list will look something like this:

```
# rpm -Va
S.5....T c /etc/printcap
.......T c /etc/securetty
S.5....T c /etc/hotplug/usb.usermap
S.5....T c /etc/sysconfig/pcmcia
.......T c /etc/libuser.conf
.......T c /etc/mail/sendmail.cf
SM5....T c /etc/mail/submit.cf
S.5....T c /usr/share/a2ps/afm/fonts.map
SM5....T c /etc/alchemist/namespace/printconf/local.adl
S.5....T c /etc/ntp/ntpservers
SM5....T   /var/mailman/pythonlib/email/Charset.pyc
SM5....T   /var/mailman/pythonlib/email/Encoders.pyc
SM5....T   /var/mailman/pythonlib/email/Errors.pyc
SM5....T   /var/mailman/pythonlib/email/Generator.pyc
SM5....T   /var/mailman/pythonlib/email/Header.pyc
```

Error codes are along the left, and the ones that end with a `c` are configuration files. Change is a normal behavior for configuration files, especially if you happen to remember changing them. But binary files shouldn't change unless you upgrade a package. And if you use RPM, an upgrade will update the database. So keep an eye out for changed binary files.

There are four error codes, and they are as follows:

- `5` MD5 sum
- `S` File size
- `L` Symlink
- `T` Mtime
- `D` Device

- `U` User
- `G` Group
- `M` Mode (including permissions and file type)

Key-based verification

RPM allows you to verify the integrity of an RPM file you have received from a dubious source. This is useful whenever it's possible that a package may have been tampered with - such as after a download. First you have to import the GnuPG public key to your key ring with the `gpg` command. You can usually find GPG keys on installation media, such as the Fedora Core installation CD-ROMs. Import it like this:

```
# gpg --import RPM-GPG-KEY
```

Then you can verify a package with it with the `-K` option:

```
# rpm -K mozilla-1.4.1.rpm
```

Using RPM for an alternate system location

Sometimes you might want to perform RPM operations on a system that is mounted beneath a running system. For example, you could mount a GNU/Linux system stored on a disk you've migrated from another computer. Or you could boot into rescue mode on your installation media, which will attempt to mount your filesystem under `/mnt/sysimage`. You can add, remove, freshen or query RPMs using the `--root` option to specify your system's root directory.

```
# rpm -ivh --root /mnt/sysimage /mnt/source/Fedora/RPMS/package.rpm
```

Source RPMs

SRPMs are a means of building an RPM package from source code. To use them you must have the `rpm-build` package installed. When you install an SRPM file from a Fedora Core supplied package its contents end up in `/usr/src/redhat/`. Beneath that directory are five directories in which the contents end up:

- `BUILD` - The uncompressed and untarred contents of SRPMs are placed here.
- `SOURCES` - Source code and its patches, usually in a tar archive, are deposited here.
- `SPECS` - "Spec" files are descriptions of the software with instructions on how to build it and a list for all the binary files that will be installed.
- `SRPMS` - If you build a new SRPM it is placed here.
- `RPMS` - If you build a binary RPM it is placed in here.

Building RPMs on older systems was done with build options of the rpm command; now you must use the `rpmbuild` command.

To build an RPM directly from an SRPM file, use the `--rebuild` argument. It will put the new RPM in `/usr/src/redhat/RPMS/i386`.

```
# rpmbuild -rebuild mozilla-1.4.1-17.src.rpm
```

If you'd like to create a binary RPM from an installed source RPM, go to the `SPECS` directory and run `rpmbuild`, citing the package's spec file. Again, the new RPM will go in the `RPMS` directory under `i386`.

```
# rpmbuild mozilla.spec
```

You can also take the source code and, using the spec file, build a new source RPM with the `-bs` ("build source") option.

```
# rpmbuild -bs mozilla.spec
```

Debian Packages

Debian-based systems use a system similar to RPM in many ways. The Debian Package Management system also uses packages that contain binary files with meta data for installing them. Its tool is very similar in its use to the RPM tools.

In addition Debian-based systems have a tool called **`apt-get`**, which allows you to easily update your system from online repositories.

Using `dpkg`

The options for the `dpkg` command are very similar to those for the `rpm` command. For example, to install a package:

```
# dpkg -i package.deb
```

Removing packages is slightly different than the analogous RPM command:

```
# dpkg -r package
```

To list all packages on your system:

```
$ dpkg --list
```

`dpkg` makes it easy to filter such results without using `grep`, like this:

```
$ dpkg --list '*foo*'
```

This would list all packages with the word *foo* contained in their names. So in general the Debian system has the same basic ingredients as the RPM system developed by Red Hat.

Converting packages with `alien`

Of course a lot of software is distributed in RPM format these days, and Debian's package format tends to be sidelined. No worries - if you use a Debian-based system you can still install RPMs: you just have to convert them to Debs first. Since RPM is an open format this is very easy to do, and a command called `alien` will actually do it for you.

```
$ alien package.rpm
```

The resulting `package.deb` file can then be installed with `dpkg` like any other Deb package. With the `-r` option this handy utility can also create RPMs from Debs.

Automated installations with `apt-get`

Of course the best thing about Debian-based systems is their ease of automated package installation. While Fedora Core and other RPM-based distributions have recently been moving toward easier package management and simplified update deployment, Debian had this area covered years ago with the Advanced Package Tool, or APT. The most common APT command is `apt-get`.

APT doesn't know anything about Deb files. It merely understands package names, and passes them to the `dpkg` system. In the process it examines dependencies and also passes them to `dpkg`. Using a list of online package sources, `apt-get` downloads your chosen packages and all of their required dependencies and installs them, quick and easy.

17. Boot Loaders

All operating systems require a boot loader to start when you power on your system. Fedora Core and most other modern distributions use GRUB, the "GRand Unified Bootloader." But older GNU/Linux systems sometimes boot using a loader called LILO, the "LIux LOader." Both loaders allow you to load GNU/Linux or, if you configure them properly, to set up your system as a dual-boot machine with GNU/Linux and another operating system.

About System Startup

When you power on an Intel-compatible x86 machine the BIOS passes its control of the hardware to whatever program it finds in the first 446 bytes of the primary hard disk, called the Master Boot Record (MBR). This program is either an initial program loader (IPL) that starts the operating system, or else a secondary boot loader. GRUB and LILO both use a secondary loader, the advantage of which is that it rests in the boot sector of the disk's primary partition. There's a lot more space in the boot sector than there is in the MBR.

Where to Put The Loader

There are two ways to install a boot loader like GRUB or LILO: as the primary boot loader started from within the MBR itself, or as a secondary boot loader. The former option is pretty standard. The latter requires a separate primary boot loader in the MBR. You'd only do the latter if you had a particular reason to use a separate boot loader when you first start your machine.

For example, if you had a system running Windows NT and you wanted to install GRUB or LILO to load GNU/Linux on a separate partition, you might for the sake of minimum change retain the NT Loader and configure it to present you with a GNU/Linux option that calls GRUB or LILO when selected. Nowadays, however, GRUB is becoming increasingly popular - to the extent that its ease of use makes other boot loaders less necessary. It is very easy to boot Windows from Grub using chainloading options, which will be described herein.

GRUB

GRUB is more advanced than LILO in that it more easily supports multiple operating systems, including many proprietary ones. It also can read filesystems, so you don't have to reinstall GRUB each time you modify its file; GRUB can actually read its configuration file during the boot process. And you OS can be stored on serveral filesystems - Extended, ReiderFS, FAT, JFS, minix or FFS. For these reasons most GNU/Linux users are switching from using LILO to GRUB.

The main file for grub is `/boot/grub/grub.conf`. It's similar in format to its LILO counterpart. There are global options followed by a stanza for each operating system. Here is a sample annotated `grub.conf` file:

```
default=1                                               # Default OS, starting with 0
timeout=10                                              # Seconds until default is loaded
splashimage=(hd0,0)/grub/splash.xpm.gz                  # Splash screen as compressed XPM

title GNU/Linux (Fedora Core with 2.4.22-1.2115.nptl)# Title of OS 0 (first stanza)
        root (hd0,0)                                    # Location of root directory
        kernel /vmlinuz-2.4.22-1.2115.nptl ro root=/dev/hda1 hdd=ide-scsi
        initrd /initrd-2.4.22-1.2115.nptl.img

title GNU/Linux (Fedora Core with 2.4.22-custom)
        root (hd0,0)
        kernel /vmlinuz-2.4.22-custom ro root=LABEL=/ hdd=ide-scsi
        initrd /initrd-2.4.22-custom.img

title Windows 2000 Professional
        rootnoverify (hd0,1)
        chainloader +1
```

This configuration file loads three operating systems, two of them GNU/Linux. The first, in stanza 0 (in GRUB, the first number in a set is always 0) is a GNU/Linux test system. It is on the same partition as stanza 1, so it is probably a re-compiled kernel. The default system is stanza 1. Stanza 2 boots Windows 2000 Professional. Windows cannot be loaded directly by GRUB because the Windows boot loader has many defects, however GRUB can do it via a process called chainloading. Note that some configurations of DOS or Windows may require that you hide and unhide partitions in order to boot the OS.

The `grub.conf` file can be difficult to read until you commit the device notation to memory. In GNU/Linux device names look like `/dev/hda1` or `/dev/fd0`. GRUB has a slightly different notation. As mentioned before, everything begins with 0. And commas delimit devices from partitions. Some examples:

- `hd0,0` The first partition on the first hard disk detected by the BIOS. This includes both IDE and SCSI.
- `hd0,1` The second partition on the same device.
- `hd1,3` The fourth partition on the second hard disk detected by the BIOS.
- `fd0` The first floppy detected by the BIOS.

Changes to the configuration file take effect immediately, since GRUB reads the file as it boots. This is different than LILO, which reinstalls itself to the MBR each time you run the executable. If the MBR becomes corrupted in GRUB you can boot via a floppy disk and force GRUB back into the MBR with the command `grub-install` *`device-name`*.

Custom Booting with GRUB

Like LILO, GRUB allows you to pass arguments to your system by editing the various stanzas in your configuration. To edit a stanza, highlight it and press `e`.

In menu-editing mode you can modify your stanza. For example you can boot into run level 1 by adding a 1 to the end of the kernel line. When you are finished editing, type b to boot.

You can also enter the GRUB command line, which is much like the bash `shell` and allows you to view files and perform diagnostics.

GRUB can be configured to require a password for customizations. If you have configured GRUB to use a password you must press `p` and then enter your GRUB password before you can pass arguments to your operating system or use the command line.

LILO - The Classic Linux Loader

The older LILO keeps its configuration file at `/etc/lilo.conf`. It consists of a number of "global" options that apply to LILO as a whole, followed by sections called "stanzas" for each operating system LILO must boot. Note: the `lilo.conf` file is not read at all during your system's boot process. Rather, if you've just configured LILO and would like to boot your system as per its new settings, you must first run the `lilo` command. This interprets your configuration file and converts it into a binary format that will fit in the MBR.

Here's a sample `lilo.conf` file, annotated with information about the various settings:

```
# Global options

boot=/dev/hda# The location of your first stage loader.
install=/boot/boot.b      # The location of your second stage loader.
map=/boot/map# The location of your system map file.
linear                    # As opposed to lba32.  Use linear addressing.
timeout=70                # Tenths of a second before the default OS is loaded.
default=linux# The default OS if none is selected after timeout.
message=/boot/message     # A PCX-format image or a text file to display at boot.
```

```
password=allyourbase      # A password required for boot.
prompt                    # Displays a LILO prompt (otherwise you must press Shift)

# Stanzas for operating systems on this computer

image=/vmlinuz-2.4.18-14               # Kernel image
             label=linux               # A name for this OS
             intrd=/boot/intrd-2.4.18-14            # Initial RAM disk
             read-only                 # Initial mode for mounting root filesystem
             root=/dev/hda2            # The root filesystem

image=/vmlinuz.test
             label=test
             intrd=/boot/intrd-test
             read-only
             root=/dev/hda2

other=/dev/hda1
             label=dos
             table=/dev/hda
```

As you can see from above, this installation of LILO is used to boot three operating systems. Two are GNU/Linux systems, although the second uses a different kernel image. The latter of the two is a test system, perhaps a newly-compiled beta version of Linux. The third OS is a DOS-based system, such as Windows 98.

With each update to `lilo.conf` you must run `/sbin/lilo`. You can test your configuration with the command `lilo -t` if you're uncertain of its functionality.

Custom Booting with LILO

If your system uses LILO, by default Fedora Core installs it with a graphical menu screen as its "message" file. In this screen you can select an operating system, or you can press Ctrl-X to go to a LILO text prompt.

At the prompt you can boot your operating system with arguments. For example, if you type "linux" and a number, the Linux kernel will boot into that runlevel. So typing `linux 1` will boot you into run level 1 (also called single-user mode) from which you can diagnose problems and avoid loading most of your services. For other options view the `bootparam` manual page by typing `man bootparam` in GNU/Linux.

Dual-Boot and Multi-Boot Systems

Many users of GNU/Linux use their systems in a dial-boot or multi-boot configuration with other operating systems. Examples of these configurations are shown above in the above sections regarding LILO and GRUB configurations.

GNU/Linux can happily coexist with other systems, and Fedora Core's installer even detects and configures many of them for you. Other systems are not as friendly; for example Microsoft Windows is generally unaware of other operating systems and will overwrite the MBR whether you ask it to do so or not. For this reason it's probably a good idea to install alternate systems first, and follow up with your GNU/Linux installation.

Generally you will need at least two partitions with GNU/Linux - a swap partition and a native partition. So when you install the other operating systems you should leave room for the GNU/Linux partitions, which together can take up at least 2 GB depending upon your installation.

Installing Around an Existing OS

If another operating system exists on your system and uses all of the available disk space, there are a couple possibilities.

The best is to use a partitioning utility to re-organize your hard disk. PartitionMagic is a popular utility that can resize many existing partitions, including Extended, FAT and NTFS. FIPS, a DOS utility that is included on Fedora Core's CDROM, can split a FAT partition in half. Another more recent addition is `parted`, a GNU utility that can resize Extended and FAT partitions.

A less desirable option is to back up your alternate operating system, repartition the computer and reinstall from a backup.

Booting can be tricky. The best option is to use GRUB to boot your alternate operating system if this is supported. If not, and you wish to continue using another boot loader such as the NT Loader or System Commander, you can configure it to point to a partition on which LILO or GRUB has been installed into the boot sector.

18. Rolling Out GNU/Linux with Fedora Core's Kickstart

If you plan to roll out GNU/Linux on numerous computers, you'll save yourself a lot of time by setting up an installation server. Fedora Core can be installed from FTP, HTTP or NFS much faster than it can be installed by CDROM media.

With Kickstart you can go a step further by specifying your desired configuration on a floppy disk, allowing a quick and uniform installation of the OS across many computers.

Fedora Core's Network Installation

You can set up any server with a couple GB of extra storage to be a Fedora Core Network Installation server. First you must copy all of the RPMs and other installation materials - that is, everything in `/mnt/cdrom/Fedora` - to a public location. For anonymous FTP this would be `/var/ftp/pub`. For HTTP this would be `/var/www/html/pub`. For NFS create an entry in `/etc/exports` to open a read-only share like this:

```
/path/to/share            allowed-hosts(ro)
```

The clients can then be set up using a bootable network-aware floppy; Fedora Core comes with bootable floppy images, and floppy disks made from the `bootdisk.img` and `drvnet.img` files can load the second stage installer from the server.

Kickstart

Kickstart is a very useful tool for setting up multiple systems. It allows the `anaconda` installer to read its configuration from a special text file rather than prompting the user. If a line is missing here or there, the installer will prompt the user for the required information and then get on with the installation. Kickstart can be used with either CDROM media or a network installation.

Making a Kickstart file is not particularly hard. In fact, once you install Fedora Core you'll find one ready for you at `/root/anaconda-ks.cfg`. This file is specific to your installation, so if you perform your installation the same way you'd like to perform additional installations on other systems you can hold onto this file as a recipe for future roll-outs.

For a floppy-based Kickstart installation, customize your Kickstart file the way you like it and save it as `ks.cfg` in the root directory of a floppy made from the `bootdisk.img` file, and also the `drvnet.img` file if you plan to do a network installation. Then boot the machine with the disk, and at the LILO prompt type `linux ks=floppy`.

For a network-based Kickstart installation, connect the computer to a network that has a DHCP server (the network-based installation depends upon DHCP). The DHCP server can be configured to store the Kickstart file, or it can provide the client with the location of the Kickstart file if it is on a different server. It can also provide the location of the installation server itself. Booting with floppy disks made from `bootdisk.img` and `drvnet.img` will allow you to start `anaconda` using this DHCP-supplied Kickstart, and then to install Fedora Core using a network share or the CDROM media.

The Kickstart File Part 1: Commands

The beginning of a Kickstart file is comprised of commands. These are configuration tasks the installer needs to perform prior to installing packages, along with arguments to guide installation. A lot of these are simple options. Take a look at the first few lines of a typical Kickstart configuration file:

```
# Kickstart file automatically generated by anaconda.

install
cdrom
```

```
lang en_US.UTF-8
langsupport --default en_US.UTF-8 en_US.UTF-8
keyboard us
xconfig --card "ATI Mach64" --videoram 8192 --hsync 30-95 --vsync 50-160 --resolution
1024x768 --depth 24 --startxonboot  --defaultdesktop gnome
rootpw --iscrypted $1$luljAvMM$kc92aoaCIuAmG9HFmlifW1
```

So the first command starts the installation, the second chooses the data source (in this case your CD-ROM drive), the third and fourth set language options, the fifth launches the `kbdconfig` utility to set keyboard options, and so forth.

You can fully customize this file, and use it to save time on future installations. For example, take out the `cdrom` line and replace it with this one:

```
nfs --server 10.0.15.4 --dir /var/ftp/pub
```

Now Kickstart will make `anaconda` use an NFS server at 10.0.15.4 as the data source, so you can set up an installation server and roll Fedora Core out across many machines without carrying around all of the installation CD-ROMs. (You can also set up an FTP or web server to do this – more on this later in the guide).

Some Kickstart commands are worth customizing from machineto machine. Chief among these are any instances of the `part` command. This is used by DiskDruid to partition your system and to set mount points. For example:

```
part swap --size size
part ext3 --size size
```

Well worth changing, unless all of your systems have the same hard disk configuration.

The Kickstart File Part 2: `%packages`

After the command section of a Kickstart file, the RPM packages listed in the section entitled `%packages`. In this section each section is either a package name (without its version number), such as `mozilla`, or an @ symbol followed by the name of a group of packages, such as `@ GNOME`.

If you plan to roll out your systems with non-Fedora Core RPMs included you must rebuild `anaconda`'s RPM `hdlist` database. Put the packages in the `/Fedora/RPMS` directory of the CD or network share and then run this command on the `/Fedora` directory:

```
# /usr/lib/anaconda-runtime/genhdlist /path-to/Fedora
```

You'd specify the Fedora Core directory of the share being used, not necessarily that of the CD-ROM media.

The Kickstart File Part 3: `%pre` and `%post` Scripts

The `%pre` and `%post` sections of a Kickstart file run as Bash shell scripts before and after installation. the `%pre` section runs after the file is read, but prior to partitioning and package installation. The `%post` section is potentially far more useful, as it runs in a chrooted environment in `/mnt/sysimage`, which is the installed system. This gives you access to all of the commands available in the installed system. You can use `%post` to customize files using `echo`, like this:

```
echo "nameserver 10.0.15.4" >> /etc/resolv.conf
```

Or install an entire file, downloaded from a remote server via FTP:

```
lynx -source ftp://10.0.15.4/pub/passwd > /etc/passwd
```

(insert an example for rolling out)

19. Linux Operations and Customization

The Linux kernel is the core of GNU/Linux. It boots the system, loads drivers and manages running processes. Without it the GNU system would almost certainly not exist today.

The /proc Filesystem

The `/proc` filesystem is not a real filesystem. In fact, you'll find that the files and directories mostly have a size of 0 bytes. This is because `/proc` is part of the Linux kernel. Using the `cat` command you can view the contents of these files, which contain information about the processes running on your system. They are treated like real files because in UNIX everything is a file.

Some of the important sections:

- `/proc/cpuinfo` - Information about the processor(s).
- `/proc/dma` - DMA settings.
- `/proc/interrupts` - IRQ settings.
- `/proc/ioports` - I/O settings.
- `/proc/loadavg` - Load average.
- `/proc/meminfo` - Information on available memory, including swap space.
- `/proc/uptime` - System uptime in econds, as well as idle time.
- `/proc/version` - The Kernel version, build host and build date.

Avoid viewing `/proc/kcore`. This is kernel memory, and displaying it can render your terminal temporarily unusable.

Beneath the `/proc` directory there are also subdirectories. The numerical directories are linked to processes of the same ID. For example, `init` always has a PID of 1, so the `/proc/1` directory is devoted to `init`. Other directories:

- `/proc/ide` - Information about the system's IDE devices.
- `/proc/net` - Network activity information.
- `/proc/scsi` - Information about the system's SCSI devices.
- `/proc/sys` - Linux kernel configurtion.

The `/proc/sys` directory can actually be modified on a running system. If the option CONFIG_SYSCTL is enabled in Linux, a change to the contents of a file in `/proc/sys` can change the parameters of the kernel. For example, doing this ...

```
# echo 1 > /proc/sys/kernel/ctrl-alt-del
```

will tell the system to reboot after a Ctrl-Alt-Delete keystroke, rather than trap the command and deal with it gracefully. Other instructions are available in the source code documentation which, if you install the source, is available in `/usr/src/linux-(version)/Documentation/sysctl/`.

Modifications to these files are temporary, and not saved after your system shuts down. But some values are set in a manner similar to that seen above every time you reboot your machine. The `rc.sysinit` script contains this line:

```
# Configure kernel parameters
action $"Configuring kernel parameters: " sysctl -e -p /etc/sysctl.conf
```

This calls `sysctl`, which sets a number of values. If you'd like to set values each time your system boots, you can add an entry to `/etc/sysctl.conf`. To find the entry needed for your desired kernel option run the command `sysctl -a` to view the list, or look in the kernel source documentation under `sysctl`.

Software RAID

Generally RAID is done at the hardware level and the operating system is generally unaware of disk redundancy. GNU/Linux can use software-controlled RAID, and on Fedora Core it comes in the `raidtools` RPM, which is installed by default with the `@ base` package group. Software RAID is not as efficient as its hardware equivalent, but it can be useful in situations where you'd like to create cheap redundancy.

If you are new to a system using software RAID you can run the `lsraid` command to view information about its configuration. You'll also find information in `/proc/mdstat`.

Software RAID is controlled from the `/etc/raidtab` file, which you have to create. Examples are available in `/usr/share/doc/raidtools-1.00.2/`. If you wanted to set up disk mirroring between two partitions, each on a separate hard disk, you would use this format to specify RAID 1.

```
raiddev                 /dev/md0
raid-level              1
nr-raid-disks           2
nr-spare-disks          0
chunk-size              4

device                  /dev/hda1
raid-disk               0

device                  /dev/hdb1
raid-disk               1
```

Set up your devices to act as a single RAID volume by creating a device node with `mkraid`.

```
# mkraid /dev/md0
```

Then use the `raidstart` command, and you'll be able to use the device node `/dev/md0` for your new redundant filesystem. First you'll have to make a filesystem on it:

```
# mke2fs -j /dev/md0
```

You'll also have to add it to the `/etc/fstab` file to remount it automatically each time you boot. RAID will restart automatically from the `/etc/rc.d/rc.sysinit` script. You can check on the status of your RAID by looking in `/proc/mdstat`.

You'll also want to test your RAID configuration by booting without one of the disks installed or, in the case of hot-pluggable SCSI drives, taking a drive offline and removing it during operation.

Recovering from a Disk Error

If a single disk in a software RAID array fails, power down the system. Replace the dead disk and reboot. RAID will be aware that a disk is missing, and you'll have to tell it to start using the new disk with raidhotadd. If the above example's `/dev/hda` was replaced, you'd execute this:

```
# raidhotadd /dev/md0 /dev/hda1
```

You can see the recovery process in progress in `/proc/mdstat`. This includes the estimate time it will take to rebuild the RAID once the new disk has been added. You can keep using the array during the rebuild process, but it will be slower than usual.

20. Understanding and Compiling the Linux Kernel

The Linux kernel was built to be a *monolithic* kernel. This means that the entirety of the kernel's code is contained in one continuously-running program. Linux garnered much criticism early on for its monolithic structure.

Another more advanced type of kernel is a *microkernel*. This type of kernel only implements lower-level functions like device-drivers, virtual memory and process scheduling in kernel space, and puts higher-level facilities like networking and filesystems in user-space. This architecture is more extensible, allowing innovations to be added to the kernel very easily, and gives the microkernel the potential for longevity. The Hurd kernel, which with GNU Mach is the official kernel of the GNU system, is an example of a microkernel.

Many efforts have been made to render Linux less monolithic. The best example is its *kernel modules*. Device drivers and other functionality can be compiled as modules rather than as a part of the kernel itself. Fedora Core and other distributions have customized their kernels to support a wide range of hardware, with some features in the kernel and others as loadable modules.

Kernel Versions

Kernel versions are numbered with a major number, a minor number and a patch level. You can find your kernel version with this command:

```
$ uname -r
```

There are many other options for this command, but this one specifically tells you the version number running your system. On the system I'm using to write this document that would be `2.6.5-1.358`, so in my case the major number is 2, the minor number is 6, and the patch level is 5.

Everything after the hyphen is the kernel's EXTRAVERSION, a custom string of text added by whomever compiled the kernel (perhaps you) to distinguish it from other builds of the same patch level. The need to distinguish seeminly identical kernels arises from the ability to *customize* the software prior to compiling it – for example, by adding SMP support for multi-processor machines, or adding support for the Windows NTFS filesystem, or removing support for SCSI devices on computes that don't require it.

Minor numbers that are even, in this case 6, are stable releases. Odd numbered minor numbers are developmental. At the time of writing the latest stable kernel version is 2.6.7, and the entire 2.6 series was based off of the 2.5 developmental series.

This section geared toward helping you to compile both a 2.4 series and a 2.6 series kernel. They are somewhat different to compile, so we'll start with 2.4 and then outline some of 2.6's changes later on.

Kernel Modules

Kernel modules are dynamically-loaded components; generally drivers for devices not needed during boot time or not used frequently are modularized rather than compiled into the kernel file itself.

Modules are kept in `/lib/modules/`*`kernel-version`*`/`. The final directory name matches the kernel version because different versions of the kernel may have different modules. The running kernel loads and removes modules on demand, but they can be added or dropped manually with `insmod` and `rmmod`. The advantage of loading a module manually is that you can pass arguments to it; removing one can come in handy if it is idle.

Modules are often dependent upon other modules, and the `depmod` command will generate a dependency database. It's run during system boot as `depmod -A` by `/etc/rc.d/rc.sysinit`. Once the database is built the kernel loads modules with `modprobe`, which is better than `insmod` in that it uses the database to load a module along with any modules upon which it is dependent, and in that it consults

/etc/modules.conf for any arguments that should be used on any particular module. Therefore if a module requires special arguments you can put them in /etc/modules.conf. For example

```
post-install sound-slot-0 /bin/aumix-minimal -f /etc/.aumixrc -L >/dev/null 2>&1 || :
```

With most newer modules the command modinfo -p *module-name* can tell you what arguments are supported.

Some modules, such as support for RAID or SCSI devices, are often needed for mounting the root filesystem during boot. In this case an initial RAM disk image, or *initrd*, is needed to make sure the modules can load before the filesystem containing the modules is available. mkinitrd can be used to make an initrd, and you must configure your boot loader to recognize the image. For example, this GRUB stanza in /boot/grub/grub.conf loads the Linux kernel along with an initrd:

Figure X.1 - A stanza from the GRUB configuration file

```
title GNU/Linux (Fedora Core 1) (2.4.22-1.2115.nptl)
        root (hd0,1)
        kernel /vmlinuz-2.4.22-1.2115.nptl ro root=LABEL=/ hdc=ide-scsi
        initrd /initrd-2.4.22-1.2115.nptl.img
```

Compiling - Should You or Shouldn't You?

There aren't many operating systems for which you are allowed to download the source code of the kernel and customize it. It's a rather nice novelty, but most users will not need to recompile the kernel. For one thing you don't have to compile the kernel to get updates because updated kernels are frequently made available for Fedora Core as RPM files. So you can always download a new kernel RPM and install it, and the new kernel will be added to your boot loader configuration so that you can choose either the old or new version when you boot your system.

If you do choose to update using a new kernel RPM, avoid using the rpm -U command. This upgrade command will remove your previous kernel and all related files, which would be bad. The best way to install a kernel RPM is with rpm -i, which will preserve your existing kernel in case things go awry. You can always rpm -e to remove the old kernel if you really want it to be gone.

But in some situations recompiling the kernel can be useful. Or fun, or educational - or some combination of the three. You get the opportunity to remove unnecessary features, add features that your distribution didn't include, or customize the kernel to run on your particular system (sometimes necessary for some laptops).

Or maybe you'd like to install the latest bleeding-edge kernel from source code. Why not?

Step 1: Prerequisites

To compile Linux you'll need the following RPM packages installed on your system:

- A kernel source RPM. The files named kernel-*version.architecture*.rpm are Fedora Core's customized kernels, whereas the files named kernel-*version*.src.rpm are the unaltered kernels as available from *http://www.kernel.org*.
- glibc-kernelheaders
- gcc
- dev86
- cpp
- make
- binutils
- glibc-devel
- ncurses
- ncurses-devel

Verify that you have all of these packages with `rpm -q` before trying to build the kernel.

Step 2: Name your Kernel

Once you've verified that you have the above packages are available you have to decide on your own EXTRAVERSION name. In this example we'll call it `-special`. Go to the `/usr/src/linux-version/` directory and edit the source code's Makefile. On line 4 you should see the option:

```
EXTRAVERSION = -special
```

Of course you can name it whatever you'd like.

Step 3: Bring the Kernel to a Clean State and Get a Configuration File

First, if you've worked with the source before you should use `make mrproper` to bring the source to its natural state.

```
# make mrproper
```

Now you can copy a configuration file of your choice to the top-level source directory. There are a number of different configuration files, including ones for the the AMD Athlon, the Intel i386 (including the 486), the i586 (Pentium), the i686 (Pentium Pro and later), and 64-bit Intel processors. There are also SMP versions of each file for multi-processor systems.

```
# cp -p configs/kernel-2.4.18-i686.config .config
```

Now use `make oldconfig` to finalize the configuration file.

```
# make oldconfig
```

This is a text-mode interface that takes your configuration file and compares its variables with the variables found in the kernel source. If it finds variables in the source that are erroneously absent from the configuration file it will prompt you for them. When using a Fedora-supplied `.config` file you will probably not run into any missing variables.

Step 4: Customize the Kernel

Configuration is the meat and potatoes of kernel compilation. You can add needed features, remove unwanted ones or compile features as modules. Your customization choices include `make config`, `make menuconfig` or `make xconfig`. The first is a text-based interface, the second has text-based menus and the third presents a graphical menu system and is the easiest to use.

```
# make xconfig
```

Configuration gives you an opportunity to enable features your distributor didn't include, such as experimental support for the NTFS filesystem, or to remove features you don't need, like Firewire devices if your system doesn't have any such hardware. Other features that may be worth shutting off are support for old CDROM drives, ISDN and IrDA. Note that you should customize the configuration for your particular processor family.

Step 5: Propagate Your Configuration

Your shiny new `.config` file should be propagated throughout the various source directories with this command:

```
# make dep
```

Step 6: Compile Linux

Now you compile the kernel itself. You'll do this with some further uses of the `make` command. For each example command you may choose to add this to the end of the command: `2>filename`. This will take any activity interpreted as an compile error (error code 2) and redirect it to the file named `filename`. You may want to keep tabs on your compiler's error messages; though it's normal to see a few of them.

You build the kernel with this command:

```
# make bzImage
```

The kernel image will end up as a compressed file in the bzip2 format, with a name like `/usr/src/linux-(version)/arch/i386/boot/bzImage`. Additionally, the `System.map` file will be placed in the directory `/usr/src/linux-(version)`. You'll want to check to make sure both files exist.

Step 7: Compile the Kernel Modules

Once you've got the kernel file itself you still have some modules to build. They come from the same source code, but are compiled in a separate command:

```
# make modules
```

This can take quite a bit longer than the compilation of the kernel itself. Better go make tea while you're compiling. When the operation is finished you'll want to male sure the `drivers` directory is full of useful-looking binary files. Those are your modules.

Step 8: Install

Nothing is where it should be, so move the modules to the `/lib/modules/(version)` directory. This simply copies all of the modules, which is why it was important early on to customize your EXTRAVERSION value in the Makefile (see Step 2). If your EXTRAVERSION is the same as another kernel on your system you will overwrite existing modules.

```
# make modules_install
```

Now install Linux itself with this command:

```
# make install
```

Kernel installation is more than a simple copy operation. `make` will copy both the kernel and the `System.map` file, update the symlinks in the `/boot` directory to your new kernel (e.g, from `/boot/vmlinuz` to `/boot/vmlinuz-(version)`), run the `depmod` command to build the module database and create the initial ramdisk.

If you're using the GRUB boot loader you're in luck: this installation command will update your GRUB configuration, giving you a new boot loader option. You can then reboot your system and choose to boot with the new kernel. As long as it has a different EXTRAVERSION name than your old kernel you can revert to the previous system if things didn't go well during compilation. But if a reboot into the new kernel is successful you'll probably want to edit `/boot/grub/grub.conf` so that the new image is the default stanza.

LILO users will unfortunately have to edit the `/etc/lilo.conf` file manually and run the `lilo` command. this edit is not too rough: simply copy another Linux stanza and modify it to suit your needs. Run the `lilo` command and reboot.

Changes in Compiling the 2.6 Kernel

The Linux kernel version 2.6 is brand new as of 2004. It adds a lot mo automation to the build process, and improves the user interface for its graphical configuration.

As with the 2.4 series, 2.6 supports `make config`, `make oldconfig`, `make menuconfig` and `make xconfig`. Now, however, it is no longer necessary to use `make dep` or `make modules`. The standard `make` command will resolve dependencies and create your kernel modules. This makes the build process much easier.

As far as configuration goes, you can now use a new addition to this mix: `make dconfig`. This command will generate a new configuration file for you by using the default values for each kernel variable.

Graphical configuration

If you use the recommended `make xconfig` to customize the configuration of your kernel you now have two improved options. The default `xconfig` option uses the Qt toolkit to display a very KDE-like configuration environment. This new interface is extremely easy to use, though you need the `qt-devel` package installed in order to run it. For those without Qt there is a GTK+ interface that is compatible with the GNOME desktop. You can launch it with the command `make gconfig`.

While both `xconfig` and `gconfig` present excellent configuration interfaces, the default Qt-based xconfig is recommended because it has a slightly more mature set of options and features.

Many changes

The changes from 2.4 to 2.6 are many and varied. While the GUI interfaces make configuration much easier, there are a plethora of new options to configure. Some previously built-in options, like module unloading, are now separate options. It's important to browse and understand all of your options if you'd like a kernel that can boot and be usable. For example, if you don't enable module unloading your kernel will be unable to unload a modular driver when it is no longer needed. It's very easy under 2.6 to create a kernel that will not work properly, so due caution is required.

Importing a 2.4 configuration

The configuration for your favorite 2.4 kernel can be used in building 2.6. As you'll recall from compiling 2.4, the `make oldconfig` command will compare the kernel's variables with those specified in the configuration file. If any variables are missing from the file you'll be prompted for them. This means that you can take an existing 2.4 `.config` file and use it with `make oldconfig` to fill in the new variables.

Installing the compiled kernel

Like the 2.4 kernel, the installation commands are `make install` and `make modules_install`. If you're new to the 2.6 kernel it's entirely possible that your kernel will not boot, or will have some problems recognizing your hardware (for example, if you didn't include the necessary networking support). For this reason it's important to keep your 2.4 kernel as the default until you are sure you want to switch to using 2.6.

21. X.Org: The X Window System

The X Window system is the basis for the graphical user interface (GUI) on pretty much every UNIX system, with notable exceptions like Mac OS X. GNU/Linux distributions tend to use a free implementation of X called X.Org, which is based upon the old Xfree86 project. Think of X.Org as your display and its most basic essentials: the monitor, the keyboard and the mouse.

X is based upon a client/server model, though this is difficult to see on a standalone system, or even in most environments running many UNIX systems. X generally only serves the system on which it runs, so its client/server nature is somewhat hidden most of the time. Part of the purpose of this section will be to show you ways in which you can take advantage of this feature, which is absent from the graphical interfaces of major non-UNIX operating systems.

X is a vendor-independent server that runs on many different hardware types, and is currently on version 11. X11 release 4.x is the current default on most GNU/Linux distributions.

The X Protocol

Yes, protocol. X is a standard for communication, like SMTP is for e-mail or TCP/IP is for data packets. A graphical program running on your system is an X client. The display is the X server. The protocol allows the client and the server to communicate. Because it is an open protocol, and X client on any system can run on an X server on any other system. So you can run a graphical program remotely over X, even if the operating systems are not identical. There is even an X server that runs on Microsoft Windows.

Remote X Clients

The upshot of this is that you can run an X client (a program) located at one host on the X server (display) of another host.

For example, let's say that you you have installed the Mozilla web browser on the host named `dagobah`. But you're currently working at a host named `endor` where Mozilla is not installed. On endor, you can set the X server to allow connections from `dagobah`.

```
$ xhost + dagobah
dagobah being added to access control list
```

Then on `dagobah` you can launch Mozilla, telling the program to launch to the X server on `endor` rather than to the local one.

```
$ DISPLAY=endor:0 mozilla &
```

Note that this operation does not have to be done as the superuser - both the `xhost` command and the launch of the client application itself are performed by normal user acounts.

In this example Mozilla will launch on `dagobah`, but appear (and be usable) on `endor` instead. So you don't have to install the application on `endor` (or, in fact, on any host - some people use the X Window system to run *application* servers). The Mozilla browser will run with the preferences, Internet cache, Bookmarks, saved passwords, plugins and other attributes of the user who launched the clientapp. If you try to use the browser to download a file from the Internet, it will see the filesystem of the system on which it's really running, not the filesystem of the X server where it is being used.

In the command above the DISPLAY variable is used along with the command. You can also use `export` to set this value for all commands ending in the & symbol. So on `dagobah` you would type:

```
$ export DISPLAY=endor:0
```

Then type any future commands like this to send your applications to endor:

```
$ phoenix &
[1] 1296
$ konqueror &
[2] 1324
$ swriter &
[3] 1402
```

The feedback of in each command is the job number and process ID.

Host-Based X Security

Because X runs over TCP/IP networks it is subject to security considerations. After all, you don't want anyone to be able to access your display. So X is configured to reject all clients except local ones, and to only allow clients to be launched by the owner of an X session. The xhost command in the above example adds clients to the access control list, which is the exception to the rule. But just as you can give a client access with the + symbol, you can also take it away with the − symbol:

```
$ xhost + tatooine
tatooine being added to access control list
$ xhost - tatooine
tatooine being removed from access control list
```

If you fail to add the hostname and simply type xhost + you'll effectively remove all restrictions, so use that command either with caution or not at all. Similarly, the xhost − command will block all access.

Note that this security mechanism is host-based. It does not control which users can launch remote X clients, it just limits where those sessions can come from.

User-Based X Security

To control which users can launch X clients on your X server you need to use an MIT-MAGIC-COOKIE. MIT-MAGIC-COOKIE is a protocol used by X to authenticate users. It consists of a 128-bit string of text stored in a user's home directory the .Xauthority file, which can contain multiple cookies. If a remote user has a cookie entry identical to one on the local .Xauthority file, the remote client can launch applications to the local display.

So for example, on your local machine you can display the local MIT-MAGIC-COOKIE with the xauth command:

```
[user1@endor user1]$ xauth list $DISPLAY
endor.test.com/unix:0  MIT-MAGIC-COOKIE-1  dc32710612476ca3822f0a9d645a1208
```

Then you can connect to the remote host and add the above key:

```
[user2@speakwrite user2]$ xauth add $DISPLAY . dc32710612476ca3822f0a9d645a1208
```

Now the user named user2 on speakwrite can launch X clients that will display on endor's X server.

Easy and Secure X Sessions with SSH

SSH (Secure Shell) provides user-based security for the X Window system. With the pam_xauth.so module installed in PAM, you can tunnel X clients though SSH sessions. This is slightly better than the MIT-MAGIC-COOKIE protocol on public networks like the Internet because the cookies are sent between machines without encryption, whereas SSH is entirely encrypted.

SSH is smart enough that it will build a secure tunnel for X whenever you establish an SSH session. You can SSH to a remote machine and launch a graphical program, such as Mozilla. Because SSH is smart

enough to tunnel X sessions, Mozilla will appear on your local X server.

In fact, you can also SSH through multiple hosts to launch remote programs. So you can SSH from Computer A to Server B, and from Server B to Server C, and launch a graphical program on Server C. It will appear right in front of you on Computer A because C will tunnel it to B, which will tunnel it to A.

X Modularity - The Components of the GUI

The X Window system is very modular, so a GNU/Linux system running X can use either of two different desktop environments, three different graphical logins and many dozens of window managers and widget sets. No two graphical environments need look alike.

Desktops

Specified in the file `/etc/sysconfig/desktop`, the two most common desktops are KDE and GNOME.

Why two desktops? And which should you use? Welcome the most contentious subject of the free software world.

KDE was started in 1996 as a free (as in *price*) desktop for UNIX. It was based upon the Qt library, which for was distributed under a non-free license, so free software advocates deemed it unusable and sought out an alternative. Enter GNOME (pronounced *guh-NOME*), which was started in 1997 to provide a free (as in *freedom*) alternative based upon the GTK+ library.

KDE was, and arguably still is, the more developed desktop of the two, and the Qt library is now available under the GNU General Public License. However a lot of momentum and support exists at the GNOME project, and its recent visual and speed improvements have gained it much support. Add to this the fact that Red Hat works primarily with GNOME, actively contributes to its development and ships its distribution with GNOME as the default desktop, and you have a rapidly emerging schism. For while KDE is slightly nicer looking and more customizable, GNOME's user base is growing.

Which desktop you choose to use is up to you. After all, choosing one desktop does not bar you from using the applications associated with the other: you can run KDE applications on GNOME and vice-versa, provided the Qt and GTK+ libraries are both installed. But whichever you choose, don't ever tell anyone that one desktop is quantifiably better than the other. This will get you in the middle of a flame war, and nobody wants that.

Display Managers

Display managers allow you to choose your desktop, set your default desktop and, of course, log in. There are three different graphical login managers available with GNU/Linux. The X display manager, `xdm`, is the default login that comes with X.Org, and has been a part of UNIX systems for many years. `kdm` is a login that comes with the KDE desktop. `gdm` comes with GNOME. There is no particular reason to choose one over another; you can choose, for example, to log in to KDE with `gdm`. At the time of writing `gdm` is probably the most eye-catching because it builds its layout and appearance with very customizable XML themes, distributed in tarred and gzipped archives.

All three managers have the capability (turned off by default) to arrange remote logins, so you could log in with `xdm`, `gdm` or `kdm` on one host and view your desktop on another. For example, the login manager on `host1` can run the command `X -query host2` if X is not yet running, or `X - query host2 :1` to start an additional X session. Or it can run `X -broadcast`, which will broadcast a login request and establish a session with the first X server that replies. This allows you to set up "dumb terminals" that will connect to one or more X servers in a distributed environment. A third option is to run `X -indirect host2`, which will give `host1` a list of servers stored on `host2`, and the user can pick from amongst them.

Widgets

A widget is a visual component of an application window. Menus, scroll bars, radio buttons, check boxes and other graphical elements are all part of widget sets. Qt and GTK+ each have their own widgets; additionally there are many other widgets sets that can be applied to X, with or without a desktop environment.

Window Managers

A window manager is an X client that can encapsulate other X clients. Think of it as a big window that can be filled with other windows. A window manager decorates windows, adds special toolbars, docks or menus and allows you to move, resize or minimize windows.

The GNOME desktop comes with the Sawfish manager. KDE has its own manager called `kwin`. However each desktop allows you to select other window managers, significantly increasing the level to which you can customize your system's look and feel.

Starting X in Run Level 3

While run level 5 will start X automatically, you can launch a session manually from run level 3 using the `startx` command. `startx` will get its configuration from a number of sources, in this order:

- The `~/.xinitrc` file in the user's home directory.
- The `/etc/X11/xinit/xinitrc`. file, if the home directory version doesn't exist.

Whichever file is used, `xinitrc` may also get information from these files:

- `/etc/X11/Xresources`
- The `~/.Xresources` file in the user's home directory
- The keymap files `/etc/X11/Xkbmap` and `~/.Xkbmap`, or the files `/etc/X11/Xmodmap` and `~/.Xmodmap`.

Finally `xinitrc` runs any shell scripts it finds in the directory `/etc/X11/xinit/xinitrc.d`. For example, the `xinput` script sets the input language from your system settings.

Finally, `xinitrc` turns to `~/.Xclients`, or `/etc/x11/xinit/Xclients`, which will load your default desktop, or choose a desktop for you if one is not set as the default.

X Startup in Run Level 5

Run level 5 is the graphical run level, so X will start automatically via the script `/etc/X11/prefdm`, which also selects your display manager (login). The desktop preference in `/etc/sysconfig/desktop` will tell it which manager to launch; for example, the line `DESKTOP="GNOME"` will cause it to launch `gdm`. This does not mean you have to use the GNOME desktop; it simply means that you'll use GNOME's display manager, `gdm`, which can be used to start a session in other desktops. If `prefdm` can't determine the default it will use `gdm`, or `kdm` if `gdm` is not present, or `xdm` as a failsafe.

Still, all three of the display managers use the same configuration file, `/etc/X11/xdm/Xsession`. Prior to the login the `/etc.X11/xdm/Xsetup_0` script is run, and after the login `/etc/X11/xdm/Xsession` starts the session, launching the scripts in `/etc/X11/xinit/xinitrc.d`. This is similar to what `xinitrc` does when you launch `startx` from run level 3.

When you log out in run level 5 the display manager restarts the X server and presents its login again.

X-tensions

X allows for the addition of extensions, and you'll find many of them installed on an average GNU/Linux system if you run the `xdpyinfo` command. The extensions are designed to allow X to perform functions it was not originally designed for. For example, the SHAPE extension allows irregularly-shaped windows, and the GLX extension lets you run OpenGL graphics in X.

Configuring the X server

Fedora Core ships with a utility called `system-config-display`, which checks your video card and can suggest the best X server to use. `system-config-display` also modifies your configuration file, located at `/etc/X11/xorg.conf`.

Note that if you are using an older distribution of GNU/Linux, such as Fedora Core 1, you will have Xfree86, which has somewhat differently-named configuration files. Also, all tools named `system-config-*` were formerly called `redhat-config-*`.

Fonts in X

`xfs` is the X Font Server, and it allows X to render fonts. X version 4 does not need this: it supports TrueType fonts natively, as long as the `freetype` module is included in the Module section of the configuration file `/etc/X11/XF86Config`. But to use TrueType on version 3 servers you need to use `xfs`.

You can also use PostScript fonts, as well as other types. Fonts are stored in more than one place, but are generally contained in subdirectories of `/usr/X11R6/lib/X11/fonts/`. You can specify additional locations in the `xfs` configuration file `/etc/X11/fs/config`.

Because some distributions of GNU/Linux ship with a limited number fonts, people often add freely-available or specially-licensed TrueType fonts. To do this, copy the fonts you want to use into the TrueType directory as the superuser:

```
# cp (location-of-fonts)/*.ttf  /usr/X11R6/lib/X11/fonts/TTF/
```

The enter that directory and rebuild the font directory.

```
# cd /usr/X11R6/lib/X11/fonts/TTF
# ttmkfdir > fonts.scale
# mkfontdir
```

Then reload the font server.

```
# service xfs reload
```

Many use this method to install fonts from another operating system, such as Microsoft Windows, on their GNU/Linux system. Keep in mind if you do this that you need a valid license to such fonts in order to use them.

This method will provide fonts to most X applications, including OpenOffice.

Fontconfig

A new font subsystem called `fontconfig` now ships with Fedora Core and other distributions, and will eventually replace the old core X font system. It allows applications direct access to the font files. `fontconfig` is often used with the `Xft` library, which allows antialiased fonts in applications.

Adding new fonts to `fontconfig` is very easy. Copy the font files into the appropriate subdirectory of

`/usr/share/fonts/` (for True Type fonts you'd create a directory named `ttf` and put them in there. You can also update the font configuration of an individual user by copying them to a directory under the user's `~/.fonts/` directory. After the new fonts have been copied, use `fc-cache` to update the font information cache:

```
# fc-cache <directory>
```

This method will provide fonts to most new applications, and should take effect almost immediately.

22. Mail Services

Sendmail is the Internet's most popular mail transfer agent (MTA). Though some newer MTAs have introduced better security and faster throughput, Sendmail has arguably kept the pace and supports many advanced features, such as the masquerading of users and machines and the use of virtual domains for sending e-mail using one server but multiple DNS domain names.

The Nuts and Bolts of e-mail Delivery

Most users send e-mail using a mail user agent (MUA). MUAs are sometimes called mail clients, and they include programs like Netscape Messenger, the text-based PINE, Eudora and Microsoft Outlook. An MUA sends outgoing messages to a default SMTP server, which then forwards the message to the recipient's MTA.

Sometimes the recipient's MTA is included in the e-mail address, such as with the address wsmith@speakwrite7.minitrue.gov. However, when the server isn't specified it's up to the MTA to do a DNS lookup for the domain's *mail exchanger* (MX). An MX is a default mail server established in the DNS. So for the address wsmith@minitrue.gov, the mail server might find out that the main MX is speakwrite7.minitrue.gov, and it will try to deliver to a user named wsmith on that server first.

The destination MTA might be reachable via one or more intermediate MTAs, but once the message reaches the final MTA, that server hands the message to a mail delivery agent, or MDA. In most configurations the MTA and MDA are one and the same program, and this is the case with Sendmail. However Sendmail can also be configured to pass messages to a separate MDA, such as Procmail for mail filtering.

The SMTP Protocol

Like all MTAs, Sendmail transfers e-mail using a protocol called the Simple Mail Transfer Protocol, or SMTP. SMTP is basically a dialog between two SMTP servers by which e-mail is moved. It operates in clear text on TCP port 25, so you can actually mimic an SMTP server by typing commands manually via telnet. Look up a domain's MX server with the `dig` command:

```
$ dig minitrue.gov mx | grep "MX"
```

Now you can use the `telnet` command to connect to this mail server, specifying the port number after the hostname:

```
$ telnet mx1.minitrue.gov 25
```

You'll probably be greeted with a response like this one:

```
Trying 10.0.2.24...
Connected to  mx1.minitrue.gov.
Escape character is '^]'.
220 mx1.minitrue.gov ESMTP Sendmail 8.12.8/8.12.5; Sun, 23 Mar 2003 18:45:36 -0500
```

Type some commands to the SMTP server. When you are finished type ^] (that's the Ctrl button plus the right square bracket). Here are some useful SMTP commands (in bold) and their replies.

```
HELO whitehouse.gov
250 mx1.minitrue.gov Hello localhost.minitrue.gov [10.0.2.220], pleased to meet you
MAIL FROM: president@whitehouse.gov
250 2.1.0 president@whitehouse.gov... Sender ok
RCPT TO: wsmith
250 2.1.5 wsmith... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
```

On the next line you can begin your e-mail. Note that in the above commands the only two e-mail

headers you need are the sender (MAIL FROM:) and recipient (RCPT TO:) addresses. They are essential, which is why they're in the opening dialog. Other headers are optional, and are placed at the beginning of the e-mail DATA, separated from the message body by a blank line. A common optional header is the subject line, so we'll add that, followed by a blank line.

```
Subject: Test from the White House
```

Then we add the blank line, after which we can type the message itself. When we're done we'll type a single dot on the last line. This tells the remote MTA to accept the message.

```
This is a test from the White House.  But not really.
.
250 2.0.0 h2NNjawU029410 Message accepted for delivery
```

After the MTA accepts the message you should type ^] to end your session. You can then check the mailbox of the user named wsmith, and read your hoax e-mail from the President of the United States. As you can see, this protocol is very simple indeed; whenever you send an e-mail, your SMTP server conducts this dialog with a remote SMTP server, much as as you see above. Usually this dialog also involves complex headers and the addition of MIME for attachments or HTML formatting. But it's all just text over telnet.

User Access

Once the MTA and MDA have done their work, users can access their mail locally, or from other machines via remote mail protocols like POP and IMAP.

Sendmail's Configuration Files

Sendmail reads its configuration from the file `sendmail.cf`. The file contains header rewriting functions, alias directives, relaying rules and other important MTA options. It is a severely complicated file, so you should edit it with caution, comment it liberally and back it up occasionally. Even so, it's recommended that you not edit the file directly, but rather use a special `m4` configuration method described later on.

Additionally the file `/etc/mail/local-host-names` contains valid host names that Sendmail is allowed to use. You can edit this to customize your server's domain name; for example the server `mx1.minitrue.gov` can be made to send and receive mail for the domain `otherdomain.org`.

The file `/etc/aliases` sets aliases for local users. Whenever you update this file you should hash its data into the file `aliases.db` using the `newaliases` command.

The `/etc/mail` directory also contains the access control list in the `access` file, the virtual users database in `virtusertable` and the m4 configuration source file `sendmail.mc`.

Configuration with `m4`

As hinted above, the `sendmail.cf` file is awful. Its syntax is incomprehensible to anyone but experts and there are so many options your head is likely to start spinning if you get too involved in the file. This is where `m4` comes in. `m4` is a macro language. You can use it to help you configure `sendmail.cf` without directly editing it. Fedora Core uses the macro configuration file `sendmail.mc` file to edit `sendmail.cf`, and recommends this as a starting point. To use m4, install the `m4` and `sendmail-cf` RPM packages.

The macro configuration file contains information about the operating system, the location of important files, user table locations and other features.

Comments can be made with the letter sequence `dnl`, which appears at the end of each line; putting it at the beginning of the line causes `m4` to ignore the rest of the line.

A must-do for mail servers

Fedora Core's default configuration doesn't allow any connections to Sendmail except from the local machine. If you want to configure your machine is an SMTP server for other machines you can fix this by editing this line in the `sendmail.mc` file:

```
DAEMON_OPTIONS(`Port=smtp,Addr=127.0.0.1, Name=MTA') dnl
```

Move the letters `dnl` to the beginning of the line, effectively commenting it out. Then run the m4 command to rebuild the Sendmail configuration file.

Other `sendmail.mc` options

Enabling `FEATURE(`relay_based_on_MX')` allows relaying if the same Sendmail server is the MX of the target domain. So if your machine is the MX of more than one domain, this option can allow relaying internally between the domains. After all, if you can't trust yourself, who can you trust?

Uploading Changes to `sendmail.mc`

Sendmail is not aware that the sendmail.mc file exists. To implement your changes to that file you have to merge it with your `sendmail.cf` file. This requires the the `m4` command:

```
# m4 /etc/mail/sendmail.mc > /etc/mail/sendmail.cf
```

You will have to do this each time you make changes.

Anti-Spam Features in Sendmail

The latest versions of Sendmail are designed with features to help you guard against unsolicited commercial e-mail, also known as *spam*. Sendmail disallows e-mail relaying (i.e., sending mail from a remote location to another remote location using your local machine), it checks an access database before accepting mail, and it can also perform checks on e-mail headers.

Blacklisted Recipients

The `FEATURE('blasklist_recipients')`, enabled by default, will block all mail for specified recipients in the `/etc/mail/access` file. Recipients in that file that have a REJECT setting will be blacklisted.

Rejecting Unresolvable Domains

Many spammers send mail from phony domains. By default Sendmail will look up each sender and reject the mail if the domain name is invalid. Unfortunately this can cause problems with laptop and dial-up users on the local network, since they may have domain names that are invalid. Because of this, Fedora Core disables this feature with this line:

```
FEATURE(`accept_unresolvable_domains')dnl
```

So completely bogus e-mail will be accepted. If you think you won't encounter these problems because your machine will have constant DNS access, move the `dnl` to the beginning of the line to effectively comment it out.

DNS Blacksists

Enabling a line with `FEATURE('dnsbl')` will point Sendmail to a DNS Blacklist. This is an online resource for DNS lookups that tells Sendmail whether a given sender has been identified as a spammer, and accepts or rejects their mail accordingly. By default the feature will try to use the MAPS (Mail Abuse

Prevention System) resource at `blackholes.mail-abuse.org`, but with a few options you can use other blacklists. To configure your server for the popular Spamcop service you'd do this (one one line):

```
FEATURE(`dnsbl', `bl.spamcop.net', `"Spam rejected -
see: http://spamcop.net/bl.shtml?"$&{client_addr}')dnl
```

You are allows to have more than one `dnsbl` line, so you can configure set your server to use multiple blacklists.

Other Files in `/etc/mail`

In the Sendmail directory `/etc/mail`, a few other files are used by the mail server. The files in this directory have to be hashed into DB files in order for Sendmail to use them. To hash the entire directory, run the `make` command. This uses the Makefile conveniently located in the directory to rebuild all the databases. You can also restart Sendmail via its System V script:

```
# service sendmail restart
```

The System V script will rebuild the databases as part of its startup process.

The `/etc/mail/access` File

The `access` file allows you to set up special access rules for incoming mail. First an foremost this lets you set up rules for relaying mail. This configuration would allow relaying from the local host, from the `minitrue.gov` domain or from the 10.0.0.0/16 subnet.

```
# by default we allow relaying from localhost...
localhost.localdomain           RELAY
localhost                       RELAY
127.0.0.1                       RELAY
# Allow local senders by domain and IP range
minitrue.gov                      RELAY
10.0                            RELAY
```

You an also block unwanted senders by adding their addresses, domains or IP addresses to your table like this:

```
spamminitrue.gov          REJECT
idiot@virus.com           ERROR:"550 E-mail rejected due to viruses."
evildomain.gov            DISCARD
```

The specific values you can use are:

- `REJECT` rejects the sender with a general message.
- `OK` accepts mail from the sender even if other rules, such as a failed DNS lookup, would cause a rejection.
- `RELAY` accepts mail for RELAYING
- `DISCARD` rejects messages but does not send a reply message. Very harsh.

You can also use a specific error code as described in RFC 821. Error 550 is a common error code for rejections, and can be followed by an error message to be returned to the sender. Feel free to be rude if you sender is a spammer.

The `/etc/mail/virtusertable` File

If your machine is the mail server for one or more virtual domains, this file allows you to map the virtual domains and specific virtual addresses to real addresses. Some examples:

```
manager@domain1.com       user1
manager@domain2.org       user2
```

```
admin@domain3.com          xyz@aol.com
@domain4.com               abc@yahoo.com
@domain5.com               %1@example.org
```

Note that the user name in both of the first two e-mail addresses is *manager*. Sendmail has no problem with this because both are in different domains. The last two lines map entire virtual domains; the first one to a specific user. The final example sends mail to a user with the same name in the mapped domain, so `wsmith@domain5.com` will map to `wsmith@example.org`.

The `/etc/mail/local-host-names` File

This file must contain any FQDN for which the mail server will receive mail. This includes the server's host name and any aliases it may have. If the server is the MX server for a domain, the domain must be in this file as well.

SASL Authentication in Sendmail

Since the release of Fedora Core 1 Sendmail has been compiled using version 2 of the Cyrus SASL librar. This is a general purpose authentication library that you can use if you plan to provide authenticated SMTP - the use of a username and password for sending e-mail. Such a service would allow you to control who uses your mail server, regardless of where they are located. You could let authenticated users send mail from anywhere on the Internet without making your server an open relay.

To get it up and running, do the following:

1. If you use an authentication service like NIS or LDAP, change the System V script located at `/etc/rc.d/init.d/saslauthd` to use PAM rather than your local `/etc/shadow` file. You can do this by finding the the line that reads `MECH=shadow` and change it to `MECH=pam`. You must also edit the `/etc/sysconfig/saslauthd` service configuration file with the same information.

2. Configure `saslauthd` to be on and then start the service. `saslauthd` is a System V service, so it can be started with the `service` command and enabled by default with the `chkconfig` command.

3. In Sendmail, make sure you have these three lines:

```
define(`confAUTH_OPTIONS', `A')dnl
TRUST_AUTH_MECH(`DIGEST-MD5 CRAM-MD5 LOGIN PLAIN')dnl
define(`confAUTH_MECHANISMS', `DIGEST-MD5 CRAM-MD5 LOGIN PLAIN')dnl
```

4. Test your authentication system directly before you implement it by using the `testsaslauthd` command, like this:

```
# testsaslauthd -u user -p pass
```

Debugging and Troubleshooting Sendmail

The `mail` command is one of the simplest MUAs, or mail clients. Using it with the `-v` argument can help you to troubleshoot Sendmail. It sends mail verbosely, and will show you all of the steps in the process including any errors encountered. Run the command like this:

```
$ mail -v user
```

Type your message, and then press Ctrl-D when done.

You can also troubleshoot Sendmail by checking the logs. The tail command can show you the end of a file; using the `-f` argument will show you the changing log in real time. For example:

```
# tail -f /var/log/maillog
```

Procmail for Local Delivery

Procmail is a post-delivery tool that allows you to manipulate or filter incoming mail. It has a number of different uses, including:

- Forwarding mail to other addresses
- Starting a script or program whenever e-mail is received
- Sorting incoming mail into different folders or files.
- Preprocessing (and possibly filtering) e-mail

Configuring Sendmail to use Procmail

To get started with Procmail you have to be sure your mail server is configured to use it. For Sendmail, make sure the following line is in `/etc/mail/sendmail.mc`:

```
FEATURE(local_procmail,`',`procmail -t -Y -a $h -d $u')
```

If you add this line, make sure that you use `m4` to build your `sendmail.cf` file and then restart Sendmail. See the above sections on Sendmail for information on how to do this.

Your `.procmailrc` file

The most simple way to configure Procmail is to create a `.procmailrc` file in the home directory of the user who needs mail filtered. We'll create such a file, and put these values in it:

```
VERBOSE=no
MAILDIR=$HOME/mail
PMDIR=$HOME/Procmail
INCLUDERC=$PMDIR/rules.rc
```

This sets up a folder for Procmail folder in your home folder and allows you to create a file for your list of rules. It also establishes the `mail` folder under your home directory as the location of your mail folders. This is a common location for mail folders on many UNIX systems, though yours may vary.

Now you can create the `Procmail/rules.rc` file, and in that put a list of rules for incoming mail. To start with a simple one, if we want to filter all mail from a particular sender into a folder for filtered mail, we'd do something simple like this:

```
:0
* ^From.*filtered.user@filtered-domain.com
Filtered
```

Now any mail with the address `filtered.user@filtered-domain.com` will be moved into the folder named Filtered.

For a more complex example, if you want to forward any mail that Winston sends you about HTML to your friend Lando, and also send a copy to your HTML folder, you'd put something like this in your rule file:

```
:0
*^From.*winston
*^Subject:*HTML
{    :0 c
! lando@bespin.mil
:0
HTML
}
```

This not only looks for an address containing the word "winston," it also looks at the subject line and filters the mail appropriately. For more examples of using this filter see the man page for `procmailex`. I contains many examples of filters for you to use in a rule file.

Configuring Sendmail as a Client

You can use Sendmail on client machines and point them to a central SMTP server. This is useful in that it allows you to set filters and rules centrally, and more easily troubleshoot mail transfer issues.

To configure Sendmail to use a central gateway, edit the `sendmail.cf` file directly. Change the following DR (forwarding agent), DS (smart relay), DH (forwarding host) and DM (masquerade) lines as follows:

```
DRmail.minitrue.gov
DHmail.minitrue.gov
DSmail.minitrue.gov
DMminitrue.gov
```

Some of these values might not yet exist. If they are absent or blank, Sendmail assumes the values to be the local host.

You must also set the central SMTP server to allow relaying from the client hosts.

Setting up POP and IMAP Mail

If you are running a mail server, chances are you'll have to set up your system so that users can get their mail. You can get mail from your server on a client machine with the POP3 or IMAP4 protocol. Whichever protocol you choose, both are `xinetd`-based services.

If you are using a firewall, please keep in mind that POP3 runs on port 110 and IMAP runs on port 143, so if you set up either server you'll have to open the appropriate port.

POP vs. IMAP

The POP and IMAP protocols are inherently different. POP is designed to allow the mail client program to download e-mail from the mail spool on the server (where on GNU/Linux it is generally stored in `/var/spool/mail`) to the client. By default this process removes the e-mail from the server, reducing the mail spool file to zero size. This is ideal for a server that has limited storage capacity.

The downside to POP mail is that if users check mail from more than one client, each client gets a different set of mail. Once messages are downloaded on one client host, they will be unavailable on another. To make up for this POP has the ability to leave messages on the server, either permanently or for a set period of time. As long as messages are left on the server they can be downloaded elsewhere, but this introduces the added caveat of having duplicate e-mail stored on multiple hosts.

Enter IMAP. The IMAP protocol sets up multiple folders for e-mail on the server, and keeps the messages on the server even when you view them with a mail client. In this way the entire folder structure for each user's e-mail will look identical when viewed from any client machine. The spool file effectively becomes the Inbox, and other folders are created as files in the users' home directories. The downside to IMAP is that it tends to require more disk space on the server.

POP and IMAP Clients

There are countless client programs for GNU/Linux that are compatible with the POP and IMAP protocols, including Netscape, Mozilla, Ximian Evolution, and KDE's KMail client.

Installing POP or IMAP from RPMs

On Fedora Core systems the POP and IMAP daemons are both provided in the `imap` RPM. This package installs `ipop3d`, `ipop2d` (for compatibility with the older POP2 protocol) and `imapd` (the Cyrus IMAP daemon).

Once installed, you'll have to configure `xinetd` to accept connections for POP or IMAP mail. The easiest way to do this is to use `chkconfig`:

```
# chkconfig ipop3 on
# chkconfig imap on
```

This modifies the appropriate file in the `/etc/xinetd.d/` directory for each service by changing the `disable =` line from `yes` to `no`.

Dovecot

Fedora Core 1 was released with an POP3 and IMAP daemon called Dovecot. Dovecot uses index files to store a mailbox's state, so even with large mailboxes it is reportedly much faster than the UW IMAP or Cyrus IMAP daemons.

To use Dovecot, you have to first make sure that the default IMAP and POP3 servers are turned off, with these commands:

```
# chkconfig ipop3 on
# chkconfig imap on
```

Both run under `xinetd`, so they do not need to be stopped.

Now you can use Dovecot without running into a conflict. Unlike the default services, Dovecot runs continuously using a System V script. You can configure it in `/etc/dovecot.conf`. Its configuration file is very straightforward, and in general you can comment or uncomment the lines you need, and examples are provided for alternative configurations.

Vacation Messages

If one or more of your users want to leave an outgoing vacation message on their e-mail they can do it with the Vacation program. Vacation can be installed from source, or from RPM packages available online. You can instruct Sendmail to use Vacation for a given user by adding a line to a `.forward` file in his or her home directory.

Sendmail Restricted Shells

Sendmail will not blindly run programs that your users add to their `.forward` files. For security's sake each program you want Sendmail to run must be specified as one of its restricted shells, which are specified using symbolic links in the `/etc/smrsh/` directory.

After you install Vacation, change to `/etc/smrsh/` and create a symbolic link to the Vacation binary like this:

```
# ln -s /usr/bin/vacation vacation
```

Sendmail will then allow the Vacation program to run from a `.forward` file in the user's home directory. But before we create that we should first initialize the Vacation program.

The message

Create a file in the home directory called `.vacation.msg`. At the top of the file create an e-mail header to customize the subject and other fields. Then skip a line and write your message. The result should look something like this:

```
Subject: Out of the office reply (Re: $SUBJECT)
Delivered-By-The-Graces-Of: The Vacation Program
Precedence: Bulk
```

```
Hello,

I will be out of the office until Wednesday, April 3rd.  I have received
Your e-mail and will respond to it upon my return.  If you have an
emergency please contact Jane Doe at x4321.
```

Now run the vacation command:

```
# vacation -I -t0
```

This will create the `.vacation.db` file, which is a database that the vacation program will keep to remember which users have received auto-replies, and same address repeatedly.

The `.forward` file

To start the vacation messages create a file named `.forward` in the user's home directory. In it, add a line like this:

```
\username, "|/usr/bin/vacation -a username@domain.com -t0 username"
```

This specifies two processes for all incoming mail, separated by a comma. The first (`\username`) tells Sendmail to deposit the message locally. The second (in quotes) pipes the message to the vacation program., which has its own syntax.

Vacation's `-a` option lists an alias, which is useful it the user has more than one e-mail address. List any applicable addresses, each in its own `-a` option. The `-t` option followed by a 0 is the number of days to wait before sending a vacation reply to a repeat sender, so `-t0` will send out a reply every time, whereas the default setting is to wait one week and not annoy the sender multiple times.

A Vacation-only shell

The only potential problem you may encounter when using the Vacation program is that it requires that users have a valid shell to execute the program, since Vacation runs under the user's UID. On some systems, particularly mail servers that support many users, you may have a policy of withholding shell access for security reasons. But without it, Vacation won't run.

To create a workaround for this you can create a special shell for Vacation users. Create a file called `/etc/vacation-only`, and add it to the `/etc/shells` file to establish it as a valid shell. In it, add a script like this:

```
#!/bin/bash
#
trap "/bin/echo Sorry; exit 0" 1 2 3 4 5 6 7 10 15
#
/bin/echo
/bin/echo "*******************************************************************"
/bin/echo "   You are NOT allowed interactive (shell) access on this server."
/bin/echo
/bin/echo "    User accounts are restricted to e-mail and webmail access."
/bin/echo
/bin/echo " For shell access or other questions please contact the Sysadmin."
/bin/echo "*******************************************************************"
/bin/echo
#
# C'ya
#
exit 0
# end of script
```

This shell is a simple script that will allow programs to be executed under a user's UID, but will not allow the user to log in via Telnet or SSH. It presents a more secure way to allow users to use the Vacation program without giving them a real shell.

Mangling Your E-mail With a Sendmail Milter

Recent Sendmail versions allow you to set up a mail filter, or *milter*. A milter can examine messages and modify them according to rules you specify. For example, you could strip dangerous Windows attachments from all incoming mail, add an annoying legal disclaimer to all outgoing mail, or prevent people from using HTML mail.

This section will help you install a popular Perl-based milter called MIMEDefang. MIMEDefang has many uses, but we'll focus on two and show how to block potentially harmful attachments and set up an annoying legal disclaimer, which in MIMEDefang is called a *boilerplate*.

We don't recommend that you use a boilerplate, however you may have no choice. Many organizations, prodded by legal concerns, have erroneously chosen to implement boilerplates even though they decrease the performance of mail servers, serve no demonstrable purpose and are a nuisance for postmasters and users.

Required Components

In addition to the basic Sendmail components, you'll also need Sendmail's configuration and development utilities. On Fedora Core these are in the `sendmail-cf` and `sendmail-devel`. Make sure that these are installed.

Perl Modules

Before you can configure and use MIMEDefang you'll need to install Perl 5.001 or later. You'll also need the following Perl modules:

- MIME-tools 5.411a or higher
- IO-stringy 1.212 or higher
- MIME-Base64 2.11 or higher
- MailTools 1.1401 or higher
- Digest-SHA1 2.00 or higher
- HTML-Parser
- HTML-Tagset
- libnet
- Mail-Audit
- Time-HiRes

Some of these are available in RPM packages that came with your distribution, but at least a few are not. The best place to start downloading them is on the Roaring Penguin Software website's MIMEDefang page at *http://www.mimedefang.com*. Under the heading “Prerequisites” you’ll find some of these modules, including a special patched version of the MIME-tools module without which MIMEDefang will not properly run.

For the remaining modules you should go to the CPAN website at *http://www.cpan.org/modules/01modules.index.html*. This is the official source for Perl resources on the web.

The modules come as tarballs, and when you download the modules you should put them somewhere temporary like `/tmp` and unpack them. Then go into each module's directory and run the following commands to install them:

```
# perl Makefile.PL
# make
# make test
# make install
```

Once all of them install you can remove both the tarballs and the source directories.

Install MIMEDefang

Once the modules are installed you can install MIMEDefang, but first you should create a system account for it:

```
# useradd -r defang
```

Now create and customize the spool directory:

```
# mkdir /var/spool/MIMEDefang
# chmod 700 /var/spool/MIMEDefang
# chown defang.defang /var/spool/MIMEDefang
```

Then create the directory `/var/spool/MD-Quarantine` with the same ownership and permissions.

Put the tarball containing the MIMEDefang software, which you can download from *http://www.roaringpenguin.com/mimedefang*, in your `/usr/src` directory and uncompress it. Change to that directory and compile your software:

```
# ./configure
# make
# make install
```

This will compile and install MIMEDefang. You should then configure it to start automatically by putting a System V init script into your `init.d` directory. There is one included with the source, so just copy it like this:

```
# cp /usr/src/mimedefang-2.X/examples/init-script /etc/init.d/mimedefang
# chmod 755 /etc/init.d/mimedefang
```

This lets you start and stop MIMEDefang with the `service` command, but for reasons of automation you'll want to create symlinks form the various run level directories to make sure MIMEDefang will automatically start and stop depending on your system's run level. Run levels 3 and 5 are probably the only two in which you'll want to start it, but you should also configure it to stop whenever you shut down (run level 0), reboot (run level 6), or go into run level 1 for maintenance.

Later, when you configure Sendmail to use MIMEDefang as its milter, you'll want to make sure that MIMEDefang is be running in the background whenever Sendmail is running. This means that MIMEDefang must start before Sendmail starts, and stop after Sendmail stops.

Since `init` processes System V scripts alphabetically, choose a number for the beginning of your symlink names that are one number *lower* than Sendmail's for the start links, and one number *higher* than Sendmail's for the stop links. For example, if Sendmail starts with a link called `S80sendmail` and stops with a link named `K30sendmail` (the default for Fedora Core systems) you should go into `/etc/rc.d/rc3.d` and `/etc/rc.d/rc5.d` and make start links with this command:

```
# ln -s ../init.d/mimedefang S79mimedefang
```

And then go into `/etc/rc.d/rc0.d`, `/etc/rc.d/rc1.d` and `/etc/rc.d/rc6.d` to make stop links with this command:

```
# ln -s ../init.d/mimedefang K31mimedefang
```

Once you've done this, test it by starting and stopping MIMEDefang. There is no danger in doing this because Sendmail is not yet aware it exists.

Tweaking the filter

Before you start using MIMEDefang you should tweak the filter. The `mimedefang-filter` file in `/etc/mail` is a Perl script, and so knowledge of Perl is handy. But even without a Perl background it should be fairly simple to add or remove snippets of code to suit your needs. Do this with caution: each time you make a change you should back up old versions of your filter so that you can restore the previous working version at a moment's notice.

The first thing you might want to do is remove some settings, including those for SpamAssassin and antivirus features. In these instructions we did not install SpamAssissin, nor did we install an antivirus database, however the default filter looks for both. Some of this may be unnecessary on your system – in fact it could seriously slow it down, particularly if your hardware's capabilities are limited, or if your e-mail throughput is noteworthy.

Make a backup copy of your filter, then go in and remove bits of it. Whenever you remove a section, pay close attention to the { and } brackets to ensure that you are not removing excess brackets used by a higher-level function.

To disable the virus scanning, delete the sections that begin with the following comments:

```
# Scan for a virus using the first supported virus scanner we find.
# Scan for a virus using the first supported virus scanner we find.
# Scan for viruses if any virus-scanners are installed
# Lower level of paranoia - only looks for actual viruses
# Higher level of paranoia - takes care of "suspicious" objects
# Virus scanremove SpamAssasin, delete the section led by this comment:
```

To disable SpamAssasin, delete the section that begins with this comment:

```
# Spam checks if SpamAssassin is installed
```

The very last function in the default filter removes HTML e-mail parts of a message if a suitable text part is available. This can be a good thing: HTML e-mail is bloated and can carry malicious scripts, so disabling it can be a good security measure that also saves storage space on your server. However, many people enjoy using fancy fonts and colors in their mail, so you might not have the option of banning HTML. If you want to allow HTML mail, comment out this line:

```
remove_redundant_html_parts($entity);
```

Loading filter changes

Once you make changes it can be a pain to restart your mail server. You shouldn't restart MIMEDefang without first stopping Sendmail, so it stands to reason that every time you want to load changes to your filter you have to stop Sendmail, restart MIMEDefang and the n start Sendmail again. Fortunately this is not the case. If you use a System V init script to start and stop MIMEDefang, as we have in this example, you can load your changes like this:

```
# service mimedefang reread
```

This will will cause MIMEDefang to rescan the filter and start using the new version immediately, but will not interfere with existing slave processes. So your mail can continue uninterrupted.

The boilerplate

Your annoying corporate disclaimer can be added using the `append_text_boilerplate` and `append_html_boilerplate` functions. Each is separate. First you should define your boilerplates. In the `sub_filter_end` section, look for this text:

```
# No sense doing any extra work
return if message_rejected();
```

Under this, define your boilerplates as string variables. For example:

```
# Text and HTML Boilerplates
    my($textboilerplate) = "\nThis is a disclaimer\. Just ignore it\.";
   my($htmlboilerplate) = "\n<p><small>\nThis This is a disclaimer\. Just ignore
it\.</small>";
```

Note that the backslash symbol is included before all periods, which in Perl-speak "escapes" the periods so that Perl does not interpret them as code. Note also that while the text itself is identical, we've added HTML tags to the HTML part of the message, which in this example makes it appear smaller than normal text. Finally, the \n denotes a new line. You should use this for long lines to break them up in the final message, or use several in a row to set the disclaimer further below your e-mail text.

Now we'll use the boilerplate function, like this:

```
    if (($RelayAddr =~ "^10\.0\.") && ($Helo !~ "server2\.example\.org") && ($Helo !~
"server3\.example\.org")) {
      append_text_boilerplate($entity,$textboilerplate, "0");
      append_html_boilerplate($entity,$htmlboilerplate, "0");
    }
```

This is a complex conditional, and is worth explaining and tweaking. In fact, you can remove the if statement entirely and remove the { and } brackets from the functions if you'd like. But some admins will want to keep these conditions intact or tweak them, because they do some important things.

- The `($RelayAddr =~ "^10\.0\.")` section causes MIMEDefang to add the boilerplates only when e-mail is sent from within the 10.0.0.0/16 IP address range. Since this applies only to local machines, this statement prevents it from being added to *incoming* messages, which would be even more annoying and pointless than the disclaimer itself. Tailor it to suit your own network's range.
- The `($Helo !~ "serverX\.example\.org")` sections are special add-ins. They look to see if the host names server2.example.org and server3.example.org are included in the HELO section of the SMTP session. If they are, the boilerplate is not added. The reason this is included is to show you how to prevent the boilerplate from being added multiple times to a single message in environments that use multiple mail servers. Since other mail servers are likely to be in the same IP address range as that specified in the `$RelayAddr` section explained above, the boilerplate would be added to incoming mail from those local servers.

Test the Filter

Before using a filter you should test it like this:

```
# /usr/local/bin/mimedefang.pl -test
```

You can test other versions of your filter with the -f option:

```
# /usr/local/bin/mimedefang.pl -f mimedefang-filter-20030501 -test
```

It's always good to verify that a program will work before turning it on.

Telling Sendmail about MIMEDefang

The final step is to configure Sendmail to route all mail through MIMEDefang. Edit your `/etc/mail/endmail.mc` file and add this line to the file:

```
INPUT_MAIL_FILTER(`mimedefang', `S=unix:/var/spool/MIMEDefang/mimedefang.sock, F=T,
T=S:1m;R:1m')
```

Keep in mind that the above should appear on a single line, not wrapped as it may appear above.

Now re-run the `m4` configuration command to rebuild your `sendmail.cf` file.

```
# m4 /etc/mail/sendmail.mc > /etc/mail/sendmail.cf
```

That's it. Sendmail should now be routing your mail through MIMEDefang.

Making MIMEDefang Faster

Because it runs in Perl and involves ripping apart and then reassembling e-mail messages, MIMEDefang can be slow, particularly on older machines. One thing you can do to improve its performance is to create a RAM disk for its spool directory. A RAM disk is a virtual disk that exists in memory, and that should be very fast for read and write operations.

To set up a RAM disk for MIMEDefang, first calculate the amount of space needed. You can do this by looking in your Sendmail and MIMEDefang settings to see what the largest allowed message size is (Sendmail's `MaxMessageSize` parameter) and the maximum number of Perl slave processes allowed. Multiply them. So for example, if you limit mail messages to 10 MB, and MIMEDefang allows up to of ten slave processes, your spool directory would have to be 100 megabytes in size

So stop MIMEDefang and Sendmail, if they are running, and empty the MIMEDefang spool directory. Then add the following line to `/etc/fstab`:

```
none  /var/spool/MIMEDefang  tmpfs  mode=700,size=128M,uid=defang,gid=defang 0 0
```

Then run this command to mount it:

```
# mount /var/spool/MIMEDefang
```

When you restart MIMEDefang and Sendmail they should use the RAM disk for message filtering, which in theory will be a great deal faster than using a physical disk.

Postfix - A Better MTA?

As you may recall, Fedora Core is capable of switching between equivalent programs for particular services. This can be done with the `alternatives` command. The alternative to Sendmail is called Postfix. It is designed as an improvement upon Sendmail in a few key areas:

- It's faster. A desktop-class system running Postfix can send ant receive up to a million messages per day. It does this by using tricks developed for web servers to reduce the overhead of process spawning and filesystem use.
- It's easy. You can actually read and understand the configuration files for Postfix, unlike Sendmail's brutal `sendmail.cf`.
- It's secure. There is no direct connection between the agent that accepts mail from the network and the local delivery agent. Postfix consists of many small programs that run independently and each perform a single task.
- It's compatible. Postfix supports `.forward` files, the `/etc/aliases` file and other common Sendmail staples. So you can migrate without much planning.

If you'd like to use Postfix you can activate it with the `alternatives` command:

```
# alternatives --set mta /usr/sbin/sendmail.postfix
```

Configuring Postfix

The configuration file for Postfix is `/etc/postfix/main.cf`. The default values are suitableto most environments, but there are a few you'll have to modify:

Set the domain to use for outgoing mail:

```
myorigin = minitrue.gov
```

Set a list of domains for which this server will accept mail:

```
mydestination = minitrue.gov mx1.minitrue.gov virtualdomain.org
```

Establish some relaying rules to allow locals but reject spammers:

```
mynetworks = 10.0.0.0/16 127.0.0.1
```

Which network interfaces should receive mail:

```
inet_interfaces = allows
```

To start Postfix, start the mail server with `service postfix start`.

23. NFS: The Network File Server

NFS is the Network Filesystem, a protocol that lets you share volumes amongst multiple hosts on a network. It was created by Sun Microsystems in 1984 and licensed for free to the industry with the hope that it would eventually become the industry standard for networked file sharing. Today it is ubiquitous on UNIX and UNIX-like operating systems.

NFS is UNIX-specific, and behaves the same way any other filesystem behaves on a UNIX system: an NFS volume from the server is mounted to a mount point on a client. The client machine treats the directory beneath this mount point as if it were local. For this reason a common directory to share via NFS is the `/home` directory, since it lets users view the same data, desktop and other settings on multiple machines.

A Security Note

The most important thing to remember about NFS is that it is entirely insecure, and requires a fair amount of user account coordination for it to be useful on machines that serve multiple users. Security is enforced by the client, not by the server, so a file owned by the user with UID 500 on the server is fully accessible to a user with the same UID on the client - even if those two users are not the same person. For all intents and purposes the users are identical, so it is important that the security account information on the client is synchronized with that of the server.

Keeping password and group files synchronized can be a chore, so most system administrators use NIS or LDAP to centralize user account administration. More information on NIS and LDAP are available elsewhere in this guide.

Server Configuration

The NFS server is an RPC daemon, so it requires that the `portmap` service be running. `portmap` responds to NFS requests by calling the `rpc.mountd` service, which can mount and unmount filesystems. If you're using Fedora Core, the NFS server also requires utilities included in the `nfs-utils` RPM. This includes the `nfs` System V service, which translates network NFS requests into requests for the local filesystem.

The `/etc/exports` file

The `/etc/exports` file contains information about how file systems should be exported. It is read by the `exportfs` utility, which will be described a bit more further on. The directories listed in `/etc/exports` must follow a simple format: the shared directory on the left and the allowed hosts (and permissions in parentheses) on the right. For example, let's say we want to export users' home directories and a special public NFS directory, each to separate hosts. We might do this:

```
/home        *.domain.com(rw)
/pub/nfs     10.0.2.0/255.255.255.0(ro),10.0.2.14(rw)
```

This shares the home directories to everyone in the `domain.com` domain with full read/write permissions. The `/pub/nfs` directory is available to everyone in the 10.0.2.0 subnet as a read-only share, but is also available to a special privileged host at 10.0.2.14 with full permissions. As you can see, host names are separated by commas and can include wildcards. IP address ranges can be expressed with an address and netmask pair, as was done in the second line of this example.

Avoid whitespaces, except between the shared directory and its hosts. There should be no spaces between the hosts and their permissions, or between two hosts listed for the same share.

Root Squashing

Because of the security concerns of NFS, the service is configured to thwart connections by the superuser

by default. It does this by converting UID 0 to a more harmless UID. If you'd like the superuser to be able to connect to items in an NFS share you can include the `no_root_squash` option in the share's `/etc/exports` permissions.

To see if the local server is running, use the rpcinfo command:

```
# rpcinfo -p
```

Or use it to see if another host's NFS server is running:

```
# rpcinfo -p nfs-server-name
```

If you can't get the server to work, remember that `nfs` and `portmap` must both be running and any firewall you may be using should allow connections on the RPC port 111.

Server Utilities

The `exportfs` utility, aside from providing the exporting service that reads `/etc/exports`, can also help you work with your exported filesystems. You can use the command with a couple useful options like these:

- `-r` refreshes the export list in the server once the /etc/exports file is updated.
- `-v` displays a list of exported directories on a host.

The command `showmount -e hostname` also lists the available shares on a host.

Client Configuration

On the client side NFS support is included in the Linux kernel, so `nfs` is a valid filesystem type for the mount command. For ease of use volumes can be added to the `/etc/fstab` file along right alongside local filesystems. These are then mounted during boot by the `netfs` service.

The NFS shares from the server example might look like this in `/etc/fstab`:

```
nfs-server-name:/home     /home        nfs          defaults     0 0
nfs-server-name:/shared   /mnt/shared  nfs          defaults     0 0
```

There are a number of NFS filesystem options that can come in handy with NFS mounting (and in this example would be put in the file in place of the word `default`):

- `soft` - A processes that can't access this share will return an error.
- `hard` - A process that can't access this share will be blocked.
- `intr` - NFS requests can be interrupted or killed if the server can't be accessed.
- `nolock` - File locking is disabled
- `nosuid` - Disables the SUID bit on any files in an NFS mount.
- `rsize=8192` and `wsize=8192` - These Buffer IP packets, speeding up NFS throughput.

NFS Clients and the Automounter

The `autofs` RPM contains the Automounter service. This service can mount volumes whenever theyre needed and then unmount them when they are not being used. The `/etc/auto.master` and `/etc/auto.misc` files allow you to configure NFS mounts to be automounted.

24. FTP: File Transfer Protocol

FTP is the File Transfer Protocol, one of the oldest protocols on the Internet. It is widely regarded as an outdated protocol for a couple reasons. For one thing, is insecure. It sends its data in clear text and requires somewhat fancy firewall settings on the FTP server. It also cold be replaced by the suitably secure protocols SFTP (Secure FTP) and SCP (Secure CP); the only obstacle to its replacement is that so many people still use it. In our discussion of the server we'll ignore the firewall considerations, but we'll get to them at the end.

Fedora Core comes with both the `wu-ftpd` and `vsftpd` daemons, and at least a couple FTP clients.

The Server

`wu-ftpd` runs as an `xinetd` service. To enable if you have to open ports 20 and 21 on your firewall and tell `xinetd` about the service with this command:

```
# chkconfig wu-ftpd on
```

The server can be configured for three different kinds of FTP access.

- User access - The FTP server authenticates you with local user names and passwords. File permissions work just as they do on a local filesystem.
- Anonymous access - The server provides service to a user named anonymous, but any password is valid (by convention anonymous users input their e-mail addresses as passwords). The FTP environment is chrooted, meaning the root of the filesystem is restricted to one area of the server's real filesystem for security reasons.
- Guest access - Users get access with permissions defined in the `/etc/ftpaccess` file. User accounts are for the FTP service only, and are not valid for logging in or other services.

Anonymous access requires you to install the `anonftp` RPM. The `wu-ftpd` daemon also has a number of other useful features. If a user requests a normal file but adds the .gz extension in the FTP request, `wu-ftpd` will gzip the file automatically to speed up download time.

Other important files include `/etc/ftphosts`, which allows or denies access to clients on a host-by-host basis, and `/etc/ftpusers`, which contains users who are denied FTP access.

Firewall Considerations for the Server

There are actually two versions of the FTP protocol. The older one is sometimes called Active FTP. It is very simple, and therefore requires a proportionately simple configuration for any server or client firewalls that may be present. It works like this:

1. The client connects to the FTP daemon on port 21. This is the “control” connection.
2. The client gives a command that requires data to be transferred, such as an upload or download.
3. The server opens a TCP session to the client on port 20.

This clearly demonstrates why FTP is theoretically faster than HTTP as a data transfer protocol: login time notwithstanding, once data is flowing the commands and data are transferred on separate ports. So to support this on your firewall the server must have port 21 open for FTP connections (and allow outgoing), and the client must have port 20 open because the *server* is responsible for opening the data connection to the client on that port.

But the complexity of the protocol increased when passive-mode FTP was introduced. It works like this:

1. The client connects to the server on port 21. This, again, is the control connection.
2. The client gives the PASV command, indicating that it would like to transfer data in passive mode.
3. The FTP server acknowledges that it will run in passive mode, and tells the client which port it is

monitoring.
4. The client then starts the data transfer on the port that the server has specified.

The main difference here is the *client*, not the server, starts the data transfer; the server only tells the client which port it would like to use. This can lead to a headache when it comes to configuring your server's firewall. If it isn't done properly the client will successfully connect and execute commands like `ls` and `dir`, but the replies to these commands will not work because the server won't accept connections on its passive FTP ports.

Perhaps the simplest workaround for this is to configure your server's firewall to recognize the "state" of incoming packets, thus allowing established sessions to switch ports without being rejected. To do this on the `iptables` firewall, these lines to your `/etc/sysconfig/iptables` file, right below your existing customized firewall rules, but above the default ACCEPT and REJECT commands:

```
-A RH-Lokkit-0-50-INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Next you'll have to load the FTP connection tracking module into the Linux kernel using the `insmod` command.

```
# insmod ip_conntrack_ftp
```

This will tell Linux to start keeping track of the statefulness of FTP packets. To make this setting permanent, configure Linux to load this module automatically by adding these lines to your `/etc/modules.conf` file:

```
post-install ip_tables /sbin/modprobe ip_conntrack_ftp
post-install ip_tables /sbin/modprobe ip_nat_ftp
```

Now whenever you start `iptables` your system will load the appropriate module and passive FTP should function.

25. DNS on GNU/Linux: Using BIND and other Tools

The Domain Name System (DNS) is the service that allows you to refer to a TCP/IP-enabled computer by its host name rather than by its IP address. In the early days of IP networking each computer maintained its own list of hosts at `/etc/hosts`; this functionality still exists in most machines, but it is far easier to rely on the DNS to supply authoritative translations between addresses and host names. This makes DNS the most mission-critical service on the Internet, as well as on many private networks.

There are two types of translations. The conversion of a host name to an IP address is called a *forward-lookup*. Finding a host name for a given IP address is called a *reverse-lookup*.

Levels of Domains

A domain name is made up of multiple elements, each separated from the other by dots. The rightmost element is called the top level domain (TLD). Examples include .com, .org, .edu, .mil., .gov and many others.

These topmost classifications in the DNS hierarchy are managed by the root name servers, which are maintained by the Internet Corporation for Assigned Names and Numbers (ICANN), who also accredit domain registrars - companies that are allowed to distribute of domain names. It is from these ICANN-approved registrars that you can reserve a domain name. The registrars then take the domain name registration and map the name to the IP address of a lower-level DNS server, per the requests of the domain's owner.

A complete domain name, such as `host1.domain.com`, refers to a specific host and is called a *fully-qualified domain name* (FQDN). Within a local network it is often redundant to use a FQDN, as the host name (in this example, host1) is often enough to locate a specific machine, but on the Internet an FQDN is necessary for finding the resources you need.

Each part of a domain name can be up to 63 characters in length, and an entire FQDN can be no more than 255 characters long.

Domains and Zones

A domain is a complete namespace. In the above example, `domain.com` is a domain that is a subtree of the top level domain `.com`.

Within this `domain.com` subtree is a *zone*, and it is managed by a single name server. Think of a zone as any part of the DNS managed by a single server - in this case it could include the entirety of the `domain.com` domain, including subdomains like `research.domain.com`. Or for reasons of practicality the name server for `domain.com` could establish separate zones for any subdomains and delegate authority to other name servers.

Name Server Masters and Slaves

For any given zone there is a *master* name server. Optionally, you may also set up *slave* servers. Slave servers contain the same authoritative data as their masters, however a slave server is not updated manually. It instead synchronizes with its master server after a change has been made.

The advantage of having one or more slave servers is that it provides a backup for the master.

DNS Clients

A DNS client is called a *resolver*, and name resolution is available as a library to any program that requires it. The resolver uses two configuration files:

- `/etc/host.conf` contains configuration information specific to the resolver library. Each line in the file starts with a keyword, followed by configuration. The most common keyword is `order`, which sets the order in which resolution resources are used. The configuration `hosts,bind` will tell the resolver to check the local hosts file first, then check the DNS server(s).

- `/etc/resolv.conf` contains the IP addresses of the local name servers, plus a `search` line containing the default domain name. The latter option allows the resolver to determine whether a domain name should be assumed for host names. With `domain.com` as the search setting, the hostname `endor` can be resolved into `endor.domain.com`.

When a program requires a name or IP address and the resolver is set to use DNS, it sends out a DNS request to its default server on port 53. It will try to use the UDP protocol, a component of the TCP/IP suite that is very fast but limited in functionality. If the packet content exceeds 512 bytes the resolver will forgo UDP 53 and operate via TCP 53, since the slower TCP protocol can send larger packets.

The name server will sent its reply. This can either be *authoritative* or *non-authoritative*. An authoritative reply is one that comes from a master or slave server for the specified zone. A non-authoritative answer is one that comes from a DNS server that has cached the value. So if, for example, a client in the `domain.com` zone needs to know the IP address of `www.google.com`, it might be able to get it from the DNS server for `domain.com` rather than asking one of the root name servers directly. This cuts down on bandwidth use on the Internet, however a client can be configured to reject non-authoritative replies.

Automatic Configuration via DHCP - or Not

If your network interface uses DHCP to automatically configure its network settings, the DNS servers in `/etc/resolv.conf` will be set automatically. To avoid this in environments where you'd like to leave your DNS settings as-is, set the option `PEERDNS=no` in your adapter's configuration file. For example if you use a dial-up service with a modem on ppp0, you can update `/etc/sysconfig/network-scripts/ifcfg-ppp0`.

DNS Servers

When a DNS server receives a request, it will answer only if it is authoritative for that zone, or if it has cached the answer after receiving an identical request earlier. If these conditions are not met, the server can either forward the request or ask a root name server. Root name servers are specified in `/var/named/named.ca`.

If it forwards the request, the next name server "upstream" may either respond, or forward. This can go on for a few forwards before resulting in a reply; that reply will be cached in all "downstream" name servers as it is passed back down to the resolver, so that the request will not take as long the next time it is made.

BIND

The name server software covered in this section is called the Berkeley Internet Name Daemon, or simply *BIND*. BIND is the most popular name server, and is the standard upon which other name servers are built. It is currently on the version 9 series, which was introduced in 2000 and includes many security enhancements over previous versions.

The BIND name daemon is called `named`. It is configured in `/etc/named.conf` and its various zone files are kept in the `/var/named` directory. It is a System V daemon, tied to the various run levels.

Fedora Core adds an additional file named `/etc/sysconfig/named`. It can contain options that will be passed to the daemon when it starts up.

BIND Configuration

Configuration of the BIND server is kept in `/etc/named.conf`. This configuration includes zones, access controls, and other options.

It is recommended that you comment the `named.conf` file whenever you make changes to it. Comments can be in C format:

```
/* This is a comment */
```

Or in C++ format:

```
// This is also a comment
```

Or in the more common shell script format:

```
# This, too, is a comment
```

Directives for BIND are simple single-word commands such as `options`, `server` or `zone`. The configuration file contains these directives, each followed by blocks of text in arrow brackets: {}. It is important not to forget the brackets. It is also important to remember that in `named.conf`, everything ends in a semicolon.

The `options` Directive

```
options {
   directory              "/var/named";
   forwarders{ 123.45.67.89; };
   allow-query            { 10.0.12/24; };
   allow-transfer         { 10.0.12/24; };
};
```

As you can see, not only does every line end with a semicolon, but sections within brackets also end in semicolons, and their subsections also end in semicolons. The commands seen above are `directory`, which is the directory used to store the DNS zone files; `forwarders`, which is a list of name servers to which BIND can forward requests it can't resolve; `allow-query`, which lists hosts allowed to send queries to the server; and `allow-transfer`, which specifies hosts that are allowed to copy the database. This latter option can limit zone transfers.

The `zone` Directive

The `zone` directive establishes zones for the BIND server. For each zone the server is configured as either the master or a slave, and the zone's database file is cited.

```
zone "domain.com" {
   type      master;
   file      "db.domain.com";
};
```

The filename of the zone database is whatever you'd like it to be, but it's helpful to make it match the zone in some way, so in this case `db.domain.com` makes it clear that the file contains the `domin.com` database. Note that this file must be kept in whatever directory is specified by the `directory` command within the `options` directive. By convention this is `/var/named`.

The example above shows us a zone master configuration; a slave zone's configuration looks very similar. The only way in which it differs is that it specifies the masters for its zone.

```
zone "domain.com" {
   type      master;
   masters   { 10.0.12.3; };
```

```
   file      "db.domain.com";
};
```

When BIND server starts a slave zone, it asks the master for an up-to-date copy of the zone database. It can do this without saving it to a file; however if you do configure a slave zone to save into a database file, as in the example above, the server will only ask the master for the serial number of the zone. If the local file contains the same serial number it will not request the entire zone file.

Reverse Lookup Zones

The previous example shows the configuration of the forward-lookup zone `domain.com`. This resolves hostnames into IP addresses. It is also important for a DNS server to do the opposite. For this reason most zones come in pairs; for every forward-lookup zone there should ideally be a reverse-lookup zone.

The names of reverse-lookup zones end in a special domain name: the `in-addr-arpa` domain. So an example analogous to the `domain.com` example above would look like this:

```
zone "12.0.10.in-addr.arpa" {
   type      master;
   file      "db.12.0.10";
};
```

Note that the IP address is reversed. In the original example we allowed queries and transfers for hosts in the IP addressing block 192.168.1./24. So the domain for the reverse-lookup becomes `1.168.192.in-addr-arpa`, and the zone database file is named appropriately.

The Root Zone

Every DNS server should have special zone called a root zone. This is basically a default zone for queries: if a client's query does not resolve to one of the customized zones, the root zone forwards the request to one of the DNS root servers. They're in the `named.ca` file, and any rare updates that take place amongst the root servers are reflected at `ftp://rs.internic.net/domain`.

A root zone's configuration looks similar to other zone configuration files, except the type of zone is neither master nor slave. It's called a `hint`, as seen below:

```
zone "." {
   type      hint;
   file      "named.ca";
};
```

The Loopback Zone

The local loopback IP address, which tells a machine that an outgoing packet is actually intended to be delivered locally and therefore allows "internal" TCP/IP, is 127.0.0.1. Though not required, this can be reflected in a master zone named `0.0.127.in-addr-arpa`, which looks like other reverse-lookup zones.

Setting Client Lists with `acl`

There are two ways to restrict client access to BIND: a list of addresses or an access control list. A list of addresses can be semicolon separated, like this:

```
allow-query  { 10.0.12/24; 10.0.13/24; };
```

In this case the /24 ending in place of the final octet specifies the network mask of 255.255.255.0, telling BIND to allow hosts in the entire subnet.

But this can get cumbersome with longer lists - and in cases where you'd like to use the same list of addresses for multiple features, such as allow-query and allow-transfer, your BIND configuration file can reach a mesmerizing size and complexity. Fortunately you can avoid this using access control lists - either built-in lists, or custom lists with the `acl` command. Lets look at the four built-in lists:

- `none` - No other systems.
- `all` - Any other system.
- `localhost` - Ony an IP address on the local system.
- `localnets` - Any IP address on a network where the local host has an IP address.

So you could replace the long list used above with a built in list and ensure that only hosts on the local network can query the name server.

```
allow-query  { localnets; };
```

To build a custom list, use the `acl` command in your configuration file:

```
acl "list-name" { 192.168.1/24; 192.168.2/24; }
```

And then use the name wherever you need it:

```
allow-query  { list-name; };
```

Zone Files

The zone files in `/var/named` are simple to read but difficult to generate, as their syntax is rather specific. First off, a commented line can only be done with as semicolon, like this:

```
; 20030213 - EPH - this is a zone file comment
```

No hashes, no slashes. But comment whenever necessary - you'll find it useful later on. Some diligent sysadmins even comment with the date and their initials, like in the sample comment above, because a file that frequently changes would otherwise be hard to keep under control. If you find yourself wondering whether a record is still in use months or years after you assigned it it can be a great help to see something like this:

```
; 20010502 - EPH - winston's color laser printer in room 227
labprinter227   IN      A       10.0.12.13
```

Time to Live

The opening of the file is a `$TTL` directive - this is short for "time to live." By default it will look like this:

```
$TTL 86400
```

This is the default time in which you want a resolver or another DNS server to cache the zone's data; beyond that time the resolver or server should re-query. This ensures that all hosts keep abreast of changes. `$TTL` is expressed in seconds, and in this case there are 86,400 seconds in a full day.

After this come the resource records. Any fully-qualified domain name should end in a dot, like this:

```
ns1.domain.com.
```

Don't forget the dot, because if you omit it BIND will assume that the name should end in the *domain name* for that zone. `ns1.domain.com` without a dot would be interpreted as `ns1.domain.com.domain.com`. The good news is that this feature of the syntax allows you to do refer to the same host mentioned above by the simple name `ns1` with no dot.

Resource Records

Though resource records look a bit different depending upon their purpose, they all have the following syntax: `domain ttl class type rdata.`

The `domain` is the domain name to use. If you put an `@` symbol in this place BIND will use the current domain name for the zone.

The `ttl` specifies a custom time to live, overriding the default `$TTL`.

The `class` is the type of network resource. It is almost always `IN`, which stands for *Internet*. Anything else is pretty much obsolete, so the `class` is the vestigial tail of resource records.

The `type` specifies what the record is used for. There are six common values for a record's `type`:

- `SOA` - Sets the start of authority for the zone
- `NS` - Identifies a name server
- `A` - Maps a host name to an IP address
- `CNAME` - Aliases one host name to another
- `PTR` - Points and IP address to a name
- `MX` - Identifies a mail exchanger

Finally, `rdata` is special extra data for the record.

Types of Records

DNS can be used to list a few different types of records, let's look at each in turn, starting with the first item in any zone file, the Start of Authority (SOA).

SOA: The Start of Authority

The very first resource record after the `$TTL` is the Start of Authority (SOA) record. Here's an example:

```
@    IN   SOA   ns1.domain.com.  root.domain.com (
                        20030220       ; serial
                        28800          ; refresh
                        7200           ; retry
                        604800         ; expire
                        86400          ; minimum ttl for negative answers
                        )
```

The first character is an `@` symbol, referencing the domain; in this case it specifies that the record is for a host in `domain.com`. In fact, the domain could be used in place of the symbol.

After the record type is specified with `IN SOA` we find the name of the Start of Authority server, and then we see `root.domain.com`. This is an e-mail address: the first dot replaces the usual @ symbol, since that symbol has a specific meaning in zone files.

In parentheses we see five values. The *serial number* is used as a version control mechanism; it is increased whenever data changes in the file, and slave servers use this progressive numbering to determine whether or not to update. The *refresh* is the duration a slave should wait before performing an update. The *retry* field is the amount of time to wait to repeat a failed refresh. The *expire* field is the amount of time a slave should bother serving out this data in absence of a successful refresh. Finally, the *minimum time to live for negative answers* tells the server how long it should cache a "no such host" response.

The last four are time fields, and traditionally are expressed in seconds, as is the case above. However you can now simplify things by specifying the time in weeks (W), days (D), hours (H) or minutes (M). Use no

spaces in your notation: so for example 7M would be seven minutes.

NS: The Name Servers

Every master or slave server in a given zone should have an NS record. Here's an example of a couple name servers in the `domain.com` zone:

```
@           IN  NS  ns1.domain.com.
domain.com. IN  NS  ns2.comain.com.
```

Note that you can't make these up from scratch: NS records must match regular host records, which will be described below. Also, when using the domain name remember that is must end in a dot if it is a fully-qualified domain name, so in the second line `domain.com.` ends in a dot. Without this BIND will try to append the domain name (so `domain.com` would become `domain.com.domain.com`).

A, CNAME nd PTR: The Host Records

Most resource records in a DNS zone will be A, CNAME and PTR records. The A records are basic host mappings, and are prerequisites for NS records (described above) and MX records (described below).

```
printserver1 IN A 10.0.2.24
mailserver1  IN A 10.0.2.25
ns1          IN A 10.0.2.26
```

CNAMES, or "canonical names." are aliases that point one name to another. The almost always point to a host that has an A record, though a CNAME could point to another CNAME if you really wanted it that way.

```
pop          IN CNAME mailserver1
smtp         IN CNAME mailserver1
```

PTR records are for reverse-lookup zone files, and specify the octets of an IP address in reverse order. for example:

```
24.2.0.10.in-addr-arpa.  IN PTR printserver1.domain.com.
```

Of course that example lists the whole reversed IP address, translating into 10.0.2.24. If the zone were called 0.10.in-addr.arpa. you could omit the full name and replace it with a truncation like this:

```
2.24  IN  PTR  printserver1.domain.com
```

MX: The Mail Exchanger

MX records tell you the default e-mail server for a given zone. Like NS records, they must refer to a host with a valid A record.

```
domain.com.  IN  MX 10  mailservr1.domain.com
domain.com.  IN  MX 20  mailserver2.domain.com
```

As you can see above, a zone can have more than one mail server. Each one has a priority number, and the lowest-numbered server receives mail first. In this case mail destined for `wsmith@domain.com` will first go to mailserver1. Should the SMTP request for e-mail delivery fail for `mailserver1`, the sending server will next query `mailserver2` for a recipient of that name. This allows you to set up backup mail servers.

MX records do not appear in reverse-lookup zones.

HINFO: Host Info Records

The vestigial organ of the DNS system, HINFO records allow you to announce your preferred name server

platform to the entire world. They tell interested parties what type of hardware and operating system you are using, so no one uses them anymore because of the security risks involved.

Delegating Subdomains to other Name Servers

While you can manage a subdomain from within its parent domain's name server (i.e., as part of the same zone), it's sometimes useful to delegate it to another name server.

Let's say the `domain.com` domain needs to have a separate zone for a subdomain called `research.domain.com`. The first step in setting up a new zone is to set up a record for that zone in the parent zone's file. In this case it would look like this:

```
research.domain.com.  IN  NS   ns1.research.domain.com.
```

Of course you can't have an NS record without an A record to match it, so we must create an A record to establish the IP address of `ns1.research.domain.com`. Of course an A record would naturally go in the subdomain's zone file because the delegated server is in that zone - but if that were the only available A record, a querying resolver would have to find the zone in order to query the server in charge of the zone. An impossibility, so a recordmust be kept in the parent zone as well:

```
ns1.research   IN  A  10.0.3.2
```

The Caching-Only Name Service

Aside from the commonly-implemented *authoritative* name servers, there are also *caching-only* servers. A caching name server has no zone files, but rather acts as a proxy to make queries on behalf of its pool of clients and cache the results. They also provide references called *hints* that point to the root name servers. These hints are kept in the `named.ca` zone file.

To set up a caching name server on a Fedora Core system you should install the `caching-nameserver` RPM. This contains neither BIND nor a zone file, but rather a configuration for use with BIND, which must be pre-installed. For best performance a `forwarders` entry should be added to the options of `/etc/named.conf` and the name server setting in `/etc/resolv.conf` should be set to 127.0.0.1.

The Fedora Core `caching-nameserver` package might interfere with `system-config-bind`, since each utilities introduces its own configuration to BIND.

Configuring the caching-only name server

Since a caching-only server will forward queries to higher-level servers, the first thing to do is to edit the `/etc/named.conf` file that was provided in the RPM. In the options section add the following reference to your higher-level server:

```
forwarders { server-address; };
forward only;
```

With this IP address your server will forward requests to the higher server, but cache name resolutions to ease traffic to the main server. This can be useful in environments with many hosts, since different subnets can receive local caching-only servers via DHCP.

Round-Robin Load Balancing

DNS can be manipulated to set up a non-hardware load balancing system for overloaded hosts. For example, if you had an overworked web server you could augment it with a second server and then add two entries with identical hostnames:

```
www          0   IN  A   10.0.2.30
```

```
www          0   IN  A   10.0.2.31
```

The 0 is a TTL value, so the results of queries will never be cached. This means that queries of the host `www` will lead web visitors to each physical host roughly 50% of the time, and if one server fails the other will take over. Note, however, that traffic to the name servers will increase significantly, so it's a trade-off.

BIND Utilities

There are many utilities available to help you work with the DNS. Three are included in the Fedora Core `bind-utils` RPM.

host

The `host` command shows information about a host. For example, this command will tell you all information about the host named `mailserver1`.

```
$ host -a mailserver1.domain.com
```

And this example will tell you everything about all the hosts in the domain called domain.com. This is called a total zone transfer.

```
$ host -al domain.com
```

dig

The `dig` or "domain information groper" utility is a powerful tool that sends commands directly to the name server, rather than relying upon the local resolver libraries. The syntax of dig looks like this:

```
dig @name-server zone query-type
```

So if we wanted to perform much the same zone transfer as the one done with the `host` command above we would type this:

```
$ dig @ns1.domain.com domain.com axfr
```

The query type can be any of the various DNS record types, so you could also use `dig` to find all of the mail servers in a given zone, like this:

```
$ dig@ns1.domain.com domain.com mx
```

nslookup

The `nslookup` command is deprecated in Fedora Core, but it has been used in the UNIX world for years.

rndc

The `rndc` (name daemon control) utility allows you to control a name server remotely. It runs on port 953 and has the configuration file `/etc/rndc.conf`. It uses ISIG encryption for security.

system-config-bind

`system-config-bind` is a graphical configuration utility for BIND. It keeps its master copy of the configuration in a subdirectory of `/etc/alchemist`, since it uses the Alchemist facility to store BIND data in XML Once you start using the utility you should no longer edit the zone files manually, since the master copy is maintained by `system-config-bind`.

26. Web Services: Using Apache and Tux

There are two web servers freely available for GNU/Linux, and both of them are included in Fedora Core.

Apache is the most widely-used web server in the world, and has been since 1996. It started as a bugfix release for NCSA's original `httpd`, but it quickly grew to eclipse its parent project. At the time of writing it runs on over 62% of the World Wide Web's many millions of servers.

Tux is the world's fastest web server. Extremely lightweight, it runs as a component of the kernel. Tux is only capable of serving static content, so it's not good for dynamic websites. But it can be configured to work with Apache, so it is ideally suited for both static sites and sites with mixed content.

Apache

The Apache web server is a versatile daemon. While it is not multi-threaded, it can deal with multiple simultaneous sessions by spawning processes as needed. It can also pre-spawn, allowing it to deal with increased demand. Apache supports many auxiliary behaviors, such as secure HTTP sessions, by loading modules dynamically.

Apache can support many virtual hosts, allowing a singe GNU/Linux server to act as a web server for multiple domains.

Configuration

Apache's configuration files are stored in `/etc/httpd/conf`. The main configuration file is `httpd.conf`. Over time the server has evolved to support a wealth of web hosting options. For example:

- You can fully customize the number of spare HTTP daemon processes, using minimum and maximum settings. This lets you balance memory utilization (important on older machines) with readiness for a high level of HTTP requests (important for busy machines).
- You can configure the log files in great detail, creating different files for different types of events. Virtually everything that happens to Apache can be logged.
- You can set Apache to look up host names of machines making requests. However, this is turned off by default since it can cause a lot of network overhead.
- Like the Linux kernel, Apache can load modules for additional functionality. These are stored in the `/etc/httpd/modules` directory.
- Apache can support many virtual hosts on a single server. These can be set up via separate IP addresses or just separate host names.

The `UserDir` Directive

The `UserDir` directive in `httpd.conf` allows other users of your machine to set up pages in their home directories. For example, adding this line ...

```
UserDir pub/index.html
```

... would enable a user to host a personal website at `http://servername/~username`. All he or she would have to do is to create a directory named `pub` under the home directory, as well as a file in that directory called `index.html`.

Index Files

The default file served for any given directory is the index file, specified with the `DirectoryIndex` directive. On most systems this file is configured to be `index.html`. But you can add more filenames to this line if you have other needs. For example:

```
DirectoryIndex default.htm
```

Access Control

Apache has access control features that let you specify access restrictions for files or entire directories. You can do this through `order` statements in your `httpd.conf` file. Order statements have two options, `allow` and `deny`. The order of these options controls the access rule for the directory.

- `order allow,deny` will permit access for specified hosts but deny it for everyone else.
- `order deny,allow` will deny access for specified hosts but allow it for everyone else.

For example, to make the directory `/var/www/html/private/` available only to people in your domain, you'd specify access for local users in `domain.com` by doing something like this:

```
<Directory /var/www/html/private>
     order allow,deny
     allow from domain.com
</Directory>
```

Or to allow access for everyone except those creeps in the legal department who made you put a disclaimer on everyone's e-mail, you'd do this:

```
<Directory /var/www/html/private>
     order deny,allow
     deny from legal.domain..com
</Directory>
```

Hosts specifications can be done with IP addresses like 10.0.2.53, or IP ranges like 10.0.2.0/24.

Access Control and Configuration via `.htaccess`

Sometimes users other than the system administrator need to make changes to the configuration of a particular directory in a web server, perhaps for a specific department's web site that is maintained by a user in that department. You obviously can't give the user write access to your `httpd.conf` file, so as a workaround Apache allows the user to add customizations by placing a file named `.htaccess` in a web directory.

The end user can then add many customizations, including special MIME types, access restrictions (or permissions) for certain hosts, or even password access to the directory's contents. In short, a user can express in a `.htaccess` file almost anything the system administrator could specify in the master `httpd.conf` file.

The first thing you need to do to allow a `.htaccess` file is to configure Apache to recognize the directory. You do this in `httpd.conf` file. Any directory can have its own `AllowOverride` option; usually it can be set to `None`, but it can also be `All`, or any combination of the keywords `Options`, `FileInfo`, `AuthConfig` and `Limit`.

In this case we'll open a directory for customization by the marketing department. In `httpd.conf` we'll add this special configuration:

```
<Directory /var/www/html/marketing>
     AllowOverride authconfig
</Directory>
```

Now if a Marketing user wants to set up authorization by password, he or she can create a `.htaccess` file in that directory and fill it with a custom configuration. The file might look like this:

```
# Password authentication for directory access
AuthName     "Password for Marketing Directory Access"
```

```
AuthType     basic
AuthUserFile /home/wsmith/www-passwd
require user wsmith
```

The user wsmith could then create a special password file (matching the location specified in the `AuthUserFile` directive) and add his password to it. To do so, he'd run this command:

```
$ htpasswd -c /home/wsmith/www-passwd wsmith
```

The `-c` option creates the file if it does not already exist, and if you're paranoid you can also add the `-m` option to force MD5 encryption. The `.htaccess` file specifies that only the user named wsmith can access the directory, but you can add additional users to this file as well.

Note that the password file is kept in wsmith's home directory. In fact it can be kept anywhere; it is only kept in the home directory because in this example wsmith is the maintainer of the file. The only place you *shouldn't* put the password file is in a directory available over the web, since a hashed password file can cracked with the right tools if it is available.

One thing the end user should keep in mind is that the Apache server must be able to read the password file. In this case wsmith would want to make his directory executable and the file readable. IF this is not ideal the administrator should find another, more private place to put it.

MIME Types

Whenever a web server sends a file to a browser, it also sends a MIME type associated with that file. The `AddType` directive will tell the web server to guess the MIME type based upon the file extension. You can include as many `AddType` directives as you'd like. For example, if you want to host a site including files with the `.htm` extension for HTML files, you'd include this type:

```
AddType text/html .htm
```

Dynamic Web Pages with CGI

Common Gateway Interface, or CGI, is a method for passing information from a web user to server-side programs, which can generate content dynamically. There are many forms of CGI scripts and a few languages in which you can write them, but Perl is the dominant CGI scripting language. Apache has a module called `mod_perl` that, if enabled, will remain in memory so that Perl scripts execute more quickly.

In `httpd.conf` the `ScriptAlias` directive sets up the directory for CGI scripts. On a default installation the directive looks like this:

```
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

Of course you can set up custom `ScriptAlias` directives for special directories.

Virtual Hosts

The virtual hosts section of the `httpd.conf` file lets you set up addition domain names supported by your server. For example, if you wanted to set up a virtual host for a second IP address you'd do this:

```
<VirtualHost 10.0.2.30>
    ServerName      www2.domain.com
    DocumentRoot /var/www/html2
</VirtualHost>
```

There are many other options you can add, including `ErrorLog` to have you virtual host record to a different log file.

Sample Virtual Hosts Section

Many web hosting companies use virtual hosts to let many logical sites share a single server. In our example we're using virtual hosts for a different purpose: to host a webmail site for the same company.

The first step in setting up virtual hosts is to create one that corresponds to the default configuration of the server (i.e., the configuration expressed earlier in the file). So we'll set it up much like we set up the regular `www.domain.com` site.

```
# We MUST include a VirtualHost identical to the existing host
<VirtualHost www.domain.com:80>
    ServerAdmin webmaster@domain.com
    ServerName  www.domain.com
    UseCanonicalName On
    DocumentRoot /var/www/html
</VirtualHost>
```

Now we're going to do an option step to make things conveneint for the webmail users. Webmail will run on a secure site. This would ordinarily require that users type the URL for the site with the `https://` protocol identifier. This tells the browser that the session will be secure HTTP on port 443.

Many users will be confused by this. In fact very few users type the protcol identifier at all, and instead type only the server's hostname (in this case `webmail.domain.com`) and let the browser do the rest. In this case if users do not specify `https://` the browser will try regular HTTP on port 80. So we'll include a virtual host that doesn't point to a real site, but rather redirects the user by using Apache's rewrite engine.

```
# VirtualHost to catch Webmail HTTP port 80 - redirects to HTTPS
<VirtualHost webmail.domain.com:80>
    RewriteEngine on
    RewriteRule ^/(.*)$ https://%{SERVER_NAME}/$1 [R,L]
    ServerAdmin webmaster@domain.com
    ServerName webmail.domain.com
    UseCanonicalName On
    ErrorLog logs/webmail.domain.com-error_log
    CustomLog logs/webmail.domain.com-access_log common
</VirtualHost>
```

Note that we omitted the document root directive, since there are no files in this site. Users are simply redirected by a transformation of the URL. Now that we're catching the slackers and redirecting them, we'll create the webmail site's actual virtual host.

```
# Real Webmail Site
<VirtualHost webmail.domain.com:443>
    RewriteEngine on
    RewriteOptions inherit
    SSLEngine On
    SSLCertificateFile conf/ssl.crt/server.crt
    SSLCertificateKeyFile conf/ssl.key/server.key
    ServerAdmin webmaster@domain.com
    DocumentRoot /var/www/squirrelmail/
    ErrorLog logs/webmail.domain.com-error_log
    CustomLog logs/webmail.domain.com-access_log common
</VirtualHost>
```

The only thing that will stop this configuration from working right now, aside from the absense of the sites' files in their respective document roots, is the absense of SSL keys and certificates. That will be covered later on.

Secure Hosting with SSL

Apache allows you to encrypt web traffic using the Secure Sockets Layer (SSL) encryption protocol. SSL is a form of asymmetric cryptography, often referred to as *public key cryptography* (PKI). With PKI two keys are created, a private key for the server and a public key the for the clients. Network traffic encrypted with one key can only be decrypted using the other key, so this ensures that that incoming data could only

have come from the other party. SSL is used in the https protocol, which by default operates on port 443. To use SSL, your Apache server must be configured to run `mod_ssl`.

The SSL program that ships with Fedora Core and many other distributions is OpenSSL. OpenSSL is a free PKI utility that can create self-signed certificates, private keys, or certificate signature requests.

To set up your secure site, first create a certificate signing request and a private key.

```
# openssl req -new -out server.csr
```

This will ask you some questions. When asked for a Common Name give the *exact* domain name of your web server (e.g. *www.domain.com*). The certificate must match the specific server name, and most web browsers will complain if the URL does not match. Make sure you keep your private key and signing requests, as they will be necessary if you want to make more certificates.

The passphrase

While making your signing request and private key you are asked to create a passphrase for the private key. By default you'll have to input the passphrase whenever you load the key, but unfortunately this is necessary every time you start your web server. That's not convenient at all in environments; if there isn't an administrator to type the passphrase after a reboot or a crash, this can be inconvenient. So you can remove the passphrase from the private key like this:

```
# openssl rsa -in privkey.pem -out server.key
```

However, this is dangerous. While it allows Apache to load the key and run the secure site without user intervention, it also means that you *must* make the `server.key` file readable only by the apache server. You should also delete the `.rnd` file because it contains the random data used to create the keys and could therefore be used to crack your private key.

Now you should create a self-signed certificate that you can use until you get a "real" one from a certificate authority.

```
# openssl x509 -in server.csr -out server.crt -req -signkey server.key -days 365
```

Of course getting a certificate from an authority is purely optional. If you know your users well enough you can tell them to install the self-signed certificate into their browsers so they can access the site. And you can increase the `-days` option to something larger than 365 if you'd like it to expire after a longer period than a single year.

When you have finished creating the keys and certificate, move the `server.crt` file into `/etc/httpd/conf/ssl.crt/` and move the `server.key` file into `ssl.key`.

Considerations for Macintosh Internet Explorer Users

If you have web users with Internet Explorer for Macintosh, they can avoid being blocked from the site by installing your certificate manually. To allow this, create a DER-encoded version of your certificate:

```
# openssl x509 -in server.crt -out server.der.crt -outform DER
```

You can call the `server.der.crt` file whatever you'd like, but it should end in `.crt`. Put the new DER-encoded file into a web-accessible directory. You might want to leave a copy in the ssl.crt directory for safe keeping.

Now have a Mac user open Internet Explorer and go to the URL for the DER certificate file. The user will be prompted to import the certificate into their browser's database of trusted certificates and authorities.

Of course if you can get them to use Mozilla or another browser that recognizes self-signed certificates these steps are not necessary.

The Tux Web Server

Tux is a web server developed by Red Hat. It is the world's fastest web server because it is kernel-based and has an extremely small footprint. If you'd like to host a website with static content, Tux is the ideal choice. However in its default mode it cannot serve dynamic content.

However Tux can also play a role in sites with a mix of static and dynamic content. In its default mode it will forward any HTTP request it can't deal with to port 8080. Therefore if you configure Apache to accept connections on port 80, Tux can tram up with Apache to create a server that is quick and efficient with static content but also capable of dealing with dynamic content.

Configuring Tux

The Tux server is started by a System V script, and its configuration file is `/etc/sysconfig/tux`. There is really very little configuration required beyond installing the RPM and makins sure it is configured to start in the appropriate run levels.

To set up Apache as Tux's dynamic-content server, simply configure apache to listen on port 8080 rather than on port 80 and restart Apache. This team-up will be invisible to end users, and in fact if you use a firewall you don't need to open port 8080 since the forwarding happens internally across the local loopback.

27. VNC: Virtual Network Computing

VNC is an open standard supported by many platforms. It is separate from X11, and has some advantages over it. One is that many separate clients can connect to the same machine in a "view only" mode. And a client can disconnect from the VNC server and then reconnect later on without loss of state in the session. In fact no data regarding your session's state is stored at the viewer's end, so you can connect to a VNC server and work, then leave that client and reconnect to your VNC desktop from another machine and continue working. If you were working on a text document, you'll find the cursor in the same place, even if the clients are miles apart.

More information on VNC is available at *http://www/uk.research.att.com/vnc*.

VNC Server

Install the VNC-related packages on your system. Once that's done, as the superuser you should edit the configuration file `/etc/sysconfig/vncservers`. If you wanted to give access to the user `wsmith` you'd edit the line to look like this:

```
VNCSERVERS="1:wsmith"
```

If you'd like to add another user simply add the username as user 2 and separate them by spaces:

```
VNCSERVERS="1:wsmith 2:hsolo"
```

Simply separate the usernames by spaces. The numbers are VNC user numbers. Then let's set up the first user. As `wsmith`, create a new directory called `.vnc` in the home directory and then run the `vncpasswd` command to set the user's VNC password. This is separate from the user's system password.

```
$ mkdir ~/.vnc
$ vncpasswd
```

As the superuser, start the service.

```
# service vncserver start
# service vncserver start
Starting VNC server: 1:wsmith                [  OK  ]
```

VNC Clients

Now you can launch a VNC client from another machine. For a quick test you can launch the `vncviewer` client from your local machine. The format is `vncviewer hostname:user-number`, so the command to launch a local client would be:

```
# vncviewer 127.0.0.1:1
```

This will launch the client locally, and connect to the server locally. But there is no reason you couldn't connect from halfway around the world. If you close the window it will end your current session; however, as long as the `vncserver` is running, the state of the session is maintained. You can reconnect from another machine and continue youw work; your cursor will even be in the same location.

Note that the desktop environment the VNC server chooses is based upon what is specified in `/etc/sysconfig/desktop`. So if you have this configuration file set to GNOME because you like to use the `gdm` display manager (login screen), but you use that login to load KDE, your VNC sessions will be GNOME sessions.

The `vncviewer` client can also run a number of other ways. Launched with the `-viewonly` option it will not allow you to use the keyboard and mouse, so you're basically "sharing" the desktop. The `-noshared` option prohibits sharing so no one else can connect. Use `-fullscreen` to use the entire

screen area for the VNC client. Finally, for slow connections you can use the `-comresslevel #` option to further compress the screen image.

Other VNC clients are available for a variety of systems, like WinVNC for Windows.

In addition, any Java-enabled web browser can act as a client since `vncserver` serves a Java applet viewer. All you have to do is connect to the correct port for your user number. The ports used are 5800=N, where N is your user number. So to see the VNC desktop for user 1 you'd go to *http://hostname:5801*.

Firewall considerations

VNC operates over ports 5900+N, where N is your user number. So if you'd like to enable VNC on your server you'll have to open ports 5901 and any additional ports for the number of users you'd like to support.

As mentioned above, the Java applet web client uses ports 5800+N, so to use a browser as your client you'll have to open up those ports on the server as well.

In either case VNC is not terribly secure. If you plan to VNC across the Internet or any other untrusted network you may want to set up a secure tunnel. This is covered in the section on SSH, but here is the general idea:

Set up an SSH session to the remote machine that is the VNC server, like this:

```
$ ssh -L a:localhost:b vncserver
```

This tells your machine to start its SSH connection to the VNC server, but to listen on the local machine's port *a* and forward all connections to the VNC server's port *b*. Since the VNC protocol normally uses port 5800+N, where N is the user number of the server, and the web client uses 5800+n, you might want to use the same local port to keep things simple.

So if you're using the web client and connecting to user 1's display, you'd do this:

```
$ ssh -L 5901:localhost:5901 vncserver
```

Then you can point your browser to *http://localhost:5801* and SSH should securely tunnel your web-based VNC session through SSH.

28. Samba: Opening Windows to a Wider World

The ability to connect with other operating systems can be an important requirement, and for this reason the Samba project is extremely important for many users of the GNU/Linux and other flavors of UNIX. Named for the Server Message Block (SMB) protocol that is also called CIFS, Samba is a client and server for the protocol used by Microsoft for file sharing and host naming services. It allows you to run services on GNU/Linux for Windows clients, or access resources on Windows servers from GNU/Linux. You can even set up a Samba-based Primary Domain Controller (PDC) for a Windows domain. Since version 3.0 it has offered functionality for setting up trust relationships with other Windows servers and the use of an LDAP/Kerberos backend for compatibility with Microsoft's Active Directory system.

Information on Samba is available at *http://www.samba.or*g.

Package Options

There is more than one option for installing Samba. Fedora Core's distribution of the daemon is split across three RPM packages: `samba-common`, `samba-server` and `samba-manual`. But if you want the latest version you can download the most up-to-date RPM from the Samba web site's Binary Packages section. The Samba binary package uses a single RPM, so if you decide to use it you should first remove the Fedora Core Samba packages in order to avoid conflicts.

Samba Basics

Samba's services are provided by two separate daemons, `nmbd` and `smbd`.

`nmbd` is the NetBIOS name server. NetBIOS is an old DOS resource-naming protocol that allows you to browse the network and lets Samba serve as a WINS server. WINS is the Windows Internet Naming Service, which maps Windows NetBIOS names to IP addresses and provides name resolution to clients. It is independent of DNS.

`smbd` is the SMB server itself. It provides authentication of users and file and printer sharing.Samba provides four main services: authentication of users, file and printer sharing, Windows name resolution and Windows network browsing.

These services require ports 137, 138 and 139.

Additionally Samba comes with the `smbclient` program for command-line access to the Windows network, and uses a kernel component called `smbfs`, which allows Linux to mount a Samba share in much the same was as it can mount an NFS share.

Samba as a PDC

Every Windows computer on a network belongs to a workgroup or a domain; a workgroup is a collection of one or more computers that appear together when browsing the network because they share a common workgroup name. A domain is a managed security environment, and as a Primary Domain Controller (PDC) Samba offers management for this type of environment. A Samba PDC can also manage roaming profiles for Windows NT and 2000 clients, and set basic user-level security for Windows 9x/ME clients.

However, while Samba performs very well as a domain authentication and file and print server, it does have some limitations. Version 2.x cannot interact with a Backup Domain Controller (BDC), and cannot establish a trust relationship with a Windows NT/2000 controller. It also cannot replicate Security Accounts Manager (SAM) authentication information with a Windows server (or visa-versa), nor integrate into the Microsoft Active Directory (AD). These functions will not be available until the release of Samba 3.0, which is currently in beta.

In this guide we'll show how to set up Samba 2.x as a Windows PDC, however the configuration is very

similar for a normal Windows server. Setting up a more modest file and print server simply involves omitting a few options, which will be outlined when applicable.

Server considerations

Samba's efficiency allows you to operate a simple file server on relatively dated machinery, however if you plan to implement a server for a large number of users you should plan on having a good amount of RAM and a quality SCSI drive.

The `smb.conf` File

The file `/etc/samba/smb.conf` is the repository for Samba's many options. Its contains two main sections: at the beginning is the `[global]` section, which has Samba-wide options, and this is followed by a "shares" section in which shared resources can be established. These resource names appear in square brackets, like `[homes]` and `[printers]`, followed by resource-specific options. The comments in this file can begin with either a hash (#) or a semicolon (;).

Global Options

The beginning of the `smb.conf` file establishes server options such as the workgroup name, the NetBIOS name and other networking options:

```
[global]
;basic server settings
workgroup = research
netbios name = fileserver1
server string = Samba PDC running %v
socket options = TCP_NODELAY IPTOS_LOWDELAY SO_SNDBUF=8192 SO_RCVBUF=8192
```

The next section is important in establishing the server's function. The first three settings have to do with the network browser functionality. Since we're setting up a PDC the options look like this:

```
;PDC and master browser settings
os level = 64
preferred master = yes
local master = yes
domain master = yes
domain logons = yes
```

The "domain master" option tells Samba that it is the PDC; switch its value to `no` if you'd like a domainless server. The other three options have to do with browse lists and elections. When a Windows user browses the network to find other computers, his or her computer gets its information from a "browse list" maintained by a computer known as the master browser. Windows machines determine which should be the master browser by holding an election, not unlike a game of "rock, scissors, paper." Elections happen all the time and it is generally useful to have the domain's PDC be the winner each time, so it's important to get the settings right; in this case we have set its OS level to 64 and made it the master; if it were not a PDC the OS level would be lower and the subsequent options should be set to `no`.

One option to add to this section is to add support for WINS. This can be useful on a network that does not have an existing Windows server because WINS will keep a database of machines and IP addresses . We enable it with these two settings:

```
wins support = yes
name resolve order = wins bcast
```

The first option is required to run WINS, and the second simply sets the name resolution order. `bcast` instructs Samba to issue NetBIOS network broadcasts to find host addresses. You can also add `lmhosts` as a resolver option, and if the `/etc/samba/lmhosts` file is present Samba will use it to resolve statically-assigned names in the file, much like the `/etc/hosts` file does for UNIX host names.

The next part of the `[global]` section establishes security and logging options.

```
;security and logging settings
security = user
encrypt passwords = yes
log file = /var/log/samba/log.%m
log level = 2
max log size = 50
hosts allow = 127.0.0.1 10.0.0.0/255.255.255.0
```

The "`security = user`" and "`encrypt passwords = yes`" options are required for Samba to be a PDC. The `host allow` option is important to ensure that only users from your local network can access your PDC.

Finally, the roaming profiles feature allows Windows NT and 2000 clients to save user settings on the PDC to be retrieved for later sessions on other machines. This lets users log in on multiple machines and see the same desktop, documents and other settings, and is enabled with these settings:

```
;user profiles and home directory
logon home = \\%L\%U\.profile
logon drive = H:
logon path = \\%L\profiles\%U
```

However, these consume a lot of bandwidth because the files are synchronized between the client and server during logout, and users who keep a lot of files on their Windows desktop or in their My Documents folder will find that synchronization can take a long time. If your bandwidth is limited, of if you simply don't require this functionality. you can disable roaming profiles by leaving the home and profile lines intact, but with values omitted:

```
logon path =
logon home =
```

Note that the directories referenced in the `[global]` section are Samba shares - for example, the profiles share is referenced as `\\%L\profiles\%U`. This should match a share called `profiles` or else the roaming profiles will not work, so it's important to get the shares configured correctly. Also, just for information, note that these two options are identical in concept but differ in their audience: the `logon home` is for Windows 9x/ME clients, and the `logon path` is for Windows NT/2000 clients.

The final option to add is the logon script. Used only if your server is a PDC, the logon script is a DOS batch file similar in concept to a shell script. It lists commands to be run on a client machine when it connects to the PDC. We'll make it netlogon.bat.

```
logon script = netlogon.bat
```

This file resides in a share named netlogon, and can contain any Windows commands you'd like to execute, including updates to the Windows registry or network drive mapping. We'll set up a simple netlogon file when we set up its share later on.

Shares

After the [global] section, all subsequent parts of the file establish "shares" on the Samba server. Some shares are core components of the Windows domain model; others can be made up however you'd like. There are a few options that can be useful when setting up shares.

- `public` - The user guest can access this share.
- `browseable` - The share is visible when browsing.
- `writable` - The share has read and write access.
- `printable` - The resource is a printer rather than a volume.
- `group` - Sessions on the share use the group specified as the primary group.

The Homes Share

The first share we'll add is called [homes], and is a standard share for the UNIX home directories:

```
[homes]
comment = Home Directories
browseable = no
writeable = yes
```

This maps each user's home directory to a share called `[homes]` and therefore differs for each user who accesses the share. A user named `wsmith` will see the contents of the `/home/wsmith` directory in the `[homes]` share. Note that we have set the browseable option to no, which is unusual for a Samba share. This is a special case, because `[homes]` will also appear as `\\server-name\wsmith`., so `\\server-name\homes` is redundant (and can actually cause some problems with Samba 3.x).

While this comes in handy from a security standpoint because it prevents users from even seeing other home directories. However it bears the caveat of exposing the standard UNIX home directory contents to Windows sharing, and Windows users whose desktops are configured to show hidden files will see important files like `.bashrc` or their mail folders, and might be inclined to move or delete them without first seeking the wise advice of the administrator. To avoid this we can set up a special directory under the home directory of each Samba user - for example, a subdirectory called `documents` - and specify it as the `[homes]` share by adding a final line to the `[homes]` configuration:

```
path = /home/%u/documents
```

Now the `[homes]` share will contain the contents of `/home/wsmith/documents`, a directory that can be devoted exclusively to browseable files.

The Profiles Share

The next section controls an optional setting for PDC implementations. Non-PDC Samba servers can omit the `[profiles]` section entirely; PDC servers should keep it regardless of whether they use the option.

```
[profiles]
path = /home/samba/profiles
writeable = yes
browseable = no
create mask = 0600
directory mask = 0700
```

If you'll look back at the logon path option from the `[global]` section you'll see variables that cite the `[profiles]` share. When a user tries to log onto the domain, the Samba PDC finds profile directory under `/home/samba/profiles/` (or it creates it if it's not yet there). For security we have disabled browsing on the share so that Windows users won't notice it while browsing the network, but it is writeable so that the profiles can be kept in sync. The permissions masks are set to allow only the user to write to his or her own profile directory.

The Netlogon Share

The next share we'll add is another PDC-specific share that can be omitted for domainless servers. The `[netlogon]` share is a tool used for updating or modifying client machines.

```
[netlogon]
comment = Network Logon Service
path = /home/netlogon
read only = yes
browseable = no
write list = root
```

This is a DOS batch file, much like a shell script but with DOS commands. A popular thing to do with login scripts is drive mapping. With the `NET USE` command you can add network resources for Windows

clients, including drive letters for your shares. So we'll script drive mapping and save it in the `netlogon.bat` file:

```
net use H: \\server-name\%username%
net use S: \\server-name\shared
```

Note that the `[homes]` share was mapped from `\\server-name\homes` for previous versions of Samba, but this caused some access problems in Samba 3.x that can be avoided by using the `%username%` variable, which Windows recognizes as the username.

We'll put this in the `/home/netlogon` share, and any users ho log in will find H: and S: drives for their home directory and shared folders. Be sure to set your script's permissions to executable.

Printers

The `[printers]` share gives Windows users access to all of the printers in the `/etc/printcap` file.

```
[printers]
comment = All Printers
path = /var/spool/samba
printer = raw
browseable = no
guest ok = yes
writable = no
printable = no
public = yes
```

Of course if you have printers you do not want Windows users to access, you can set up a separate printcap file and add a line like this one instead:

```
printcap = /etc/printcap-win
```

You can also establish printers as separate resources by creating a share-like stanza. This would allow you to set access rights and other attributes on a printer-by-printer basis.

```
[research1]
comment = First Floor Research Printer
printer = research1
path = /var/spool/research1
public = no
writable = no
printable = yes
valid users = wsmith hsolo lorgana
```

Setting up the Samba Server

Once you've finished your `smb.conf` file there are a few server-side tasks remaining. First you should create the directories necessary for the shares in `smb.conf`, as well as set the permissions on these directories. If you've chosen to run Samba as a PDC you'll need to take the following optional steps by creating administrative and machine groups. The administrative group lets you establish a way to manage the domain, and the machine group will establish a way to authenticate machines as members of the domain.

```
# group -g 200 admins
# group -g 201 machines
```

Note that these two GIDs are chosen to avoid conflict with other groups on our Samba server; your chosen GIDs may differ. Now that the groups are ready we'll create three PDC-specific directories and set their permissions and ownership.

```
# mkdir -m 0775 /home/netlogon
# mkdir /home/samba
# mkdir /home/samba/profiles
```

```
# chown root.admins /home/netlogon
# chmod 1757 /home/samba/profiles
```

Setting the permissions and ownership is extremely important for many reasons. For one thing, the `/home/netlogon` directory will contain the logon script that is executed by clients as they log on to the domain. A malicious user with accidental write access could easily propagate damaging code through this script. Similarly, the ownership on `/home/samba/profiles` sets the permissions such that the superuser owns everything in `/home/samba` down to `profiles`, and each user owns everything below that. This mean a user cannot change directory above his or her own profile.

User accounts

Now it's time to set up user accounts and, if you're setting up a PDC, machine accounts. One caveat on the user front is that Windows passwords and GNU/Linux passwords are stored in entirely different formats. This means that for every local user who will have Samba access you'll need to maintain two separate accounts - the local user account in `/etc/passwd` and the Samba user account in the `/etc/samba/smbpasswd` file. The pain involved in this is that you'll also end up maintaining two separate passwords for each user.

Unfortunately there are no easy utilities for combining these two steps, so they must be performed separately. First add the user in the usual way:

```
# useradd wsmith
# passwd wsmith
Changing password for user wsmith
New password:
Retype new password:
passwd: all authentication tokens updated successfully
```

Then use the `smbpasswd` command to add the user to Samba. This command on its own lets you change Samba passwords, but in this case the `-a` option adds the user.

```
# smbpasswd -a wsmith
New SMB password:
Retype new SMB password:
Added user leah.
```

For a password change to an existing user, just remove the `-a` option.

Samba understands the connection between the Samba user and the regular GNU/Linux user because it is specified in the file. Another way to add users that lets you specify the GNU/Linux user associated with the Samba user is with the `smbuseradd` command:

```
# smbuseradd wsmith:wsmith
```

Samba Password Encryption

If you choose not to use password encryption, you can comment out the `encrypt passwords =` and `smb passwd file =` options (if present). You will then be able to use the GNU/Linux passwords and accounts for authentications. However this is not recommended unless you really, really trust your network.

For newer versions of Windows than the original Windows 95 (OSR2 and later) you'll have to enable unencrypted passwords. For Windows 9x. open the registry editor (`regedit.exe`) and find this key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\VNSETUP
```

For Windows NT, find this key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Rdr\Parameters
```

In whichever key you select, create a new DWORD value in that key named `EnablePlainTextPassword` and set its value to `0x01`.

Keeping user accounts in sync

Keeping passwords in sync between GNU/Linux and Samba is kind of a pain. To help alleviate it Samba has a password synchronization option. This allows a user to change their Samba password from a Windows client, or at the shell with the `smbpasswd` command, and his or her GNU/Linux password is also changed to match it. Just add this to the `[global]` options:

```
;sync UNIX passwords
unix password sync = yes
passwd program = /usr/bin/passwd %u
passwd chat = \
 *password* %n\n \
 *password* %n\n \
 *successful*
```

This "catches" the Windows password and sends it to the UNIX passwd command. Important note: the password chat option should be on a single line, though it may span multiple lines in this document.

On the other hand, you might also want to synchronize the Windows password whenever the UNIX `passwd` command is given. This ensures that users don't get confused by the existence of two passwords, and two password commands. To do so, add the following line to `/etc/pam.d/system-auth` below the `pam_cracklib.so` invocation:

```
password required /lib/security/pam_smbpass.so nullok use_authtok try_first_pass
```

This configures the GNU/Linux pluggable authentication module (PAM) to catch the `passwd` command and send its passwords to `smbpasswd`. - pretty much the reverse of the password chat option above.

If you use `authconfig`, a component of the Fedora Core `setup` utility, keep in mind that your changes will be overwritten by the utility whenever you use it.

Machine Accounts for PDCs

Machine accounts for Windows clients of a PDC are called "trust accounts," and require entries in both the `passwd` and `smbpasswd` files just as Samba user accounts do. When you create a machine account a "secret" is generated and subsequently used to secure communications between the PDC and the client, as well as to stop another machine with the same NetBIOS name from accessing the machine account.

You'll first need to create a superuser account in order to join machines to the domain:

```
# smbpasswd -a root
New SMB password:
Retype new SMB password:
Added user root.
```

Be careful with this account - it has the same authority as the GNU/Linux superuser and the password should be heavily guarded.

While Windows NT/2000/XP supports trust accounts, Windows 9x/ME does not, so older DOS-based Windows machines can only log in, run logon scripts, download a system policy file and store a profile on the server. The trust account does not exist. In fact these DOS-based systems are so archaic that they don't support more than one user name at a time, so while you are logged into the domain you can't access a separate file server under an alternate name. The creation of machine accounts is therefore only for NT-based Windows clients, and Windows 9x/ME clients have a different requirement: system policies. These will be discussed later on.

Manually adding machine accounts

Samba will not allow you to add an entry to the smbpasswd file unless there is a GNU/Linux account of the same name. To create a machine account you'll have to first set up the normal passwd account:

```
# useradd -g machines -d /dev/null -c "Machine Description" -s /bin/false hostname$
# passwd -l hostname$
Changing password for user hostname$
Locking password for user hostname$
```

The first command creates the account for the computer named hostname, and the dollar sign tells Samba it's a trust account. It's also member of the machines group and has no home directory and no shell access. The passwd command creates the "secret" for the machine to authenticate against.

Now we'll add the machine's Saamba account:

```
# smbpasswd -a -m hostname
Added user hostname$
```

The dollar sign is not required with smbpasswd. as it is with useradd and passwd. After setting up the account you should try to connect the client to the PDC immediately. Until the client formally connects to the PDC (and thus changes the "secret," the domain is vulnerable to a foreign machine with the same NetBIOS name as the account you've set up.

Auto-generation of machine accounts

The alternate (and probably superior) approach to generating machine accounts is to configure Samba to create them automatically when the client first joins the domain. Add the following to the [global] options section:

```
add user script = /usr/sbin/useradd -d /dev/null -g machines -s /bin/false -M %u
```

This adds both the GNU/Linux account and the Samba machine account.

Adding the machine to the domain

Setting up a machine is not terribly difficult. Before you try to add machines to the domain you'll first want to try to browse the network for the Samba server and see if you can access the shares. Look in the homes and shared shares and any others you may have created, and type the path to netlogon manually in Windows Explorer to see if you can see the login script file. In Windows syntax it should be at \\server-name\netlogon. Try running it.

When you are convinced that Samba is working, go into the Windows network identity settings and configure the machine to be a member of the domain. When you save your changes you should be prompted for a login to authenticate. This is the superuser login, so you should log in as root with root's Samba password (this may differ from root's GNU/Linux password).

If these steps fail it's worth making sure your root account password is correct. If you fail to get a login entirely, check your network settings. If your client is on a different subnet than your Samba server you may have to coax your client into recognizing the server by setting it as a WINS server or trying to browse to it by its IP address (e.g., \\10.0.2.27\homes\). If you get a resource error the problem may be that your testing caused a redundant connection to the server under a different set of credentials (in other words, you're trying to authenticate as root but you've already opened a share as another user). If this is the case you should reboot to clear the authentication cache.

Special Problems with Windows XP

Windows XP has its own special considerations. First off, forget about user authentication of you are using Windows XP Home Edition. The Home version is identical to the Professional edition, but has

several features "broken" to better suit its cheaper price tag. If you need to access a Samba share from XP Home you'll need to configure Samba to use share authentication instead of user authentication, since Home will not join a domain.

Professional will join a domain, and it is exactly like joining a Windows NT or 2000 client to the domain. With XP, however, you will not be able to log in unless you first perform the following steps:

> 1) Go to the XP Start menu and select Run. Type `gpedit.msc` then hit enter.
> 2) In the policy console, go to "Local Computer Policy \ Computer Configuration \ Windows Settings \ Security Settings \ Local Policies \ Security Options."
> 3) Look for the item named "Domain member: Digitally encrypt or sign secure channel data (always)." Disable this feature.
> 4) Look for the item named "Domain member: Digitally sign secure channel data (when possible)." Disable this as well.

Reboot the machine and you'll be able to log into the machine with a domain user account.

System policies for Windows 9x/ME clients

As mentioned before, DOS-based Windows clients require a system policy file in order to log onto a domain. This can be accomplished by putting a *policy file* named `config.pol` in the `[netlogon]` share. The `config.pol` file is created and edited with a utility called `poledit.exe`, the Policy Editor. It is found on the Windows 9x/ME CD but is not installed by default.

Creating a policy is not hard to figure out, but you have to make the file on Windows and then move it to your Samba server. You can use your policy to restrict what your Windows users can do to their machines, preventing them from being able to use the machine without logging on, locking the desktop settings on permanent Windows drab turquoise, and so forth. But if you want the desktop to be fully useable you should enable everything in the Policy Editor menu structure.

Save the file as config.pol and upload it to your `/home/netlogon` directory, giving it the same permissions and ownership as a logon script - superuser and `admins` group ownership with permissions set to 755 (executable by all). Your Windows 9x/ME users should then be able to log in; joining them to the domain is not possible as it is with NT and 2000 clients.

Other Samba authentication methods

In our example we have set the `security =` setting in the `[global]` section to `user`. This validates users by their user name and password. There are three other options for this security setting:

- `security = domain`

  Samba contacts a workgroup with a collection of authentication data. If you use domain authentication you should also set these options:

  ```
  workgroup = WORKGROUP
  encrypt passwords = yes
  password server = server1 server2
  ```

  The password encryption option should be set to no if you are using an older Windows (such as the original release of Windows 95) that does not support the feature, or if you have it turned off in the Windows registry. The password server option can specify multiple servers, separated by spaces.

- `security = server`

  Another server should be contacted to perform authentication. If you use `server` authentication you should also include this line to specify another server as the password server:

```
password server = server1 server2
```

- `security = share` - User validation is on a per-share basis.

Testing your Configuration File

The `testparam` program is a command-line utility used to test your Samba configuration file for errors. It will test all of your options, including global options built into Samba but not included in the `[global]` section of your file.

Use the command with an IP address on your network, and `testparam` will tell you which Samba resources will be available to a host at that address:

```
# testparam /etc/samba/smb.conf 10.0.2.100
```

Samba Clients

The previous section discussed a Samba server; GNU/Linux can also use Samba from the client side. Let's take a look at some client apps.

`smbclient`

The `smbclient` tool is much like FTP for Samba. After initiating commands an FTP-like file tool is available.

```
$ smbclient //servername/service
> cd shared
> get data.txt
```

The term *service* is used interchangeably with *share*. In SMB parlance this is called a UNC path (short for Universal Naming Convention).

As used above, the `smbclient` command will borrow the `$USER` or `$LOGNAME` environment variables. It will also look for the `$PASSWORD` variable. This is much the way Windows machines work; iit attempts to login to the remote resource using local credentials, and if the usernames and passwords are identical it allows unrestricted access between the two resources.

To specify a password you can use the -U option. Everything after the % symbol will be used as the password:

```
$ smbclient -U wsmith%goldstein //servername/service
```

Of course you don't need to log into see the resources available on the machine. The -L option will list the resources without authentication.

```
$ smbpasswd -L
```

One nice feature of `smbclient` is the ability to create tar archives of remote shares. For example

```
$ smbclient //servername/sharename -U -Tcq filename.tar
```

Putting these tarballs on your local filesystem allows you to incorporate them into other backups, such as a tape backup system.

nmblookup

The `nmblookup` utility can query a WINS server and obtain IP addresses and hostnames, much as a DNS resolver can look up DNS hostnames and addresses. You can query a particular WINS server like this:

```
$ nmblookup -U winsserver -R 'queryname'
```

Or you can look up machines using broadcast if you don't know of a specific WINS server.

```
$ nmblookup \*
```

The backslash prevents the asterisk from being interpreted as a shell wildcard.

smbmount

The Linux kernel supports SMB as a filesystem type, allowing you to mount Samba and Windows shares. To manually mount a share, use the `smbmount` command:

```
$ smbmount //filserver1/shared /mnt/samba mountpoint -o username=wsmith
```

Or better, specify the filesystem type and use the regular `mount` command:

```
$ mount -t smbfs -o username=wsmith,password=r2d2c3po /fileserver1/shared /mnt/samba
```

Automatic mounting with `fstab`

You can make Samba shares mount automatically using the `/etc/fstab` file. Just add a line like this:

```
//fileserver1/shared /mnt/samba/ smbfs defaults,username=nobody 0 0
```

(Note: try this to see what this username does)

Special Windows Client Problems

Sometimes you'll run into issues with Windows clients. On some networks Windows may not want to join your perfectly acceptable domain. In some cases you may change the network settings of your domain controller and find that the clients have trouble authenticating to it.

One workaround is to set up and `lmhosts` file. This is a static mapping of local network hosts similar to the /etc/hosts file in UNIS and UNIX-like systems, though its syntax is a little more complex.

To set it up, create a plain text file named lmhosts. If you are using Windows 95, 98 or Me you can put this in your `C:\WINDOWS` folder. If you are using NT, 2000 or XP it would go in `%SystemRoot%\System32\Drivers\Etc`. The file should look somewhat like this:

```
10.0.15.17     fileserver1            #PRE #DOM:workgroup #Workgroup PDC
10.0.15.17     "RESEARCH       \0x1b" #PRE
# IP Address   "123456789012345*7890"
```

Substitute your IP address, as well as your Windows domain name in place of `workgroup`. The reason the final, commented line is included is that the middle line must have rather exact spacing. The commented line helps you make sure the the string `\0x1b` begins directly above the asterisk (assuming you're using a fixed-width font). Note that the quotations marks also line up. Without this very specific spacing your file will not work.

Once the file is in place on the client, open a DOS command line and issue the following commands:

```
C:\> nbtstat -R
```

Next you can load the new NBT Remote Cache name table:

```
C:\> nbtstat -c

Node IpAddress: [10.0.13.22] Scope Id: []
```

```
              NetBIOS Remote Cache Name Table

    Name              Type       Host Address    Life [sec]
---------------------------------------------------------------
WORKGROUP      <1B>  UNIQUE      10.0.15.17          -1
FILESERVER1    <03>  UNIQUE      10.0.15.17          -1
FILESERVER1    <00>  UNIQUE      10.0.15.17          -1
FILESERVER1    <20>  UNIQUE      10.0.15.17          -1
```

Once this is in place your clients should honor your domain controller.

Setting up Samba Printing with CUPS

If you use the CUPS printing system instead of LPRng you can also add a `printing` option to the `[global]` section of your Samba configuration file, along with a couple other settings like this:

```
[global]
printing = cups
printcap name = cups
load printers = yes
```

But this doesn't make your printers available immediately. As with anything else in Samba, you need to create a share for your printers. This appears to be a folder to Windows users, just like a file share. But unlike a file share it will be populated with your machine's local printers. Our `[printers]` section looks like this:

```
[printers]
comment = All Printers
path = /var/spool/samba
browseable = no
public = yes
guest ok = yes
writable = no
printable = yes
printer admin = root
```

And you're all set. But if you're really into automation you can go an extra mile by *pushing* printer drivers to your Windows clients. This is a nice incentive to printing with CUPS rather than with LPRng. Windows clients will install the printer drivers from a share on your Samba server, and no special user privileges are required. This makes it easy to leave it up to your users to decide where they'd like to print.

The `cupsaddsmb` utility, which is provided with CUPS, will export your printers to Samba. First you have to download the proprietary Postscript driver from the Adobe website, extract it and put its files into a `drivers` directory under the CUPS data directory, `/usr/share/cups/`. Different Windows clients require different files:

```
[Windows 95, 98, and Me]
ADFONTS.MFM
ADOBEPS4.DRV
ADOBEPS4.HLP
DEFPRTR2.PPD
ICONLIB.DLL
PSMON.DLL

[Windows NT, 2000, and XP]
ADOBEPS5.DLL
ADOBEPSU.DLL
ADOBEPSU.HLP
```

Make sure the files have uppercase filenames. CUPS also comes with its own PostScript drivers, and they're actually recommended over the Adobe drivers, however the CUPS drivers will not support Windows 9x / ME clients. If you won't need to support older Windows systems you can use them, but be sure to name them with lowercase letters:

```
cups.hlp
```

```
cupsdrvr.dll
cupsui.dll
```

Now create a Samba share in `smb.conf` like this:.

```
[print$]
comment = Printer Drivers
path = /etc/samba/drivers
browseable = yes
guest ok = no
read only = yes
write list = root
```

The `cupsadsmb` utility can then be employed to copy the needed files for Windows client installations from the previously prepared CUPS data directory to your new `[print$]` share. To share the drivers for the printer `research1`, we'd do this:

```
# cupsadsmb -U root research1
```

To share the drivers for all printers we'd replace the printer name with an `-a`.

Setting up a PDF Printer

It is possible to train the CUPS printer daemon to print to PDF files rather than to printers, and then to share this feature for your Windows clients as a pseudo-printer in Samba.

The basic ingredient in such a scheme is a program called Ghostscript. This provides a utility called `ps2pdf`, which converts Postscript data to a PDF. Since the above examples configure Samba to provide access to Postscript printing, incoming data from your Windows clients is in Postscript format already. A freely-available shell script can use `ps2pdf` to take that data and convert it to a PDF file. Before starting this installation you'll want to make sure you have this command on your system by typing "`which ps2pdf`".

The shell script that performs the conversion is called `pdfdistiller`, and you can download it from the KDE website at *http://printing.kde.org/downloads/pdfdistiller*. For our purposes it should be renamed `pdf`, and then placed in `/usr/lib/cups/backend/`. Change its mode (permissions) to 755 and then restart CUPS. This should make it available as a printing function in CUPS. To test this, run this command:

```
# lpinfo -v | grep pdf
direct pdf
```

If you got the output specified, you're doing well. Next test the backend and its ability to accept arguments by running the script itself in the following two ways:

```
# /usr/lib/cups/backend/pdf
direct pdf "Unknown" "PDF Writing"
# /usr/lib/cups/backend/pdf 1 2 a b
Usage: pdf job-id user title copies options [file]
```

Now we need to get a PPD distiller. Most users recommend the Adist4.ppd file found at *http://www.pentondigitalads.com/PentonDigitalAds/creating_postscript_files.htm* under the heading "Acrobat Distiller PPD for Windows." This PPD file should be placed in `/usr/share/cups/model`. You might want to rename it `pdf.ppd`.

Restart CUPS, then add the pseudo-printer to CUPS with this command

```
# lpadmin -p PDF-Printer -E -v pdf:/tmp/pdf_out -m pdf.ppd
```

Now create the export area, /tmp/pdf_out, specified in your command. This is where the PDFs will be created. It must be writeable for all users or the backend script will not function, so change its mode to

777.

Test the output. Create a file called `test.txt` and populate it with some text, then send it to your PDF printer like this:

```
# lpr -P PDF-Printer test.txt
```

In the `/tmp/pdf_out` directory you should find your PDF.

Modifications

This is not very convenient, however, because the simplest way to give people access to their documents is to give them access to a Samba share that points to `/tmp/pdf_out`. If you don't want to create a new share, or if you don't want people to be able to see each others' PDF files, it's much more convenient to simply modify the `pdfdistiller` script at `/usr/lib/cups/backend/pdf` to copy the file to the user's home directory.

Open the file in a text editor, and at the beginning of the file under the line that starts with PRINTTIME, add this line:

```
TARGET=/home/$2/documents
```

This creates the variable for the location. `$2` is the user's name, so we're putting all PDFs under the `documents` directory of the user's home directory. But if course you could put them anywhere.

Next we'll skip to the end of the file and look for these three lines:

```
if [ "$2" != "" ]; then
        chown $2 "$OUTPUTFILENAME"
fi
```

We'll add three more lines to this, right under the line containing the `chown` command:

```
if [ "$2" != "" ]; then
        chown $2 "$OUTPUTFILENAME"
        if [ "$TARGET" != "" ]; then
        mv "$OUTPUTFILENAME" "$TARGET"
        fi
fi
```

This moves the file to the target directory.

Finally, to share this printer via Samba you'll want to use the `cupsaddsmb` command to copy the driver to Samba and make it available to Windows clients:

```
# cupsaddsmb -U root PDF-Printer
```

29. Setting up a DHCP Server

DHCP is the Dynamic Host Configuration Protocol. It is a service that lets you pass configuration options to all of your network hosts. Prior to DHCP each machine on a TCP/IP network had to be configures independently - each with its own IP address, subnet mask, default gateway, DNS servers and so forth. With DHCP this can be administered centrally and hosts can "lease" a configuration from a DHCP server.

DHCP supports older BOOTP clients, which can get DHCP configurations but do not understand the concept of expiring leases and therefore never give up their assigned configuration.

The DHCP protocol runs on ports 67 and 68, and these ports can be configured to be open when configuring most firewalls.

The Server

The DHCP server that comes with GNU/Linux distributions is `dhcpd`. It is a a System V-launched daemon. To configure the server, make sure that a broadcast address is specified in your server's IP configuration. You can do this with the `ifconfig` command. If it is not present you will have to reconfigure your network adapter to use a broadcast address.

Then create a file called `/etc/dhcpd.conf`. This file will contain the settings for lease duration, IP address ranges and other settings you'd like to push to DHCP clients.

For example, let's say you need to set up a DHCP server for the folks in 10.0.3.0/24. First set up your global variables:

```
option domain-name "domain.com";
option domain-name-servers 10.0.3.12 10.0.2.12;

default lease time 21600;
max-lease-time 43200;
```

Then add the client configuration options for your subnet:

```
subnet 10.0.3.0 netmask 255.255.255.0 {
    range 10.0.3.100 10.0.3.254;
    option routers 10.0.3.1;
    option subnet-mask 255.255.255.0;
}
```

If you have a specific host within that subnet that you'd like to send a special configuration you can tack on its MAC address within the {} brackets of the subnet:

```
host special-host-name
{
hardware ethernet 00:20:AF:A2:BD:34
fixed-address 10.0.3.2
}
```

There must be at least one subnet block in your configuration file. Once IP addresses are leased they'll be recorded in `/var/lib/dhcp/dhcpd.leases`.

30. Using LDAP

Lightweight Directory Access Protocol (LDAP) is a proposed standard for hosting and accessing directories. A directory is much like a telephone book, and indeed LDAP is most commonly used to associate names with e-mail addresses, phone numbers and other personal data. But it can also handle other data, even user authentication data as a replacement for NIS (covered in Chapter 12).

An LDAP directory stores data in a database. Unlike most other types of databases, an LDAP directory is not a relational database - that is, it doesn't contain lots of independent records with common fields, but rather is more structured like a filesystem's directory structure. Clients can search through this information, and most programming languages now have APIs for LDAP access so that programmers can make software LDAP-aware. As a result many e-mail clients have at least rudimentary support for LDAP.

Fedora Core and other GNU/Linux distributions come with OpenLDAP, a free software implementation of the LDAP standard.

LDAP's Extensibility

LDAP is a distributed service. Much like the DNS system that resolves hostnames on the Internet, LDAP supports propagation across multiple servers, and it is designed to grow into a ubiquitous Internet service. While there are not many public LDAP servers available like there are DNS servers, the idea of LDAP is much like the DNS. There are high-ranking servers that delegate lower-ranking servers, stretching down in an inverted tree to include countless organizational LDAP servers. Except that instead of storing host names, LDAP stores *any* information you'd like.

While LDAP can forward queries to higher-ranking servers like DNS, in practice most LDAP servers are standalone machines because the protocol is not as widespread as DNS. The protocol is mainly used for directory services within organizations.

LDAP operates on port 389 (or 636 for the secure "ldaps" version of the protocol).

Installing OpenLDAP Daemons and Utilities

The OpenLDAP suite comes with a number of utilities, daemons and libraries. They are split into the following three RPM packages:

- `openldap` - The libraries needed to run the server and client applications.
- `openldap-clients` - The command-line tools for querying and modifying LDAP directories.
- `openldap-server` - The servers and other utilities needed to configure and run the LDAP server.

There are actually two servers in the `openldap-servers` package; both are System-V launched services. `slapd`, the Standalone LDAP Daemon, is the only one you really need to understand in order to get started with LDAP. `slurpd`, the Standalone LDAP Update Replication Daemon, synchronizes changes between LDAP servers, so it is useful in environments in which there will be more than one LDAP server and fault tolerance is important.

Authentication

If you plan to configure your LDAP server for authentication you must install not only the OpenLDAP packages above also the `nss_ldap` package. This allows your LDAP server to be an option for authentication, much like NIS.

Planning

It is easy to set up a simple LDAP server, but ensuring its longtime usefulness requires planning a good, extensible structure for your directory. This structure does not have a standard format, so your directory

could be unique in the LDAP world.

There are three basic terms you'll need to understand in oder to understand LDAP:

- A *Distinguished Name* (DN) is a unique name for an LDAP entry. No two entries can have the same DN. An example might be something like this: `cn=winston smith,ou=editors,dc=minitrue,dc=gov`
.
- A *Canonical Name* (CN) is the human-readable general name that appears in the directory as you browse it. Following the above example the CN would be `winston smith`.

- A *BaseDN* is the most basic parameter of an LDAP server. The server is responsible for all entries with DNs that fall underneath the BaseDN. This is generally configured to mirror the DNS domain name of your organization; this example's BaseDN is `dc=minitrue,dc=gov`.

Organizational Units

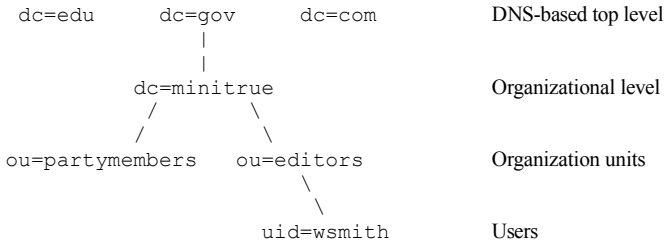
Underneath your directory's base DN you will create organizational units (OUs). These are basically containers that categorize your data. You can further organize your data by creating second-tier OUs. For example, let's say that Winston Smith would like to set up a directory for the Ministry of Truth. He needs to organize people who are editors, security staff, members of the Party and maintenance workers. But amongst the editors there are different groups that handle publications for London, Edinburgh and Manchester. We could organize the directory much like this:

```
dc=minitrue,dc=gov
  ou=editors
    ou=london
    ou=edinburgh
    ou=manchester
  ou=security
  ou=partymembers
  ou-maintenance
```

For sake of ease it's important to keep the directory structure as simple as you can. And while sub-tiers of OUs will make your directory more granular in its organization, it can also make LDAP searches somewhat slower.

An LDAP Visualization

Just as with a filesystem or any other directory-based organization of data, it can help to look at the tree-structure involved. A partial version of the above example might look like this:



Understanding Directory Entries

An LDAP directory entry, like Winston Smith's entry, is one unit in the directory. An entry can be an entire organization, a department or an individual person. Just like in any database, every LDAP entry must have a unique identifier - the DN. Every entry is made up of a DN followed by a list of *attributes*.

Attributes are bits of information associated with the entry like fax numbers, e-mail addresses, and office

phone numbers. Some are mandatory, while others are optional - the precise configuration is determined by an `objectclass` definition, which sets the required and optional attributes for each entry.

Sample LDAP entry

Let's take a look at a sample LDAP entry for Winston Smith. We'll break his entry into three sections and start with the first four lines.

```
dn: cn=Winston Smith, ou=editor, dc=minitrue, dc=gov
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
```

The first line is the DN, the unique identifier of the entry. The `cn` is a *Relative Distinguished Name* (RDN), which is followed by the entry's location within the directory's structure. This starts with Winston's OU. Under this line are three object classes. While `objectClass` is really a single attribute, it is listed three times because a directory entry can be in multiple object classes at the same time.

Since Winston is included in the `person` object class we can give him attributes that are available to any member of that class, like this:

```
sn: Smith
cn: Winston Smith
telephoneNumber: 555-1234
userPassword: {crypt}:$1$H5RIBxAu$OdM.CkSgnzp5dct1VZRMU/
description: Speakwrite Editor
Next we give him organizationalPerson attributes:
ou: Pets
title: HRH
street: 125 N. Main Street
postOfficeBox: 4224
st: CT
postalCode: 06512
facsimileTelephoneNumber: 888-555-1212
```

Those are all specific to objects of the type `person`. Next we'll add some `inetOrgPerson` attributes:

```
departmentNumber: 20
employeeType: permanent
givenName: Winston
initials: WS
mail: wsmith@minitrue.gov
jpegPhoto: wsmith.jpg
audio: daphne.wav
homePhone: 555-1234
pager: 555-4321
preferredLanguage: Newspeak
userCertificate: certs/df_cert.pem
```

All of this data is presented in the LDAP Directory Interchange Format (LDIF). LDIF data allows you to import and export LDAP entries using plain text files. In practice the `person` and `inetOrgPerson` attributes could be mixed together in an LDIF file, but we've separated them to make it clear where the data in an entry comes from.

Object classes and LDAP schema

Every entry belongs to at least one object class. Winston belongs to three, and they determine what attributes the entry can contain. Here is an example of an LDAP object class:

```
objectclass ( 2.5.6.6 NAME 'person' SUP top STRUCTURAL
        MUST ( sn $ cn )
        MAY ( userPassword $ telephoneNumber $ seeAlso $ description ) )
```

Each object class has five components: a unique object identifier (OID), a unique name, a parent (SUP), a

list of required attributes (MUST) and a list of allowed attributes (MAY).

OIDs look similar to IP addresses, and like IP addresses and OID is registered by the individual or entity that created them. ICANN maintains the central registry of LDAP OIDs. These numeric strings are used by the the LDAP daemon's database. Some LDAP software requires you to use registered OIDs, others do not.

A schema is collection of object classes. A given schema groups object classes by their similarity; for example, in this chapter we'll make use of a schema named `inetOrgPerson`, which contains object classes like `givenName`, `homePhone`, `departmentNumber` and so forth. It also inherits object classes contained in *parent* schemas.

Objects in an LDAP directory are based upon schema, which define the list of possible record types, or object classes. OpenLDAP schema files are stored in /etc/openldap/schema. The attributes of these object classes, which are also stored in the schema, can be required or optional, and the same attribute can be used by multiple object classes.

Object classes are arranged hierarchically, with the top object class at the root of the tree. First level object classes are children of this class, and lower level object classes are their descendants. Child classes automatically contain all the attributes of the parent class.

Once you get the LDIF data in your directory, it will be organized much like an inverted tree structure. The root of the tree is usually based upon a DNS domain name, such as `dc=domain,dc=com`.

Each component of the domain name becomes the value for a dc (domain component) attribute, and all of them are collected into a comma separated list. This is known as the directory's base, corresponding in this case to ahania.com.

Setting up your slapd LDAP server

The `slapd.conf` file

Configuration of OpenLDAP server starts with the `/etc/openldap/slapd.conf` file. Before you start to edit this file, it's important to back it up. It's easy to make damaging changes to this file before you understand how the software works. Backing up the original is good for reference and is also worth doing when you make further changes.

Schema

At thetop of your file you should have *include* references to various schema. This provides the various object classes to help structure your data. The default schema should work:

```
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/inetorgperson.schema
include         /etc/openldap/schema/nis.schema
include         /etc/openldap/schema/redhat/rfc822-MailMember.schema
include         /etc/openldap/schema/redhat/autofs.schema
include         /etc/openldap/schema/redhat/kerberosobject.schema
```

Database and user customizations

One thing to do before you start customizing this file is to secure the `rootdn`. This line represents the directory entry that can write to the database. Toward the the top of the `slapd.conf` file you may see this text:

```
# if no access controls are present, the default is:
#       Allow read by all
#
# rootdn can always write!
```

This is the access control section of the file. Take it seriously. The entry specified in `cn=` part of the `rootdn` line is the user who has full access to the database. The default configuration uses a plain text password password of "secret." Changing this to another plain text password is fine for a standalone machine, but if your LDAP server is accessible on a network you do not entirely trust you should probably encrypt the password. This can be done with the slappasswd utility like this:

```
# slappasswd
```

The default is encryption algorithm is SSHA. You can also use other hash options like SMD5, MD5, and SHA with the `-h` option (e.g., `-h MD5`). The utility will ask you for a password and then provide an encrypted string, which you can then paste string into `slapd.conf`. In this example we'll use an encrypted password; if you choose to leave you password in plain text, at least change it from "secret" to something else.

Your finished database configuration should look something like this.

```
database     ldbm
suffix       "dc=minitrue,dc=gov"
rootdn       "cn=root,dc=minitrue,dc=gov"
rootpw       {SSHA}cEQm/g/+6a+oDJHF+Yj2c759RyYXm0Yr
directory    /var/lib/ldap
index        objectClass,uid,uidNumber,gidNumber,memberUid   eq
index        cn,mail,surname,givenname                       eq,subinitial
```

Initial Server Test

When your first configuration is saved it's worth testing the server. Start the daemon and run a basic search:

```
# service ldap start
# ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts
```

If your configuration is working you should see output like this:

```
version: 2

#
# filter: (objectclass=*)
# requesting: namingContexts
#

#
dn:
namingContexts: dc=minitrue,dc=gov

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

Configuring Access Control Lists

Once your server is working you should establish some access control lists (ACLs). This will help make sure that users will be able to access the entries that they need, but nothing more.

You can either put your ACLs in the `slapd.conf` file or keep them in a separate file using an include line, much like you do with schema. In this example we'll simply put them in the main configuration file.

```
# Define ACLs

access to dn=".*,dc=minitrue,dc=gov" attr=userPassword
```

```
        by dn="cn=root,dc=minitrue,dc=gov" write
        by self write
        by * auth

access to dn=".*,dc=minitrue,dc=gov" attr=mail
        by dn="cn=root,dc=minitrue,dc=gov" write
        by self write
        by * read

access to dn=".*,ou=people,dc=minitrue,dc=gov"
        by * read

access to dn=".*,dc=minitrue,dc=gov"
        by self write
        by * read
```

Take a close look at each section. The first allows the `rootdn` to write the password attribute for any entry. However, aside from this user, only the owner of the particular `userPassword` can modify his or her own entry.

The second section entry allows users to modify their own `mail` attribute. This will let a user can change his or her e-mail address.

The third section specifies that any DN in `ou=people,dc=minitrue,dc=gov` is read-only to everyone – except, of course, the `rootdn`, who has full access specified in the first section. It stops users from changing their username, UID, home directory, and other administrative attributes.

The fourth section is a catch-all setting. It still allows basic editing of settings, but prevents any action not covered in the previous three controls.

One you have added these settings you'll need to a restart the LDAP daemon. This holds true for most settings updates.

Populating the directory with manually or with LDIF

If you've gotten this far you should have a working LDAP server and a general handle on how your data will be organized. So it's time to populate the directory with data.

If you have more than a small handful of users on your system you will almost certainly want to try using a script to import your flat UNIX authentication files into your LDAP directory. This is discussed later on, but in fact there are two other ways to to enter data: manually at the command line, or using an LDIF file.

Manual entry

Populating your database manually is the most straightforward of the three approaches (although, as the word "manually" implies, it's also the most labor intensive). To add a single record, go to a command line interface and, enter the following:

```
# ldapadd -x -D "cn=root,dc=minitrue,dc=gov" -h localhost -W
dn: uid=wsmith,ou=editors,dc=minitrue,dc=gov
uid: wsmith
cn: Winston Smith
givenname: Winston
sn: Smith
mail: wsmith@minitrue.gov
objectClass: top
objectClass: mailRecipient
objectClass: person
objectClass: inetOrgPerson
^D

adding new entry uid=wsmith,ou=editors,dc=minitrue,dc=gov
```

Of course in place of `localhost` you could use whatever name your LDAP server uses and send your updates from a remote location.

The process you see above starts with a binding operation. You must bind to the LDAP server as an administrative user in order to have write access to the entire directory. Next there is an update operation which you type data into the directory manually. Finally, the process ends with an unbind operation when you press Ctrl-D. The LDAP server responds with either a verification message, or in some cases an error.

Errors are easy to encounter when you are getting used to LDAP. Sometimes you'll run into them when you try to add a value without having specified the necessary correct object class, or when you add a user or RDN that already exists. Sometimes an error could happen if you forget to include a "MUST" attributes, such as the `givenname` and `sn` attributes required by the `person` object class.

Manual entry is important to understand, but it has many caveats. You have to know which object class holds the attributes for the data you're adding, the process is very labor-intensive if you have more than a few records to add, and it is easy to introduce typos to your entry.

LDIF file entry

A more common approach to adding LDAP data is to use an LDIF file. An LDIF file is a plain text file that contains the data you'd like to insert. It can contain one LDAP entry or thousands; in this case we'll just take the same sample date used above:

```
dn: uid=wsmith,ou=editors,dc=minitrue,dc=gov
uid: wsmith
cn: Winston Smith
givenname: Winston
sn: Smith
mail: wsmith@minitrue.gov
objectClass: top
objectClass: mailRecipient
objectClass: person
objectClass: inetOrgPerson
```

If we save this data into the file `smith.ldif` we can then import it at a command line like this:

```
# ldapadd -x -D "cn=root,dc=minitrue,dc=gov" -W -f smith.ldif
```

You will be prompted for the password of the `rootdn` and then the data will be written to the directory.

Using an LDIF file is a much more controlled means of entering data. However, it is still labor-intensive because you have to write the LDIF.

Script-based data migration

The easiest (and recommended) way to enter data into your LDAP server is by using a script to take the data in your UNIX security files and use it to build LDIF input for your server. Fedora Core and other vendors provide scripts written for this purpose that are provided by PADL software.

Create a main LDIF file

Before you run the scripts it's worth creating a main LDIF file for parent objects. The scripts will generate entries that depend upon these parent objects, so you'll have to have this data ready before you can import the user and group data. You'll need organizational units named People and Group, as well as a BaseDN entry and an entry for your administrative account. So create a file named `main.ldif` and make it look something like this:

```
# BaseDN entry
dn: dc=minitrue,dc=gov
objectClass: top
```

```
objectClass: dcObject
objectClass: organization
o: Ministry of Truth
dc: minitrue

# Organizational unit for people
dn: ou=People,dc=minitrue,dc=gov
objectclass: top
objectclass: organizationalUnit
ou: People

# Organizational unit for Groups
dn: ou=Group,dc=minitrue,dc=gov
objectclass: top
objectclass: organizationalUnit
ou: Group

# Manager
dn: cn=root,dc=minitrue,dc=gov
objectClass: organizationalRole
cn: root
description: LDAP Directory Administrator
```

Now you're ready to run some scripts.

Preparation

The scripts are designed to migrate your flat files, like `passwd`, `group` and `shadow`, or an NIS database into an LDAP database. They re located in `/usr/share/openldap/migration`. Alternatively you can download the latest version of the tools from the PADL web page at *http://www.padl.com/OSS/MigrationTools.html*. When you look through the various scripts, give the README file some attention. It's worth understanding.

Each of these Perl scripts relies on the `migrate_common.ph` script, which you must modify to fit your own settings. Open the file in an editor and set these two options appropriately:

```
$DEFAULT_MAIL_DOMAIN = "minitrue.gov"
$DEFAULT_BASE = "dc=minitrue,dc=gov"
```

An important line in this file is one that enables support for extended schema. Extended schema go in your `/etc/openldap/schema/extension.schema` file, and provide attributes that help support access by programs like Microsoft Outlook and Mozilla or Netscape. Where the usual setting is 0, you'll want to change it to look like this:

```
# turn this on to support more general object clases
# such as person.
$EXTENDED_SCHEMA = 1;
```

Two problematic files the scripts will process are `/etc/protocols` and `/etc/services`. These are not security files *per se*, but like the NIS service LDAP can share more than user information. Whether you choose to share protocol and service information via LDAP or not, you'll have to comment out the following line in `/etc/protocols` ...

```
tp++          39 TP++       # TP++ Transport Protocol
```

... and the following lines in `/etc/services`:

```
whois++                   63/tcp
whois++                   63/udp
```

Just put a # symbol at the beginning of each line.

Notes on bugs and permissions

The scripts can work well, however they are prone to bugs. For this reason before running a far-reaching script like `migrate_all_offline.sh` we recommend you consider skipping it and instead running the specific scripts needed for your purposes. For authentication these are probably `migrate_passwd.sh` and `migrate_group.sh`.

Regardless of your script choice, one glaring bug is that the scripts cannot handle more than two domain names in your base DN. A domain like `minitrue.gov` configured as `dc=minitrue,dc=gov` will probably work fine, however a base DN `dc=security,dc=minitrue,dc=gov` will cause errors. The easy workaround for people using three or more domain names is to configure both the scripts and the `slapd.conf` file to use a two-level domain name. When you send the output to an LDIF file you can use a text editor to find and replace the base DN before changing the `slapd.conf` file back and uploading the LDIF data to the directory.

Another problem is that the `$EXTENDED_SCHEMA` variable does not seem to work on all systems. If you run the scripts and notice that key attributes like `mail` and `givenName` are not showing up in the LDIF output it's possible that the object class `inetOrgPerson` is being ignored. Edit the `migrate_passwd.pl` script and find the part in which it handles the `$EXTENDED_SCHEMA` variable. At the beginning there is a line containing an if statement, and at the end there are three lines for an else statement, like this:

```
        if ($EXTENDED_SCHEMA) {
...
        } else {
                print $HANDLE "objectClass: account\n";
        }
```

Comment out these four lines, but leave the intermediate lines intact. The statement will become non-conditional and the script will import your extended schema attributes.

Also, a note about permissions: some of these scripts can be used to send data directly into your LDAP database files in `/var/lib/ldap/`. This directory must be owned by the LDAP user and group or you'll encounter errors when trying to input data. Also the script-generated database files might be owned by the entity that ran the script - in other words, the superuser. So it's important to occasionally check on this directory and change ownership like this if needed:

```
# chown -R ldap.ldap /var/lib/ldap
```

Running only the recommended scripts

You probably don't need to migrate a great deal of your system's data into your LDAP directory. If you do want to migrate all of your flat UNIX files instructions for doing that are below. But it is recommended that you take a limited approach and create two LDIF files - one for your users and one for your groups.

The thing to keep in mind is that the scripts will import users and groups indiscriminately - even users like root and your system accounts will be pulled along for the ride. Since you're better off keeping these accounts in the flat UNIX security files, there is no need to add them to the LDAP directory. So you should probably start by organizing your `/etc/passwd` and `/etc/group` files so that all the system accounts are at the top; you can then very easuily remove them from the LDIF output using a text editor. This will keep your system accounts, Samba machine accounts, and other non-user entities separate.

When you are satisfied you can run the passwd and group scripts, like this:

```
# ./migrate_passwd.pl /etc/passwd passwd.ldif
# ./migrate_group.pl /etc/passwd group.ldif
```

Trim the LDIF

Open up the LDIF files you've created and look at your data. The first thing you *must* do before importing the data is delete entries you don't want in your directory. This includes the superuser account and any

accounts or groups related to services. Those can remain in the regular UNIX security files on each machine.

Import the data

Now you should have three LDIF files. You can import them with the `ldapadd` command, starting with the main file that includes the BaseDN, administrative account and OUs ...

```
# ldapadd -f main.ldif -x -v -D "cn=root,dc=minitrue,dc=gov" -W
```

... and then doing the passwd and group files:

```
# ldapadd -f passwd.ldif -x -v -D "cn=root,dc=minitrue,dc=gov" -W
# ldapadd -f group.ldif -x -v -D "cn=root,dc=minitrue,dc=gov" -W
```

Keep an eye out for errors. `ldapadd` will stop importing if it encounters a problematic entry. If your UNIX `passwd` file had text typed into the telephone field the LDAP server will reject it. You can either edit the LDIF file or fix your UNIX security files and re-export.

If you need to run the migration and import again you can remove all of the files in `/var/lib/ldap/` and then restart the LDAP daemon.

Running the global script

An alternative to running the `passwd` and `group` scripts as shown above is to migrate all of the basic flat files. If this sounds like a good idea and you'd like to share this information across multiple servers you should run this script:

```
# ./migrate_all_offline.sh
```

Modifying LDAP Entries

Once you've got your data in place, you will need to change the data in it as people come and go, as passwords change and so forth. For example, let's say we want to add some new attributes. We'd do it with the `ldapmodify` command like this:

```
# ldapmodify -x -v -D "cn=root,dc=minitrue,dc=gov" -W
password:
dn: uid=wsmith,ou=People,dc=minitrue,dc=gov
changetype: modify
add: mail
mail: wsmith@minitruee.gov
```

So we first specify the DN we wish to modify, then we set the change type as `modify` and use the `add` line to insert the `mail` attribute. The last line contains the attribute. But look it over carefully: we accidentally mistyped Winston's e-mail address. So we'll use the `ldapmodify` command to change that attribute:

```
# ldapmodify -x -v -D "cn=root,dc=minitrue,dc=gov" -W
password:
dn: uid=wsmith,ou=People,dc=minitrue,dc=gov
changetype: modify
modify: mail
mail: wsmith@minitrue.gov
```

Using LDAP for authentication

Configuring GNU/Linux Systems as LDAP Clients

A GNU/Linux system can authenticate to an LDAP server using NSS. When you install the `nss_ldap` packages you add libraries and PAM modules that facilitate authentication using LDAP lookups.

When configuring your client system, make sure you know the following parameters:

- `host` - The LDAP server.
- `base` - The BaseDN of the server.
- `binddn` - NSS identity to use when making lookups.
- `bindpw` - Password for lookups.
- `rootbinddn` - Identity to use for the superuser's directory access.
- `ssl` - Whether or not to use TLS encapsulation for secure lookups.

The easiest way to configure a client system is to use `authconfig`, which can be used on its own or as a component of the `setup` utility. First you must choose LDAP as you authentication method, which requires that you specify the server name and BaseDN. On the subsequent screen you must enter these values a second time. Once you've completed this change, take a look in your `/etc/nsswitch.conf` file and see how the automated process incorporates the LDAP option into the authentication configuration.

In the future if you want to set up your system manually instead of using `authconfig`, you can simply edit the `/etc/nsswitch.conf` file to include LDAP in the search order, like this:

```
passwd: files ldap
shadow: files ldap
```

Whichever method you choose, you can test your new configuration by deleting a local user and then logging in as that user. If you can log in without the user account in your normal UNIX `passwd` file, you're doing so via LDAP.

Authentication won't work yet, however, because PAM is not configured. Make sure that you have the PAM module `/lib/security/pam_ldap.so` and then edit your `/etc/pam.d/login` file. You should add the following lines:

```
auth       sufficient   /lib/security/pam_ldap.so
account    sufficient   /lib/security/pam_ldap.so
password   sufficient   /lib/security/pam_ldap.so
```

The `password sufficient` line is ideal for authentication testing because it will allow logins via LDAP but notactually *require* them. So you can take one test user, delete them from the `/etc/passwd` file (and from the NIS database, if you're using NIS) and then try to log in as that user. If you can, LDAP is working. In the mean time, other users can still use non-LDAP services for their logins. When you are ready to eliminate the other authentication mechanisms and embrace LDAP you can change this line to `password required` instead.

Once LDAP authentication is working you can use the `passwd` command to update LDAP passwords.

Mail Clients

You can configure e-mail clients to your LDAP server to use it as an address book. Just make sure that you set the BaseDN to be specific to your users, like this:

```
ou=People,dc-minitrue,dc=gov
```

This will weed out the groups and other organizational units so that you'll only search through user accounts.

Rolodap: LDAP as your address book

Rolodap is an free software project that uses an LDAP directory to store your contact information. It includes a PHP-based web interface so you can keep it up-to-date using any web browser.

Because of this you'll need to install and configure the Apache web server, PHP, and a PHP add-on module called PHP-LDAP. In the future this section will explain how to set up Rolodap.

Samba via LDAP

Computers using Samba for file and printer sharing often have configuration files that store special user account information, including Samba-specific passwords for each user (although this is not the case if Samba is configured to leave passwords unencrypted). When used as a primary domain controller it must also store machine accounts to maintain the client/server relationship between Windows machines and the Samba server.

Samba 3.0 and later support the ability to store this information in an LDAP directory. In fact an LDAP back end is a requirement if you plan to use Samba to emulate a Microsoft Active Directory domain.

Before you can do this you must install and understand Samba. If you have never set up Samba as a primary domain controller it is worth reviewing Chapter 28. Samba, like LDAP, has its own concepts and challenges, and it's best to tackle it separately from LDAP before taking the big plunge.

Once you're familiar with Samba, adding LDAP support is not terribly hard, and you can even migrate an existing server.

Special Samba schema

The first step is to add a special Samba schema to your OpenLDAP directory. The file `samba.schema` can be downloaded from the Samba website, and the Samba organization owns all of the attributes in it. Create an `include` line and put it in your `slapd.conf` file, then restart OpenLDAP.

Next you'll create an organizational unit for your computers. We'll call it `Machines` and add it to our global LDIF file.

```
# Organizational unit for Samba Clients
dn: ou=Machines,dc=minitrue,dc=gov
ou: Machines
objectClass: top
objectClass: organizationalUnit
objectClass: domainRelatedObject
associatedDomain: editors
```

We've bound this organizational unit to the Samba domain called `editors`.

Command Reference

Server Commands

Note: Be sure to stop `slapd` by issuing the command "`service slapd stop`" before using the `slapadd`, `slapcat` or `slapindex` commands described below. Running these commands while the directory server is running risks damaging the consistency of the directory.

The `openldap-server` package installs the following utilities:

- `slapadd` - Uploads an LDIF file's directory entries to an LDAP directory. The command `slapadd -l test.ldif` will upload the new entries in the file `test.ldif`.
- `slapcat` - Queries entries from an LDAP directory in the Berkeley DB format and saves them in an LDIF file. The command `slapcat -l test2.ldif` will output the directory into an LDIF file.
- `slapindex` - Re-indexes the `slapd` directory.
- `slappasswd` - Generates an encrypted user password value for use with `ldapmodify` or the `rootpw` value in the `slapd` configuration file `/etc/openldap/slapd.conf`.

Client Commands

The `openldap-clients` package installs these utilities for adding, modifying and deleting entries:

- `ldapmodify` - Changes directory entries using a file or the standard input.
- `ldapadd` - Adds entries using a file or standard input (actually a hard link to `ldapmodify -a`).
- `ldapsearch` - Searches for entries using a special shell prompt.
- `ldapdelete` - Deletes entries according to user input or via a file.

With the exception of ldapsearch, each of these utilities is more easily used by referencing a file containing the changes to be made rather than typing a command for each entry you wish to change in an LDAP directory. See the man page for each of these utilities for more information about how to use them.

31. GNU/Linux Security

Security is defined as the protection of your hardware and data. A secure system is reasonably safe from physical harm, and has the following qualifications with regards to its important data:

- The data is secret. Only people who should be allowed to access information can get to it. Any other users are blocked.
- The data is reliable. The data is not corrupt, and users can access it in its proper form.
- The data is available. The system doesn't crash and is properly configured.

To ensure that you satisfy these needs you should develop a security policy. The level of detail in this policy depends on whether you are dealing with a home system or a multi-host environment as a systems administrator, but in either case your policy should take these points into consideration:

Physical security: should you lock your system away from users in a physically secure environment? What environmental systems, such as air conditioning and backup power systems, should you use?

User security: are users allowed to access your system from multiple machines, and at any time?

Service security: who will you allow to access services, and how should those services be secured?

Network restrictions: what types of network traffic will you allow? Should it be restricted by a firewall of other security measure?

Encryption: should you use secured protocols, such as `stunnel`, `ssh`, and other measures?

Creating Your Security Policy

Your system's security policy is a set of rules and procedures. They should not specify programs, people or hardware. Rather, the policy should be generalized, referring to people by their roles or titles and deferring to more specialized policies for specific systems.

Responding to a System Break-in

Most detections of intrusion result from suspicious processes, missing or corrupt data, information in the log files, and other signs of trouble. When you suspect that someone has accessed your system without authorization, it is important that you document everything that you do, and what tools you use to do it. Think carefully before you touch anything.

The first thing to do is to secure the system. In doing so you should strive to maintain the current system state as much as possible. This could require that you remove the system from the network, however an intruder might implement a malicious script that will run in such a circumstance. You could also cut power to the system, but this would lose the current state with its running processes. Still, it may be better than using the `shutdown` command, which might be trojaned.

Whatever you choose to do, the outcome should be (a) recovery, (b) an image of the infected system for analysis, and (c) a detailed report of your findings.

The image is especially important, because if your organization choose to pursue legal actions against your attacker you will be well-served by having your original, unaltered disk. The image allows you to do your analysis without harming the original.

Once you have imaged the disk you should recover from the break-in by reinstalling your system. You could also replace suspect packages, but this leaves some room for error. The best approach is to rebuild the system from scratch, reconfigure it and restore user data.

Analyzing the Suspect Image

An analysis of the suspect image should be done using a variety of methods. The most simple is to review log files for any unusual activity. You can also use the RPM Package Manager to test the validity of installed packages against the database:

```
# rpm -V -a
```

Another method is to use a security tool such as `tripwire`, a utility that will be explained later in this chapter.

Controlling System Access

The best step you can take to prevent downtime and stop unwanted users on your system is to restrict access. There are many approaches to limiting use access to your system.

Hardware Restrictions

Keeping your system in a secure location can prevent unauthorized access or theft, but other considerations include physical damage by natural disasters or extreme heat or cold. In short, keep your data center as secure as you can, and as far from physical harm as possible.

BIOS Restrictions

On most machines the system BIOS can be used to restrict booting a machine using a password. Some systems also allow you to disable booting from removable media like CD-ROMs and floppy disks. On these systems you can often set boot passwords and administrative passwords: the latter prevents unauthorized changes to the BIOS.

A BIOS password and media restrictions are the only combat against Single User Mode, a mode of Linux that foregos logins and user security and gives the you superuser access. In this mode there are no running network services, however you can start any services you'd like. You can also install any software you'd like, erase or change system and user data, and so forth.

A boot loader password in LILO or GRUB (discussed below) can also help prevent unauthorized access, but a user with a LILO-enabled boot floppy can boot to their own disk and start Single User Mode with the LILO command `linux single`, thus bypassing the local boot loader. For this reason BIOS boot and administration passwords are recommended.

Boot Loader Restrictions

The LILO and GRUB boot loaders both allow you to set boot passwords.

For LILO, edit the `/etc/lilo.conf` file. After the `install=` line, add this:

```
password=yourpasswordhere
```

Then issue the command `lilo -v` to implement your changes. Keep in mind that your `lilo.conf` file contains the password in plain text. To avoid this you should move or read-protect this file.

As a means of customizing your boot options you can add the `restricted` directive to `lilo.conf`. This allows anyone to boot your system without a password, but requires the password if special parameters are used in the LILO command, such as an alternate shell or run level

For GRUB you can use an encrypted password. Run the `grub-md5-crypt` command, and at the promt enter your password.

```
# grub-md5-crypt
Password:
$1$HoGGo/$PsJrS62.k1HFj.VEQoS///
```

The output of the command is your encrypted password. Copy it and put it in your `/boot/grub/grub.conf` file like this:

```
password --md5 $1$HoGGo/$PsJrS62.k1HFj.VEQoS///
```

This should be done in a stanza, not to GRUB's global options. When you reboot you can type p and enter your password. Without the password you can only select the boot options provided by GRUB. With it you can go to command line mode to customize your boot process.

Session Restrictions

During a session you can lock your system by using `vlock`, `xlock` and `xscreensaver`. All three will prevent passers by from issuing commands or launching applications from your session by requiring a password to unlock the system.

`vlock` will lock a virtual console, whereas `xlock` will lock your X Window environment. `xscreensaver` will do the same thing as `xlock` but display the screensaver in the interim. To launch it in locked mode use the `xscreensaver` command with the `--lock` option. You can also configure it from within GNOME or KDE.

Keep in mind that a locked system can be rebooted, so session locking should probably be backed up by BIOS and boot loader restrictions.

Enforcing User Account Security

In most environments your user account system should be configured to use the maximum protection available. This includes MD5 passwords and shadow passwords, which should be chosen during installation, as well as password aging.

MD5 passwords eliminate an old shortcoming in UNIX systems that limited user names and passwords to eight characters or fewer. With MD5 you can have passwords that are up to 256 characters in length - and the longer, the better.

Shadow passwords eliminate a shortcoming in older systems that stored passwords in the `/etc/passwd` file. This allowed encrypted passwords to be read by anyone who had access to the system, whereas passwords stored in the `/etc/shadow` file are accessible only by the superuser.

If you are using shadow passwords you can use password aging. This lets you set rules for password changing; for example, to make a user's password expire in 60 days, you could use the `chage` command:

```
# chage -M 60 username
```

You can also prevent passwords from being used repeatedly by setting a minimum interval between password changes. Note that this feature is unavailable in environments that use NIS.

Authentication With NSS and PAM

GNU/Linux uses two authentication methods: Name Service Switch (NSS) and Pluggable Authentication Modules (PAM).

PAM is a set of dynamically loadable modules configured through files stored in in the `/etc/pam.d/` directory. In its default configuration the `pam_unix` module causes PAM to authenticate using instructions in the `glibc` library, which simply directs the system to use NSS.

NSS is the older configuration method. It looks in `/etc/nsswitch.conf`, a configuration file that lets you choose amongst traditional password and shadow files, local hashed databases, NIS or LDAP for

authentication. If you use the default pam_unix module, you are effectively using NSS. While this is very functional there are many other modules available.

PAM Basics

PAM relies upon the contents of two directories. In /etc/pam.d/ you'll find configuration files specific to an application or service, such as SSH. You will also find general files with common authentication configurations that are borrowed by many other files, which is useful for preventing reinventing the wheel for each program of service that uses PAM. These configuration files list the security libraries that PAM should use when a service needs to authenticate.

A program like Ethereal, for example, consults the file /etc/pam.d/ethereal, which can be configured to access these seven libraries:

```
auth    sufficient      /lib/security/pam_rootok.so
auth    sufficient      /lib/security/pam_timestamp.so
auth    required        /lib/security/pam_stack.so service=system-auth
session required        /lib/security/pam_permit.so
session optional        /lib/security/pam_xauth.so
session optional        /lib/security/pam_timestamp.so
account required        /lib/security/pam_permit.so
```

This causes Ethereal to prompt you for a superuser password before you can run the program. It will let you run the program as a normal user, but the program will run in an unprivileged mode. You will notice that all of the security libraries listed here are in stored in /lib/security/.This configuration causes Ethereal to prompt you for a superuser password before you can run the program.

Virtually every function that prompts you for a password, including setting the date and time in GNOME or KDE, relies on PAM for authentication. This prevents each application from having to create its own security system.

Pam Services

PAM provides four different types of services:

- auth - Authenticates the user, usually with a password.
- account - Allows or restricts access based upon factors like login location or the time of day.
- password - Updates the user's password information if this is necessary.
- session - Sets tasks hat PAM performs before or after a user is granted a service.

When a client accesses PAM, the client identifies itself by name; for example, it may be login or ethereal. PAM then consults the file in /etc/pam.d/ that has the same name as the service, and executes the libraries indicated in this file. It generally gets a pass or a fail for each library call, and this determines the overall success returned to the client.

The most important file to PAM is /etc/pam.d/system-auth. Many services - including login and the display managers xdm, GDM and KDM - refer to this file for their security policies. The authconfig utility, itself a subcomponent of the setup utility, updates security settings by simply changing the contents of the system-auth file.

Of course each service, even login, can have a custom behavior based upon settings in its own file. But in the case of login (and many others) you won't have to reiterate all of the settings used in system-auth because the login service calls the system-auth PAM service whenever it runs. Take a look at the /etc/pam.d/login file:

```
#%PAM-1.0
auth       required     pam_access.so
auth       required     pam_securetty.so
auth       required     pam_stack.so service=system-auth
```

```
auth       required     pam_nologin.so
account    required     pam_stack.so service=system-auth
password   required     pam_stack.so service=system-auth
session    required     pam_selinux.so multiple
session    required     pam_stack.so service=system-auth
session    optional     pam_console.so
```

Notice the number of instances of the string `pam_stack.so service=system-auth`. The use of the `pam_stack.so` library is sort of like using an include. It directs PAM to include the features of another PAM service. With Fedora, anything you do to `system-auth` can affect to many other services.

PAM Configuration

A PAM configuration file contains a few lines, each specifying one of the four services, and a specific library. Between these values is a *control flag*. The flags effect the return value of the PAM invocation; that value is either *pass* or *fail*. They work like this:

- Any `required` service must return a pass. If one fails, PAM will still run all the others so it will not be clear to the end user at which stage authentication failed.
- A `sufficient` service, if it passes, makes it unnecessary for any subsequent libraries to be called.
- An `optional` service does not affect the return value.

The `/etc/security/` directory contains many other configuration files. Some PAM libraries rely on these files for their configuration.

Noteworthy PAM Modules

There are three primary PAM modules with which you should probably be familiar. The first we've already mentioned a few sections ago. But the other two are equally important to understand.

- `pam_unix` - Provides traditional UNIX authentication. In its configuration file the `account` service ensures that the account and password are active, and the `session` service logs the opening and closing of the session. The other two services each have special options. `auth`, which sets up authentication using Name Service Switch (NSS), has the `nullok` and `nodelay` options. `password` has `nullok`, `md5`, `shadow`, `remember=n` and `nis` options.

- `pam_env` - Allows the superuser to initialize environment variables during authentication calls. It is used by `system-auth` by default, but its configuration file is commented out.

- `pam_cracklib` - Consults a cracking library to sense when passwords are good or lousy, and therefore enforces the use of good passwords.

Restricting User Logins by Location

You can restrict user logins by user name, group or location. PAM will only allow logins defined in the file `/etc/security/access.conf`, which it consults because `/etc/pam.d/login` has this in its first `account` line:

```
account required /lib/security/pam_access.so
```

Remember, of course, that you can also add this to the `/etc/pam.d/system-auth` file, which is included in the `login` service. The caveat to doing that is that it will affect other services, so if you're targeting the restriction of logins and not, for example, POP mail, you'll want to put this line in the `login` service file specifically.

Doing either will cause PAM to match the user, host, or user and terminal to the permissions listed in `access.conf`. If they fail to match then the user will be denied access. The format of the access permissions in this file consists of three fields separated by colons, in this order:

```
permission : users : origins
```

In the permission field a plus symbol will grant access and a minus will deny it. In the users and origins fields you can use user names or group names, host names and a number of operatives. For example, to stop a user from logging on at any location, you might do this:

```
-:wsmith:ALL
```

Or if you want to allow the user to access your system, but only from the host named `speakwrite5`, you could do this:

```
-:wsmith:ALL EXCEPT speakwrite6
```

Or perhaps we don't want him to come in via `speakwrite6`, but any other host is OK.

```
+:wsmith:ALL EXCEPT speakwrite6
```

Restricting User Login by Time of Day

User logins can be restricted by time - a useful thing if you want to keep users out of your system after hours. The configuration file for this is `/etc/security/time.conf`, and PAM can be configured to consult this if the `/etc/pam.d/login` file contains this line:

```
account required /lib/security/pam_time.so
```

The `time.conf` file is restrictive by nature – you are not granting rights in it, but rather taking away rights with each set of users, locations and time ranges. A line consists of four fields separated by semicolons. They are:

```
services;ttys;users;times
```

The fields are logical lists, and they take logical operators like the `&` character for a logical "and" and the `|` (pipe) character for logical "or" (this is an exclusive or, not an inclusive or - an important thing to keep in mind because while it can be Monday or Tuesday, it should never be Monday and/or Tuesday). Finally, the the `!` character is used to indicate "not," or perhaps more accurately, "everything except."

In the `ttys` field you can use an asterisk (*) to apply to all possible members of the field; the symbol can be on its own or attached to a word, such as with `ttyS*` for all serial ports.

In the `times` field you can use `Mo`, `Tu`, `We`, `Th`, `Fr`, `Sa`, and `Su`. Additionally there are `Wk` for weekdays, `Wd` for weekends and `Al` for all days. Concatenate them to make a list - for example `FrSaSu` for Friday, Saturday and Sunday.

For example, to restrict logins on all terminals for two specific users on weekends and evenings, you might do this:

```
login;tty*;wsmith|egoldstein;!Wk1800-2400|!Wd
```

Note, though, the first word in that line: `login`. That's a PAM service, and only one of many. You may notice that if you use the word `login` your users will still be able to access the machine via SSH. This is because `sshd` is a separate PAM service, so we need to add it:

```
login|ssh;tty*;wsmith|egoldstein;!Wk1800-2400|!Wd
```

The pipe character is a logical "or," necessary because each field is a logical list. Wecould also use & for "and," but no one logs in and connects via SSH in one sincle action.

We need to do add SSH to regulate SSH access even if we explicitly call the `pam_time.so` library in

PAM's /etc/pam.d/sshd file, since sshd won't respond to rules applying to login. The nice thing about this, though, is that it allows you to make up highly customized rules. You could configure it, for example, so that your users can only log into their terminals during business hours, and can only SSH in the evenings and weekends.

Killing User Logins, Starting Now

In addition to regulating user logins as described above, there is one action you can take that will keep all users from logging into your system. If you create a file named /etc/nologin all logins will be denied from all locations. Only root, and users already logged in before the file was created, will have access to the system.

GNU/Linux will remove this file during system shutdown, so the barrier is not permanent.

Restricting Superuser Logins

Superuser logins can be limited to specific terminals and serial ports; this can prevent remote logins as root in case you're paranoid enough to want it that way. The PAM module pam_securetty.so parses the file /etc/securetty for a list of valid superuser login locations. If you comment the line containing pam_securetty.so in the /etc/pam.d/login file, the configuration file will be ignored and you'll enable superuser logins from anywhere.

By default the /etc/securetty configuration includes virtual consoles and terminals. If you remove tty1, a user who presses Ctrl-Alt-F1 and tries to log in as the superuser on terminal 1 will be rejected.

If you've enabled logins over serial ports, which is useful on machines that lack monitors, you'll have to add your serial ports to securetty for the superuser to log on to them, like this:

```
ttyS1
ttyS2
```

And if you'd like to remove some login locations, simply delete them from the file. But keep in mind that a normal user can always issue the su command to become the superuser, and it doesn't matter where they've logged in. If you'd like to disable this you can commend out the pam_securetty.so line from /etc/pam.d/su.

Limiting Processes

PAM can be used to set limits on your system's resources. The /etc/security/limits.conf file contains lines that limit how much of a system's resources the a user or group can use. The format for a line is <domain> <type> <item> <value>. The domain is any entity that can use system resources, inclyding user and groups. The type is the sort of limit the domain will experience: "hard" will cut it off cold, and "soft" will give a warning. The item is one of many types of resources, including CPU time and the maximum file size the user can create.

For example, you could limit the number of processes the user wsmith can generate to 50. Just add this line to limits.conf.

```
wsmith hard nproc 50
```

User Lists

PAM can allow or deny access to users based upon a user list. For example, if you want to give shell access to all osers but you want to deny logins to a select few, you can edit the login configuration file and add a reference to the pam_listfile.so library. Just add them to /etc/pam.d/login.

```
auth required /lib/security/pam_listfile.so onerr=succeed sense=deny file=/etc/nologin
```

Any users whose names you place in this file will find themselves unable to login.

Console Users

The console group is a list of users who can log into the console, which is a fancy way of saying they can log in physically at a machine rather than just connecting to it over the network. A PAM module named `/lib/security/pam_console.so` lets members of this group perform certain administrative-level tasks even though they are not superusers: they can shut down and reboot the machine, mount and unmount removable media, and launch the X Window server. The unique thing about the console users group is that you don't add members to it: instead, users are automatically added when they log in at the console.

For most personal computers this makes sense. You don't want to prevent a desktop or laptop user from shutting down their computer or using the X Window server simply because he or she is an unprivileged user. On a server, however, you might want to restrict which users are allowed to log in at the console. You can do this by editing the `/etc/security/access.conf` file. If you disallow everyone except the superuser you'll prevent a passer-by from shutting down the system - although this setting will not guard your computer's power button or electrical outlet.

Alternatively, you could allow full console access but restrict what a console user can do. This can be customized by deleting files from the `/etc/security/console.apps/` directory. Files in this directory have the same names as programs that console users are allowed to run. Deleting the `halt` and `reboot` files, for example, will prevent users from being able to shut down the system. Inside each file is a set of parameters by which each program should run, including the name of the user under whose account the program should be run (usually the superuser) and the full path of the program.

Finally, the `/etc/security/console.perms` file specifies which devices count as local consoles, then goes on to define the devices available to console users and the permissions for each device.

Privileged Sudoers with `visudo`

Sometimes it's necessary for an unprivileged user to launch programs as the superuser. If you administer a system and spend most of your time logged into an unprivileged account but need to occasionally run administrative commands, you could add yourself to the `/etc/sudoers` file. You can edit this file by logging in as the superuser and issuing the `visudo` command. This not only lets you edit the file, but also checks syntax when you save the file.

Fedora Core and other distributions have preformatted `sudoers` files. For example, in Fedora's user privilege section of the file you can add a line like this:

```
# User privilege specification
username ALL=/bin/vi,/bin/cat,/bin/more
```

The ALL in that line refers to the login location, so it will allow the speified user to run the commands `vi`, `cat` and `more` as the superuser no matter where that user is logged in by issuing the `sudo` command:

```
$ sudo cat /etc/shadow
```

Be careful: this is more dangerous than it looks. Use `sudo` with the same care you'd give to the superuser account itself.

There are many complex things that you can do with the `sudoers` file. For example, if you plan to give `sudo` access to multiple users you create classifications of users in the User Alias section. For example:

```
# User alias specification
User_Alias ADMINS=username1,username2,username3
```

This is like setting up a group, only it is specific to the `sudo` command. Note that the ADMINS name is in caps, whereas the member names are in lowercase. Now you can go back to the user privilege from the first example and use the alias name in place of your user name:

```
# User privilege specification
ADMINS ALL=/bin/vi,/bin/cat,/bin/more
```

Again, the alias name is in caps. To take things a step further you can set classifications for your commands in the Command Alias section. For example, take the above `sudo` commands and create an EDITING alias:

```
# Cmnd alias specification
Cmnd_Alias EDITING=/bin/vi,/bin/cat,/bin/more
```

Now members of your user alias can `sudo` commands specified in your command alias. Justspecify the command alias in place of the commands.

```
# User privilege specification
ADMINS ALL=EDITING
```

Similarly you can set up aliases for hosts on your network, helping you specifying where users can run `sudo` instead of specifying "ALL." You may wish to only allow local users to run `sudo`, and not to be able to run it over an SSH session. Or you may want to allow SSH users to `sudo`, unless they are coming from outside your network.

You can also specify which commands users are forbidden to use by using an exclamation point to specify "not." So if you want anyone in the ADMINS group to be able to issue all commands except the Bash shell you'd do this:

```
ADMINS ALL=ALL,!/bin/bash
```

So now the ADMINS group can `sudo` anything, except run the Bash shell. This prevents them from having a `superuser` command line.

Additionally, in place of user names you can specify real UNIX security groups by including a percent symbol before their names. One common practice on UNIX and UNIX-like operating systems is to have a security group named wheel. This group, established in the `/etc/group` file, is an administrative group. Once your administrative users are added to wheel you can set up a rule for them in the `sudoers` file:

```
# Uncomment to allow people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL

# Same thing without a password
%wheel  ALL=(ALL)       NOPASSWD: ALL
```

This is similar to how administrative security works on Apple's Mac OS X.

32. System Monitoring

It is important to make a habit of monitoring your systems. If you don't keep tabs on their status it will be very simple for a malicious attacker to breach your security measures without your knowledge, or for a problem with a service or other process to become critical when it could have been prevented by careful monitoring.

File System Monitoring

Monitoring your file system can help you avoid running out of disk space. There are two basic tools you can use for monitoring. Both report on all currently mounted filesystems. The most common utility is `df`, which shows disk space in 1 KB blocks. You can also use `-h` to encourage more human readable units like MB and GB, and `-l` option to display only local filesystems.

`du` is the disk usage utility, which tells you more about specific files or directories on your system. By default it will show you statistics on all files in and beneath the present working directory, but you can also specify a different directory if you'd like. For example, if you know that a user is using a lot of storage on a server you can see how that resource is being used by getting an analysis of each file in that user's home directory:

```
$ du -hca /home/username
```

The `-h` option puts the file sizes in human-readable units, the `-a` option list all files, not just the count for each subdirectory and the `-c` option creates a grand total at the end. You can also get a count for specific files.

Problematic File System Permissions

An important consideration for filesystems is permissions. It is important not to assign file and directory permissions haphazardly: never give world-readable permissions unless you're sure you want a file or directory to be accessed by everyone. When installing software, leave you configuration files readable only by root, or for a daemon, only by the user that should run the program.

Unwanted S-Bits

Be very be careful of SUID and SGID permissions on programs. Some programs should run as superuser; but most programs should require superuser access to run with the UID 0. Any program that runs as the superuser will have unlimited permissions as long as the program lasts. The `passwd` command is a good example of this. While running, `passwd` lets you change your password. To do this it must edit the `/etc/shadow` file, which is only readable by the superuser. The only thing stopping an unprivileged user from changing other people's password with this command is the security *built into* the `passwd` command.

To make sure that no programs have SUID and SGID bits without your knowledge, use the find command to filter for them.

```
# find / -type f -perm +6000
```

This finds all files under the root directory, and therefore anywhere on your filesystem, that are of the type "file" and have S-bit permissions for the user or group (a 4 for the user and 2 for the group). If you see any suspicious programs in the resulting list it could mean you have been cracked.

A related consideration is "imported" SUID or SGID programs. Programs on removable media, such as a floppy disk, could be owned by the superuser or another user on another system, but then brought to your system and executed with the UID of their owner. To prevent this, removable media should be mounted with the `nosuid` option. This can be done in the `/etc/fstab` file.

Unowned Files

Keep an eye out for files on your system that have world-readable permissions. It is very rare that a file will need to be written by anyone other than the owning user or group. To search for them do this:

```
# find / -type f -perm -2
```

Another problem is the presence of files that are not associated with a valid user or group. These "unowned" files might be an indication that your system has been cracked. Or they might just mean that you installed software that was tarballed by a user whose UID and GID do not exist on your system. Either way you should probably correct the problem and assign real ownership.

Search for them like this (the `-o` option is an inclusive "or"):

```
# find / -nouser -o -nogroup
```

Special Extended Filesystem Attributes

The ext2 and ext3 filesystems have special attributes, which can be set with the `chattr` command and listed with the `lsattr` command. These were discussed in the Chapter 11 and are separate from the UNIX permissions. Using these attributes can help you protect important files.

Using `tripwire`

`tripwire` is a security program that keeps track of changes in the filesystem. Changes to configuration files might indicate a security violation; changes to binary files are an almost certain sign of replaced files.

`tripwire` operates from an encrypted database that stores file modifications, including ownership, file or directory size, and timestamp. Because of the nature of its monitoring it is usually best to install it immediately after building a system.

Configuration

After installing `tripwire` you can configure it via files in the `/etc/tripwire` directory. The `twcfg.txt` file contains configuration options. You can use it to set the default editor, the default e-mail program, the location of the policy file and other options.

The `twpol.txt` file is the one to customize. It contains a sample policy that establishes which files and directories to monitor, as well as the level of scrutiny by which they are watched.

There are likely to be dozens of files in this policy that don't exist on your system because the file is based upon a full installation of Fedora Core. If you're not sure which files to comment out of the policy you might want to skip to the next step; the output of your database initialization will tell you which files are not present, and you can remove them from the policy and use the `twadmin` utility to update the database.

Installation

One you've configured the policy file, or given up trying, run the installation script located in the same directory.

```
# ./twinstall.sh
```

This creates the keys and passwords that an administrator can use to run and modify the program. It also takes the contents of the two configuration files and creates encrypted versions of these files named `tw.cfg` and `tw.pol`. The original configuration files are kept, but are not used by `tripwire`. You'll need them later on if you want to create new versions of the encrypted configuration files.

The databases

After running the installation script, create the initial databases using the `--init` option.

```
# tripwire --init
```

The database is kept in `/var/lib/tripwire`, and it is password protected. In this way tripwire guards against attacks by malicious users who manage to "root your box," which is slang for unauthorized superuser access on your system.

The installation script should never be run again. You must make any further changes using the administration utility.

Tweaking

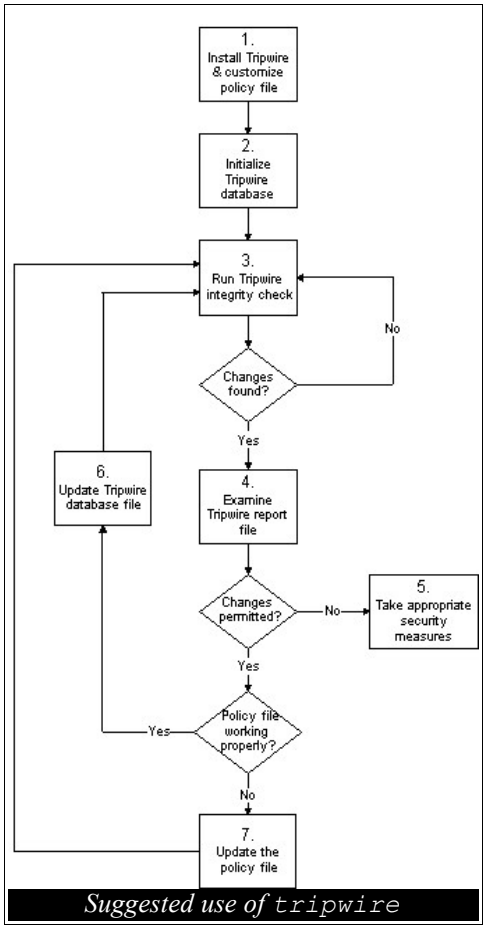
The administration utility is called `twadmin`. If you modify the policy in `twpol.txt` (for example, you might have added or removed software, or you might be removing unneeded files during installation) you can use it to inform `tripwire` of your changes.

```
# twadmin --create-polfile -S
/etc/tripwire/site.key /etc/tripwire/twpol.txt
```

Using `tripwire`

The `-check` option will run a check on your system.

```
# tripwire --check
```



Suggested use of `tripwire`

After the check you may find files that have been changed. Check into each of them. If you detect a problem, respond appropriately. If the changes are expected, update your database with the results of the check using the `--update` option.

```
# tripwire --update
```

Reporting

The `twprint` command can generate a report for you. Just specify the report's filename:

```
# twprint -m r --twrfile file-name
```

Automatically generated reports are kept in the `/var/lib/tripwire/report/` directory.

Automatic Checks

The Fedora Core RPM will install a shell script in `/etc/cron.daily`. This will run a nightly check on your system, comparing it with the previous database. The check will yield a comparison report, which will tell you which files and directories have changes since the last check. The report is e-mailed to the superuser.

System Logs

The simplest way to monitor your system's daily state is to check its log files. This can help you detect hardware problems and security breaches. It is also the first place to go when you're having a software

issue.

By convention the log files are kept in the `/var/log/` directory. Each application can have its own log, however there are a few common ones that should be on all systems:

- `maillog` - This logs e-mail transactions, including incoming mail and POP or IMAP sessions.
- `messages` - This logs general system messages.
- `secure` - Stores authentication messages and messages from `xinetd` daemons.
- `xferlog` - An FTP log.

Configuring the Logging Daemons

There are two log daemons, which write the log files in response to messages from applications and services. `syslogd` is the general log daemon used by most services. `klogd` logs Linux kernel messages. Both are configured from the file `/etc/syslog.conf`.

In the configuration file you'll find three fields. They can be summarized like this:

```
facility.priority log-file
```

The first two, separated by a dot, are the facility and priority; the facility is a classification of log messages (such as mail) and the priority is the severity level of the message. The third field is the name of the log file to which messages matching the facility and priority combination should be written. For example, this line will log all mail-related messages, regardless of priority:

```
*.info;mail.none;authpriv.none;cron.none    /var/log/messages
```

Obviously this also logs all `info` priority, `authpriv` facility and `cron` facility messages. It is the default entry for recording entries to the `messages` file. For each entry the log daemons will record not only the specified priority, but also any messages at higher priorities as well.

Facilities and Priorities

There are numerous available facilities. You can also make you own if you have software that generates messages with its own facility name. Here are the default ones:

- `authpriv` - Security and administration messages
- `cron` - Messages from both `at` and `cron`
- `daemon` - Other daemons
- `kern` - Kernel messages
- `local` - Reserved for local use facility.
- `lpr` - Printing messages
- `mail` - E-mail messages
- `news` - News messages
- `syslog` - Internal daemon messages for `syslogd`
- `user` - User-level messages

There are also a number or priorities:

- `debug` - Debugging messages
- `info` - Informative messages
- `notice` - Messages indicating a significant condition
- `warning` - Warning messages
- `err` - Error messages
- `crit` - Critical condition messages
- `alert` - Messages calling for an immediate response
- `emerg` - System no longer available

Special Characters

You can tweak your system logging even further with the use of three special characters.

The * (asterisk) symbol is a wildcard. It causes the daemons to log all items that apply to the field it occupies. The example line above uses the string *.info to log all messages with a priority of info or higher to /var/log/messages. You reverse this and log all priorities for a given facility:

```
mail.*       /var/log/maillog
```

And you can use the asterisk in the log file field as well. Rather than log to a file, it will send the log message to all consoles so that local users will be aware of it. This is the case with all emergency messages:

```
*.emerg      *
```

The = (equal) symbol can be used with the priority field only. It will cause the daemons to log messages of precisely the specified priority, rather than use the default behavior of logging the specified priority and higher. For example, if you want to log all mail-related error messages to a special file, but not also log messages of higher priorities, you could do this:

```
mail.=warning/root/mail-warnings
```

The ! (exclamation mark) symbol is used to exclude a facility or a priority. So you could modify the above example to log all mail messages that are *not* warnings:

```
mail.=!warning            /root/mail-non-warnings
```

Advanced Logging

In addition to logging to a file, you can also send log messages to users by using a comma-separated list. Or you can log to a terminal. This example will log all events to virtual console 8, so at any time you can press Ctrl-Alt-F8 to see the most recent log messages:

```
*.*          /dev/tty8
```

Perhaps the most creative thing you can do with the log daemons is pipe the log message to a named pipe. That is, you can pipe the output like this:

```
*.emerg      |/name/of/program
```

Logging to other machines

Your syslog daemon can operate as a log server, accepting log entries sent to it from other machines. Or it can be configured to send its logs to another server. To send you messages to another machine, simply use the hostname after a an @ symbol. For example:

```
mail.*       /var/log/maillog,@logserver.domain.com
```

This line sends mail logs to both the local file and to the host named logserver. Note that it does not specify the remote filename; how the log is processed by logserver is up to the syslogd configuration on that host.

Centralizing your logging can simplify analysis. It can also be good from a security standpoint, since it makes it hard for an attacker to hide his or her tracks after causing log output. But if you choose to centralize make sure you don't ignore you local files. The logging host could crash, or your network could go down, and it's important not to lose log output.

On your log server you'll have to configure `syslogd` to accept log entries on a log server you'll have to edit the `/etc/syslog/syslog` file to include the `-r` option like this:

```
SYSLOGD_OPTIONS="-r -m 0"
```

You'll also have to make sure that your firewall allows connections on the syslog port.

(ToDo: find out which port syslog uses)

Analyzing Your Log Files

You should read through some of your system's log output on a regular basis. Unfortunately your log files may tend to get very large, depending on what your computer is doing. So to cut to the chase you can install a log analysis utility, such as `logwatch`. By default the `cron.daily` job on Fedora Core systems will run `logwatch` daily. Systems with a heavy workload might be better candidates for more frequent reports, perhaps even every hour.

You can configure `logwatch` using the files in the `/etc/log.d` directory. When configuring it, keep in mind that the idea is to filter out anything that might be considered "normal." Your filter should be negative, constantly refined to eliminate log messages that are of little interest. What you're looking for are failed logins, attempts to access a service from a surprising location, error messages and other anomalies.

`logwatch` is written in Perl, a language ideally suited for quickly writing programs that can analyze text files. Most analysis tools for GNU/Linux are written in Perl, and it's a good language to learn if you'd like to develop some yourself.

Keeping an an Eye on Processes

You should periodically monitor the processes running on your system. If any take up too many resources then you need to either get a faster computer or consider using a different program. Otherwise they may seriously hamper your system's ability to host its important services.

While the `ps -aux` command will give you a good list of running processes, a better tool for analysis is `top`. `top` will show you a real-time list of active processes, and by default it updates every five seconds. It will tell you CPU time, RAM usage, elapsed time and other stats for each process, and it will let you sort by memory usage if you press M (yes, a capital M), processor time if you press P or elapsed time (the age of the process) if you press T.

You can also interact with the processes by pressing r to renice them or k to kill them. `top` will prompt you for a PID and take care of the rest. to list the processes of a single user, press u and then enter the user's name. For a list of functions, press h. Press q to quit.

For a graphical experience, KDE user will want to check out `kpm`, the KDE Process Manager. GNOME users can try `gtop`. Both are graphical interfaces to `top`.

Setting Limits

It is useful to limit the resources that your processes are allowed to use. The most effective way to do this is to use PAM's `system-auth` module in `/etc/pam.d/` to set global limits on resources. `pam_access.so` and `pam_time.so` can all be added to this module to restrain your system's availability, and `pam_limits.so` will let you limit CPU time.

Monitoring Users

To keep track of what your users are doing it's handy to use the `last` command, which will show your system's login history. You can also use `ac`, which tells you your user's login connect times. It gets this

information from `/var/log/wtmp`.

A more advanced approach is to install the process accounting utilities, available in the `psacct` RPM. Once installed, you can turn it on by creating the accounting log file and then activating it:

```
# touch /var/log/pacct
# accton /var/log/pacct
```

To analyze this file, use `lastcomm` for information on previously entered commands and `sa` for a summary sorted by user.

To turn off process accounting, use the `accton` command with no arguments.

33. Service Security

One of the key back doors into your system is the set of running services. Daemons listen for input and give their programmed response, but cases they can be crashed remotely, giving an attacker elevated access to the system.

Another service-related concern is network security; can an attacker monitor the activity on a service and use it to crack your machine? Some solutions are provided herein.

Evaluating System Services

It is important to determine which services are configured to listedn for input. Aside from those reasons mentioned above, there are other practical considerations. For example, there is no point in using processing power by running your SSH daemon if you don't intend for users or administrators to connect via that protocol. It's a good idea to account for each active service and disable it if you don't need it.

The best way to do this is with `chkconfig`. This utility will list both System V and `xinetd` daemons. The best way to start out with this command is to use the `--list` argument. In fact, for a quick fix you can use this command to filter for services that are configured to run:

```
# chkconfig --list | grep on
```

Scan through the running services and disable the ones you do not need. This will free system resources as well as close potential security holes.

To shut off a service, such as Sendmail:

```
# chkconfig sendmail off
```

Keep in mind that this changes the *configuration* of services. It doesn't actually start or stop them. You should follow a `chkconfig` spring cleaning by manually shutting down services with the `service` command. For `xinetd` services you are changing the settings of the service, which `xinetd` will re-check each time a request for that service is made. For this reason you do not need to restart `xinetd` after enabling or disabling its services.

Finding Running Services

Since services can run from xinetd, System V scripts or a variety of other methods, it can be useful to use a more general network-based command to check for running services on your system. You can do that with this command:

```
# netstat -taupe
```

The output of this command tells you a variety of things, as summarized below:

- `-t` Shows all TCP connections
- `-a` Shows all active sockets, both connected and listening.
- `-u` Shows all UDP connections
- `-p` Shows the process ID and name of the program.
- `-e` Shows extended information. Add a second e for even more data.

Note that this only shows processes using the network.

Finding Services on a Remote Host

To find services running on a remote host you should use `nmap`. `nmap` is a very powerful tols with many options. It can not only tell you about open ports, but also can determine the operating system of the

remote host and scan a large range instead of a specific machine.

The only caveat to `nmap` is that malicious users also have access to it. So the same tool that you'd use to help secure your system can be used by an intruder to break into it ... but this is all the more reason to understand the intricacies of the program.

Let's take a look at a sample `nmap` command and its very verbose output:

```
[root@localhost root]# nmap -sSUR -P0 -vO 192.168.1.1

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host  (192.168.1.1) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.1.1)
Adding open port 80/tcp
The SYN Stealth Scan took 7 seconds to scan 1601 ports.
Initiating UDP Scan against  (192.168.1.1)
The UDP Scan took 8 seconds to scan 1468 ports.
Adding open port 520/udp
Adding open port 53/udp
Adding open port 67/udp
Adding open port 1400/udp
Adding open port 69/udp
Adding open port 5050/udp
Initiating RPCGrind Scan against  (192.168.1.1)
The RPCGrind Scan took 7 seconds to scan 0 ports.
For OSScan assuming that port 80 is open and port 1 is closed and neither are
firewalled
Interesting ports on  (192.168.1.1):
(The 3062 ports scanned but not shown below are in state: closed)
Port       State       Service (RPC)
53/udp     open        domain
67/udp     open        dhcpserver
69/udp     open        tftp
80/tcp     open        http
520/udp    open        route
1400/udp   open        cadkey-tablet
5050/udp   open        mmcc
Remote operating system guess: Linksys BEFW11S4 802.11B WAP
TCP Sequence Prediction: Class=trivial time dependency
                         Difficulty=1 (Trivial joke)
IPID Sequence Generation: All zeros

Nmap run completed -- 1 IP address (1 host up) scanned in 23 seconds
```

This command scanned the host at 192.168.1.1, and adds the following options:

- `-sS` Performs a TCP SYN scan, which is also called a "half-open" scan because there is no full TCP connection; rather `nmap` sends out a SYN packet as if it were going to open a real connection and then waits for a response.
- `-sU` Performs a scan to determine which UDP ports are open.
- `-sR` Performs a scan on open TCP and UDP ports to determine whether any of them are used by RPC (remote procedure call) daemons.
- `-P0` Refrains from pinging the target hosts before scanning them. This allows the scanning of networks that block ICMP echo requests.
- `-v` Outputs data in verbose mode.
- `-O` Attempts to identify the operating system using TCP/IP fingerprinting. It compares subtleties in the underlying network stack with a database of known OS fingerprints.

In this case we've found a few open services, including a web server on port 80, a DHCP server on port 67 and a DNS server on port 53. We've also determined that the operating system is "Linksys BEFW11S4 802.11B WAP," or a Linksys wireless hub and router appliance. This is not exactly correct because it's not wireless, but it is indeed a Linksys router.

You can scan multiple machines by specifying a range. For example, 192.168.1/24 will scan the whole subnet. See the man page for more options.

To make things easier `xnmap` and `nmapfe` provide graphical interfaces for `nmap`.

`identd`

When a client connects to a remote daemon, that daemon can query the client for information about the user who has initiated the TCP connection. The reply is encrypted. Using this service, many services record user names, and a scan through a log file will show you the value of having `identd`.

However, `identd` is more useful on systems in a trusted environment. On a system you don't trust, `identd` may provide false information. But in a trusted environment it can tell you a great deal about who did what.

Host-based Security with TCP Wrappers

Many applications are configured to use a special library named `libwrap`, which provides a functionality called *TCP wrappers*. This lets you set up a host-based access policy for your system that all `libwrap`-linked services can share. Of course many dome daemons can be customized with their own security settings, but wrappers provide a nice central mechanism, should you wish to use it.

All `xinetd` daemons use TCP wrappers automatically, so the use of the `libwrap` configuration files is important to understand. Other `libwrap`-enabled daemons include Sendmail, `gdm`, `sshd` and `portmap`.

`libwrap` depends upon two files in `/etc` named `hosts.allow` and `hosts.deny`. These files contain access control lists that services can use to decide whether to accept or reject connections. The access control lists, which are usually based upon allowed and denied IP addresses, can be specified for individual services.

The `libwrap` checking order

When a remote client tries to connect to a `libwrap`-enabled daemon, the daemon first checks the `hosts.allow` file to determine whether the connection is explicitly allowed. It uses a simple policy of "first match." That is, if it finds a match it will stop checking and not move further on in the file, nor will it move onto the second file, `hosts.deny`.

However, if it doesn't find a rule allowing the connection in `hosts.allow` it will check `hosts.deny` to see if the service is specifically denied. Again, its policy is simply "first match."

If the daemon gets through both files and doesn't find anything it will allow the connection.

An ideal default policy

Because a match must occur for a denial to take place, it's easy to err on the side of insecurity by allowing unforseen access. Better to err on the side of security: if you put this simple line in `hosts.deny` you will deny all access:

```
ALL: ALL
```

Since `hosts.allow` is checked first, any services specifically permitted in that file will work fine. But all other access will be denied.

`libwrap` file syntax

The basic syntax of configuration in both `hosts.allow` and `hosts.deny` is basically this:

```
daemon_list : client_list [ : shell_command ]
```

The daemon list can consist a single item or multiple items separated by commas. For example:

```
sshd, sendmail:           10.0.2.3
```

It can use the special wildcard ALL. For example you can give a green light to a host for all services by using ALL:

```
ALL: 10.0.2.3
```

Or you can allow that host access to all daemons except one:

```
ALL EXCEPT sshd: 10.0.2.3
```

In some cases you might have more than one network interface on your system, and want to provide services differently on each network. In that case you can use the `@` symbol to specify the interface by IP address:

```
sshd@10.0.2.20:           10.0.2.
sshd@10.0.3.10:           10.0.3.
```

The only exceptions to this method are -based services, which use `portmap`. To block them you have to block `portmap`, though it will take a short time for any changes to `portmap` to take effect. NIS and NFS are two daemons that can be blocked via `portmap`.

The client section of the TCP wrapper syntax can also be customized. As we see in the example above, leaving off the last octet of an IP address can allow or deny access to that entire subnet. You can also use a netmask for much the same effect (e.g., 10.0.2.0/255.255.255.0). Of you can use a domain name, a host name or a network name. The network name can be defined in `/etc/networks` or in NIS, and it looks something like `@networkname`.

You can also filter by user name, though this uses `identd` so it is not entirely trustworthy.

Special syntax

Aside from ALL, there are a few other potentially useful wildcards:

- `LOCAL` - Matches when a remote host does not have a dot in its hostname.
- `KNOWN` – Matches when a host or user can be looked up.
- `UNKNOWN` - Matches when a host or user cannot be looked up.
- `PARANOID` - Matches hosts where a DNS lookup fails to match a reverse DNS lookup.

Additionally you can use the EXCEPT operator, which can be used to create rule exceptions and can be used multiple times in a given line, as in:

```
ALL : ALL EXCEPT 10.0.2 EXCEPT 10.0.2.3
```

Running Programs with TCP Wrappers

`libwrap` allows for the use of some additional options with TCP wrappers. They are discussed in the `hosts_options` man page, but the `spawn` option is the most commonly used. This option will run programs or scripts whenever a match is detected on a `libwrap`-enabled service.

For example, you could add this to `hosts.deny`:

```
in.ftpd : ALL | spawn echo "FTP attempt from %c to %s" | mail -s FTP-warning root
```

This will send an e-mail advisory to the postmaster whenever someone attempts to FTP to your host. Because the rule is in `hosts.deny`, it will also prevent access. The `%c` variable will be substituted with

client information, and the `%s` with server information.

TCP Wrappers in Action

The following is an example of a TCP Wrappers configuration. To clarify the use of the two configuration files we'll show each in turn, then highlight some of the outcomes of this configuration.

First we'll start with the `hosts.allow` file:

```
in.ftpd: 10.0.3
in.telnetd, portmap: 10.0.3.4
```

And then `hosts.deny` file:

```
ALL: .evil.com EXCEPT trusted-host.evil.com
in.ftpd, portmap: ALL
imapd: 10.0.3. EXCEPT 10.0.3.45
```

And now some points to observe within these rules. In the first file we've configured `libwrap` such that only hosts on the 10.0.3. subnet can connect to the system via FTP. And only 10.0.3.4 can connect via Telnet and mount NFS shares (the latter because of `portmap`).

In the second file we've prohibited any connections from the evil.com domain. Probably a good idea. But one trusted host in evil.com is still allowed. Similarly, all hosts on the 10.0.3 subnet are prohibited from using IMAP except for one.

(ToDo: why is the FTP and portmap line in hosts.deny when denial is implied in hosts.deny?)

`xinetd` Security

Aside from using `libwrap`, `xinetd` services have their own special set of security functions. If a service is allowed through TCP Wrappers, the `xinetd` security kicks in. Unlike TCP Wrappers, `xinetd` security settings are stored in the configurations for each service.

Host and Network Security

The special security options are `only_from`, which establishes valid hosts, and `no_access`, which establishes hosts to deny.

For example, the IMAP service's configuration in `/etc/xinetd.d/imap` can be reconfigured like this:

```
{
        disable = yes
        socket_type             = stream
        wait                    = no
        user                    = root
        server                  = /usr/sbin/imapd
        log_on_success          += HOST DURATION
        log_on_failure          += HOST
        only_from               = 10.0.3.0/24
        no_access               = 10.0.3.4
}
```

In the last two lines we've restricted access to hosts on the 10.0.3 subnet, with the exception of 10.0.3.4, which can IMAP to its heart's content.

The hosts used in these rules can be simple numeric IP addresses like 10.0.3.4, IP addresses with netmask ranges like 10.0.3.0/24, host names, domain names, network names pulled from the `/etc/networks` file, and special "factorized" addresses. These latter are sets of addresses in arrow brackets, like 10.0.3.{3,4,5,7}.

Access by Time

`xinetd` services can be configured to work only within specific time frames. For example, if we add this line to the IMAP configuration file above, we can set the IMAP service so that it will only allow connections for the three hours between 9 AM and 12 PM.

```
access_times                = 09:00-12:00
```

Access by Source

An additional per_source option will limit the number of connections from a given host. For example, if we wanted to limit sessions to ten per connecting host, we could add this:

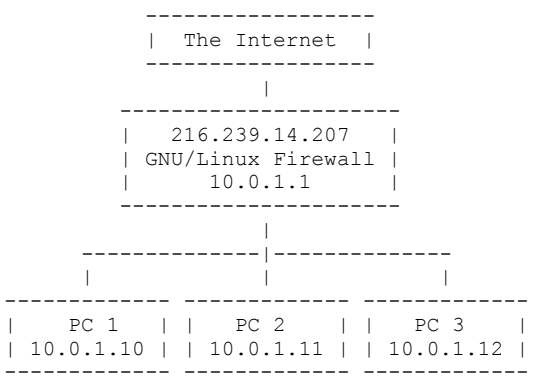
```
per_source                  = 10
```

The eleventh simultaneous connection would be dropped. This can be useful in preventing a denial of service attack from maximizing your system's resources.

34. Firewalls, Routers and IP Masquerading

Linux on the Edge of Your Network

Most people think of routers and firewalls as specialized appliances: a router connects two or more networks, while a firewall keeps private networks out of harm's way. In fact neither functionality is unique to commercial network appliances: any decent GNU/Linux system can act as a router and firewall for your entire network. For example, if you wanted to connect your private network to the Internet you could design a topology that looks something like this:



Of course any given GNU/Linux system can have its own firewall, so if you have a small network it's not necessary to have a dedicated machine. But for environments with many networked computers a configuration like this one will allow all of the machines to access the Internet using a single IP address, and allow you to set up a single point of network security for multiple machines.

The Linux Router

A router is a computer that connects two different networks. It listens for TCP/IP packets on both networks and if a it hears a packet on one that is addressed to a host on the other, the router passes it from one network interface to the other. If you've configured a network interface you know that all machines need a default gateway. The router is that gateway.

A router can not only pass packets to another network, but it can also also transform packets from one network type, such as Ethernet, to another type like ATM. And it can resize packets if the destination network supports a smaller packet size.

The Linux kernel can act as a router when IP forwarding is enabled. You can turn it on by adding this line to the `/etc/sysctl.conf` file:

```
net.ipv4.ip_forward = 1
```

You have to reboot to activate the setting. If you don't want to reboot, or if you just want to try the setting temporarily, you can edit the running kernel by changing the value of the `ip_forward` file in the `/proc` file system with this command:

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
```

Basics of Linux Routing

Once IP Forwarding is enabled the `route` command will let you to manipulate the kernel's routing table. The most basic use of the command is to list your current routing table.

```
# route -n
Kernel IP routing table
```

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         192.168.0.1     0.0.0.0         UG    0      0        0 eth0
```

You can add new hosts to this table and specify their gateways with the `add` option.

```
# route add -host 10.0.4.2 gw 10.0.3.2 eth0
```

Now if your router hears a packet destined for 10.0.4.2 it will forward it to 10.0.3.2, another router. Obviously the second router must also be configured properly for such an exchange to work. You can use the `traceroute` command to troubleshoot network routing problems.

To permanently add routes you can add them to `/etc/sysconfig/static-routes`.

Dynamic Routing

Instead of adding static routes and determining your network topology by thorough testing you can do dynamic routing. Adding the `routed` or `gated` daemons allows you to use the RIP, RIPng, OSPF and BGP4 routing protocols. These can route dynamically, negotiating routes between multiple routers. Routes are chosen using a number of options, including the lowest latency.

Network Address Translation

The Linux kernel can also do Network Address Translation (NAT). A NAT translates an IP address that is valid on one network into an address that is valid on another. One of the most common uses for this function is IP Masquerading, wherein multiple hosts access the Internet with a single IP address. This is necessary mostly because of the short supply of IP addresses in TCP/IP version 4,but it also has some security benefits as well.

With masquerading enabled, a private address like 10.0.3.4 can be translated into a real Internet IP address like 216.239.14.207.

`iptables`: The Linux Firewall

There are two types of firewalls. The first is a *packet filtering router*. This type of router screens network traffic, filtering out select packets while it also performs the functionalities of a router. Packet filtering is done on the protocol layer of your network. This means that it pays attention to the types of packets passing through it by reading packet headers, but it pays no attention to the packet contents. Very simple computers can act as packet filters, as the system requirements are not major.

The other type of firewall is the *proxy*. Proxies accept requests from clients and perform their own requests to the destination servers, then forward the server's response back to the client. Because proxies reside on both networks they are sometimes called *dual-homed hosts*. Unlike packet filters, proxies work on the application layer of your network. They pay no attention to network routing rules and packet headers, but they tend to be smarter about filtering content. Unfortunately in order to filter content and accept and receive many simultaneous connections, proxies require much more heavy-duty hardware than packet filters.

Advice on Routing and Firewalling

A firewall can exist on any standalone machine, and the discussion that follows is useful to secure a single GNU/Linux computer. But a more common use of a firewall is as a packet filtering router for multiple machines. When setting up such a firewall, be sure that your network is working by testing it thoroughly. It's silly to troubleshoot firewall problems when your router functionality isn't working properly to begin with. Once you're sure everything is fine, start making rules. It is important to make rules in a positive sense - start out by denying access to everything, then add rules for specific purposes.

iptables

Most modern distributions of GNU/Linux use the `iptables` firewall. `iptables` is an advanced filtering module designed to let you manipulate the Linux netfilter stack, which is the part of the kernel responsible for routing, IP masquerading and NATing. It is the fourth packet filtering technology to appear in Linux; in the 2.2 kernel it was preceded by the `ipchains` firewall. While 2.4 still supports `ipchains`, the netfilter stack and `iptables` firewall are now the default in most distributions.

`iptables` can filter traffic with rules based upon MAC addresses, source and destination network interfaces, connection frequency and connection tracking. The latter feature is important because it makes it possible to have `iptables` pay attention to sessions and determine the "statefulness" of packets, allowing the passage of packets that would otherwise disobey firewall rules if they are part of an established session on another port.

The basic path of a packet through `iptables` might be summarized like this:

1. The packet reaches the firewall

2. A *checksum* feature drops the packet if it is corrupt.

2. A *sanity* drops any malformed packets

3. NAT begins with PREROUTING, wherein a destination NAT (DNAT) may be performed on the packets before any routing occurs.

4. `iptables` routes the inbound packet via either the FORWARD or INPUT chain.

5. If FORWARD, the FORWARD chain evaluates the packet. If ACCEPT, it is sent to POSTROUTING and skips steps 5-7 entirely.

5. If INPUT, the INPUT chain evaluates the packet. If ACCEPT, it is sent to a local process. When this happens the packet leaves netfilter and is accepted by the local system.

6. The OUTPUT of a local process is evaluated, and if ACCEPT, a n OUTPUT packet is routed based upon relevant netfilter rules.

7. An OUTPUT NAT may occur wherein the outgoing packets is DNATed before it is sent to the output chain.

8. POSTROUTING may perform a source NAT (SNAT) on the packet before sending it on its way.

Tables and Other Syntax

Your `iptables` firewall uses different tables for different types of actions. The *filter* table does packet filtering, and the *NAT* filter does network address translation. The rules for these tables are stored in separate *chains*. All of the functions listed above - INPUT, OUTPUT, FORWARD, PREROUTING and POSTROUTING - are built-in chains. You can also set up user-defined chains, however the built-in chains are all most people tend to need.

`iptables` chains can be used as commands at a shell prompt, and this can be a good way to learn how they work before you try to put together a file. The basic syntax of the `iptables` command looks like this:

```
# iptables [-t table] <action> [pattern] [-j target]
```

The `-t` option specifies a table, either `filter` or `nat`. The actions are performed on the chains described above, and include `-A` for add, `-D` for delete, `-L` for a list of the chain's rules and `-F` for a flush

of all rules from the chain. The actions and patterns are many and varied, and the target is the resulting action. In its most basic form the command looks like this:

```
# iptables -t filter -A INPUT -i lo -j ACCEPT
```

This command uses the filter table (in fact we could omit the `-t` option here, since the filter table is the default option for the command). It then applies the INPUT chain to it. The chain examines each incoming packet to determine if it is from the interface named `lo`. This is the local loopback address; if a packet is from the loopback it's accepted. This is a pretty standard rule that appears on most systems; without it your system would be unable to communicate with itself.

The one thing omitted from the above example is the pattern option. Patterns specify when the rule matches a given packet's attributes. For example, an `-s` patter matches to a source IP address. A `-p` pattern matches a protocol. If we wanted to allow incoming SSH connections, we'd allow incoming TCP connections on port 22 like this:

```
# iptables -A INPUT -p tcp -m tcp --dport 22 --syn -j ACCEPT
```

Admittedly a bit more complicated. This example introduces some new options. And they have to be in this precise order. That is, the general extensions must be added first, then the protocols, then the rules.

Examples

Let's say you want to set up a rule to deny access by a particular machine with the IP address 10.0.4.23. You can specify its source IP address with the `-s` (source) option:

```
# iptables -A INPUT -s 10.0.4.23 --j DROP
```

Or we could do the opposite, and alow this host but deny everyone else using the exclamation point as a negator.

```
# iptables -A INPUT -s ! 10.0.4.23 --j DROP
```

Of course depending on your network topology you could have multiple machines with this address. In that case you might want to specify the network interface, which happens to be `eth1`.

```
# iptables -A INPUT -s 10.0.4.23 -i eth1 -j DROP
```

Or perhaps you're seeing ICMP packets flooding your machine from the multiple machines on the 10.0.4. subnet. In that case you'd block ISMP from the whole Class C subnet:

```
# iptables -A INPUT -s 10.0.4.0/255.255.255.0 -p icmp -j DROP
```

Or you could deny TCP SYNc packets, used to initiate TCP sessions, to your server to prevent remote access to a particular service:

```
# iptables -A INPUT -s ! 10.0.2.0/255.255.255.0 -p tcp --dport 25 --syn -j DROP
```

This will drop all attempted connections from outside the 10.0.2 subnet, in this case allowing only local e-mail to reach an SMTP server.

Adding and Removing Rules

In the examples above we created a number of separate rules, but it's important to understand that they are all part of the same *chain*: the INPUT chain of the filter table. This is because they're all part of the same single function: examining and filtering packets as they enter your system.

Because these rules are all part of a single chain you can list them all at the same time. To list a chain, use the `-L` option.

```
# iptables -t filter -L INPUT
```

To list all of your rules simply omit the name of the chain. Note that in some cases in which you have specified one or more hosts the `iptables` command will attempt a DNS lookup. To avoid this you can use the `-n` option for numerical addresses. And to make things more interesting you can add the `-v` option for verbose output, which will tell you the number of packets and bytes that have passed through each chain

The `iptables` command also lets you delete rules from your chains. Just pay attention to the order of the output. If you'd like to delete the fourth rule of the chain, you'd do this:

```
# iptables -D INPUT 4
```

Instead of using the rule's number you can do a little more typing and exactly replicate the rule's syntax, though this is really more useful for scripting than for interactive use.

If you want to quickly clear, or "flush," all rules form a chain, use the `-F` option:

```
# iptables -A INPUT -F
```

Permanent Rules

When you reboot, any chains that you have created will be washed away. It would be a bummer if you had to input the rules each time you started your system, but fortunately there is a way to record them more permanently. When you are finished tweaking your `iptables` rules and want to record them, use the `service` command to save all chains in memory into the `/etc/sysconfig/iptables` file:

```
# service iptables save
```

Though it's treated like a System V service, and responds happily to the `service` and `chkconfig` commands, `iptables` is based upon the netfilter stack in the kernel, so it never really goes away. When you use the `service` command to start it or stop it, you're really just using the startup script to add the chains in your configuration file, or to flush them from memory.

While it's good to understand the use of the `iptables` command to add and remove rules, you can also use the file like a standard service configuration file. Its contents look more or less like a series of `iptables` commands, minus the word `iptables`. For example, a typical desktop configuration might look like this:

```
-A INPUT -p tcp -m tcp --dport 22 --syn -j ACCEPT
-A INPUT -p udp -m udp -s 0/0 --sport 67:68 -d 0/0 --dport 67:68 -i eth0 -j ACCEPT
-A INPUT -p udp -m udp -s 0/0 --sport 67:68 -d 0/0 --dport 67:68 -i eth1 -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m tcp --syn -j REJECT
-A INPUT -p udp -m udp -j REJECT
```

This set of rules prevents certain types of nasty packets while allowing all traffic within the local machine (`-i lo -j ACCEPT`) as well as incoming SSH connections (`--dport 22 --syn -k ACCEPT`).

Saving and Restoring

Not only can you save the current ruleset to your configuration file, but you can also save to an alternate file to create a backup, and import the backup later on. You can do this with two special commands. First, to save:

```
# iptables-save > filename
```

And to restore later, or on another computer:

```
# iptables-restore < filename
```

Connection Tracking

`iptables` supports connection tracking. The idea of connection tracking is that the firewall is aware of continuous sessions, so only the first packet of a given session needs to be filtered. This is especially useful in complex sessions like passive FTP. In passive FTP a session on port 21 is switched to a different port of the server's choosing. If the FTP client uses firewall that does not support connection tracking the passive mode data transfer may be blocked; `iptables` is capable of allowing the session to continue despite the port change.

The firewall does this by watching the "statefulness" of packets. There are four connection states:

- `NEW`
- `ESTABLISHED`
- `RELATED`
- `INVALID`

To allow for connection tracking you have to load one of the `conntrack` modules. These are three kernel modules, so you have to load them with `modprobe` like this:

```
# modprobe -a ip_conntrack
# modprobe -a ip_conntrack_ftp ip_nat_ftp
# modprobe -a ip_conntrack_irc ip_nat_irc
```

You can configure these modules to load automatically when `iptables` loads. For passive FTP you'd add these lines to your `/etc/modules.conf` file:

```
post-install ip_tables /sbin/modprobe ip_conntrack_ftp
post-install ip_tables /sbin/modprobe ip_nat_ftp
```

Once the modules are loaded you can add two rules to your firewall's INPUT chain that look like these:

```
# iptables -A INPUT -m state --state NEW -i ! eth0 -j DROP
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

This sets the default behavior of incoming `NEW` packets to `DROP` unless they come from eth0, and creates a special rule for allowing `ESTABLISHED` and `RELATED` packets.

Network Address Translation (NAT)

As discussed earlier, Network Address Translation is a method of masquerading a large group of machines as a single IP address. It can also be used to masquerade multipe IP addresses for multiple machines, all from a single gateway. There are two types of NATs: Destination NATs (DNATs) and Source NATs (SNATs).

A DNAT is generally used to redirect requests for one resource to another location - to a proxy machine, for example. You could set up an HTTP proxy to filter all web traffic, or an SMTP proxy for e-mail.

A SNAT, on the other hand, changes the source address of outgoing traffic. A common use of this is IP masquerading, in which multiple machines share a single IP address assigned to the gateway. Under a SNAT configuration the following will happen:

- When a local machine sends a packet outside the network, the router changes the real source address into its own external IP address.

- When a remote machine replies, the router changes the packet address back to the local host's private IP address. It does this by comparing the port address in the packet contents to its own masquerading table. This way it can keep track of which internal host the packet should reach.

NAT rules are their own table, separate from the filter table. While the filter table uses the INPUT chain, the NAT table uses the PREROUTING, POSTROUTING and OUTPUT chains.

Implementing a NAT

To use NAT to redirect HTTP requests on `eth0` (both normal and secure) to a proxy server that filters web traffic, we'd do this:

```
iptables -t nat -A PREROUTING -p tcp -m multiport --dport 80,443 -i eth0 -j DNAT --to-
destination 10.0.3.15
```

Note that this should be on a single line. It starts by specifying the NAT table, then the PREROUTING chain, then the set of TCP ports. Then it calls for a match of multiple ports and specifies 80 and 443, the traditional HTTP and HTTPS ports. Then it specifies the interface on which to listen. Finally it uses -j to make an objective for the rule - to DNAT the packet to 10.0.3.15, the web proxy.

If you want an SNAT, you might do this:

```
iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to-source 10.0.3.14:1024-65535
```

With this rule in place, all outgoing traffic will have its source address changed to 10.0.3.14, and it will go out from the ports 1024 through 65,535. It will rotate ports as it passes packets.

IP Masquerading

If you want to set up IP masquerading, things get slightly more complex. You need to have the `conntrack` kernel modules to perform connection tracking, and thus you need them for masquerading. Make sure you use the modprobe command to add the modules to the kernel, or else configure `modules.conf` to add them as needed.

(ToDo: Figure out examples of masquerading).

35. Data Encryption

Most Internet users do not fully understand the importance of encrypting data transmissions. It is partly for this reason that encryption is taking a long time to become the rule rather than the exception. In the interests of rendering data private, of keeping passwords secret and of preventing data from being manipulated by malicious parties, encryption is gaining ground.

Encryption Methods

Encryption is usually performed by combining multiple techniques, including ..

- One-way hashes
- Random numbers
- Symmetric Algorithms (DES, 3DES, Blowfish, etc.)
- Asymmetric Algorithms (Public Key Encryption)
- Public Key Infrastructures
- Digital Certificates

Some GNU/Linux distributions, including Fedora Core, include implementations of encryption. The most popular are GNU Privacy Guard (GPG) and OpenSSL.

One-way Hashes

A one-way hash can be used to verify the integrity of information. A block of data is run through an algorithm that generates a fixed-length output called a *fingerprint*. Later, the data can be used to regenerate the fingerprint; statistically is is extremely unlikely that the same fingerprint can be generated using different data sources, so if the finger prints match you can be fairly certain the data has not been tampered with. The reason these fingerprints are called "one-way" hashes is that fingerprint cannot be used to guess at the contents of the original file.

One-way hashes are used by RPM to verify package integrity, and by the password system to store encrypted passwords. Many utilities are available at the command line to generate one-way hashes; a popular utility is `md5sum`, which can be used against a file to create an MD5 hash. Aside from MD5, other algorithms include SHA and CRC-32.

Random Numbers

It is very difficult to produce truly random numbers. Linux allows you to access pseudo-random numbers generated by an *entropy pool*. The entropy pool is built from data gathered via mouse movements, keyboard strokes and other user events. Between boots GNU/Linux runs the `random` System V script, which stores the entropy pool in the file `/var/run/random-seed` file.

There are two ways to access this pseudo-random data; both are device nodes. The `/dev/random` node is the best source, however it is limited in its usefulness because it ends when the entropy pool ends. For a larger pool your system can use `/dev/urandom`, which pulls data from the entropy pool and, when it is exhausted, creates its own pseudo-random numbers from an algorithm.

OpenSSL also includes a random number generation utility. Use this command:

```
$ openssl rand <number>
```

The algorithm will be applied to the number you supply. For a more printable block of text, add the `-base64` option.

Symmetric Encryption

Symmetric encryptions are based upon a single key that is used to both encrypt and decrypt data. The key

is generally a short passphrase. The resulting cyphertext can not easily be used to guess the contents of the original plain text.

Symmetric encryption can be used to create a one-way hash. This is th type of one-way hash used to create UNIX-style passwords. The password and a randomly-generated string called a *salt* are used by an algorithm to encrypt a string of zeros. The result, with the salt appended at the beginning, is stored as the encrypted password. When a user inputs the password later on, it is used with the salt in the same algorithm, and the result must match the string in the password field. Most Linux distributions use MD5 encryption to create very secure passwords.

Asymmetric Encryption

Asymmetric encryption is based upon a pair of keys - one public and one private. Because of this it is often called *public key encryption*. What one key encrypts, the other can decrypt, however it cannot be done without the pair. As their names suggest, the public key is publicly available and the private key is kept secret.

Imagine that Bob has generated a public/private key pair, and has published his public key. Alice can take that public key and use it to encrypt data intended for Bob. Only Bob can decrypt that data, because it can only be done with his private key.

More Asymmetric Encryption

Another important use of asymmetric encryption is digital signatures. Imagine that Bob creates a public/private key pair and publishes the public key. He can then encrypt a message to Alice. Alie can decrypt the message using the public key. This is the basis of the GNU Privacy Guard, as well as the older PGP protocol. In fact users of GPG often hold key signing parties in which they exchange GPG keys for future encrypted communications.

One very secure method would be for Bob to encrypt the data with his private key, then again with Bob's public key. Alice can then do the opposite decryption. This ensures that the data cannot be decrypted by anyone but Alice, and is a unique use of both types of asymmetric encryption.

Another use of this technology is a detached signature. Rather than encrypt data, a hash of it is created to accompany the data. It can then be used to verify its authenticity. If Bob creates a document, he can create a hash of it with a command like `md5sum`. Alice can use the resulting signature to ensure that the document is authentic. She can perform the same hash and compare the resulting signature with the one Bob published; if it matches, she can be certain that the document has not been altered since Bob created his signature.

Asymmetric encryption has one flaw: it requires that public keys be revealed. In our continuing example this means that Bob must publish his key so that Alice can access it and be sure that his messages and other data really come from him. But a third party could publish a false key, and send data on Bob's behalf. People with the false version of Bob's key would assume the data is from Bob.

Fingerprints

One way to avoid this would be for Bob to create a one-way hash of his public key, called a *fingerprint*. He could then publish this everywhere - in e-mail, on his website, and on electronic documents.

If he sends some of his data to Alice, Alice can then get his key to decrypt it from a public key server. In doing so, she can verify that his key is real by creating a fingerprint and comparing it to the fingerprint in Bob's correspondence.

Digital Certificate and Certificate Authorities

If Bob wants a better way to let people verify the validity of his key, he can use a public key infrastructure that *signs* his key. This third-party infrastructure, called a *certificate authority*, issues signed keys. The signature guarantees that the key comes via the trustworthy certificate authority. Since Alice trusts the

authority, she trusts the signed key, called a *certificate*.

Certificate authorities have their own private and public key pairs; their public key is referred to as a certificate authority certificate.

To use a authority's services, Bob could create his private and public key pair and share the public key with the authority. The authority will require Bob to prove his identity; he must also provide an expiration date for the certificate.

Next, the authority will combine this information with its own identity and create a one-way hash of the entire set of data. It will encrypt this one-way hash with its private key. This is the *detached digital signature*. This signature and the information it represents are the digital certificate. While the authority is the *issuer* of this certificate, Bob is the *owner*.

End users can then access these certificates from the owners. To verify that the owners are legitimate, the end users can check with the certificate authority to verify the public key. Most web browsers come with certificate authority certificates included, allowing one's browser to establish trusted communications with certificate authorities.

Make Your Own Digital Certificate

You can make your own digital certificates very easily, and set up your own certificate authority. Internet sites that use certificates use the X.509 format. This format requires and identity, and the identity generally consists of the country, province or state, organization name, a common name and and e-mail address.

Another set of instructions for making a certificate exist in this guide. ToDo: combine these instructions with those and make a pointer to them from here, keeping the "official" instruction here.

To make your certificate, use the `openssl` command. Star by generating a 1024-bit RSA public/private key pair:

```
# openssl genrsa -out server1.key.pem 1024
```

Next you have to generate a certificate signature request (CSR):

```
# openssl req -new -key server1.key.pem -out server1.csr.pem
```

This step will make a request customized to your key pair, and it will also prompt you for identity information.

If you want to use a certificate authority you can send this certificate signature request to them. They will return a signed certificate to you, which in his example you would save as `server1.crt.pem`. Presumably such an authority would verify your identity, since third parties will rely upon them to verify your key pair as legitimate. If you choose a reliable certificate authority, they'll grill you a bit.

Another option is to set up your own certificate authority, or not to use an authority at all. The former is gradually becoming an option, and may ultimately be a popular choice. The latter is popular as well - basically without any authority you are playing both the role of owner and of issuer. The approach still provides encrypted communication, but it is not verified by a trusted authority.

If the encryption is the main thing that matters to you, going without an authority is probably fine. You'll have to generate you own self-signed certificate using the `openssl` command with the `-x509` switch like this:

```
# openssl req -new -key server1.key.pem -out server1.crt.pem -x509
```

These files can be used with any web server that supports SSL, such as Apache.

Secure Communications with OpenSSH

The secure shell (SSH) protocol is an extremely popular protocol for secure communications. Most GNU/Linux users are familiar with it, but if you've either used or heard of it and think it's just encrypted telnet you only know a fraction of its potential.

SSH was designed to replace many older, insecure protocols. This includes `telnet`, `rlogin` and `rsh`, but also other protocols like `ftp`, `rdist` (remote file distribution across multiple hosts), `rcp` (remote copy) and `rsync` (remote file synchronization). As an added bonus it can be used to wrap other insecure protocols in a secure tunnel. The more you know about SSH, the better off you are.

Basic SSH Sessions

To connect to an SSH server, open a shell interface and type the `ssh` command with a hostname.

```
$ ssh times.minitrue.gov
```

This connect you to the remote server, albeit with your local username. If your username is different at the remote host you'll want to add it either like this:

```
$ ssh wsmith@times.minitrue.gov
```

or like this:

```
$ ssh times.minitrue.gov -l wsmith
```

When you first connect to a remote host you are asked whether to save the host's public key permanently. If you choose to do so, they key will land in your ~/.ssh/known_hosts file. One line of that file might look like this:

```
times.minitrue.gov,10.0.5.4   ssh-rsa   AAAAB3NzaC1yc2EAAAABIwAAAIEAt4/Kcuo4sfs847+Kmux
LuWYyG4bFJHUdlVcBItscyJu3X1DZ6t3xzkc3JgIIbYJhHD1G8GY0Zqe5Kpggh3pEkvdlAhHQPfS2Sp83T+
ktPrJRk+5kHYoxpKUUi+IDsQY4mcDKV3Bfp0w43TGhgyqB2/9inSG9/Hr12/juHxlSuZ0=
```

This is just the hostname, IP address, encryption type (in this case, RSA) and public key. If the remote server's key pair ever changes, SSH will hassle you about connecting. If you're sure the key change is not indicative of foul play (e.g., a spoofed server), simply delete the host manually from this file and try to connect again.

SSH operates on port 22 by default. The other parts of the protocol, discussed below, also use the port.

Remote Commands

SSH can be used to send commands to a remote server without having to log in. The advantage to this is that you can send these commands to multiple machines form a single location, and they are scriptable.

If you'd like to send a command to a remote machine, just tack the command onto the end of your SSH command. for example, if you'd like to see how long one of your servers has been running you can send it the `uptime` command.

```
$ ssh wsmith@speakwrite2 uptime
wsmith@speakwrite2's password:
 11:06am  up 208 days,  9:55,  1 user,  load average: 0.67, 1.38, 1.21
```

You can also have SSH send the output to the background rather than print it to the standard output. This requires the `-f` option, and comes in handy if you plan to launch an X application form a remote server and have it appear on your local machine. For example:

```
$ xhost + server2
$ ssh -f wsmith@server2 DISPLAY=desktop4.minitrue.gov:0 ethereal
```

```
$
```

This runs `ethereal` on `server2` but redirects it to `desktop4`'s X server. And it runs it in the background so you get your shell prompt back.

Sending remote commands with SSH is especially useful with key-based authentication (described later in this chapter), which eliminates the need to provide a password.

SFTP Sessions

SFTP is Secure FTP. It is a part of the SSH protocol and the `sshd` daemon can host an SFTP session. To start an SFTP session you'd do this:

```
$ sftp wsmith@times.minitrue.gov
```

You'll then be greeted with a session in which you can use all the basic FTP commands, like `cd`, `ls`, `lcd`, `lls`, `get` and `put`.

Optionally you can insert the `-C` option into the `sftp` command to add compression. Encrypting your session adds some bulk to your data, so this compression can be a welcome bandwidth helper. Keep in mind that less powerful machines may slow down under the weight of encryption and compression.

SCP Sessions

`scp` is the Secure Copy, and is a part of the SSH protocol. This extremely useful tools lets you copy a local file to a remote host securely, or vice-versa. For example, say you want to copy your password file to your home directory on another server. You could do this:

```
$ scp /etc/passwd wsmith@times.minitrue.gov:passwords
```

The password file will end up in user wsmith's home directory, under a subdirectory named `passwords`.

To make this even more useful you can recursively copy whole directories at a time with the `-r` option. The -r option works differently depending upon whether you decide to use a trailing slash at the end of your source directory name. For example, if you add a trailing slash like this ...

```
$ scp -r localdirectory/ remotehost:remotedirectory
```

... you'll copy the contents of the local directory into the remote directory, but you won't copy the remote directory itself. However, if you omit the slash like this ...

```
$ scp -r localdirectory remotehost:remotedirectory
```

You'll copy `localdirectory` to the remote host, where it will become a subdirectory of `remotedirectory`. However, if there is no such remote directory, the results will be identical to the slash example above.

The SSH Server

The SSH daemon that comes with Fedora Core and other GNU/Linux distributions is OpenSSH. The name of the daemon is `sshd`, and on Fedora Core systems it requires the `openssl` and `openssh-servers` RPM packages.

Some distributions, including Fedora Core, compile `sshd` with support for `libwrap`. If you are using `/etc/hosts.allow` and `/etc/hosts.deny` you'll need to add a line to allow incoming SSH connections in `hosts.allow`. It could look like this:

```
sshd: addresses-of-allowed-hosts
```

OpenSSH is equipped to handle a few different methods of authentication. These methods inlude secure passwords, RSA and DSA keys, Kerberos, s/Key and SecureID for expring tokens and one-time passwords, and key pairs.

Authentication Using Key Pairs

Key pairs allow you to log into a remote host or securely exchange files without using a password. To generate a key pair for encryption, use the `ssh-keygen` command, andswering a few basic questions along the way. Use a blank password and the default filename:

```
$ ssh-keygen -d
Generating public/private dsa key pair.
Enter file in which to save the key (/home/wsmith/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/wsmith/.ssh/id_dsa.
Your public key has been saved in /home/wsmith/.ssh/id_dsa.pub.
The key fingerprint is:
0e:46:89:3f:aa:e1:9d:73:60:a5:1f:2b:98:b2:0b:29 wsmith@localhost.localdomain
```

Transfer the file `id_dsa.pub` to the remote host, in the directory `~/.ssh/`. Rename it `authorized_keys2`, or if a file of that name exists, append its contents to the existing file.

Once you've done this you can SSH to the remote host without entering a password.

Securing Key Authentication

Of course, using key pair authentication without a passphrase on the key is potentially dangerous. True, you can script it. And it saves time. But it should only be done this way in a secure environment, such as between two very safe servers. In general practice it should be avoided because anyone who gains access to your account can log into the same systems as you.

One work-around is to use `ssh-agent`, which allows you to enter your key's passphrase once per session. Thereafter `ssh-agent` will provide it on your behalf. This can be a time-saver if you find yourself logging into the same remote hosts frequently.

An example of this would be to configure your X session to run `ssh-agent`. If you run `switchdesk` from the command line on a Fedora Core system you'll be able to choose your desktop environment. This will generate an `~/.Xclients-default` file. You can add this line to the top of the file (or replace the top line if there is already a shell there):

```
#! /usr/bin/ssh-agent /bin/bash
```

When you're in your next X session you'll be able to issue the `ssh-add` command into any terminal window. You'll be asked to use your passphrase once, but thereafter you'll be able to connect to select hosts without it.

RPM Package Security

RPM contains two security mechanisms to help you ascertain file integrity.

Package File Integrity

Fedora Core and other vendors sign RPM package files with private keys. Public keys are available for you to check their signatures. Fedora Core's public key is the RPM-GPG-KEY, distributed with any installer CD. To import this key into your GPG keychain, use this command:

```
# gpg --import RPM-GPG-KEY
```

Once this is installed you can use the `rpm` command to verify any package file:

```
# rpm --checksig <package-file>
```

Checking the signature verifies that the file is the same one originally created by the Fedora Core team.

Installed Package Integrity

Whenever you install a package, the RPM database stores a catalog of every file you have installed and an MD5 hash for each. You can then use the `rpm` command to verify a package:

```
# rpm --verify <package-name>
```

Or verify every package on your system:

```
# rpm --verify -a
```

Keep in mind that some files, such as configuration files, may have been changed deliberately.

Secure Tunnels

The `stunnel` utility can be used to create an encrypted "tunnel" between a program and the TCP networking layer. It is designed in such a way that the program is not aware that the encryption layer is even present.

Private Key and Certificate

`stunnel` requires that you create a private key and a self-signed certificate, much as you would for an SSL-enabled web server. For example:

```
# openssl req -new -newkey rsa:1024 -nodes -x509 -keyout key.txt -out cert.txt
```

You can then take the contents of the private key and certificate files you created and put them in the `stunnel` file `/usr/share/ssl/certs/stunnel.pem`. The file should then look more or less like this:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDuHB+D+IrVqRE+K2gqNx4HW5jWwV7MkGp4duSAvYhwO+XdSriY
ljYxfVvnCDmgdFHp9WGqhqdythfw/depySqXPJBwiEgRIsuHx4JTH0uOMy8g6tdW
d6OIWVF/A/BwYIBz7rGGNQUNO/sUifu8P7OcNHEadsCKx+tPQhi8VL1yXQIDAQAB
AoGBAKLo3OsnFp4egWyeuDMB6Oqx0b1FKf4d5Kqs9aPbiUj4wkmIgQns5AEyOrm0
F/FfBFhlCcBpRpGmYnUAyBgjSZ5D9pTqNcxa7L5acFubu1Y65u7BWEte24cYRBh7
4ZzHVuSgCFt6FD1krPrcuW+Ul5Qa5unAk4hyCqRIGK8xI1rBAkEA+WKzwBJrq+Ga
BQspMdpZS5RVh/m4/hzE/8pmX963Us9Apx6wxE5fpYa8WU8m1Z8kkmi3Naz4Pb3o
rde2dc4ZcQJBAPRs3D3YjvXAWhfFH8x1tiacmzWbj1rOH8SAiC8VtbyU8fLEeqMI
CfEsYZbf+iez2a0gOGoXLjrBKtgTPQBB0a0CQEno3br7+psrNnluMFxNCQ/kO2Ec
Op5dKmlgs+yP1icds1yUNPFJBQHKp8GUOr8u9ijKDhpDVCMASJhVCQdq+JECQQDE
ki/lxR3j4hgSecQDCL+++b+RM1Ps8Ux3Ge9r/oez7A1Q8la2eqaUQN8TGXnzKB7t
rrXwDEhvlB5Cu+YZKNXZAkAXio8488L+C1geskb+jALrZXwTvSTvGn8LJ7MjbgOl
MNd+LZy0yPjfzcqnGLx6cOGa8tb5RKsvdZA9/ibNDQGC
-----END RSA PRIVATE KEY-----

-----BEGIN CERTIFICATE-----
MIIDJDCCAo2gAwIBAgIBADANBgkqhkiG9w0BAQQFADBwMQswCQYDVQQGEwJVUzEW
MBQGA1UECBMNTWFzc2FjaHVzZXR0czESMBAGA1UEBxMJV2F0ZXJ0b3duMRUwEwYD
VQQKEwxFZEhvbGRlbi5jb20xHjAcBgkqhkiG9w0BCQEWD2VkQGVkaG9sZGVuLmNv
bTAeFw0wMzA3MjIwMTUxNTNaFw0wMzA4MjEwMTUxNTNaMHAxCzAJBgNVBAYTAlVT
MRYwFAYDVQQIEw1NYXNzYWNodXNldHRzMRIwEAYDVQQHEwlXYXRlcnRvd24xFTAT
BgNVBAoTDEVkSG9sZGVuLmNvbTEeMBwGCSqGSIb3DQEJARYPZWRAZWRob2xkZW4u
Y29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDuHB+D+IrVqRE+K2gqNx4H
W5jWwV7MkGp4duSAvYhwO+XdSriYljYxfVvnCDmgdFHp9WGqhqdythfw/depySqX
PJBwiEgRIsuHx4JTH0uOMy8g6tdWd6OIWVF/A/BwYIBz7rGGNQUNO/sUifu8P7Oc
NHEadsCKx+tPQhi8VL1yXQIDAQABo4HNMIHKMB0GA1UdDgQWBBThOjF3PDconFy9
l/zwUoCWYvBQxTCBmgYDVR0jBIGSMIGPgBThOjF3PDconFy9l/zwUoCWYvBQxaF0
pHIwcDELMAkGA1UEBhMCVVMxFjAUBgNVBAgTDU1hc3NhY2h1c2V0dHMxEjAQBgNV
BAcTCVdhdGVydG93bjEVMBMGA1UEChMMRWRIb2xkZW4uY29tMR4wHAYJKoZIhvcN
```

```
AQkBFg9lZEBlZGhvbGRlbi5jb22CAQAwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0B
AQQFAAOBgQDB5cNlogYCKNmsYHq2O+LPj481MwWSdDrH2iSJMej+GE+Fdw37ICWt
P+4ygcbn/irFtfRLQY5msdqC4qsPa8D8K69ZIVjrIF5EXiLPA9hL1zS4+wTE6CMW
+yKnG5Em4T77Iwhkt6XU5RM4QzYC1V74mffP7rqUYFlPQAEdIiexRw==
-----END CERTIFICATE-----
```

It should be owned by the superuser and have the mode 600.

Wrapping a Service

A daemon running, such as an insecure telnet server, will generally listen to its nativ port. For telnet this is port 23. But the daemon can be forced to listen to a secure port generated by stunnel, rather than to its native port through a process called *wrapping*. A telnet daemon listening to port 23 can be wrapped to an `stunnel` port like this:

```
# stunnel -d localhost:5023 -r localhost:23
```

The telnet daemon, thinking it is listening to port 23, will instead listen to port 5023. On 5023 an SSL-enabled telnet client can then connect and communicate with the standard telnet daemon.

Wrapping a Client

Of course to connect to the server above you'll need an SSL-enabled client. Most telnet clients are not SSL-aware, so you can use `stunnel` on the client to create a complementary tunnel. To make a tunnel for the telnet client you'd do something like this:

```
# stunnel -c -d localhost:8023 -r remote-server:5023
# telnet localhost 8023
```

Your local loopback connection to port 8023 will be sent by `stunnel` to port 5023 on the telnet server.

Wrapping X

One great use of `stunnel` is to securely launch remote X applications and have them appear on your local desktop. This would be hard to configure if it weren't automatic. SSH is smart enough that it will build a secure tunnel for X whenever you establish an SSH session. You can SSH to a remote machine and launch a graphical program, such as Mozilla. Because SSH is smart enough to tunnel X sessions, Mozilla will appear on your local X server.

In fact, you can also SSH through multiple hosts to launch remote programs. So you can SSH from Computer A to Server B, and from Server B to Server C, and launch a graphical program on Server C. It will appear right in front of you on Computer A because C will tunnel it to B, which will tunnel it to A. The only caveat of wrapping an X session is that the remote machine and any relay machines must all have the `pam_xauth.so` module installed in PAM. If any remote machines lack this module, which in Fedora is in the `xauth` RPM, the application will fail to launch.

36. Troubleshooting

Try the easy things first

There is no perfect science for dealing with a problem on any computer system. However, if there must be a Rule #1 for troubleshooting, let it be "Try the easy things first." With your newfound GNU/Linuc expertise there are many deep and complex theories you can test, but if the problem with the mail server ends up being that the Sendmail daemon was not running, you'll feel pretty dumb about the hours you spent reconfiguring the firewall settings.

Check the logs

Look at your logs first. Often they'll contain pretty obvious details of why a program or service failed. If the problem is in a configuration file, it's still best to look at the logs first. A syslog error message might even specify the line in the configuration file that is mis-typed.

For BIND, DHCP, NFS and many other services, check `/var/log/messages`.

For Sendmail, IMAP and POP daemons, check `/var/log/maillog`.

For Apache check the logs in the `/var/log/httpd/` directory.

For Samba, Check the logs in the `/var/log/samba/` directory.

Debug

Many programs contain debugging options; check out the manual page for your program for details. Launching a program in debugging mode can reveal a great deal about what's going awry. Also, some programs have the ability to examine their own configuration files and verify their syntax.

X Window System

X often has problems, but fortinately UNIX systems are not completely reliant upon their graphical interfaces. You're free to leave X by switching to another virtual console - type `Ctrl-Alt-F1` to log into the first virtual console (X runs in console 7). Then you can log in and troubleshoot; while you have a working shell you might consider switching to run level 3 by typing `init 3`. In doing so you'll shut off X completely.

Of course if X gets really nasty and won't let you switch to a separate virtual console you're best rebooting into run level 3. At LILO's prompt you'd type linux 3. At GRUB's interface you'd edit the command to put a 3 after the kernel filename.

Once you get a command line, running X with the `-probeonly` option might tell you a great deal about the problem.

Many times the problem will be related to disk space, either real (full disk or partition) or policy-related (quota exceeded). Checking the user's quota with the `quota` command or running `df` will tell you if the problem is related to the inability to write to the disk.

If the X Font Server fails to run, X will also fail. So if you can't get X to start, try starting xfs. IF a font directory is corrupt, the `mkfontdir` command can fix it.

Services

Again, refer to the log files if a service fails to load. Services that allow logins can be inoperable if the PAM subsystem is misconfigured; running setup can help you reconfigure PAM appropriately. Services

that use the network can experience problems if the network is not functioning correctly. It is sometimes useful to test the service from the local loopback address 127.0.0.1. If it answers properly, the problem is related to the network: a firewall, a bad network cable, a broken NIC, a DNS lookup problem, a routing error. Or a configuration file problem - for example, Sendmail has been configured to reject outside hosts by default, and a configuration line in `/etc/mail/sendmail.mc` must be commented out for it to work.

Networking Issues

DNS is the most mission-critical service on the Internet. Name resolution or reverse-lookup problems can lead to many other issues. Probably the best tool for doing DNS lookup testing is `dig`. Make sure it's installed on your systems as a matter of routine. `nslookup` will do reverse-lookup testing, but is deprecated and may be removed from Fedora Core.

The `ifconfig` command ill tell you the system's IP configuration, and `netstat -r` and `netstat -rn` will both tell you routing table of the system. If you're not sure that your broadcast address and netmask are correct, `ipcalc` can check them for you.

Boot Errors

Determining the cause boot problem is as simple as knowing what happens when GNU/Linux boots, and figuring out how far it is getting.

If you get an "Operating System not Found" error, make sure you're not booting to a non-bootable floppy. Then make sure your boot loader is installed properly. A mis configured or corrupt boot loader cal lead to a "not found" error, or can render your boot loader incapable of loading your operating system. The recovery tools on the installation CD can help you fix your boot loader.

If the kernel fails to load, or if it loads just a bit before a kernel panic message appears, then either the boot loader is loading it improperly (with the incorrect parameters) or the kernel itself is corrupt. The only way to fix this is to reinstall the kernel, of boot to another kernel if you happen to have more than one installed. Reinstallation of the kernel RPM is covered a bit later in the CD recovery environment section.

If the kernel loads but the system panics or crashes while loading the root filesystem or while running init, there are four possible problems: the boot loader may be mis configured, or the `/etc/inittab` file may be misconfigured, or `/sbin/init` may be corrupt, or the root filesystem itself may be corrupt.

If the problem is with `init` you'll have to reinstall the RPM or fix the `inittab` file. Filesystem checking is covered in the next section.

If the kernel loads and `rc.sysinit` is interrupted, the Bash shell may be missing or corrupted, or perhaps the `/etc/fstab` file is the problem.

Filesystems

When your system crashes it's not unusual to experience filesystem corruption. To force users to run adequate checking after a crash, ext2 filesystems are marked as "dirty" the moment they are mounted in read-write mode. Only when an unmount command is run on them are they marked "clean."

When data in memory prevented from being written to disk due to a crash, the filesystem can become corrupt. Since the filesystem will be marked "dirty" it will be checked upon reboot. The problem on an ext2 filesystem is that it can take a long time to check a large partition for errors. ext3 solves this by introducing a journal, which keeps track of which files are written. Filesystem checking has been significantly improved since the introduction of ext3, since `fsck` only needs to check files recorded in the journal, rather than the whole filesystem.

If the root filesystem has a journal, and the Linux kernel detects corruption (i.e., if / is marked as "dirty"), it will examine the journal after it mounts / in read-only mode, and repair it if necessary.

Later on, if the `rc.sysinit` script detects corruption, it will flash a prompt that says "Press Y within 5 seconds to force file system integrity check..." If you press Y, `rc.sysinit` will run a check on very filesystem mentioned in `/etc/fstab`. For this reason it's probably overkill.

If you ignore the prompt, `rc.sysinit` will run an `fsck` on any filesystem that has an even number in the sixth (last) column of `/etc/fstab`.

If a file system is very badly corrupted `fsck` will fail. You should then run it manually. For ext2 and ext3 filesystems `fsck` runs `e2fsck`. This should only be run on a filesystem that is mounted as read-only, or not mounted at all. If there is a problem with the superblock, which is a part of the filesystem that contains a general description of it, you can use the -b option to have `e2fsck` use an alternate copy of the superblock (a copy is kept at the beginning of each block group). Or you can use the `-y` option to have `e2fsck` answer "yes" to every prompt it would normally give the user.

Rescue Run Levels
The rescue run levels are 1 and single-user mode. They differ slightly.

Run Level 1 - Runs `/etc/rc.d/rc/sysinit`, but then proceeds to run the scripts in `/etc/rc.d/rc1.d`.

Single user mode - Also called S, or s. Only runs `/etc/rc.d/rc.sysinit`. It ill therefore work even if `/etc/inittab` file is missing or corrupt.

Emergency mode - Runs no scripts at all.

You can also press I during boot when you see the prompt that says "Press I for interactive startup." This will allow you to approve and veto services sequentially.

The Boot Floppy

You can make a boot floppy during system installation, or use the `mkbootdisk` command. Either way the floppy you create will be unique to your system. If you use SCSI disks or a RAID array, special kernel modules will be integrated into the initial ramdisk to support this. If you use IDE disks, the boot disk you create will be specific to IDE systems with the same root parition (e.g., `/dev/hda2`). In the latter case you can always port an IDE boot disk between IDE systems by passing the `root=(device-name)` parameter at the LILO prompt.

CD Rescue Environment

If the root filesystem is corrupt or otherwise unavailable, you must use the rescue environment on the installation CD to fix GNU/Linux. The rescue environment is a scaled-down version of the operating system that contains the command line tools necessary to check your media, mount volumes, install software and so forth.

To get into the rescue environment you can simply boot your installation CD and type "`linux rescue`" at the LILO boot prompt.

If you can't boot your CD for some reason (as would be the case if you have an older system), boot a floppy made from the `bootdisk.img` image on the installation CD, then and type "`linux rescue`" at the prompt. The bootable floppy will find the CD drive and initiate the rescue environment.

Furthermore, if you lack even a CD drive you can boot floppy disks made from the `bootdisk.img` and `drvnet.img` images. When you type "`linux rescue`" you'll be prompted to supply an NFS server's

location, from which you can start the environment over the network.

It can actually be advantageous to do it over the network. In most cases a network share will be faster than a CDROM drive.

Utilities

When you get into your rescue environment, you'll be greeted with a shell. From this prompt you can run `mount`, `fsck`, `fdisk` and `df`. There are also some less common utilites like `mkfs` to make a filesystem, `mknod` to make a device node, `mkswap` to create a swap space and `mkraid` to enable software RAID. And you get basic utilities like the `vi` text editor, `tar`, `cpio`, `gzip`, `dd`, `rpm` and `chroot`.

The rescue environment is network-aware, so network utilities like `ping`, `ifconfig`, traceroute and `ftp` also work. You can also mount NFS shares.

The environment logs to the file `/tmp/syslog`.

The Filesystem

The filesystem available in the rescue environment is virtual - it exists only in memory. But during boot the environment will attempt to mount your entire *real* filesystem under `/mnt/sysimage`. This reconstruction of the system can hang the environment if the filesystem is damaged - and a damaged filesystem is one reason you might need the rescue environment.

So if your system hangs, don't panic. Reboot and at the prompt type "`linux rescue nomount`" to start the rescue environment without the filesystem. Then you can use the `mount` command to mount your partitions, and a `mount` error message will tell you more about what's wrong with it. Don't bother trying to mount all of your filesystems: the `mount` command on its own will reconstruct everything if it can.

If you need to access a device for which there is no node in the rescue environment you can create it with the `mknod` command. You may have to find out the major and minor numbers of the device you need to create, but for the most common devices the command will be smart enough to do this automatically. For example, the command `mknod /dev/fd0` will create a floppy device without any further input required.

RPM Packages

If you're fortunate enough to get your filesystem mounted under `/mnt/sysimage` you can add, remove, freshen or query RPMs. Use the `--root` option to specify your system's root directory.

```
# rpm -ivh --root /mnt/sysimage /mnt/source/Fedora/RPMS/package.rpm
```

37. Bits and Bobs (to be integrated later)

Making Driver Disks from Images

The `dd` command is a Data Dump. It is meant to convert and copy a file, and often will have an input file (written `if`), and an output file (`of`). The following command will use `dd` to send the raw data in an image file to the floppy device, creating a bootable floppy:

```
dd if=bootdisk.img of=/dev/fd0
```

Of course you can do the same thing without the `if` and `of` by using redirects:

```
dd < bootdisk.img > /dev/fd0
```

Or just forget about dumping data altogether, and catenate:

```
cat bootdisk.img > /dev/fd0
```

Customizing the UI

Adding a Message of the Day (MOTD)

Anything you place in the file `/etc/motd` will appear both a local or remote console after login for all users. So you can add a message to tell users who log on that there may be problems with the system if you are doing maintenance, or just give them general notifications or instructions

Changing the Login Banner

The login banner is contained in the file `/etc/issue`, however Fedora Core and similar systems reset this value each time the system boots. However, you can add a few lines to `/etc/rc.d/rc.local` to overwrite this file with your own custom content. Open the `rc.local` script in a text editor and find the line that says touch `/var/lock/subsys/local`. Before that line you can insert your own login banner. For example:

```
echo "Welcome to \n" > /etc/issue
echo "System last rebooted at '/bin/date'" >> /etc/issue
```

whatis

The `whatis` command can be used to get information on what a command does. It runs from a database and can be refreshed with the `makewhatis` command. It is run daily by `cron`.

38. Useful Online Resources

General GNU/Linux Sites

The Linux Documentation Project - *http://www.tldp.org/*

39. Bibliography

General Sources Used Throughout the Guide

Red Hat, Inc. *RH133: Red Hat Linux Systems Administration.* Version RH133-7.3-20020514. Red Hat, 2002. (Available as part of Red Hat's training classes.)

Red Hat, Inc. *RH253: Red Hat Linux Networking and Security Administration.* Version RH253-7.3-20020528. Red Hat, 2002. (Available as part of Red Hat's training classes.)

Chapter 1: Introduction

Stone, Mark. "Real Desktop Linux, Part II." Online at *http://www.onlamp.com/pub/wlg/4004*. O'Reilly, Inc., December, 2003.

Chapter 12: Using NIS for Authentication

Kukuk,Thorsten. "The Linux NIS(YP)/NYS/NIS+ HOWTO." Online at *http://www.linux-nis.org/*. v1.2, 4 August 2002

Chapter 20: Understanding and Compiling the Linux Kernel

von Hagen, William. "Migrating to Linux Kernel 2.6." In Linux Devices at *http://www.linuxdevices.com/articles/AT3855888078.html*. February 6, 2004.

Chapter 22: Mail Services

University of Illinois at Urbana-Champaign. "Filtering mail with Procmail" In CITES website at *http://www.cites.uiuc.edu/procmail/filters.html.* March 10, 2004.

Chapter 27: VNC: Virtual Network Computing

AT&T Laboratories, Cambridge. "Making VNC more secure using SSH." On AT&T's Research site at *http://www.uk.research.att.com/archive/vnc/sshvnc.html*. May 13, 2004.

Chapter 28: Samba: Opening Windows to a Wider World

Syroid, Tom. "Using Samba as a primary domain controller." Online at *http://www-1.ibm.com/servers/esdd/tutorials/samba.html*. IBM website tutorials, 2002.

Microsoft, Inc. "How to Write an LMHOSTS File for Domain Validation and Other Name Resolution Issues" Microsoft Knowledge Base Article - 180094. Online at *http://support.microsoft.com*. Microsoft Knowledge Base, 2003.

Sutherland, Ronald Steven. "Printing to PDF/JPG/PNG" Online at *http://epccs.com/indexes/Documents/VirtualPrinting/*, 2004.

Chaper 30: Using LDAP

Ippolito, Greg. "YoLinux LDAP Tutorial: Deploying OpenLDAP - Directory Installation and configuration (V1.2 / 2.x)." Online at *http://yolinux.com/TUTORIALS/LinuxTutorialLDAP.html*. YoLinux Information Portal, 2003.

Syroid, Tom. "Build an LDAP Address Book." Online at *https://www6.software.ibm.com/developerworks/education/l-ldap/?x=39&y=10*. IBM website tutorials, 2003.

Frisch, Æleen. "Top Five Open Source Packages for System Administrators #4: LDAP" Online at *http://www.onlamp.com/pub/a/onlamp/2002/10/17/essentialsysadmin.html?page=1*. O'Reilly & Associates, Inc, 2002.